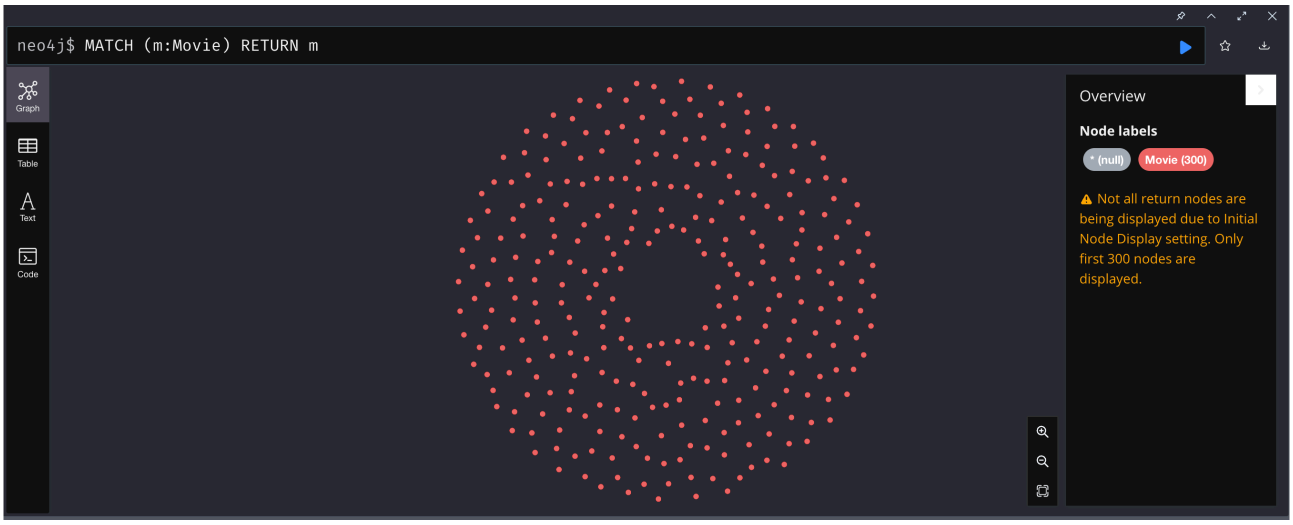
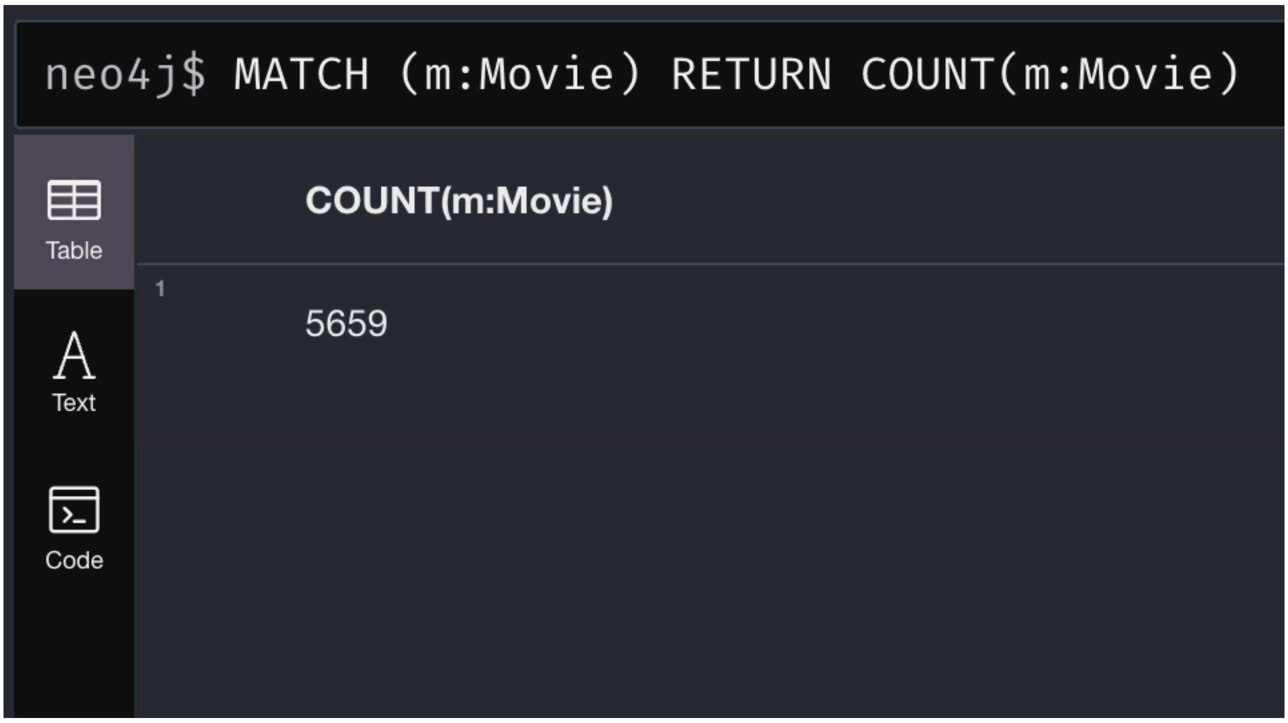
CYPHER QUIERIES:

```
MATCH (m:Movie) RETURN m
MATCH (m:Movie) RETURN COUNT(m:Movie)
MATCH (a:Actor) RETURN a

MATCH (a:Actor)-[:ACTED_IN]->(m:Movie)
RETURN a.Actor1, a.Actor2, a.Actor3, m.Name

MATCH (m:Movie)<-[:ACTED_IN]-(a:Actor)
RETURN m, a

MATCH (m:Movie {Name: "Indian"})<-[:ACTED_IN]-(a:Actor)
RETURN a

MATCH (m:Movie)<-[:ACTED_IN]-(a:Actor)
WHERE a.Actor1 = 'Sunny Deol' AND NOT m.Name = 'Indian'
RETURN m

MATCH (m:Movie)<-[:ACTED_IN]-(a:Actor)
WHERE a.Actor1 = 'Sunny Deol' AND NOT m.Name = 'Indian'
RETURN m
ORDER BY m.Rating DESC

MATCH (g:Genre) RETURN g
MATCH p=()-[m:BELONGS_TO]->() RETURN p LIMIT 2225

MATCH (m:Movie {Name: "Houseful"})-[:BELONGS_TO]->(g:Genre {Name: "Drama"})<-
[:BELONGS_TO]-(r:Movie)
WHERE m <> r
RETURN r.Name, r.Year, r.Rating, r.Genre LIMIT 10
```



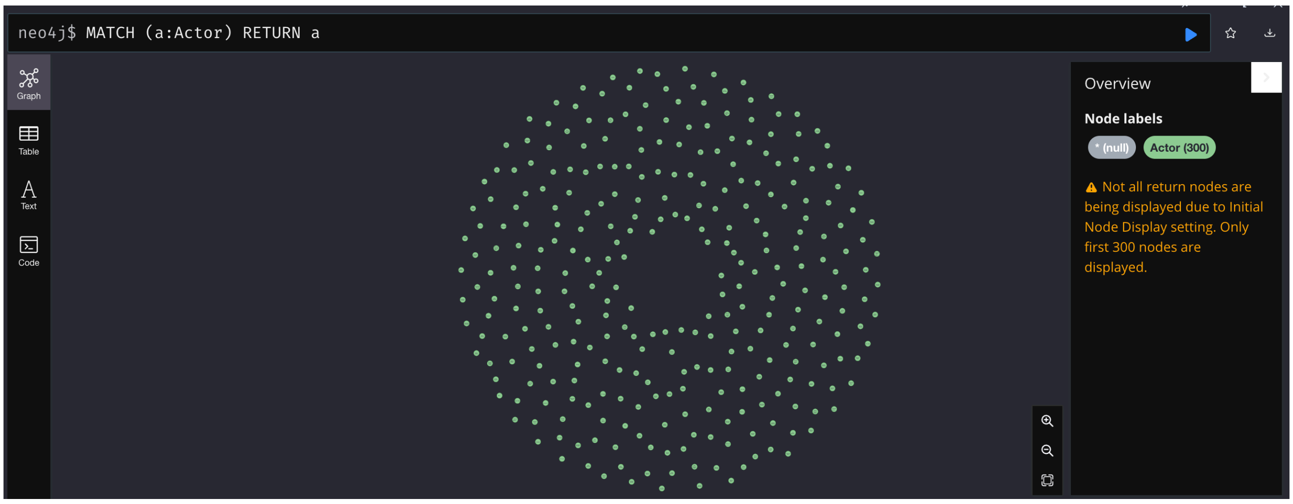
```
neo4j$ MATCH (m:Movie) RETURN m
```

⚠ Not all return nodes are being displayed due to Initial Node Display setting. Only first 300 nodes are displayed.



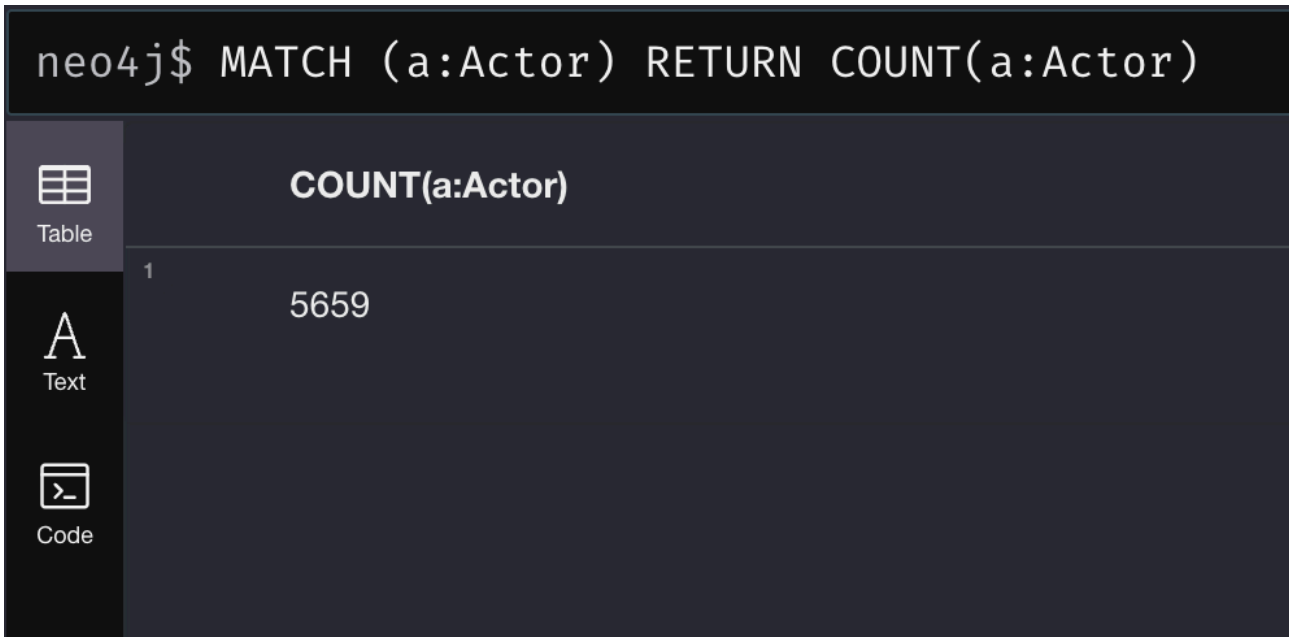
```
neo4j$ MATCH (m:Movie) RETURN COUNT(m:Movie)
```

	COUNT(m:Movie)
1	5659



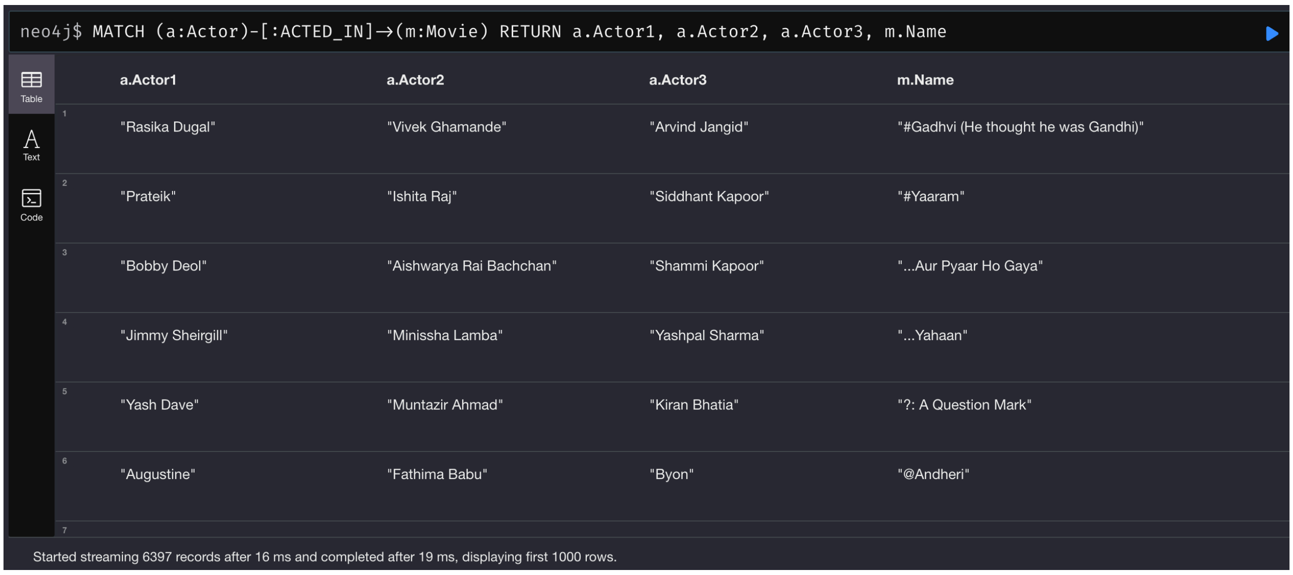
```
neo4j$ MATCH (a:Actor) RETURN a
```

⚠ Not all return nodes are being displayed due to Initial Node Display setting. Only first 300 nodes are displayed.



```
neo4j$ MATCH (a:Actor) RETURN COUNT(a:Actor)
```

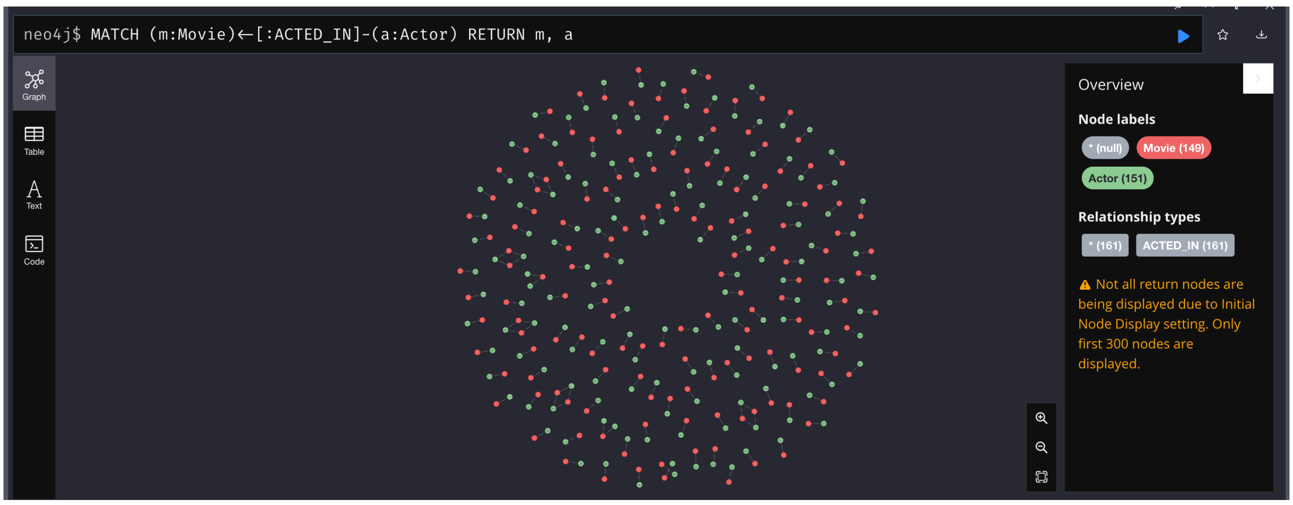
	COUNT(a:Actor)
1	5659



```
neo4j$ MATCH (a:Actor)-[:ACTED_IN]→(m:Movie) RETURN a.Actor1, a.Actor2, a.Actor3, m.Name
```

	a.Actor1	a.Actor2	a.Actor3	m.Name
1	"Rasika Dugal"	"Vivek Ghamande"	"Arvind Jangid"	"#Gadhvi (He thought he was Gandhi)"
2	"Prateik"	"Ishita Raj"	"Siddhant Kapoor"	"#Yaaram"
3	"Bobby Deol"	"Aishwarya Rai Bachchan"	"Shammi Kapoor"	"...Aur Pyaar Ho Gaya"
4	"Jimmy Sheirgill"	"Minissha Lamba"	"Yashpal Sharma"	"...Yahaan"
5	"Yash Dave"	"Muntazir Ahmad"	"Kiran Bhatia"	"?: A Question Mark"
6	"Augustine"	"Fathima Babu"	"Byon"	"@Andheri"
7				

Started streaming 6397 records after 16 ms and completed after 19 ms, displaying first 1000 rows.



```
neo4j$ MATCH (m:Movie)←[:ACTED_IN]-(a:Actor) RETURN m, a
```

Overview

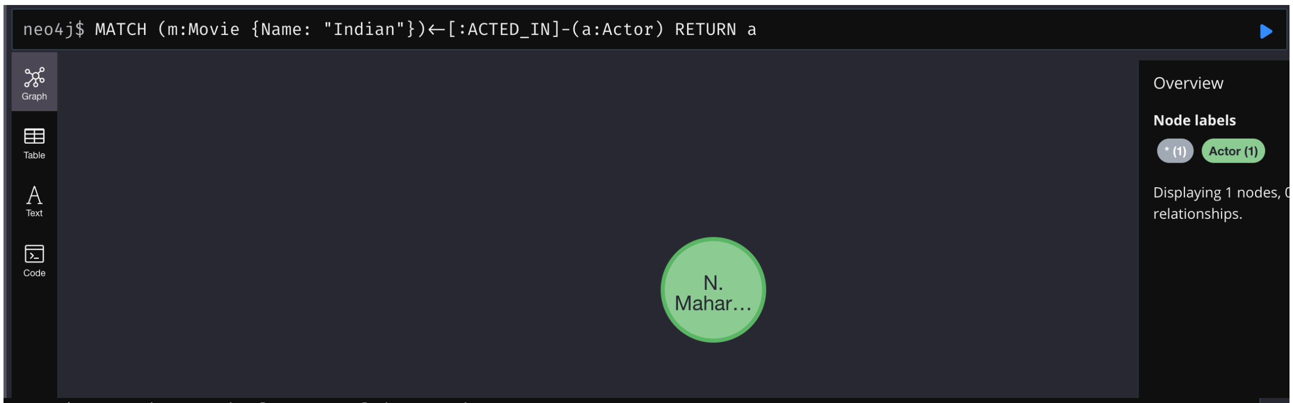
Node labels

* (null) Movie (149) Actor (151)

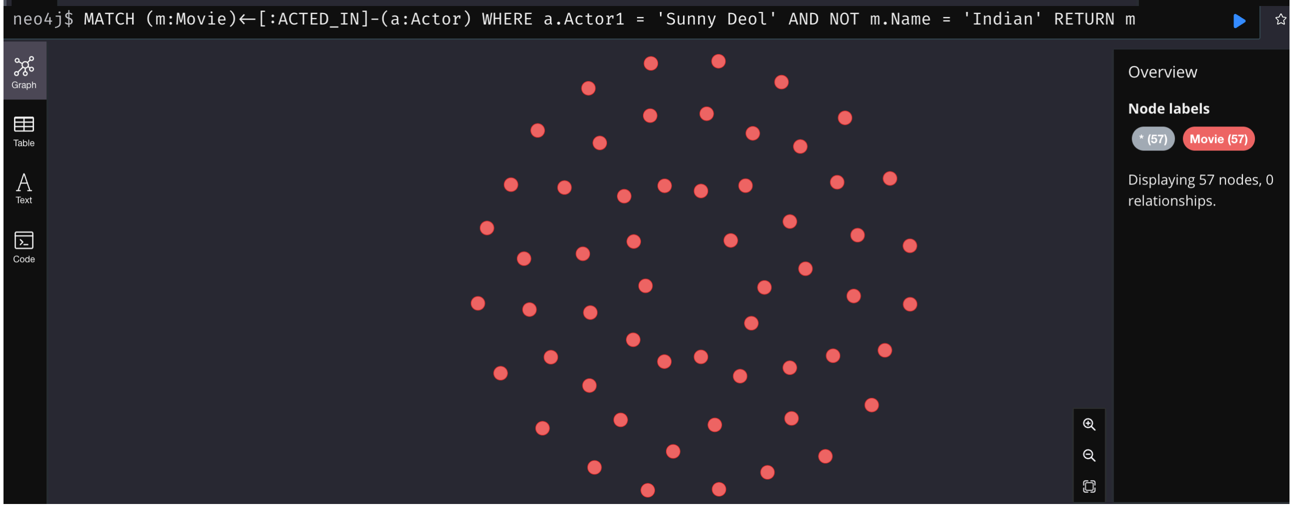
Relationship types

* (161) ACTED_IN (161)

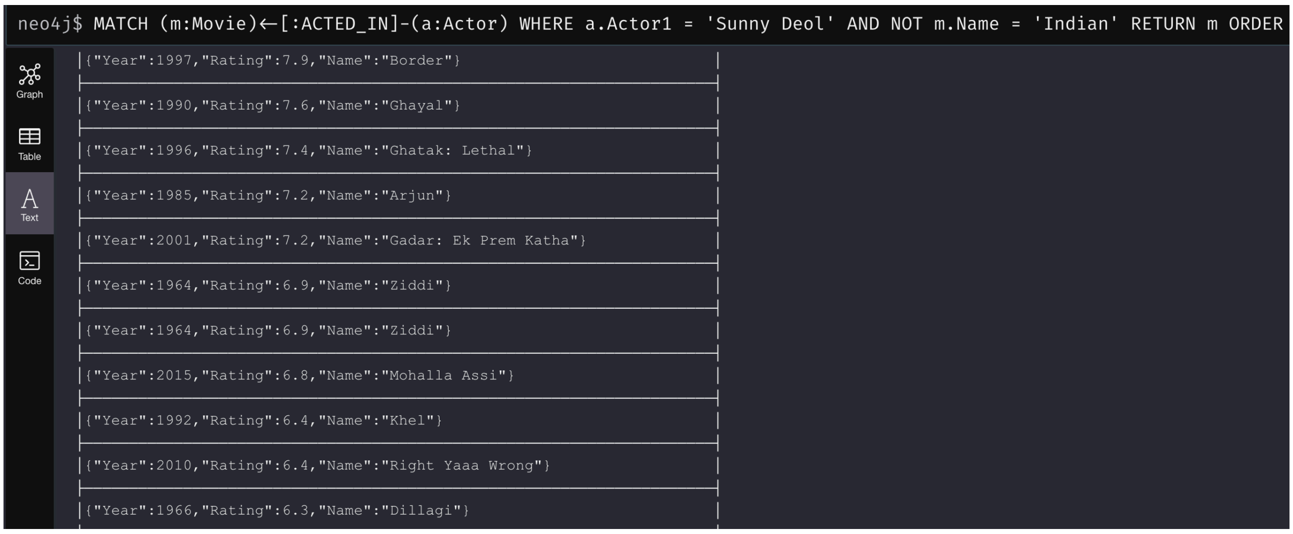
Not all return nodes are being displayed due to Initial Node Display setting. Only first 300 nodes are displayed.



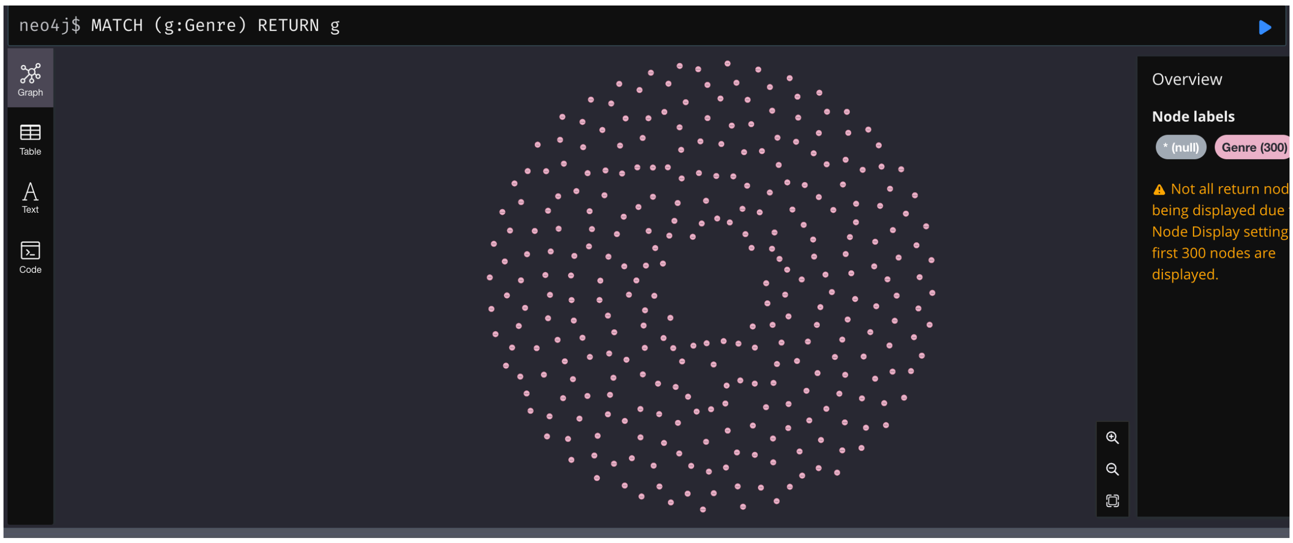
```
neo4j$ MATCH (m:Movie {Name: "Indian"})←[:ACTED_IN]-(a:Actor) RETURN a
```



```
neo4j$ MATCH (m:Movie)←[:ACTED_IN]-(a:Actor) WHERE a.Actor1 = 'Sunny Deol' AND NOT m.Name = 'Indian' RETURN m
```



```
neo4j$ MATCH (m:Movie)←[:ACTED_IN]-(a:Actor) WHERE a.Actor1 = 'Sunny Deol' AND NOT m.Name = 'Indian' RETURN m ORDER
│{"Year":1997,"Rating":7.9,"Name":"Border"}                         │
├───────────────────────────────────────────────────────────────────┤
│{"Year":1990,"Rating":7.6,"Name":"Ghayal"}                         │
├───────────────────────────────────────────────────────────────────┤
│{"Year":1996,"Rating":7.4,"Name":"Ghatak: Lethal"}                 │
├───────────────────────────────────────────────────────────────────┤
│{"Year":1985,"Rating":7.2,"Name":"Arjun"}                          │
├───────────────────────────────────────────────────────────────────┤
│{"Year":2001,"Rating":7.2,"Name":"Gadar: Ek Prem Katha"}           │
├───────────────────────────────────────────────────────────────────┤
│{"Year":1964,"Rating":6.9,"Name":"Ziddi"}                          │
├───────────────────────────────────────────────────────────────────┤
│{"Year":1964,"Rating":6.9,"Name":"Ziddi"}                          │
├───────────────────────────────────────────────────────────────────┤
│{"Year":2015,"Rating":6.8,"Name":"Mohalla Assi"}                   │
├───────────────────────────────────────────────────────────────────┤
│{"Year":1992,"Rating":6.4,"Name":"Khel"}                           │
├───────────────────────────────────────────────────────────────────┤
│{"Year":2010,"Rating":6.4,"Name":"Right Yaaa Wrong"}               │
├───────────────────────────────────────────────────────────────────┤
│{"Year":1966,"Rating":6.3,"Name":"Dillagi"}                        │
```



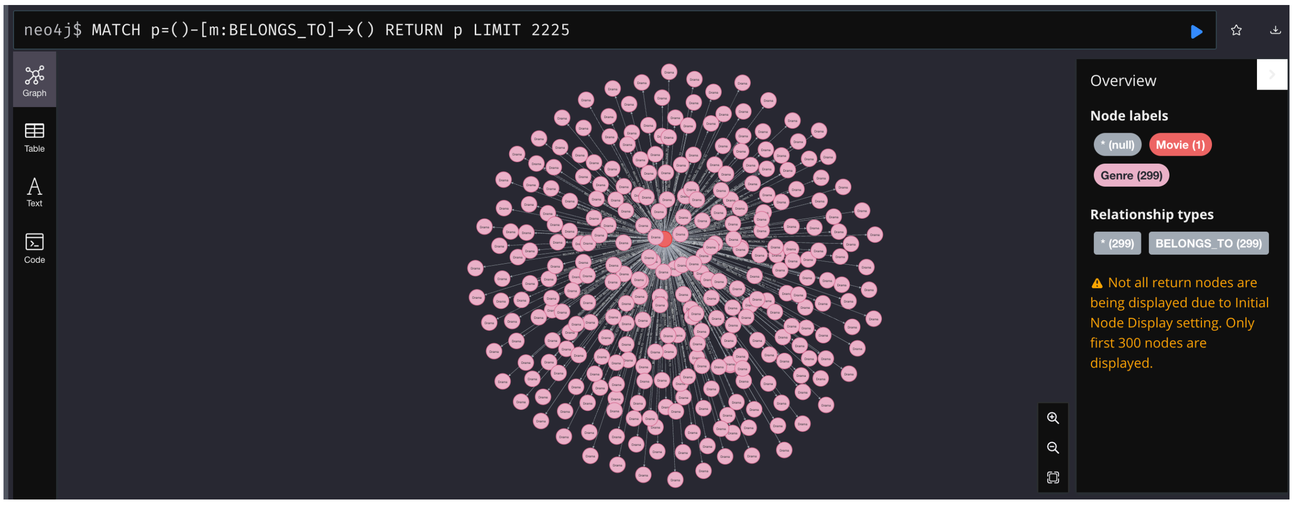
```
neo4j$ MATCH (g:Genre) RETURN g
```

Overview

Node labels

* (null) Genre (300)

⚠ Not all return nod being displayed due Node Display setting first 300 nodes are displayed.



```
neo4j$ MATCH p=()-[m:BELONGS_TO]→() RETURN p LIMIT 2225
```

Overview

Node labels

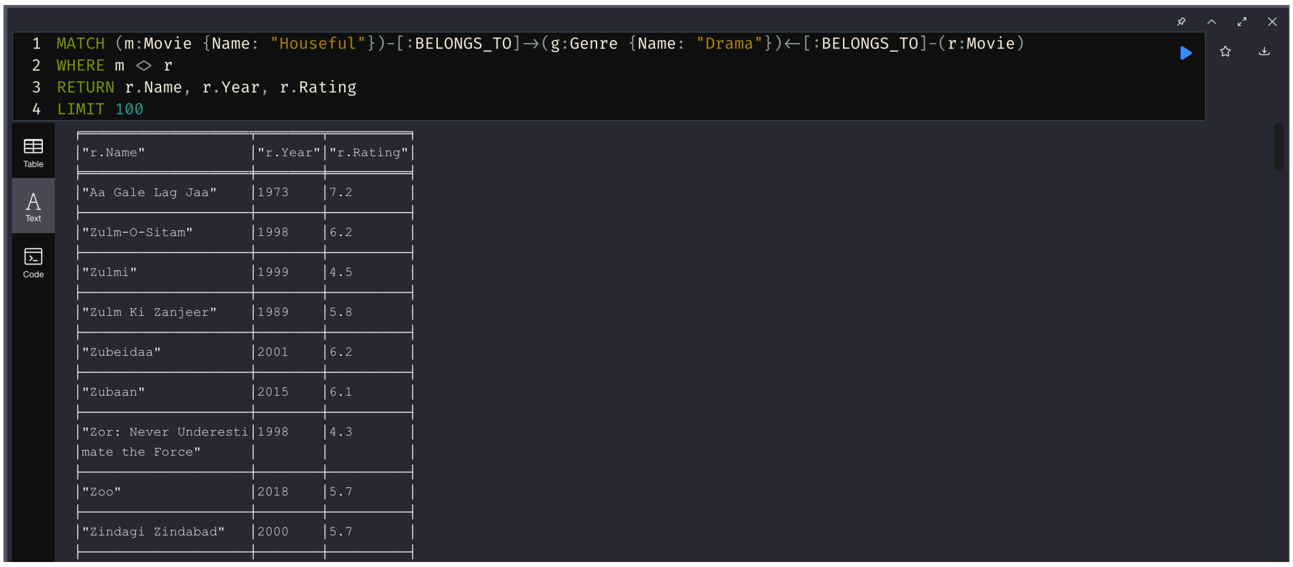
* (null) Movie (1)

Genre (299)

Relationship types

* (299) BELONGS_TO (299)

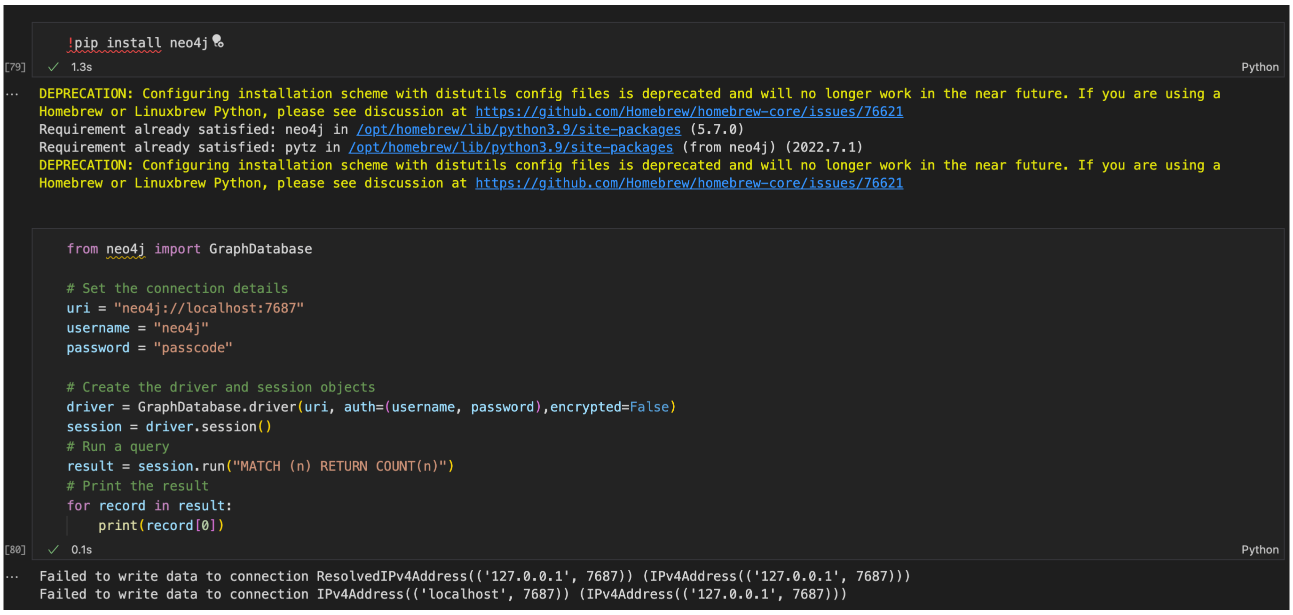
⚠ Not all return nodes are being displayed due to Initial Node Display setting. Only first 300 nodes are displayed.



```
MATCH (m:Movie {Name: "Houseful"})-[:BELONGS_TO]→(g:Genre {Name: "Drama"})←[:BELONGS_TO]-(r:Movie)
WHERE m <> r
RETURN r.Name, r.Year, r.Rating
LIMIT 100
```

```
╒══════════════════════╤══════════╤══════════╕
│"r.Name"              │"r.Year"  │"r.Rating"│
╞══════════════════════╪══════════╪══════════╡
│"Aa Gale Lag Jaa"     │1973      │7.2       │
├──────────────────────┼──────────┼──────────┤
│"Zulm-O-Sitam"        │1998      │6.2       │
├──────────────────────┼──────────┼──────────┤
│"Zulmi"               │1999      │4.5       │
├──────────────────────┼──────────┼──────────┤
│"Zulm Ki Zanjeer"     │1989      │5.8       │
├──────────────────────┼──────────┼──────────┤
│"Zubeidaa"            │2001      │6.2       │
├──────────────────────┼──────────┼──────────┤
│"Zubaan"              │2015      │6.1       │
├──────────────────────┼──────────┼──────────┤
│"Zor: Never Underesti │1998      │4.3       │
│mate the Force"       │          │          │
├──────────────────────┼──────────┼──────────┤
│"Zoo"                 │2018      │5.7       │
├──────────────────────┼──────────┼──────────┤
│"Zindagi Zindabad"    │2000      │5.7       │
├──────────────────────┼──────────┼──────────┤
```

PYTHON PIPELINE

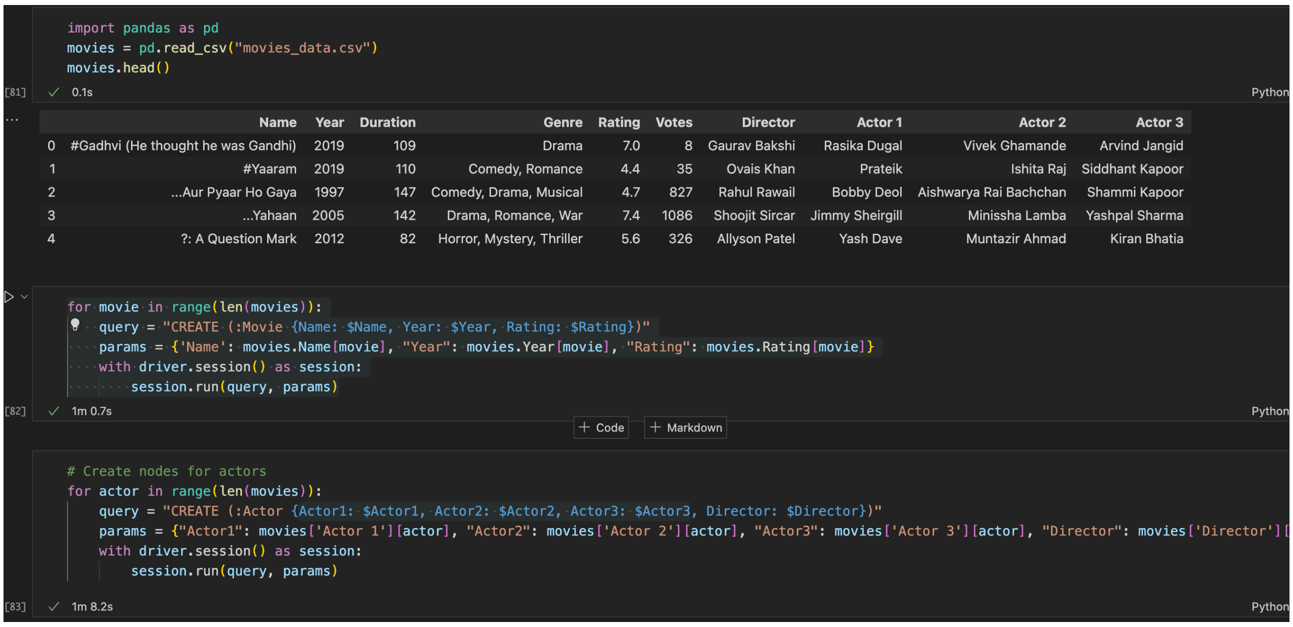


```
[79]
!pip install neo4j
```

```
DEPRECATION: Configuring installation scheme with distutils config files is deprecated and will no longer work in the near future. If you are using a
Homebrew or Linuxbrew Python, please see discussion at https://github.com/Homebrew/homebrew-core/issues/76621
Requirement already satisfied: neo4j in /opt/homebrew/lib/python3.9/site-packages (5.7.0)
Requirement already satisfied: pytz in /opt/homebrew/lib/python3.9/site-packages (from neo4j) (2022.7.1)
DEPRECATION: Configuring installation scheme with distutils config files is deprecated and will no longer work in the near future. If you are using a
Homebrew or Linuxbrew Python, please see discussion at https://github.com/Homebrew/homebrew-core/issues/76621
```

```python
[80]
from neo4j import GraphDatabase

# Set the connection details
uri = "neo4j://localhost:7687"
username = "neo4j"
password = "passcode"

# Create the driver and session objects
driver = GraphDatabase.driver(uri, auth=(username, password),encrypted=False)
session = driver.session()
# Run a query
result = session.run("MATCH (n) RETURN COUNT(n)")
# Print the result
for record in result:
    print(record[0])
```

```
Failed to write data to connection ResolvedIPv4Address(('127.0.0.1', 7687)) (IPv4Address(('127.0.0.1', 7687)))
Failed to write data to connection IPv4Address(('localhost', 7687)) (IPv4Address(('127.0.0.1', 7687)))
```

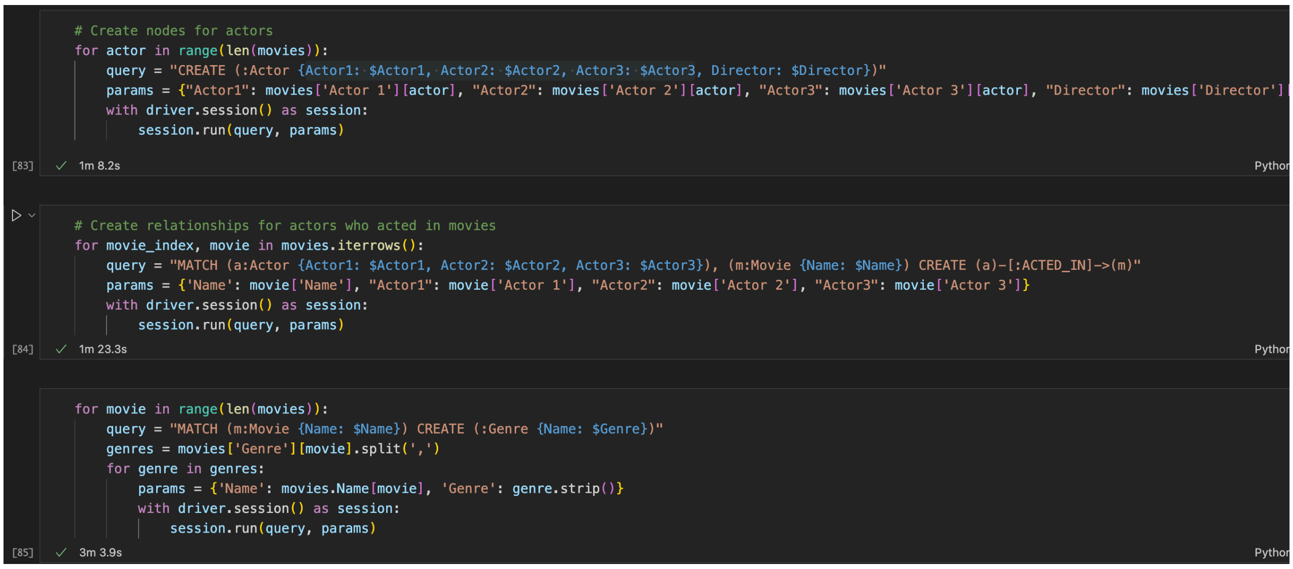


```python
[81]
import pandas as pd
movies = pd.read_csv("movies_data.csv")
movies.head()
```

	Name	Year	Duration	Genre	Rating	Votes	Director	Actor 1	Actor 2	Actor 3
0	#Gadhvi (He thought he was Gandhi)	2019	109	Drama	7.0	8	Gaurav Bakshi	Rasika Dugal	Vivek Ghamande	Arvind Jangid
1	#Yaaram	2019	110	Comedy, Romance	4.4	35	Ovais Khan	Prateik	Ishita Raj	Siddhant Kapoor
2	...Aur Pyaar Ho Gaya	1997	147	Comedy, Drama, Musical	4.7	827	Rahul Rawail	Bobby Deol	Aishwarya Rai Bachchan	Shammi Kapoor
3	...Yahaan	2005	142	Drama, Romance, War	7.4	1086	Shoojit Sircar	Jimmy Sheirgill	Minissha Lamba	Yashpal Sharma
4	?: A Question Mark	2012	82	Horror, Mystery, Thriller	5.6	326	Allyson Patel	Yash Dave	Muntazir Ahmad	Kiran Bhatia

```python
[82]
for movie in range(len(movies)):
    query = "CREATE (:Movie {Name: $Name, Year: $Year, Rating: $Rating})"
    params = {'Name': movies.Name[movie], "Year": movies.Year[movie], "Rating": movies.Rating[movie]}
    with driver.session() as session:
        session.run(query, params)
```

```python
[83]
# Create nodes for actors
for actor in range(len(movies)):
    query = "CREATE (:Actor {Actor1: $Actor1, Actor2: $Actor2, Actor3: $Actor3, Director: $Director})"
    params = {"Actor1": movies['Actor 1'][actor], "Actor2": movies['Actor 2'][actor], "Actor3": movies['Actor 3'][actor], "Director": movies['Director'][
    with driver.session() as session:
        session.run(query, params)
```



```
# Create nodes for actors
for actor in range(len(movies)):
    query = "CREATE (:Actor {Actor1: $Actor1, Actor2: $Actor2, Actor3: $Actor3, Director: $Director})"
    params = {"Actor1": movies['Actor 1'][actor], "Actor2": movies['Actor 2'][actor], "Actor3": movies['Actor 3'][actor], "Director": movies['Director'][
    with driver.session() as session:
        session.run(query, params)
```

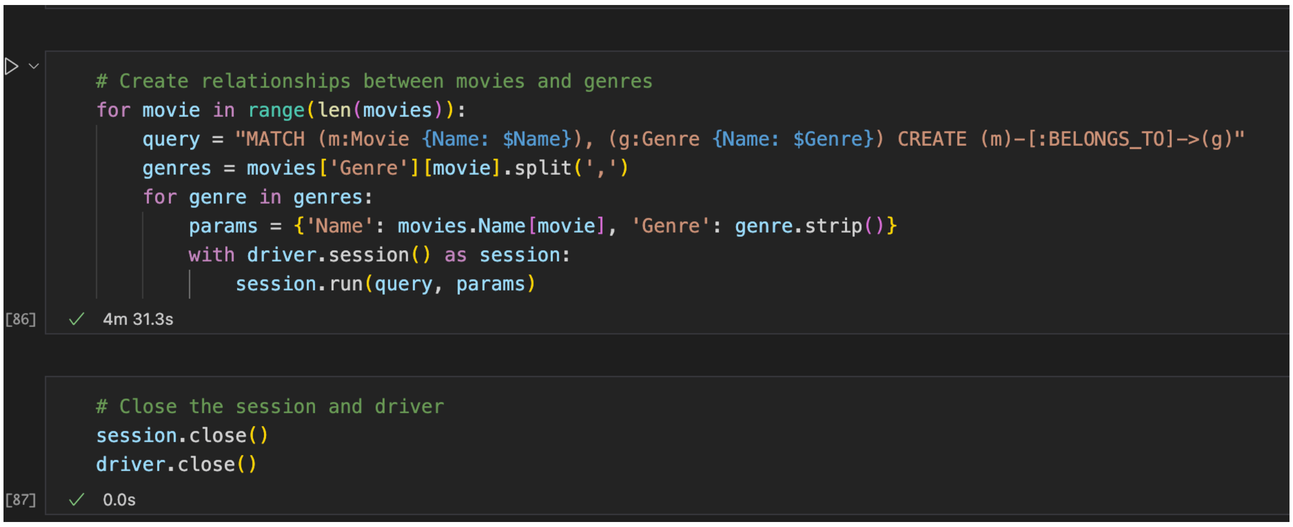
[83] ✓ 1m 8.2s Python

```
# Create relationships for actors who acted in movies
for movie_index, movie in movies.iterrows():
    query = "MATCH (a:Actor {Actor1: $Actor1, Actor2: $Actor2, Actor3: $Actor3}), (m:Movie {Name: $Name}) CREATE (a)-[:ACTED_IN]->(m)"
    params = {'Name': movie['Name'], "Actor1": movie['Actor 1'], "Actor2": movie['Actor 2'], "Actor3": movie['Actor 3']}
    with driver.session() as session:
        session.run(query, params)
```

[84] ✓ 1m 23.3s Python

```
for movie in range(len(movies)):
    query = "MATCH (m:Movie {Name: $Name}) CREATE (:Genre {Name: $Genre})"
    genres = movies['Genre'][movie].split(',')
    for genre in genres:
        params = {'Name': movies.Name[movie], 'Genre': genre.strip()}
        with driver.session() as session:
            session.run(query, params)
```

[85] ✓ 3m 3.9s Python



```
# Create relationships between movies and genres
for movie in range(len(movies)):
    query = "MATCH (m:Movie {Name: $Name}), (g:Genre {Name: $Genre}) CREATE (m)-[:BELONGS_TO]->(g)"
    genres = movies['Genre'][movie].split(',')
    for genre in genres:
        params = {'Name': movies.Name[movie], 'Genre': genre.strip()}
        with driver.session() as session:
            session.run(query, params)
```

[86] ✓ 4m 31.3s

```
# Close the session and driver
session.close()
driver.close()
```

[87] ✓ 0.0s