ADT Lab 10 (10pts)

Neo4j

In [1]:

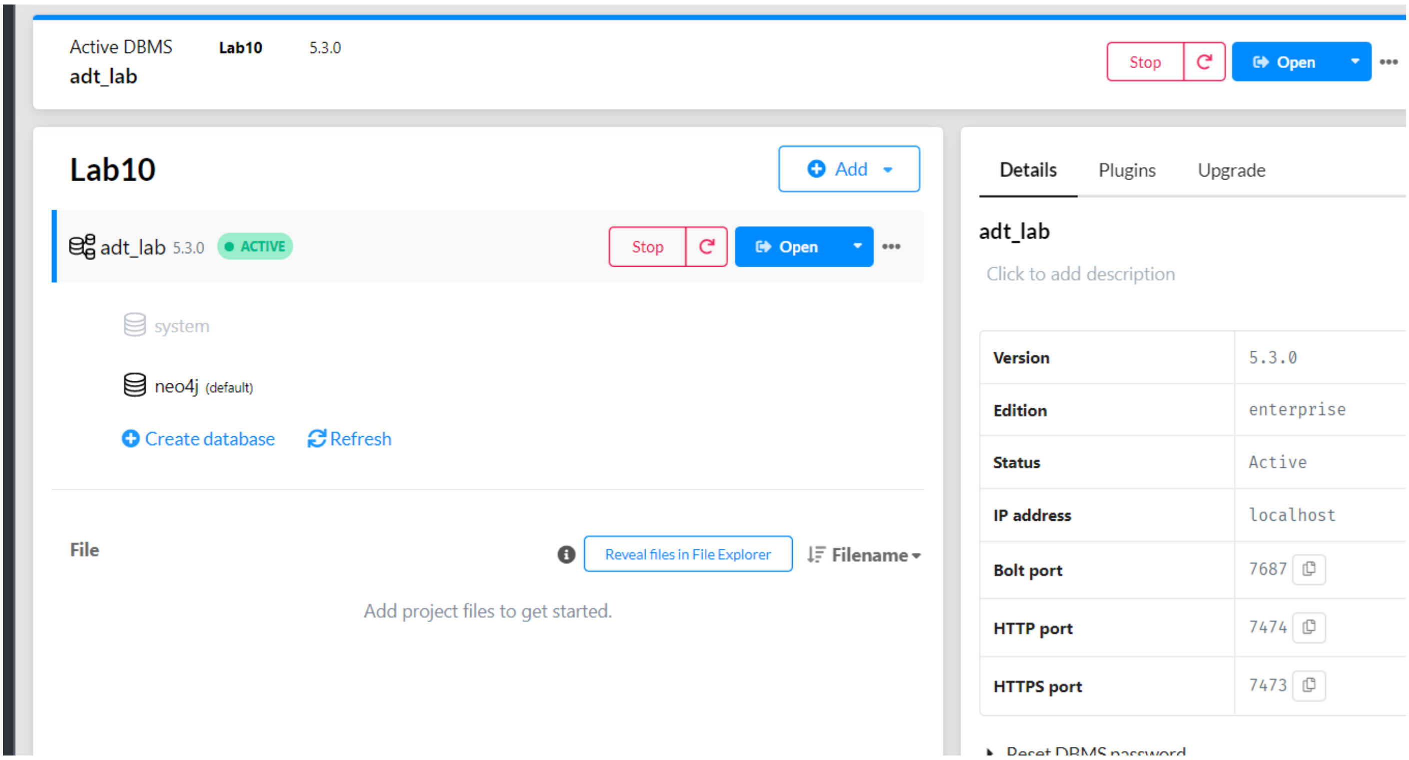
```
!pip install py2neo
!pip install pandas
!pip install scikit-learn
```

```
Requirement already satisfied: py2neo in c:\users\jsubh\anaconda3\lib\site-packages (2021.2.3)
Requirement already satisfied: urllib3 in c:\users\jsubh\anaconda3\lib\site-packages (from py2neo) (1.26.9)
Requirement already satisfied: packaging in c:\users\jsubh\anaconda3\lib\site-packages (from py2neo) (21.3)
Requirement already satisfied: monotonic in c:\users\jsubh\anaconda3\lib\site-packages (from py2neo) (1.6)
Requirement already satisfied: six>=1.15.0 in c:\users\jsubh\anaconda3\lib\site-packages (from py2neo) (1.16.0)
Requirement already satisfied: interchange~=2021.0.4 in c:\users\jsubh\anaconda3\lib\site-packages (from py2neo) (2021.
0.4)
Requirement already satisfied: pygments>=2.0.0 in c:\users\jsubh\anaconda3\lib\site-packages (from py2neo) (2.11.2)
Requirement already satisfied: certifi in c:\users\jsubh\anaconda3\lib\site-packages (from py2neo) (2021.10.8)
Requirement already satisfied: pansi>=2020.7.3 in c:\users\jsubh\anaconda3\lib\site-packages (from py2neo) (2020.7.3)
Requirement already satisfied: pytz in c:\users\jsubh\anaconda3\lib\site-packages (from interchange~=2021.0.4->py2neo)
(2021.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\jsubh\anaconda3\lib\site-packages (from packaging->p
y2neo) (3.0.4)
Requirement already satisfied: pandas in c:\users\jsubh\anaconda3\lib\site-packages (1.4.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\jsubh\anaconda3\lib\site-packages (from pandas) (2021.3)
Requirement already satisfied: numpy>=1.18.5 in c:\users\jsubh\anaconda3\lib\site-packages (from pandas) (1.21.5)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\jsubh\anaconda3\lib\site-packages (from pandas) (2.8.
2)
```

Setup Instructions

1. Create a new project in Neo4j desktop
2. Click on Add button
3. Click Local DBMS
4. Set your dbms name and password. (For simplicity keep password as password)
5. Start the db

Your neo4j screen should look something like this-



Version	5.3.0
Edition	enterprise
Status	Active
IP address	localhost
Bolt port	7687
HTTP port	7474
HTTPS port	7473

4. Once the database is started run the below cells to populate the newly created neo4j database

In [5]:

```
import os
import pandas as pd
from py2neo import Graph, Node, Relationship
from sklearn.metrics.pairwise import cosine_similarity
```

In [6]:

```
# Load the ratings data
ratings = pd.read_csv(os.getcwd()+'/ratings.csv')
ratings=ratings.loc[:10000]
# Load the movie data
movies = pd.read_csv(os.getcwd()+'/movies.csv')
movies=movies[movies['movieId'].isin(ratings['movieId'].unique().tolist())].reset_index()
```

In [7]:

```
# Connect to Neo4j
graph = Graph('bolt://localhost:7687', auth=('neo4j', 'password'))
graph.run("CREATE CONSTRAINT FOR (u:User) REQUIRE u.userId IS UNIQUE")
graph.run("CREATE CONSTRAINT FOR (m:Movie) REQUIRE m.movieId IS UNIQUE")
```

Out[7]:

```
(No data)
```

In [8]:

```
d={}
users_d = {}
movies_d = {}
```

In [9]:

```
# Create movie nodes
for index, movie in movies.iterrows():
    node = Node('Movie', movieId=int(movie['movieId']),title=movie['title'])
    graph.merge(node, "Movie", "movieId")

    movies_d[movie['movieId']]=node
    d[movie['movieId']]=movie['title']
```

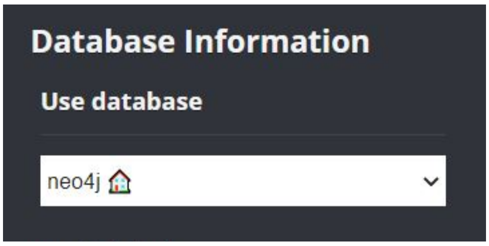
In [10]:

```
# Create user nodes
for userId in ratings['userId'].unique():
    node = Node('User', userId=int(userId))
    graph.merge(node, "User", "userId")

    users_d[int(userId)]=node
```

In [11]:

```
# Create rating relationships
for index, rating in ratings.iterrows():
    rating_float = float(rating['rating'])
    id_int = int(rating['userId'])
    movie_id_int = int(rating['movieId'])

    user_node = users_d[id_int]
    movie_node = movies_d[movie_id_int]

    rel = Relationship(user_node, 'RATED', movie_node, rating=rating_float)
    graph.create(rel)
```

Now check your Neo4j browser. Your data should reflect there-



Database Information

Use database

neo4j 🏠

Create queries for the below questions and run it

Exmaple Query 1

Find the top-rated movies

In [12]:

```
query = f"MATCH (m:Movie)<-[r:RATED]-() RETURN m.title, AVG(r.rating) AS avg_rating ORDER BY avg_rating DESC LIMIT 10"

results = graph.run(query)

for record in results:
    print(record)
```

```
'Three Colors: Red (Trois couleurs: Rouge) (1994)'	5.0
'Murder in the First (1995)'	5.0
'Canadian Bacon (1995)'	5.0
'Living in Oblivion (1995)'	5.0
'Persuasion (1995)'	5.0
"Antonia's Line (Antonia) (1995)"	5.0
'Home for the Holidays (1995)'	5.0
'Bottle Rocket (1996)'	5.0
'Flirting With Disaster (1996)'	5.0
'Farewell My Concubine (Ba wang bie ji) (1993)'	5.0
```

This query finds the top-rated movies in the dataset by calculating the average rating for each movie and sorting the results in descending order.

Query 2

Find the users who have rated the most movies

In [14]:

```
query = f"match(user:User)-[rating:RATED]->(movie:Movie) return user.userId, count(movie) as countRating order by countRating desc"

results = graph.run(query)

for record in results:
    print(record)
```

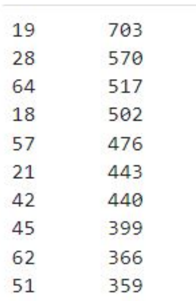
```
19	703
28	570
64	517
18	502
57	476
21	443
42	440
45	399
62	366
51	359
6	314
50	310
63	271
20	242
1	232
41	217
4	216
33	156
7	152
10	140
47	140
15	135
27	135
52	130
23	121
22	119
43	114
58	112
24	110
59	107
17	105
40	103
32	102
39	100
16	98
34	86
29	81
38	78
11	64
36	60
31	50
14	48
44	48
8	47
9	46
56	46
5	44
46	42
3	39
61	39
30	34
65	34
48	33
54	33
12	32
13	31
2	29
25	26
55	25
35	23
60	22
66	22
26	21
37	21
49	21
53	20
```

This query finds the users who have rated the most movies in the dataset by counting the number of movies each user has rated and sorting the results in descending order.

Hint-

- Use COUNT() aggregate clause

Expected result-



```
19      703
28      570
64      517
18      502
57      476
21      443
42      440
45      399
62      366
51      359
```

Query 3

Find movies that were rated the highest by a the user 2

In [15]:

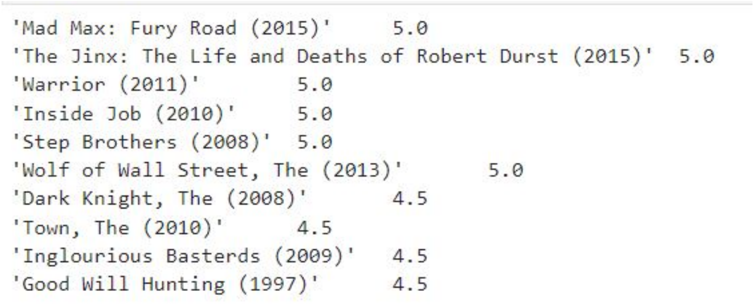
```
tch(user:User)-[rating:RATED]->(movie:Movie) WHERE user.userId = 2 return movie.title, max(rating.rating) as rating order by rating desc"

aph.run(query)

n results:
cord)
```

```
'The Jinx: The Life and Deaths of Robert Durst (2015)'  5.0
'Mad Max: Fury Road (2015)'     5.0
'Wolf of Wall Street, The (2013)'       5.0
'Warrior (2011)'        5.0
'Inside Job (2010)'     5.0
'Step Brothers (2008)'  5.0
'Town, The (2010)'      4.5
'Inglourious Basterds (2009)'   4.5
'Dark Knight, The (2008)'       4.5
'Good Will Hunting (1997)'      4.5
'Whiplash (2014)'       4.0
'Louis C.K.: Hilarious (2010)'  4.0
'Inception (2010)'      4.0
'Shutter Island (2010)' 4.0
'Departed, The (2006)'  4.0
'Talladega Nights: The Ballad of Ricky Bobby (2006)'    4.0
'Kill Bill: Vol. 1 (2003)'      4.0
'Gladiator (2000)'      4.0
'Tommy Boy (1995)'      4.0
'Ex Machina (2015)'     3.5
'Django Unchained (2012)'       3.5
'Dark Knight Rises, The (2012)' 3.5
'Collateral (2004)'     3.5
'Interstellar (2014)'   3.0
'Exit Through the Gift Shop (2010)'     3.0
'Zombieland (2009)'     3.0
'Shawshank Redemption, The (1994)'      3.0
'Girl with the Dragon Tattoo, The (2011)'       2.5
'The Drop (2014)'       2.0
```

Hint-

- Use MAX() aggregate clause

Expected result-



```
'Mad Max: Fury Road (2015)'     5.0
'The Jinx: The Life and Deaths of Robert Durst (2015)'  5.0
'Warrior (2011)'        5.0
'Inside Job (2010)'     5.0
'Step Brothers (2008)'  5.0
'Wolf of Wall Street, The (2013)'       5.0
'Dark Knight, The (2008)'       4.5
'Town, The (2010)'      4.5
'Inglourious Basterds (2009)'   4.5
'Good Will Hunting (1997)'      4.5
```

Query 4

Find the average rating for the moiveId 356

In [16]:

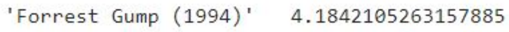
```
query = f"match(movie:Movie)<-[rating:RATED]-() WHERE movie.movieId=356 return movie.title, avg(rating.rating)"

results = graph.run(query)

for record in results:
    print(record)
```

```
'Forrest Gump (1994)'   4.1842105263157885
```

Hint-

- Use AVG() aggregate clause

Expected result-



```
'Forrest Gump (1994)'   4.1842105263157885
```

Run the queries, convert into pdf and submit it

Optional

In [17]:

```
# A function to get movie recommendations for a user
def get_movie_recommendations(userId):
    query = f"MATCH (u:User {{userId: {userId}}})-[:RATED]->(m:Movie)<-[:RATED]-(u2:User)-[r2:RATED]->(m2:Movie) WHERE NOT (u)-[:RATED]->(
    results = graph.run(query)

    # Convert the results to a pandas DataFrame
    recommendations = pd.DataFrame(results, columns=['title', 'rating'])

    return recommendations
```

In [18]:

```
# Get recommendations for user 3
recommendations = get_movie_recommendations(3)
print(recommendations)
```

```
                                      title  rating
0                       Pink Floyd: The Wall (1982)     5.0
1                         Newton Boys, The (1998)     5.0
2                            Wolf Man, The (1941)     5.0
3             All Quiet on the Western Front (1930)     5.0
4  Messenger: The Story of Joan of Arc, The (1999)     5.0
5       Winnie the Pooh and the Blustery Day (1968)     5.0
6                                    Shaft (1971)     5.0
7                        Gulliver's Travels (1939)     5.0
8                                SLC Punk! (1998)     5.0
9                            Bottle Rocket (1996)     5.0
```

- The recommendation algorithm above is a simple content-based filtering algorithm, which makes recommendations based on the similarity between items (in this case, movies).
- In a content-based filtering algorithm, the system makes recommendations based on the attributes or features of the items that the user has previously interacted with. In the above example, the system calculates the cosine similarity between the movie that the user has rated and all other movies in the dataset based on the genres of the movies, and recommends the top N most similar movies.

Understand the above query and write what you understood. How the recommendation algorithm is working?

It first finds the movies rated by a specific user. Then it tries to match another user who has rated the same movies, and only recommend those movies which are in the second users list, but has not been rated by the first user.

In [ ]: