COMS W4111-003/V003 (Fall 2022)
Introduction to Databases

Homework 4: Both Tracks

Setup Environment

MySQL

In [1]:
```
%load_ext sql
```

In [2]:
```
import pymysql
```

In [3]:
```
%sql mysql+pymysql://root:dbuserbdbuser@localhost
```

In [4]:
```
#
# Note that your list of databases will be different.
#
%sql show databases;
```

```
 * mysql+pymysql://root:***@localhost
13 rows affected.
```

Out[4]:

Database
classicmodels
db_book
f22_hw1_got
f22_hw1_got_clean
f22_hw1_got_programming
f22_hw2
f22_midterm
information_schema
lahmansbaseballdb
mysql
performance_schema
sys
w4111_f22_hw4

Neo4j

- This section assume that you have set up Neo4j on Aura DB, installed the Movie database and that your interest is running. If you do not use your instance for a while, Aura suspends it. You will have to login and restart it.
- Since these are cloud databases, I put my credentials in a file instead of a notebook that I share with everyone and place on GitHub. You will have to set the correct information for your instances.

In [5]:
```
import sys
sys.path.append('/Users/williamdas/Documents/Intro Databases Fall 2022/F22-W4111-HW4/data/')
```

In [6]:
```
import my_secrets
```

- You may have to install `py2neo` using pip or Conda.

In [7]:
```
!pip install py2neo
```

```
Requirement already satisfied: py2neo in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages (2021.2.
3)
Requirement already satisfied: certifi in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages (from p
y2neo) (2021.10.8)
Requirement already satisfied: monotonic in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages (from
py2neo) (1.6)
Requirement already satisfied: urllib3 in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages (from p
y2neo) (1.26.9)
Requirement already satisfied: packaging in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages (from
py2neo) (21.3)
Requirement already satisfied: pygments>=2.0.0 in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages
(from py2neo) (2.11.2)
Requirement already satisfied: six>=1.15.0 in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages (fr
om py2neo) (1.16.0)
Requirement already satisfied: interchange~=2021.0.4 in /Users/williamdas/opt/anaconda3/lib/python3.9/site-pa
ckages (from py2neo) (2021.0.4)
Requirement already satisfied: pansi>=2020.7.3 in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages
(from py2neo) (2020.7.3)
Requirement already satisfied: pytz in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages (from inte
rchange~=2021.0.4->py2neo) (2021.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /Users/williamdas/opt/anaconda3/lib/python3.9/site
-packages (from packaging->py2neo) (3.0.4)
```

In [8]:
```
from py2neo import Graph
```

In [9]:
```
aura_url = my_secrets.aura_url
aura_user = my_secrets.aura_user
aura_pw = my_secrets.aura_pw
```

In [10]:
```
def t1():
    graph = Graph(aura_url, auth=(aura_user, aura_pw))
    q = "match (r:Person) where r.name='Tom Hanks' return r"
    res = graph.run(q)

    for r in res:
        print(r)
```

In [11]:
```
t1()
```

```
Node('Person', born=1956, name='Tom Hanks')
```

MongoDB

- This section assume that you have set up MongoDB on Atlas.
- You need to follow instructions to enable connecting from a remote Python application.
- You also need to pip/Conda install pymongo.
- You will have to your URL for connecting.

In [12]:
```
!pip install pymongo
```

```
Requirement already satisfied: pymongo in /Users/williamdas/opt/anaconda3/lib/python3.9/site-packages (4.3.3)
Requirement already satisfied: dnspython<3.0.0,>=1.16.0 in /Users/williamdas/opt/anaconda3/lib/python3.9/site
-packages (from pymongo) (2.2.1)
```

In [13]:
```
import pymongo
```

In [14]:
```
def connect():
    client = pymongo.MongoClient(
        my_secrets.mongodb_url
    )
    return client


def t_connect():
    c = connect()
    print("Databases = ", list(c.list_databases()))
```

In [15]:
```
# Your list of databases will be different. It may, be empty.
t_connect()
```

```
Databases =  [{'name': 'hw4', 'sizeOnDisk': 835584, 'empty': False}, {'name': 'testdb', 'sizeOnDisk': 24576,
'empty': False}, {'name': 'admin', 'sizeOnDisk': 385024, 'empty': False}, {'name': 'local', 'sizeOnDisk': 632
4416512, 'empty': False}]
```

- The following will do some additional testing.

In [16]:
```
client = pymongo.MongoClient(
        my_secrets.mongodb_url
```

```
    )
```

In [17]:
```
client["testdb"]["testcollection"].insert_one(
    {
        "class": "W4111",
        "is_cool": True
    }
)
```

Out[17]:
```
<pymongo.results.InsertOneResult at 0x7ff181ad5af0>
```

In [18]:
```
client["testdb"]["testcollection"].insert_one(
    {
        "professor": "Ferguson",
        "is_cool": "Seriously? Are you kidding me? Heck NO!"
    }
)
```

Out[18]:
```
<pymongo.results.InsertOneResult at 0x7ff171da9250>
```

In [19]:
```
res = client["testdb"]["testcollection"].find()
```

In [20]:
```
some_info = list(res)
some_info
```

Out[20]:
```
[{'_id': ObjectId('639931824f3f8db5e31af891'),
  'class': 'W4111',
  'is_cool': True},
 {'_id': ObjectId('639931834f3f8db5e31af892'),
  'professor': 'Ferguson',
  'is_cool': 'Seriously? Are you kidding me? Heck NO!'}]
```

Common Tasks for Both Tracks

Load MongoDB Atlas

- The project contains two files: `data/characters.json` and `data/episodes.json`.
- The following code snippets will load the information into lists. This is a little tricky because we need to extract the embedded array from the JSON object.

In [21]:
```
import json
```

In [22]:
```
with open ("data/characters.json") as in_file:
    characters = json.load(in_file)
```

In [23]:
```
characters = characters['characters']
```

In [24]:
```
characters[0:2]
```

Out[24]:
```
[{'characterName': 'Addam Marbrand',
  'characterLink': '/character/ch0305333/',
  'actorName': 'B.J. Hogg',
  'actorLink': '/name/nm0389698/'},
 {'characterName': 'Aegon Targaryen',
  'houseName': 'Targaryen',
  'royal': True,
  'parents': ['Elia Martell', 'Rhaegar Targaryen'],
  'siblings': ['Rhaenys Targaryen', 'Jon Snow'],
  'killedBy': ['Gregor Clegane']}]
```

In [25]:
```
with open ("data/episodes.json") as in_file:
    episodes = json.load(in_file)
```

In [26]:
```
episodes = episodes['episodes']
```

- You now have two lists in the notebook. Save the documents in the array to Atlas using databases name `hw4`. You must write Python code using `pymongo` to load the data.
- Put your code in the cells below and execute it.

In [27]:
```
# Your code here.
client["hw4"]["characters"].insert_many(characters)
client["hw4"]["episodes"].insert_many(episodes)
```

Out[27]:
```
<pymongo.results.InsertManyResult at 0x7ff17249eee0>
```

- The following code will test that you have loaded the information.

In [28]:
```
ep = client['hw4']['episodes'].find_one(
    {"seasonNum": 1, "episodeNum": 1}
)
```

In [29]:
```
ep
```

Out[29]:

```
{'_id': ObjectId('639931ae4f3f8db5e31afa18'),
 'seasonNum': 1,
 'episodeNum': 1,
 'episodeTitle': 'Winter Is Coming',
 'episodeLink': '/title/tt1480055/',
 'episodeAirDate': '2011-04-17',
 'episodeDescription': "Jon Arryn, the Hand of the King, is dead. King Robert Baratheon plans to ask his olde
st friend, Eddard Stark, to take Jon's place. Across the sea, Viserys Targaryen plans to wed his sister to a
nomadic warlord in exchange for an army.",
 'openingSequenceLocations': ["King's Landing",
  'Winterfell',
  'The Wall',
  'Pentos'],
 'scenes': [{'sceneStart': '0:00:40',
   'sceneEnd': '0:01:45',
   'location': 'The Wall',
   'subLocation': 'Castle Black',
   'characters': [{'name': 'Gared'},
    {'name': 'Waymar Royce'},
    {'name': 'Will'}]},
  {'sceneStart': '0:01:45',
   'sceneEnd': '0:03:24',
   'location': 'North of the Wall',
   'subLocation': 'The Haunted Forest',
   'characters': [{'name': 'Gared'},
    {'name': 'Waymar Royce'},
    {'name': 'Will'}]},
  {'sceneStart': '0:03:24',
   'sceneEnd': '0:03:31',
   'location': 'North of the Wall',
   'subLocation': 'The Haunted Forest',
   'characters': [{'name': 'Will'},
    {'name': 'Wight Wildling Girl', 'alive': False}]},
  {'sceneStart': '0:03:31',
   'sceneEnd': '0:03:38',
   'location': 'North of the Wall',
   'subLocation': 'The Haunted Forest',
   'characters': [{'name': 'Will'}]},
  {'sceneStart': '0:03:38',
   'sceneEnd': '0:03:44',
   'location': 'North of the Wall',
   'subLocation': 'The Haunted Forest',
   'characters': []},
  {'sceneStart': '0:03:44',
   'sceneEnd': '0:05:36',
```

```
 'location': 'North of the Wall',
 'subLocation': 'The Haunted Forest',
 'characters': [{'name': 'Gared'},
  {'name': 'Waymar Royce'},
  {'name': 'Will'}]},
{'sceneStart': '0:05:36',
 'sceneEnd': '0:05:41',
 'location': 'North of the Wall',
 'subLocation': 'The Haunted Forest',
 'characters': [{'name': 'Will'}]},
{'sceneStart': '0:05:41',
 'sceneEnd': '0:05:48',
 'location': 'North of the Wall',
 'subLocation': 'The Haunted Forest',
 'characters': [{'name': 'Gared'}, {'name': 'Waymar Royce'}]},
{'sceneStart': '0:05:48',
 'sceneEnd': '0:05:58',
 'location': 'North of the Wall',
 'subLocation': 'The Haunted Forest',
 'characters': [{'name': 'Will'}]},
{'sceneStart': '0:05:58',
 'sceneEnd': '0:06:21',
 'location': 'North of the Wall',
 'subLocation': 'The Haunted Forest',
 'characters': [{'name': 'Gared'},
  {'name': 'Waymar Royce',
   'alive': False,
   'mannerOfDeath': 'Back stab',
   'killedBy': ['White Walker']},
  {'name': 'White Walker'}]},
{'sceneStart': '0:06:21',
 'sceneEnd': '0:06:39',
 'location': 'North of the Wall',
 'subLocation': 'The Haunted Forest',
 'characters': [{'name': 'Will'}]},
{'sceneStart': '0:06:39',
 'sceneEnd': '0:06:49',
 'location': 'North of the Wall',
 'subLocation': 'The Haunted Forest',
 'characters': [{'name': 'Will'}, {'name': 'Wight Wildling Girl'}]},
{'sceneStart': '0:06:49',
 'sceneEnd': '0:07:45',
 'location': 'North of the Wall',
 'subLocation': 'The Haunted Forest',
 'characters': [{'name': 'Will'},
```

```
  {'name': 'Gared',
   'alive': False,
   'mannerOfDeath': 'Decapitation',
   'killedBy': ['White Walker']},
  {'name': 'White Walker'}]},
{'sceneStart': '0:09:27',
 'sceneEnd': '0:12:38',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Will'},
  {'name': 'Jon Snow'},
  {'name': 'Bran Stark'},
  {'name': 'Robb Stark'},
  {'name': 'Eddard Stark'},
  {'name': 'Catelyn Stark'},
  {'name': 'Rickon Stark'},
  {'name': 'Sansa Stark'},
  {'name': 'Arya Stark'},
  {'name': 'Theon Greyjoy'},
  {'name': 'Rodrik Cassel'},
  {'name': 'Septa Mordane'}]},
{'sceneStart': '0:12:38',
 'sceneEnd': '0:15:41',
 'location': 'The North',
 'subLocation': 'Outside Winterfell',
 'characters': [{'name': 'Will',
   'alive': False,
   'mannerOfDeath': 'Decapitation',
   'killedBy': ['Eddard Stark']},
  {'name': 'Eddard Stark', 'weapon': [{'action': 'has', 'name': 'Ice'}]},
  {'name': 'Robb Stark'},
  {'name': 'Theon Greyjoy'},
  {'name': 'Bran Stark'},
  {'name': 'Rodrik Cassel'},
  {'name': 'Jory Cassel'},
  {'name': 'Jon Snow'}]},
{'sceneStart': '0:15:41',
 'sceneEnd': '0:18:44',
 'location': 'The North',
 'subLocation': 'Outside Winterfell',
 'characters': [{'name': 'Jon Snow'},
  {'name': 'Eddard Stark', 'weapon': [{'action': 'has', 'name': 'Ice'}]},
  {'name': 'Theon Greyjoy'},
  {'name': 'Bran Stark'},
  {'name': 'Robb Stark'},
```

```
  {'name': 'Rodrik Cassel'},
  {'name': 'Jory Cassel'},
  {'name': 'Grey Wind'},
  {'name': 'Lady'},
  {'name': 'Nymeria'},
  {'name': 'Summer'},
  {'name': 'Shaggydog'},
  {'name': 'Ghost'}]},
{'sceneStart': '0:18:44',
 'sceneEnd': '0:20:45',
 'location': 'The Crownlands',
 'subLocation': "King's Landing",
 'characters': [{'name': 'Jon Arryn',
   'alive': False,
   'title': 'Hand',
   'mannerOfDeath': 'Poison',
   'killedBy': ['Lysa Arryn']},
  {'name': 'Cersei Lannister'},
  {'name': 'Jaime Lannister'}]},
{'sceneStart': '0:20:45',
 'sceneEnd': '0:22:43',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Catelyn Stark'},
  {'name': 'Eddard Stark', 'weapon': [{'action': 'has', 'name': 'Ice'}]}]},
{'sceneStart': '0:22:43',
 'sceneEnd': '0:23:09',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Catelyn Stark'}, {'name': 'Maester Luwin'}]},
{'sceneStart': '0:23:09',
 'sceneEnd': '0:23:39',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Robb Stark'},
  {'name': 'Theon Greyjoy'},
  {'name': 'Jon Snow'}]},
{'sceneStart': '0:23:39',
 'sceneEnd': '0:29:16',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Bran Stark'},
  {'name': 'Catelyn Stark'},
  {'name': 'Maester Luwin'},
  {'name': 'Summer'},
```

```
  {'name': 'Arya Stark'},
  {'name': 'Joffrey Baratheon',
   'weapon': [{'action': 'has', 'name': "Lion's Tooth"}]},
  {'name': 'Sandor Clegane'},
  {'name': 'Rickon Stark'},
  {'name': 'Eddard Stark'},
  {'name': 'Sansa Stark'},
  {'name': 'Theon Greyjoy'},
  {'name': 'Jon Snow'},
  {'name': 'Jory Cassel'},
  {'name': 'Hodor'},
  {'name': 'Rodrik Cassel'},
  {'name': 'Robert Baratheon', 'title': 'King'},
  {'name': 'Cersei Lannister'},
  {'name': 'Jaime Lannister'},
  {'name': 'Myrcella Baratheon'},
  {'name': 'Tommen Baratheon'}]},
{'sceneStart': '0:29:16',
 'sceneEnd': '0:30:47',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Robert Baratheon', 'title': 'King'},
  {'name': 'Eddard Stark'}]},
{'sceneStart': '0:30:47',
 'sceneEnd': '0:32:57',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Jaime Lannister'},
  {'name': 'Tyrion Lannister',
   'sex': {'with': ['Ros'], 'type': 'paid', 'when': 'present'}},
  {'name': 'Ros',
   'sex': {'with': ['Tyrion Lannister'],
    'type': 'paid',
    'when': 'present'}}]},
{'sceneStart': '0:32:57',
 'sceneEnd': '0:33:45',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Robert Baratheon', 'title': 'King'},
  {'name': 'Eddard Stark'}]},
{'sceneStart': '0:33:45',
 'sceneEnd': '0:36:17',
 'location': 'Pentos',
 'characters': [{'name': 'Daenerys Targaryen'},
  {'name': 'Viserys Targaryen'}]},
```

```
{'sceneStart': '0:36:17',
 'sceneEnd': '0:38:23',
 'location': 'Pentos',
 'characters': [{'name': 'Daenerys Targaryen'},
  {'name': 'Viserys Targaryen'},
  {'name': 'Illyrio Mopatis'},
  {'name': 'Khal Drogo', 'title': 'Khal'},
  {'name': 'Qotho'}]},
{'sceneStart': '0:38:23',
 'sceneEnd': '0:40:05',
 'location': 'Pentos',
 'characters': [{'name': 'Viserys Targaryen'},
  {'name': 'Illyrio Mopatis'},
  {'name': 'Daenerys Targaryen'}]},
{'sceneStart': '0:40:05',
 'sceneEnd': '0:40:56',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Catelyn Stark'}, {'name': 'Sansa Stark'}]},
{'sceneStart': '0:40:56',
 'sceneEnd': '0:41:19',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Robert Baratheon', 'title': 'King'},
  {'name': 'Catelyn Stark'},
  {'name': 'Cersei Lannister'},
  {'name': 'Rodrik Cassel'}]},
{'sceneStart': '0:41:19',
 'sceneEnd': '0:44:14',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Jon Snow'},
  {'name': 'Benjen Stark'},
  {'name': 'Tyrion Lannister'}]},
{'sceneStart': '0:44:14',
 'sceneEnd': '0:47:28',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Rodrik Cassel'},
  {'name': 'Benjen Stark'},
  {'name': 'Eddard Stark'},
  {'name': 'Robb Stark'},
  {'name': 'Robert Baratheon', 'title': 'King'},
  {'name': 'Catelyn Stark'},
  {'name': 'Cersei Lannister'},
```

```
  {'name': 'Sansa Stark'},
  {'name': 'Joffrey Baratheon'},
  {'name': 'Jaime Lannister'},
  {'name': 'Arya Stark'},
  {'name': 'Theon Greyjoy'}]},
{'sceneStart': '0:47:28',
 'sceneEnd': '0:50:29',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Eddard Stark'},
  {'name': 'Catelyn Stark'},
  {'name': 'Maester Luwin'}]},
{'sceneStart': '0:50:29',
 'sceneEnd': '0:56:05',
 'location': 'Pentos',
 'characters': [{'name': 'Khal Drogo',
   'title': 'Khal',
   'married': {'to': 'Daenerys Targaryen',
    'when': 'present',
    'type': 'arranged',
    'consummated': True}},
  {'name': 'Daenerys Targaryen',
   'title': 'Khaleesi',
   'married': {'to': 'Khal Drogo',
    'when': 'present',
    'type': 'arranged',
    'consummated': True}},
  {'name': 'Viserys Targaryen'},
  {'name': 'Illyrio Mopatis'},
  {'name': 'Qotho'},
  {'name': 'Jorah Mormont'},
  {'name': 'Drogon', 'born': False},
  {'name': 'Rhaegal', 'born': False},
  {'name': 'Viserion', 'born': False}]},
{'sceneStart': '0:56:05',
 'sceneEnd': '0:57:48',
 'location': 'Pentos',
 'characters': [{'name': 'Khal Drogo',
   'title': 'Khal',
   'sex': {'with': ['Daenerys Targaryen'],
    'when': 'present',
    'type': 'rape'}},
  {'name': 'Daenerys Targaryen',
   'title': 'Khaleesi',
   'sex': {'with': ['Khal Drogo'], 'when': 'present', 'type': 'rape'}}]},
```

```
{'sceneStart': '0:57:48',
 'sceneEnd': '0:59:06',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Tyrion Lannister'},
  {'name': 'Sandor Clegane'},
  {'name': 'Theon Greyjoy'},
  {'name': 'Eddard Stark'},
  {'name': 'Benjen Stark'},
  {'name': 'Robb Stark'},
  {'name': 'Robert Baratheon', 'title': 'King'},
  {'name': 'Bran Stark'},
  {'name': 'Summer'}]},
{'sceneStart': '0:59:06',
 'sceneEnd': '1:00:57',
 'location': 'The North',
 'subLocation': 'Winterfell',
 'characters': [{'name': 'Bran Stark'},
  {'name': 'Summer'},
  {'name': 'Jaime Lannister',
   'sex': {'with': ['Cersei Lannister'], 'when': 'present', 'type': 'love'}},
  {'name': 'Cersei Lannister',
   'sex': {'with': ['Jaime Lannister'],
    'when': 'present',
    'type': 'love'}}]}]}
```

In [30]:
```
cc = client['hw4']['characters'].find_one(
    {"characterName": "Sansa Stark"}
)
```

In [31]:
```
cc
```

Out[31]:
```
{'_id': ObjectId('639931ad4f3f8db5e31af9bf'),
 'characterName': 'Sansa Stark',
 'houseName': 'Stark',
 'characterImageThumb': 'https://images-na.ssl-images-amazon.com/images/M/MV5BNjAwMjE2NDExNF5BMl5BanBnXkFtZTcwODAwODg4OQ@@._V1._SX100_SY140_.jpg',
 'characterImageFull': 'https://images-na.ssl-images-amazon.com/images/M/MV5BNjAwMjE2NDExNF5BMl5BanBnXkFtZTcwODAwODg4OQ@@._V1_SY1000_CR0,0,806,1000_AL_.jpg',
 'characterLink': '/character/ch0158137/',
 'actorName': 'Sophie Turner',
 'actorLink': '/name/nm3849842/',
 'royal': True,
 'siblings': ['Robb Stark', 'Arya Stark', 'Bran Stark', 'Rickon Stark'],
 'marriedEngaged': ['Joffrey Baratheon', 'Tyrion Lannister', 'Ramsay Snow'],
 'guardedBy': ['Lady'],
 'parents': ['Eddard Stark', 'Catelyn Stark']}
```

Load MySQL

- The `data` directory contains 5 CSV files:
  - `got_cast.csv`
  - `got_cast_details.csv`
  - `got_episodes.csv`
  - `got_title_ratings.csv`
  - `got_title_cast.csv`
- Create a MySQL databases `w4111_f22_hw4` and load the CSV files. You should use Pandas to load and save the information.
- Put your code in the cells below and execute it.

In [33]:
```
%%sql
CREATE DATABASE w4111_f22_hw4;
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[33]:
```
[]
```

In [34]:
```
import pandas as pd
```

In [35]:
```
got_cast_df = pd.read_csv('data/got_cast.csv')
got_cast_details_df = pd.read_csv('data/got_cast_details.csv')
got_episodes_df = pd.read_csv('data/got_episodes.csv')
got_title_ratings_df = pd.read_csv('data/got_title_ratings.csv')
got_title_cast_df = pd.read_csv('data/got_title_cast.csv')
```

In [36]:
```
from sqlalchemy import create_engine
```

In [37]:
```
engine = create_engine("mysql+pymysql://root:dbuserbdbuser@localhost")
```

In [38]:
```
got_cast_df.to_sql("got_cast", con=engine, index=False, if_exists='replace', schema='w4111_f22_hw4')
```

Out[38]:
```
730
```

In [39]:
```
got_cast_details_df.to_sql("got_cast_details", con=engine, index=False, if_exists='replace', schema='w4111_f2
```

Out[39]:
```
52
```

In [40]:
```
got_episodes_df.to_sql("got_episodes", con=engine, index=False, if_exists='replace', schema='w4111_f22_hw4')
```

Out[40]:
```
73
```

In [41]:
```
got_title_ratings_df.to_sql("got_title_ratings", con=engine, index=False, if_exists='replace', schema='w4111_
```

Out[41]:
```
73
```

In [42]:
```
got_title_cast_df.to_sql("got_title_cast", con=engine, index=False, if_exists='replace', schema='w4111_f22_hw
```

Out[42]:
```
165
```

- The following will test your loading.

In [43]:
```
%sql select * from w4111_f22_hw4.got_cast limit 10;
```

```
 * mysql+pymysql://root:***@localhost
10 rows affected.
```

Out[43]:

	id	tconst	ordering	nconst	category	job	characters
0	0	tt1480055	10	nm0230361	producer	producer	None
1	1	tt1480055	1	nm0000293	actor	None	["Eddard 'Ned' Stark"]
2	2	tt1480055	2	nm0004692	actor	None	["Robert Baratheon"]
3	3	tt1480055	3	nm0182666	actor	None	["Jaime Lannister"]
4	4	tt1480055	4	nm0265610	actress	None	["Catelyn Stark"]
5	5	tt1480055	5	nm0887700	director	None	None
6	6	tt1480055	6	nm1125275	writer	created by	None
7	7	tt1480055	7	nm1888967	writer	created by	None
8	8	tt1480055	8	nm0552333	writer	based on "A Song of Ice and Fire" by	None
9	9	tt1480055	9	nm0122251	producer	producer	None

In [44]:
```
%sql select * from w4111_f22_hw4.got_cast_details limit 10;
```

```
 * mysql+pymysql://root:***@localhost
10 rows affected.
```

Out[44]:

id	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitle
0	nm0230361	Frank Doelger	None	None	producer,director,writer	tt0985040,tt0139780,tt0944947,tt020217
1	nm0000293	Sean Bean	1959.0	None	actor,producer,animation_department	tt1181791,tt0944947,tt0120737,tt016726
2	nm0004692	Mark Addy	1964.0	None	actor,soundtrack	tt0183790,tt0944947,tt0119164,tt095530
3	nm0182666	Nikolaj Coster-Waldau	1970.0	None	actor,producer,writer	tt0110631,tt1483013,tt0944947,tt240423
4	nm0265610	Michelle Fairley	1963.0	None	actress	tt2431286,tt0944947,tt1390411,tt1118801
5	nm0887700	Timothy Van Patten	1959.0	None	director,actor,producer	tt0083739,tt0979432,tt0374463,tt014184
6	nm1125275	David Benioff	1970.0	None	producer,writer,director	tt1025100,tt0944947,tt0419887,tt030790
7	nm1888967	D.B. Weiss	1971.0	None	writer,producer,director	tt11547156,tt12141112,tt3677548,tt094494
8	nm0552333	George R.R. Martin	1948.0	None	writer,producer,miscellaneous	tt0944947,tt11198330,tt0088634,tt009231
9	nm0122251	Jo Burn	None	None	miscellaneous,production_manager,producer	tt1596342,tt4530422,tt5697572,tt279892

In [45]:
```
%sql select * from w4111_f22_hw4.got_episodes limit 10;
```

```
 * mysql+pymysql://root:***@localhost
10 rows affected.
```

Out[45]:

id	seasonNum	episodeNum	episodeTitle	episodeLink	episodeAirDate	episodeDescription
0	1	1	Winter Is Coming	/title/tt1480055/	2011-04-17	Jon Arryn, the Hand of the King, is dead. King Robert Baratheon plans to ask his oldest friend, Eddard Stark, to take Jon's place. Across the sea, Viserys Targaryen plans to wed his sister to a nomadic warlord in exchange for an army.
1	1	2	The Kingsroad	/title/tt1668746/	2011-04-24	While Bran recovers from his fall, Ned takes only his daughters to King's Landing. Jon Snow goes with his uncle Benjen to The Wall. Tyrion joins them.
2	1	3	Lord Snow	/title/tt1829962/	2011-05-01	Lord Stark and his daughters arrive at King's Landing to discover the intrigues of the king's realm.
3	1	4	Cripples, Bastards, and Broken Things	/title/tt1829963/	2011-05-08	Eddard investigates Jon Arryn's murder. Jon befriends Samwell Tarly, a coward who has come to join the Night's Watch.
4	1	5	The Wolf and the Lion	/title/tt1829964/	2011-05-15	Catelyn has captured Tyrion and plans to bring him to her sister, Lysa Arryn, at The Vale, to be tried for his, supposed, crimes against Bran. Robert plans to have Daenerys killed, but Eddard refuses to be a part of it and quits.
5	1	6	A Golden Crown	/title/tt1837862/	2011-05-22	While recovering from his battle with Jaime, Eddard is forced to run the kingdom while Robert goes hunting. Tyrion demands a trial by combat for his freedom. Viserys is losing his patience with Drogo.
6	1	7	You Win or You Die	/title/tt1837863/	2011-05-29	Robert has been injured while hunting and is dying. Jon and the others finally take their vows to the Night's Watch. A man, sent by Robert, is captured for trying to poison Daenerys. Furious, Drogo vows to attack the Seven Kingdoms.
7	1	8	The Pointy End	/title/tt1837864/	2011-06-05	Eddard and his men are betrayed and captured by the Lannisters. When word reaches Robb, he plans to go to war to rescue them. The White Walkers attack The Wall. Tyrion returns to his father with some new friends.
8	1	9	Baelor	/title/tt1851398/	2011-06-12	Robb goes to war against the Lannisters. Jon finds himself struggling on deciding if his place is with Robb or the Night's Watch. Drogo has fallen ill from a fresh battle wound. Daenerys is desperate to save him.
9	1	10	Fire and Blood	/title/tt1851397/	2011-06-19	With Ned dead, Robb vows to get revenge on the Lannisters. Jon must officially decide if his place is with Robb or the Night's Watch. Daenerys says her final goodbye to Drogo.

In [46]:
```
%sql select * from w4111_f22_hw4.got_title_ratings limit 10;
```

```
 * mysql+pymysql://root:***@localhost
10 rows affected.
```

Out[46]:

id	tconst	averageRating	numVotes
0	tt1480055	8.9	48686
1	tt1668746	8.6	36837
2	tt1829962	8.5	34863
3	tt1829963	8.6	33136
4	tt1829964	9.0	34436
5	tt1837862	9.1	34050
6	tt1837863	9.1	34564
7	tt1837864	8.9	32339
8	tt1851398	9.6	45134
9	tt1851397	9.4	39748

In [47]:
```
%sql select * from w4111_f22_hw4.got_title_cast limit 10;
```

```
 * mysql+pymysql://root:***@localhost
10 rows affected.
```

Out[47]:

nconst	tconst	characters
nm0000293	tt1480055	["Eddard 'Ned' Stark"]
nm0004692	tt1480055	["Robert Baratheon"]
nm0182666	tt1480055	["Jaime Lannister"]
nm0000293	tt1668746	["Eddard 'Ned' Stark"]
nm0004692	tt1668746	["Robert Baratheon"]
nm0182666	tt1668746	["Jaime Lannister"]
nm0000293	tt1829962	["Eddard 'Ned' Stark"]
nm0004692	tt1829962	["Robert Baratheon"]
nm0182666	tt1829962	["Jaime Lannister"]
nm0000293	tt1829963	["Eddard 'Ned' Stark"]

Load Neo4j

- You are now going to load some information into Neo4j. Specifically, the Game of Thrones episodes, the Game of Thrones cast, and the Game of Thrones title-cast information.
- You can load the data into the notebook from the RDB using Pandas and SQLAlchemy.
- You will use `py2neo` to do the loading into Neo4j. The functions are later in the notebook.
- I decided to make this easier and give you some helper functions instead of having you figure it out by yourself.

Load SQL into DataFrames

In [48]:
```
import pandas
```

In [49]:
```
from sqlalchemy import create_engine
```

In [50]:
```
engine = create_engine("mysql+pymysql://root:dbuserbdbuser@localhost")
```

In [51]:
```
sql_episodes = \
"""
SELECT
    seasonNum, episodeNum, episodeTitle,
    substr(episodeLink, 8, length(episodeLink) - 8) as tconst,
    episodeAirDate, episodeDescription
FROM
    w4111_f22_hw4.got_episodes;
"""
```

In [52]:
```
sql_cast_details = \
"""
SELECT
    DISTINCT *
FROM
    w4111_f22_hw4.got_cast_details;
"""
```

In [53]:
```
sql_titles_cast = \
"""
SELECT
    DISTINCT *
FROM
    w4111_f22_hw4.got_title_cast;
"""
```

In [54]:
```
episodes_df = pd.read_sql(sql_episodes, con=engine)
```

In [55]:
```
cast_df = pd.read_sql(sql_cast_details, con=engine)
```

In [56]:
```
titles_cast_df = pd.read_sql(sql_titles_cast, con=engine)
```

- The following tests whether or not your loading worked.

In [57]:
```
episodes_df
```

Out[57]:

	seasonNum	episodeNum	episodeTitle	tconst	episodeAirDate	episodeDescription
0	1	1	Winter Is Coming	tt1480055	2011-04-17	Jon Arryn, the Hand of the King, is dead. King...
1	1	2	The Kingsroad	tt1668746	2011-04-24	While Bran recovers from his fall, Ned takes o...
2	1	3	Lord Snow	tt1829962	2011-05-01	Lord Stark and his daughters arrive at King's ...
3	1	4	Cripples, Bastards, and Broken Things	tt1829963	2011-05-08	Eddard investigates Jon Arryn's murder. Jon be...
4	1	5	The Wolf and the Lion	tt1829964	2011-05-15	Catelyn has captured Tyrion and plans to bring...
...	...	...	...	...	...	...
68	8	2	A Knight of the Seven Kingdoms	tt6027908	2019-04-21	The battle at Winterfell is approaching. Jaime...
69	8	3	The Long Night	tt6027912	2019-04-28	The Night King and his army have arrived at Wi...
70	8	4	The Last of the Starks	tt6027914	2019-05-05	In the wake of a costly victory, Jon and Daene...
71	8	5	The Bells	tt6027916	2019-05-12	Daenerys and Cersei weigh their options as an ...
72	8	6	The Iron Throne	tt6027920	2019-05-19	In the aftermath of the devastating attack on ...

73 rows × 6 columns

In [58]:
```
cast_df
```

Out[58]:

	id	nconst	primaryName	birthYear	deathYear	primaryProfession	kn
0	0	nm0230361	Frank Doelger	NaN	None	producer,director,writer	tt0985040,tt0139780,tt0944
1	1	nm0000293	Sean Bean	1959.0	None	actor,producer,animation_department	tt1181791,tt0944947,tt0120
2	2	nm0004692	Mark Addy	1964.0	None	actor,soundtrack	tt0183790,tt0944947,tt01191
3	3	nm0182666	Nikolaj Coster-Waldau	1970.0	None	actor,producer,writer	tt0110631,tt1483013,tt0944
4	4	nm0265610	Michelle Fairley	1963.0	None	actress	tt2431286,tt0944947,tt1390
5	5	nm0887700	Timothy Van Patten	1959.0	None	director,actor,producer	tt0083739,tt0979432,tt0374
6	6	nm1125275	David Benioff	1970.0	None	producer,writer,director	tt1025100,tt0944947,tt0419
7	7	nm1888967	D.B. Weiss	1971.0	None	writer,producer,director	tt11547156,tt12141112,tt36775
8	8	nm0552333	George R.R. Martin	1948.0	None	writer,producer,miscellaneous	tt0944947,tt11198330,tt00886
9	9	nm0122251	Jo Burn	NaN	None	miscellaneous,production_manager,producer	tt1596342,tt4530422,tt56975
10	10	nm0400240	Mark Huffam	NaN	None	producer,production_manager,assistant_director	tt11214590,tt3659388,tt01208
11	11	nm1047532	Brian Kirk	1968.0	None	director,producer	tt0944947,tt1474684,tt86886
12	12	nm2643685	Bryan Cogman	1979.0	None	producer,miscellaneous,writer	tt0944947,tt4872
13	13	nm0372176	Lena Headey	1973.0	None	actress,producer,soundtrack	tt0944947,tt1374989,tt0416
14	14	nm0590889	Daniel Minahan	NaN	None	director,producer,writer	tt0944947,tt2788432,tt1856
15	15	nm0260870	Jane Espenson	1964.0	None	producer,writer,miscellaneous	tt0118276,tt1338724,tt20560
16	16	nm3592338	Emilia Clarke	1986.0	None	actress,soundtrack	tt2674426,tt3778644,tt0944
17	17	nm0851930	Alan Taylor	1965.0	None	director,producer,writer	tt0944947,tt0282768,tt1981
18	18	nm1014697	Ramin Djawadi	1974.0	None	composer,soundtrack,music_department	tt0944947,tt0475784,tt0371
19	19	nm0227759	Peter Dinklage	1969.0	None	actor,producer,soundtrack	tt0944947,tt1877832,tt0340

	id	nconst	primaryName	birthYear	deathYear	primaryProfession	kn
20	20	nm0146529	Bernadette Caulfield	NaN	None	producer,production_manager,miscellaneous	tt12141112,tt0944947,tt0421
21	21	nm0318821	Aidan Gillen	1968.0	None	actor,writer,producer	tt4500922,tt4046784,tt13458
22	22	nm0002399	Alik Sakharov	1959.0	None	cinematographer,director,producer	tt1856010,tt5071412,tt01418
23	23	nm0007008	David Petrarca	NaN	None	director,producer	tt0421030,tt0844441,tt0944
24	24	nm0961827	Vanessa Taylor	NaN	None	producer,writer,miscellaneous	tt0944947,tt1840309,tt67728
25	25	nm0638354	David Nutter	1960.0	None	director,producer,soundtrack	tt0112173,tt0374463,tt0944
26	26	nm0001097	Charles Dance	1946.0	None	actor,director,writer	tt0944947,tt0107362,tt2084
27	27	nm0192377	Liam Cunningham	1961.0	None	actor,director,producer	tt0944947,tt0800320,tt04609
28	28	nm0551076	Neil Marshall	1970.0	None	producer,director,writer	tt0483607,tt9182964,tt02806
29	29	nm0817770	Greg Spence	NaN	None	producer,production_manager,editorial_department	tt0460829,tt0944947,tt0411
30	30	nm3229685	Kit Harington	1986.0	None	actor,writer,producer	tt9032400,tt1921064,tt09383
31	31	nm0628040	Christopher Newman	1955.0	None	assistant_director,producer,miscellaneous	tt0120915,tt0107943,tt01859
32	32	nm0336241	Alex Graves	NaN	None	producer,director,writer	tt0200276,tt0112746,tt35958
33	33	nm0533713	Michelle MacLaren	1965.0	None	producer,director,production_manager	tt4998350,tt2953250,tt0944
34	34	nm0322513	Iain Glen	1961.0	None	actor,producer,soundtrack	tt6954652,tt0944947,tt68598
35	35	nm0534635	Richard Madden	1986.0	None	actor,producer,soundtrack	tt2066051,tt2368619,tt0944
36	36	nm4263213	John Bradley	1988.0	None	actor,soundtrack	tt5451690,tt0944947,tt58344
37	37	nm2356940	Hannah Murray	1989.0	None	actress,soundtrack	tt5390504,tt4180576,tt2141
38	38	nm3310211	Rose Leslie	1987.0	None	actress	tt1618442,tt0944947,tt4520
39	39	nm0806252	Michael Slovis	NaN	None	cinematographer,producer,director	tt7817340,tt0981227,tt0247
40	40	nm4984276	Dave Hill	NaN	None	miscellaneous,writer,producer	tt7937220,tt7462410,tt0944

	id	nconst	primaryName	birthYear	deathYear	primaryProfession	kn
41	41	nm0617042	Mark Mylod	NaN	None	director,producer,miscellaneous	tt0944947,tt7660850,tt1586
42	42	nm0226820	Stephen Dillane	1957.0	None	actor	tt0944947,tt0266987,tt0274
43	43	nm0687964	Jeremy Podeswa	1962.0	None	director,miscellaneous,writer	tt10574236,tt5834204,tt0374
44	44	nm0764601	Miguel Sapochnik	1974.0	None	director,producer,writer	tt1053424,tt3420504,tt0944
45	45	nm1380874	Lisa McAtackney	NaN	None	producer,production_manager,miscellaneous	tt0944947,tt0804552,tt0465
46	46	nm0755261	Daniel Sackheim	NaN	None	producer,director,editorial_department	tt0120902,tt0412142,tt0221
47	47	nm0070474	Jack Bender	1949.0	None	director,producer,actor	tt0411008,tt0944947,tt0285
48	48	nm1754059	Natalie Dormer	1982.0	None	actress,writer,producer	tt10361016,tt0944947,tt0758
49	49	nm0787687	Matt Shakman	1975.0	None	director,producer,actor	tt9140560,tt0944947,tt0472
50	50	nm2977599	Gursimran Sandhu	1986.0	None	miscellaneous,writer,director	tt2561572,tt1910550,tt094
51	51	nm7260047	Ethan J. Antonucci	NaN	None	miscellaneous,writer	

In [59]:
```
titles_cast_df
```

Out[59]:

	nconst	tconst	characters
0	nm0000293	tt1480055	["Eddard 'Ned' Stark"]
1	nm0004692	tt1480055	["Robert Baratheon"]
2	nm0182666	tt1480055	["Jaime Lannister"]
3	nm0000293	tt1668746	["Eddard 'Ned' Stark"]
4	nm0004692	tt1668746	["Robert Baratheon"]
...	...	...	...
160	nm0182666	tt6027914	["Jaime Lannister"]
161	nm0227759	tt6027916	["Tyrion Lannister"]
162	nm0182666	tt6027916	["Jaime Lannister"]
163	nm0227759	tt6027920	["Tyrion Lannister"]
164	nm0182666	tt6027920	["Jaime Lannister"]

165 rows × 3 columns

Load Data into Neo4j

In [60]:
```
# Up until now, we have been directly using query languages to interact with databases.
# There are many higher layer language libraries that make using the databases "simpler."
# I avoided using these libraries because they obscure what is happening and hide the query language.
# Learning the query language(s) is important.
#
# py2neo and other tools provide an "object-mapping" layers that maps between the data in the databases
# and classes/objects in the programming language you are using. py2neo implements classes Node and Relationsl
#
from py2neo.data import Node, Relationship
```

In [61]:
```
graph = Graph(aura_url, auth=(aura_user, aura_pw))
```

- The following are a couple of helper functions that will simplify the homework.

In [62]:
```
# The matcher classes simplify using the query language from withon Python and classes.
#
```

```
from py2neo.matching import *
```

In [63]:
```
def get_nodes(label, template):
    """

    :param label: The label of the node
    :template: A dictionary of property, value pairs to match.
    :return: A list of matching nodes.
    """
    nodes = NodeMatcher(graph)
    the_match = nodes.match(label, **template)
    result = []
    for n in the_match:
        result.append(n)
    return result
```

In [64]:
```
keanu = get_nodes("Person", { "name": "Keanu Reeves" })
```

In [65]:
```
keanu
```

Out[65]:
```
[Node('Person', born=1964, name='Keanu Reeves')]
```

In [66]:
```
def create_node(n):
    """
    :template: A py2neo Node
    :return: None
    """
    tx = graph.begin()
    tx.create(n)
    graph.commit(tx)
```

In [67]:
```
# Create an GOT_Actor node.
#
sb = Node('GOT_Actor', name="Sean Bean", nconst="nm0000293", born=1959)
```

In [68]:
```
create_node(sb)
```

In [69]:
```
mf = Node('GOT_Actor', name="Michelle Fairley", nconst="nm0265610", born=1963)
create_node(mf)
```

In [70]:
```
# Get what we created.
got_a = get_nodes('GOT_Actor', {})
```

In [71]:
```
got_a
```

Out[71]:
```
[Node('GOT_Actor', born=1959, name='Sean Bean', nconst='nm0000293'),
 Node('GOT_Actor', born=1963, name='Michelle Fairley', nconst='nm0265610')]
```

In [72]:
```
# Create an episode.
episode_data = {'seasonNum': 1,
  'episodeNum': 1,
  'episodeTitle': 'Winter Is Coming',
  'tconst': 'tt1480055',
  'episodeAirDate': '2011-04-17',
  'episodeDescription': "Jon Arryn, the Hand of the King, is dead. King Robert Baratheon plans to ask his olde
}
episode_node = Node("GOT_Episode", **episode_data)
```

In [73]:
```
create_node(episode_node)
```

- From the table `got_title_cast`, we can find out how acted in what episodes and roles.
- A little probing shows that Sean Bean and Michelle Fairley were in season 1, episode 1.
- So, I can create a relationship.

In [74]:
```
ep = get_nodes("GOT_Episode", {"seasonNum": 1, "episodeNum": 1})
ep[0]["tconst"]
```

Out[74]:
```
'tt1480055'
```

In [75]:
```
sb = get_nodes("GOT_Actor", { "nconst": "nm0000293"})
sb
```

Out[75]:
```
[Node('GOT_Actor', born=1959, name='Sean Bean', nconst='nm0000293')]
```

In [76]:
```
mf = get_nodes("GOT_Actor", { "nconst": "nm0265610"})
mf
```

Out[76]:
```
[Node('GOT_Actor', born=1963, name='Michelle Fairley', nconst='nm0265610')]
```

In [77]:
```
sb_roles = %sql select characters from w4111_f22_hw4.got_title_cast where \
    nconst='nm0000293' and tconst='tt1480055'
sb_roles[0]["characters"]
```

```
 * mysql+pymysql://root:***@localhost
1 rows affected.
```

Out[77]:
```
'["Eddard \'Ned\' Stark"]'
```

In [78]:
```
ep
```

Out[78]:
```
[Node('GOT_Episode', episodeAirDate='2011-04-17', episodeDescription="Jon Arryn, the Hand of the King, is dea
d. King Robert Baratheon plans to ask his oldest friend, Eddard Stark, to take Jon's place. Across the sea, V
iserys Targaryen plans to wed his sister to a nomadic warlord in exchange for an army.", episodeNum=1, episod
eTitle='Winter Is Coming', seasonNum=1, tconst='tt1480055')]
```

In [79]:
```
sb
```

Out[79]:
```
[Node('GOT_Actor', born=1959, name='Sean Bean', nconst='nm0000293')]
```

In [80]:
```
def create_relationship(source, label, target, properties):
    tx = graph.begin()
    r = Relationship(source, label, target)
    tx.create(r)
    graph.commit(tx)
```

In [81]:
```
r = create_relationship(sb[0], "GOT_Acted_In", ep[0], {"roles": ["Eddard 'Ned' Start"]})
```

In [82]:
```
r = create_relationship(mf[0], "GOT_Acted_In", ep[0], {"roles": ["Catelyn Stark"]})
```

- When I run a query in my Neo4j console, I get the following.

__Simple Query Result

- Your task is to now use your skills and my helper code to load episodes, actors and relationships.
- Please put your code below, and then past some screenshots.

In [83]:
```
cast_list = cast_df.to_dict(orient='record')
```

```
/var/folders/xh/jjqjnhjj0m3gv9697kb8zk0r0000gn/T/ipykernel_9278/1876656935.py:1: FutureWarning: Using short n
ame for 'orient' is deprecated. Only the options: ('dict', list, 'series', 'split', 'records', 'index') will
be used in a future version. Use one of the above to silence this warning.
  cast_list = cast_df.to_dict(orient='record')
```

In [84]:
```
import math
for c in cast_list:
    actor_node = Node('GOT_Actor', name=c['primaryName'], nconst=c['nconst'], born=c['birthYear'] if math.isna
    create_node(actor_node)
```

In [85]:
```
episode_list = episodes_df.to_dict(orient='record')
```

```
/var/folders/xh/jjqjnhjj0m3gv9697kb8zk0r0000gn/T/ipykernel_9278/3227841542.py:1: FutureWarning: Using short n
ame for 'orient' is deprecated. Only the options: ('dict', list, 'series', 'split', 'records', 'index') will
be used in a future version. Use one of the above to silence this warning.
  episode_list = episodes_df.to_dict(orient='record')
```

In [86]:
```
episode_list[0]
```

Out[86]:
```
{'seasonNum': 1,
 'episodeNum': 1,
 'episodeTitle': 'Winter Is Coming',
 'tconst': 'tt1480055',
 'episodeAirDate': '2011-04-17',
 'episodeDescription': "Jon Arryn, the Hand of the King, is dead. King Robert Baratheon plans to ask his olde
st friend, Eddard Stark, to take Jon's place. Across the sea, Viserys Targaryen plans to wed his sister to a
nomadic warlord in exchange for an army."}
```

In [87]:
```
for e in episode_list:
    episode_node = Node('GOT_Episode', **e)
    create_node(episode_node)
```

In [88]:
```
titles_cast_list = titles_cast_df.to_dict(orient='record')
```

```
/var/folders/xh/jjqjnhjj0m3gv9697kb8zk0r0000gn/T/ipykernel_9278/1931331461.py:1: FutureWarning: Using short n
ame for 'orient' is deprecated. Only the options: ('dict', list, 'series', 'split', 'records', 'index') will
be used in a future version. Use one of the above to silence this warning.
  titles_cast_list = titles_cast_df.to_dict(orient='record')
```

In [89]:
```
titles_cast_list[10]
```

Out[89]:
```
{'nconst': 'nm0004692',
 'tconst': 'tt1829963',
 'characters': '["Robert Baratheon"]'}
```

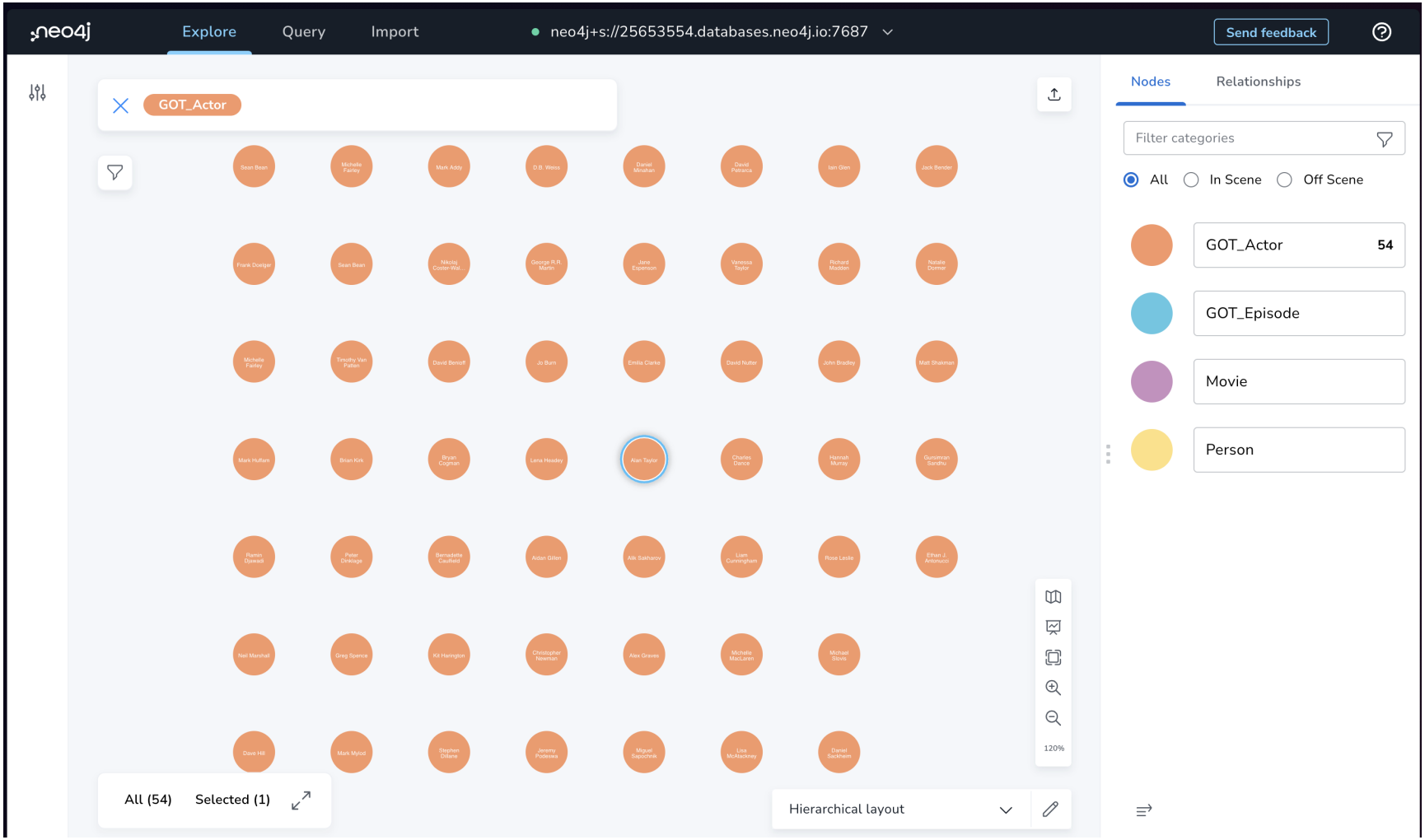
In [90]:
```
for tc in titles_cast_list:
    actor_node = get_nodes('GOT_Actor', {'nconst': tc['nconst']})
    actor_node = actor_node[0]

    episode_node = get_nodes('GOT_Episode', {'tconst': tc['tconst']})
    episode_node = episode_node[0]

    create_relationship(actor_node, 'GOT_Acted_In', episode_node,
                        {'role_played': tc['characters']})
```

In [1]:
```
from IPython.display import Image
Image("actors.png")
```

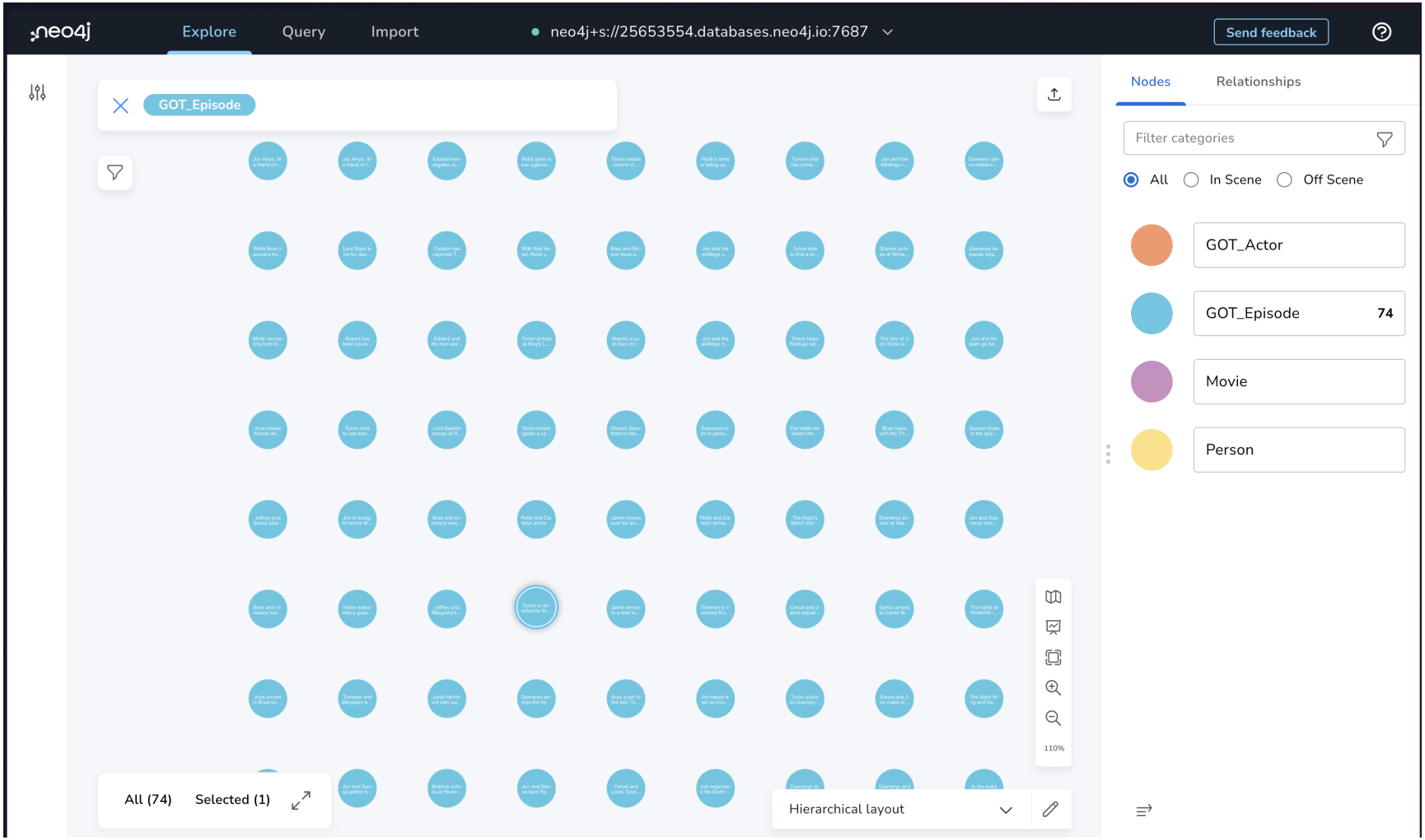
Out[1]:



In [2]:

```
Image("episodes.png")
```

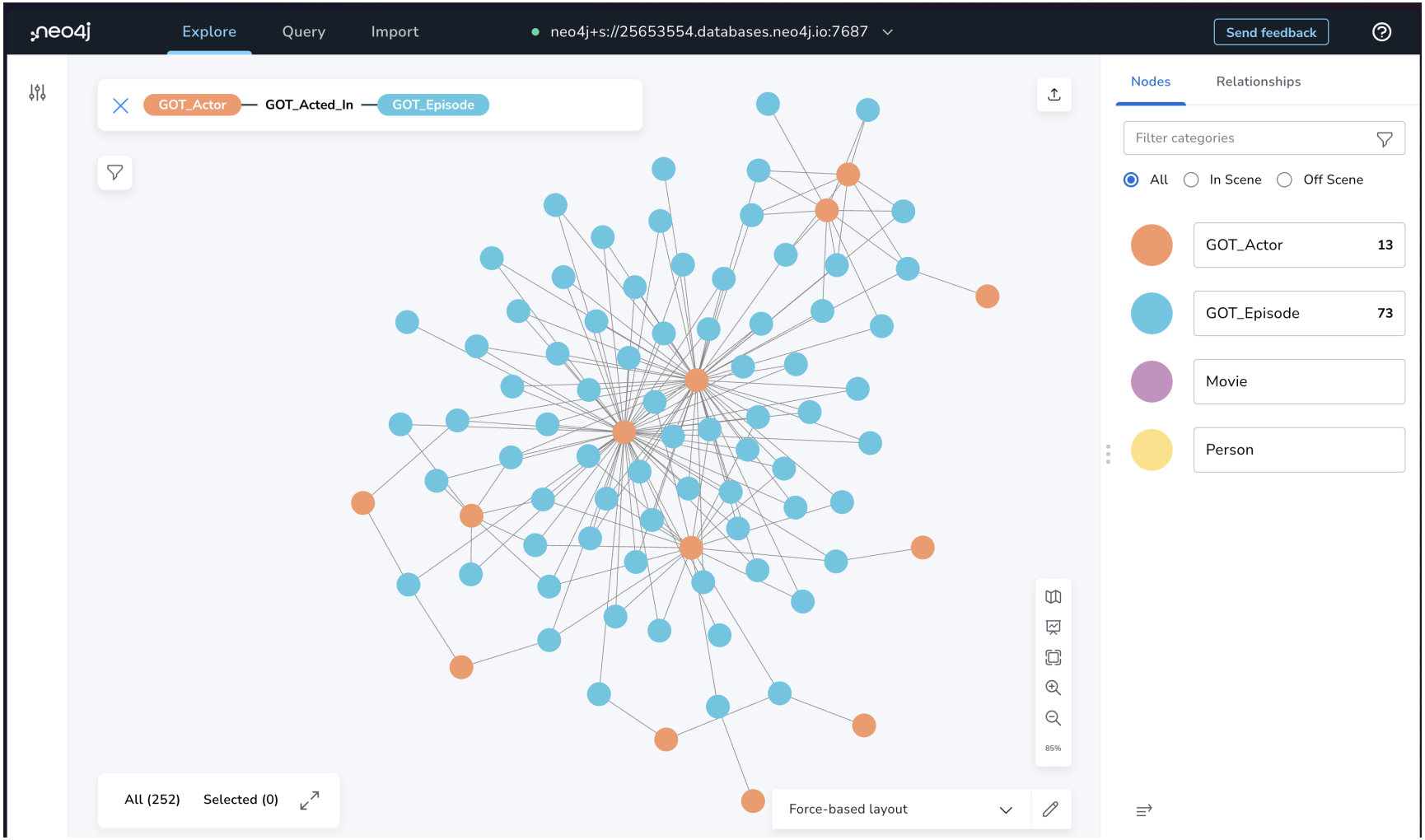
Out[2]:



In [3]:

```
Image("acted_in.png")
```

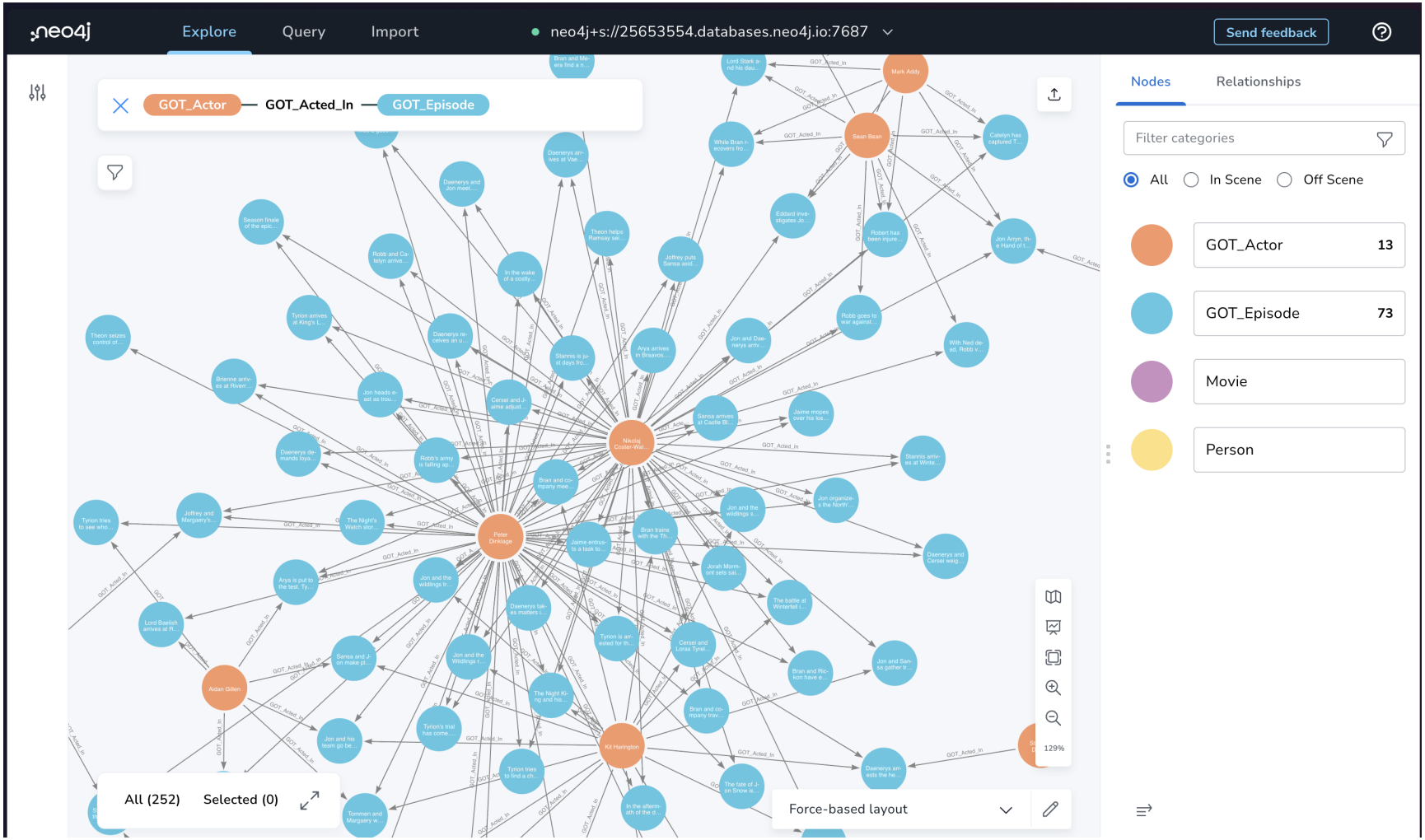
Out[3]:



In [4]:

```
Image("acted_in_2.png")
```

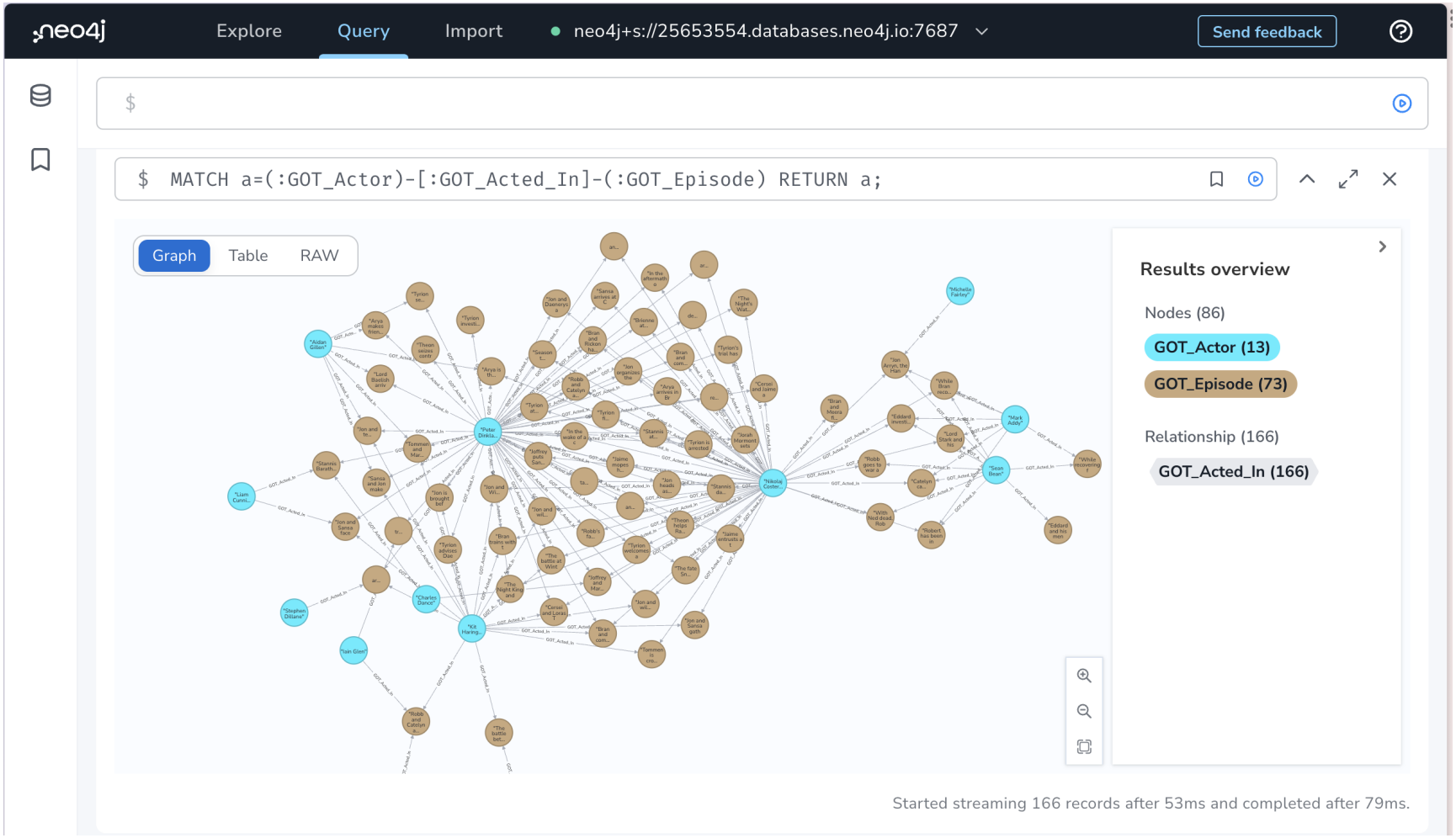
Out[4]:



In [5]:

```
Image("match1.png")
```

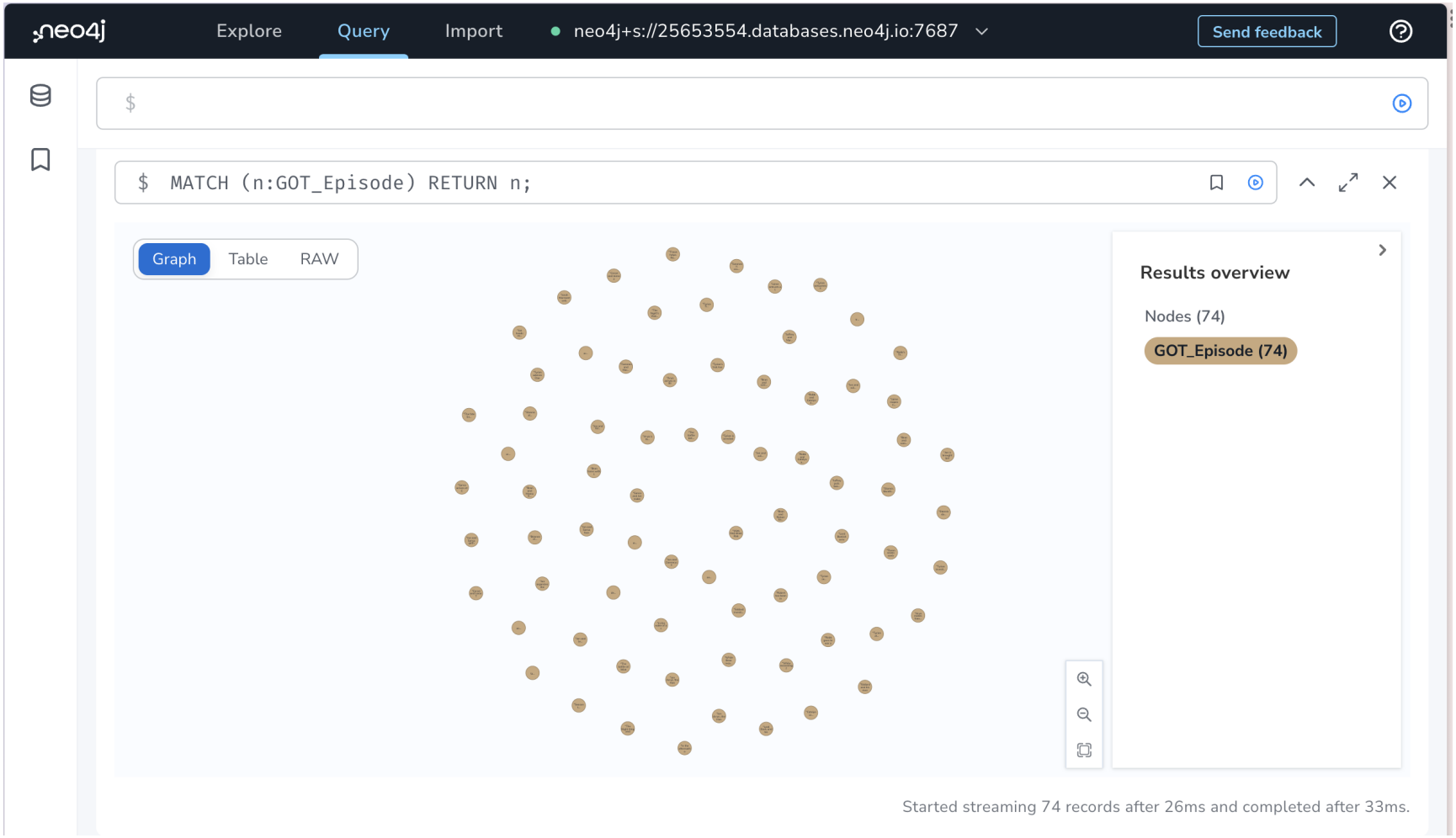
Out[5]:



In [6]:

```
Image("match2.png")
```

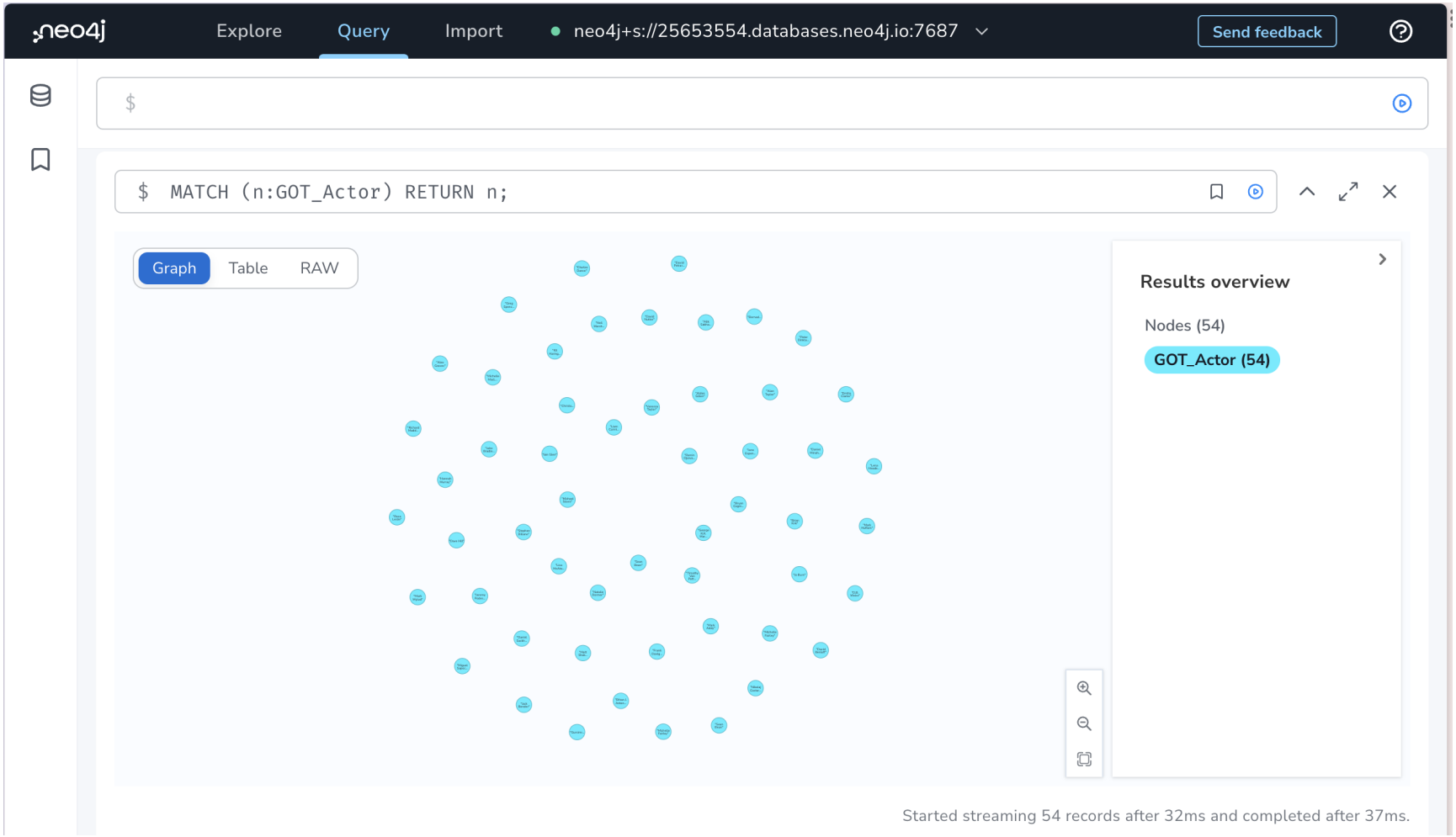
Out[6]:



In [7]:

```
Image("match3.png")
```

Out[7]:



Programming Track

- I am canceling the programming track. I will explain how to use what we just did in a web application, but piling on more work at this point would be cruel.

Non-Programming Track

- I am canceling the non-programming track. I will explain how to use what we just did in data science, but piling on more work at this point would be cruel.

In [ ]: