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:dbuserdbuser@localhost
```

In [4]:

```
#
# Note that your list of databases will be different.
#
%sql show databases;
```

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

Out[4]:

Database
classicmodels
columbia_university
db_book
f22_hw1_got
f22_hw1_got_programming
f22_hw2
f22_midterm
information_schema
lahmansbaseballdb
mysql
performance_schema
sakila
sys
w4111_f22_hw4
world

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/donaldferguson/Dropbox/00NewProjects/F22-W4111-HW4')
```

In [6]:

```
# import my_secrets
```

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

In [7]:

```
# !pip install py2neo
```

In [6]:

```
from py2neo import Graph
```

In [7]:

```
aura_url = 'neo4j+s://142430a7.databases.neo4j.io'
aura_user = 'neo4j'
aura_pw = 'p1DygPT9TKfZxnMB9poFcU0Bq1um7kw47-AHwFvg-uM'
```

In [8]:

```
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 [9]:

```
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. (https://www.mongodb.com/docs/atlas/tutorial/connect-to-your-cluster/)
- You also need to pip/Conda install pymongo.
- You will have to your URL for connecting.

In [12]:

```
# !pip install pymongo
```

```
Collecting pymongo
  Downloading pymongo-4.3.3-cp39-cp39-win_amd64.whl (382 kB)
Collecting dnspython<3.0.0,>=1.16.0
  Downloading dnspython-2.2.1-py3-none-any.whl (269 kB)
Installing collected packages: dnspython, pymongo
Successfully installed dnspython-2.2.1 pymongo-4.3.3
```

In [44]:

```
import pymongo
```

In [42]:

```
def connect():
    client = pymongo.MongoClient(
        'mongodb+srv://tedwu:W1fuq2peUOn5veXA@comp4111.rgzkcri.mongodb.net'
    )
    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': 'admin', 'sizeOnDisk': 380928, 'empty': False}, {'name': 'local', 'sizeOnDisk': 7493640192, 'empt
y': False}]
```

- The following will do some additional testing.

In [16]:

```
client = pymongo.MongoClient(
        'mongodb+srv://tedwu:No.25_Aminor@comp4111.rgzkcri.mongodb.net'
    )
```

In [17]:

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

Out[17]:

```
<pymongo.results.InsertOneResult at 0x1932d6c83a0>
```

In [18]:

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

Out[18]:

```
<pymongo.results.InsertOneResult at 0x1932d455fa0>
```

In [19]:

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

In [20]:

```
some_info = list(res)
some_info
```

Out[20]:

```
[{'_id': ObjectId('6395263843d7a3e7ae9d8a35'),
  'class': 'W4111',
  'is_cool': True},
 {'_id': ObjectId('6395263d43d7a3e7ae9d8a36'),
  '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 [35]:

```
import json
```

In [36]:

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

In [37]:

```
characters = characters['characters']
```

In [38]:

```
characters[0:2]
```

Out[38]:

```
[{'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 [39]:

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

In [40]:

```
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 [45]:

```
# Your code here.
## Connect to MongoDB
client = connect()
## Access the database hw4, or create it if is not exist
db = client["hw4"]
## Switch to episodes collection
collection = db["episodes"]
## Insert episodes document to database
collection.insert_many(episodes)
## Switch to characters collection
collection = db["characters"]
## Insert characters document to database
collection.insert_many(characters)
```

Out[45]:

```
<pymongo.results.InsertManyResult at 0x2c189ee6fd0>
```

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

In [46]:

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

In [47]:

```
ep
```

Out[47]:


```
{'_id': ObjectId('639529ac43d7a3e7ae9d8a38'),
 '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 oldest friend,
Eddard Stark, to take Jon's place. Across the sea, Viserys Targaryen plans to wed his sister to a nomadic warlord in ex
change 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'},
```


In [31]:

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

In [32]:

```
cc
```

Out[32]:


```
{'_id': ObjectId('639529ad43d7a3e7ae9d8bad'),
 '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 [19]:

```
import pandas as pd
import numpy as np
import pymysql

# Data reading and cleaning
# got_cast
got_cast_csv = pd.read_csv('.\data\got_cast.csv', index_col=False, delimiter = ',')
del got_cast_csv["id"]
got_cast_csv = got_cast_csv.where(pd.notnull(got_cast_csv), None)
got_cast_csv = got_cast_csv.astype(object).replace(np.nan, None)
# got_cast_details
got_cast_details_csv = pd.read_csv('.\data\got_cast_details.csv', index_col=False, delimiter = ',')
del got_cast_details_csv["id"]
got_cast_details_csv = got_cast_details_csv.where(pd.notnull(got_cast_details_csv), None)
got_cast_details_csv = got_cast_details_csv.astype(object).replace(np.nan, None)
# got_episodes
got_episodes_csv = pd.read_csv('.\data\got_episodes.csv', index_col=False, delimiter = ',')
del got_episodes_csv["id"]
got_episodes_csv = got_episodes_csv.where(pd.notnull(got_episodes_csv), None)
got_episodes_csv = got_episodes_csv.astype(object).replace(np.nan, None)
# got_title_ratings
got_title_ratings_csv = pd.read_csv('.\data\got_title_ratings.csv', index_col=False, delimiter = ',')
del got_title_ratings_csv["id"]
got_title_ratings_csv = got_title_ratings_csv.where(pd.notnull(got_title_ratings_csv), None)
got_title_ratings_csv = got_title_ratings_csv.astype(object).replace(np.nan, None)
# got_title_cast
got_title_cast_csv = pd.read_csv('.\data\got_title_cast.csv', index_col=False, delimiter = ',')
got_title_cast_csv = got_title_cast_csv.where(pd.notnull(got_title_cast_csv), None)
got_title_cast_csv = got_title_cast_csv.astype(object).replace(np.nan, None)

# Connect to MySQL database
conn = pymysql.connect(
    user="root",
    password="dbuserdbuser",
    host="localhost",
    autocommit=True,
    cursorclass=pymysql.cursors.DictCursor,
)
cursor = conn.cursor()

# Create database
## cursor.execute("CREATE DATABASE w4111_f22_hw4")
cursor.execute('use w4111_f22_hw4;')

# Build table go_cast
cursor.execute('DROP TABLE IF EXISTS got_cast;')
cursor.execute("CREATE TABLE got_cast(tconst varchar(255),ordering decimal(4),\
                nconst varchar(255),category varchar(255), job varchar(255), \
                characters varchar(255))")
for i,row in got_cast_csv.iterrows():
    sql = "INSERT INTO w4111_f22_hw4.got_cast VALUES (%s,%s,%s,%s,%s,%s)"
    cursor.execute(sql, tuple(row))
    conn.commit()

# Build table got_cast_details
cursor.execute('DROP TABLE IF EXISTS got_cast_details;')
cursor.execute("CREATE TABLE got_cast_details(nconst varchar(255),primaryName \
                varchar(255), birthYear decimal(4),deathYear decimal(4),\
                primaryProfession varchar(255),knownForTitles varchar(255))")
for i,row in got_cast_details_csv.iterrows():
    sql = "INSERT INTO w4111_f22_hw4.got_cast_details VALUES (%s,%s,%s,%s,%s,%s)"
    cursor.execute(sql, tuple(row))
    conn.commit()

# Build table got_episodes
cursor.execute('DROP TABLE IF EXISTS got_episodes;')
cursor.execute("CREATE TABLE got_episodes(seasonNum decimal(4), episodeNum decimal(4), \
                episodeTitle varchar(255), episodeLink varchar(255), episodeAirDate DATE, \
                episodeDescription varchar(255))")
for i,row in got_episodes_csv.iterrows():
    sql = "INSERT INTO w4111_f22_hw4.got_episodes VALUES (%s,%s,%s,%s,%s,%s)"
    cursor.execute(sql, tuple(row))
    conn.commit()

# Build table got_title_ratings
cursor.execute('DROP TABLE IF EXISTS got_title_ratings;')
cursor.execute("CREATE TABLE got_title_ratings(tconst varchar(255), averageRating double, \
                numVotes decimal(6))")
for i,row in got_title_ratings_csv.iterrows():
    sql = "INSERT INTO w4111_f22_hw4.got_title_ratings VALUES (%s,%s,%s)"
    cursor.execute(sql, tuple(row))
```

```
    conn.commit()

# Build table got_title_cast
cursor.execute('DROP TABLE IF EXISTS got_title_cast;')
cursor.execute("CREATE TABLE got_title_cast(nconst varchar(255), tconst varchar(255), \
                characters varchar(255))")
for i,row in got_title_cast_csv.iterrows():
    sql = "INSERT INTO w4111_f22_hw4.got_title_cast VALUES (%s,%s,%s)"
    cursor.execute(sql, tuple(row))
    conn.commit()
```

- The following will test your loading.

In [51]:

```
%sql select * from w4111_f22_hw4.got_cast limit 10;
```

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

Out[51]:

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

In [52]:

```
%sql select * from w4111_f22_hw4.got_cast_details limit 10;
```

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

Out[52]:

nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
nm0230361	Frank Doelger	None	None	producer,director,writer	tt0985040,tt0139780,tt0944947,tt0202179
nm0000293	Sean Bean	1959	None	actor,producer,animation_department	tt1181791,tt0944947,tt0120737,tt0167261
nm0004692	Mark Addy	1964	None	actor,soundtrack	tt0183790,tt0944947,tt0119164,tt0955308
nm0182666	Nikolaj Coster-Waldau	1970	None	actor,producer,writer	tt0110631,tt1483013,tt0944947,tt2404233
nm0265610	Michelle Fairley	1963	None	actress	tt2431286,tt0944947,tt1390411,tt11188010
nm0887700	Timothy Van Patten	1959	None	director,actor,producer	tt0083739,tt0979432,tt0374463,tt0141842
nm1125275	David Benioff	1970	None	producer,writer,director	tt1025100,tt0944947,tt0419887,tt0307901
nm1888967	D.B. Weiss	1971	None	writer,producer,director	tt11547156,tt12141112,tt3677548,tt0944947
nm0552333	George R.R. Martin	1948	None	writer,producer,miscellaneous	tt0944947,tt11198330,tt0088634,tt0092319
nm0122251	Jo Burn	None	None	miscellaneous,production_manager,producer	tt1596342,tt4530422,tt5697572,tt2798920

In [53]:

```
%sql select * from w4111_f22_hw4.got_episodes limit 10;
```

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

Out[53]:

seasonNum	episodeNum	episodeTitle	episodeLink	episodeAirDate	episodeDescription
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	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.
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.
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.
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.
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.
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.
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.
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.
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 [54]:

```
%sql select * from w4111_f22_hw4.got_title_ratings limit 10;
```

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

Out[54]:

tconst	averageRating	numVotes
tt1480055	8.9	48686
tt1668746	8.6	36837
tt1829962	8.5	34863
tt1829963	8.6	33136
tt1829964	9.0	34436
tt1837862	9.1	34050
tt1837863	9.1	34564
tt1837864	8.9	32339
tt1851398	9.6	45134
tt1851397	9.4	39748

In [55]:

```
%sql select * from w4111_f22_hw4.got_title_cast limit 10;
```

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

Out[55]:

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 [20]:

```
# Put your code here to read the data.
#
from sqlalchemy import create_engine
engine = create_engine("mysql+pymysql://root:dbuserdbuser@localhost/w4111_f22_hw4").connect()

episodes_df = pd.read_sql_table('got_episodes', engine)
episodes_df['episodeLink'] = episodes_df['episodeLink'].replace('/title/', '', regex=True).replace('/', '', regex=True)
episodes_df = episodes_df.rename(columns={'episodeLink': 'tconst'})
episodes_df = episodes_df.astype({"seasonNum":"int","episodeNum":"int","episodeAirDate":"str"})

cast_df = pd.read_sql_table('got_cast_details', engine)
cast_df['birthYear'] = cast_df['birthYear'].fillna(0)
cast_df = cast_df.astype({"birthYear":"int"})
cast_df['birthYear'] = cast_df['birthYear'].replace(0, None)
cast_df = cast_df.astype(object).replace(np.nan, None)

titles_cast_df = pd.read_sql_table('got_title_cast', engine)
```

In [21]:

```
import pandas
```

In [22]:

```
from sqlalchemy import create_engine
```

In [23]:

```
engine = create_engine("mysql+pymysql://root:dbuserdbuser@localhost")
```

- The following tests whether or not your loading worked.

In [24]:

```
episodes_df
```

Out[24]:

	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 [25]:

```
cast_df
```

Out[25]:

	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
0	nm0230361	Frank Doelger	None	None	producer,director,writer	tt0985040,tt0139780,tt0944947,tt0202179
1	nm0000293	Sean Bean	1959	None	actor,producer,animation_department	tt1181791,tt0944947,tt0120737,tt0167261
2	nm0004692	Mark Addy	1964	None	actor,soundtrack	tt0183790,tt0944947,tt0119164,tt0955308
3	nm0182666	Nikolaj Coster-Waldau	1970	None	actor,producer,writer	tt0110631,tt1483013,tt0944947,tt2404233
4	nm0265610	Michelle Fairley	1963	None	actress	tt2431286,tt0944947,tt1390411,tt11188010
5	nm0887700	Timothy Van Patten	1959	None	director,actor,producer	tt0083739,tt0979432,tt0374463,tt0141842
6	nm1125275	David Benioff	1970	None	producer,writer,director	tt1025100,tt0944947,tt0419887,tt0307901
7	nm1888967	D.B. Weiss	1971	None	writer,producer,director	tt11547156,tt12141112,tt3677548,tt0944947
8	nm0552333	George R.R. Martin	1948	None	writer,producer,miscellaneous	tt0944947,tt11198330,tt0088634,tt0092319
9	nm0122251	Jo Burn	None	None	miscellaneous,production_manager,producer	tt1596342,tt4530422,tt5697572,tt2798920
10	nm0400240	Mark Huffam	None	None	producer,production_manager,assistant_director	tt11214590,tt3659388,tt0120815,tt11138512
11	nm1047532	Brian Kirk	1968	None	director,producer	tt0944947,tt1474684,tt8688634,tt1949720
12	nm2643685	Bryan Cogman	1979	None	producer,miscellaneous,writer	tt0944947,tt4872762,tt2207542
13	nm0372176	Lena Headey	1973	None	actress,producer,soundtrack	tt0944947,tt1374989,tt0416449,tt1343727
14	nm0590889	Daniel Minahan	None	None	director,producer,writer	tt0944947,tt2788432,tt1856010,tt0251031
15	nm0260870	Jane Espenson	1964	None	producer,writer,miscellaneous	tt0118276,tt1338724,tt2056091,tt0944947
16	nm3592338	Emilia Clarke	1986	None	actress,soundtrack	tt2674426,tt3778644,tt0944947,tt1340138
17	nm0851930	Alan Taylor	1965	None	director,producer,writer	tt0944947,tt0282768,tt1981115,tt1340138
18	nm1014697	Ramin Djawadi	1974	None	composer,soundtrack,music_department	tt0944947,tt0475784,tt0371746,tt1663662
19	nm0227759	Peter Dinklage	1969	None	actor,producer,soundtrack	tt0944947,tt1877832,tt0340377,tt5027774
20	nm0146529	Bernadette Caulfield	None	None	producer,production_manager,miscellaneous	tt12141112,tt0944947,tt0421030,tt0106179
21	nm0318821	Aidan Gillen	1968	None	actor,writer,producer	tt4500922,tt4046784,tt1345836,tt0944947
22	nm0002399	Alik Sakharov	1959	None	cinematographer,director,producer	tt1856010,tt5071412,tt0141842,tt0944947
23	nm0007008	David Petrarca	None	None	director,producer	tt0421030,tt0844441,tt0944947,tt5743796
24	nm0961827	Vanessa Taylor	None	None	producer,writer,miscellaneous	tt0944947,tt1840309,tt6772802,tt5580390
25	nm0638354	David Nutter	1960	None	director,producer,soundtrack	tt0112173,tt0374463,tt0944947,tt3107288
26	nm0001097	Charles Dance	1946	None	actor,director,writer	tt0944947,tt0107362,tt2084970,tt0280707
27	nm0192377	Liam Cunningham	1961	None	actor,director,producer	tt0944947,tt0800320,tt0460989,tt0986233
28	nm0551076	Neil Marshall	1970	None	producer,director,writer	tt0483607,tt9182964,tt0280609,tt0435625
29	nm0817770	Greg Spence	None	None	producer,production_manager,editorial_department	tt0460829,tt0944947,tt0411951,tt0416044
30	nm3229685	Kit Harington	1986	None	actor,writer,producer	tt9032400,tt1921064,tt0938330,tt0944947
31	nm0628040	Christopher Newman	1955	None	assistant_director,producer,miscellaneous	tt0120915,tt0107943,tt0185906,tt0944947
32	nm0336241	Alex Graves	None	None	producer,director,writer	tt0200276,tt0112746,tt3595806,tt0944947
33	nm0533713	Michelle MacLaren	1965	None	producer,director,production_manager	tt4998350,tt2953250,tt0944947,tt0903747
34	nm0322513	Iain Glen	1961	None	actor,producer,soundtrack	tt6954652,tt0944947,tt6859806,tt10370380
35	nm0534635	Richard Madden	1986	None	actor,producer,soundtrack	tt2066051,tt2368619,tt0944947,tt1661199
36	nm4263213	John Bradley	1988	None	actor,soundtrack	tt5451690,tt0944947,tt5834426,tt10223460
37	nm2356940	Hannah Murray	1989	None	actress,soundtrack	tt5390504,tt4180576,tt2141751,tt0944947
38	nm3310211	Rose Leslie	1987	None	actress	tt1618442,tt0944947,tt4520364,tt3177316
39	nm0806252	Michael Slovis	None	None	cinematographer,producer,director	tt7817340,tt0981227,tt0247082,tt0903747
40	nm4984276	Dave Hill	None	None	miscellaneous,writer,producer	tt7937220,tt7462410,tt0944947,tt3391176

	nconst	primaryName	birthYear	deathYear	primaryProfession	knownForTitles
41	nm0617042	Mark Mylod	None	None	director,producer,miscellaneous	tt0944947,tt7660850,tt1586680,tt2699110
42	nm0226820	Stephen Dillane	1957	None	actor	tt0944947,tt0266987,tt0274558,tt4555426
43	nm0687964	Jeremy Podeswa	1962	None	director,miscellaneous,writer	tt10574236,tt5834204,tt0374463,tt0944947
44	nm0764601	Miguel Sapochnik	1974	None	director,producer,writer	tt1053424,tt3420504,tt0944947,tt11198330
45	nm1380874	Lisa McAtackney	None	None	producer,production_manager,miscellaneous	tt0944947,tt0804552,tt0465670,tt0774030
46	nm0755261	Daniel Sackheim	None	None	producer,director,editorial_department	tt0120902,tt0412142,tt0221218,tt2356777
47	nm0070474	Jack Bender	1949	None	director,producer,actor	tt0411008,tt0944947,tt0285333,tt0103956
48	nm1754059	Natalie Dormer	1982	None	actress,writer,producer	tt10361016,tt0944947,tt0758790,tt1951266
49	nm0787687	Matt Shakman	1975	None	director,producer,actor	tt9140560,tt0944947,tt0472954,tt2235759
50	nm2977599	Gursimran Sandhu	1986	None	miscellaneous,writer,director	tt2561572,tt1910550,tt0944947,tt9737326
51	nm7260047	Ethan J. Antonucci	None	None	miscellaneous,writer	tt0944947

In [26]:

```
titles_cast_df
```

Out[26]:

	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 [11]:

```
# 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 Relationship.
#
from py2neo.data import Node, Relationship
```

In [12]:

```
graph = Graph(aura_url, auth=(aura_user, aura_pw))
```

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

In [13]:

```
# The matcher classes simplify using the query language from withon Python and classes.
#
from py2neo.matching import *
```

In [14]:

```
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 [15]:

```
keanu = get_nodes("Person", { "name": "Keanu Reeves" })
```

In [16]:

```
keanu
```

Out[16]:

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

In [15]:

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

In [18]:

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

In [19]:

```
create_node(sb)
```

In [20]:

```
mf = Node('GOT_Actor', name="Michelle Fairley", nconst="nm0265610", born=1963)
create_node(mf)
```

In [21]:

```
# Get what we created.
got_a = get_nodes('GOT_Actor', {})
```

In [22]:

```
got_a
```

Out[22]:

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

In [23]:

```
# 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 oldest friend, Eddard Stark, to
}
episode_node = Node("GOT_Episode", **episode_data)
```

In [24]:

```
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 [25]:

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

Out[25]:

```
'tt1480055'
```

In [161]:

```
sb = get_nodes("GOT_Actor", { "nconst": "nm0000293"})
sb
```

Out[161]:

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

In [27]:

```
mf = get_nodes("GOT_Actor", { "nconst": "nm0265610"})
mf
```

Out[27]:

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

In [28]:

```
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[28]:

```
'["Eddard \'Ned\' Stark"]'
```

In [162]:

```
ep
```

Out[162]:

```
[Node('GOT_Episode', episodeAirDate='2011-04-17', episodeDescription="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.", episodeNum=1, episodeTitle='Winter Is Coming', seasonNum=1, tconst='tt1480055'),
 Node('GOT_Episode', episodeAirDate='2011-04-24', episodeDescription="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.", episodeNum=2, episodeTitle='The Kingsroad', seasonNum=1, tconst='tt1668746'),
 Node('GOT_Episode', episodeAirDate='2011-05-01', episodeDescription="Lord Stark and his daughters arrive at King's Landing to discover the intrigues of the king's realm.", episodeNum=3, episodeTitle='Lord Snow', seasonNum=1, tconst='tt1829962'),
 Node('GOT_Episode', episodeAirDate='2011-05-08', episodeDescription="Eddard investigates Jon Arryn's murder. Jon befriends Samwell Tarly, a coward who has come to join the Night's Watch.", episodeNum=4, episodeTitle='Cripples, Bastards, and Broken Things', seasonNum=1, tconst='tt1829963'),
 Node('GOT_Episode', episodeAirDate='2011-05-15', episodeDescription='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.', episodeNum=5, episodeTitle='The Wolf and the Lion', seasonNum=1, tconst='tt1829964'),
 Node('GOT_Episode', episodeAirDate='2011-05-22', episodeDescription='While recovering from his battle with Jaime, Edda
```

In [30]:

```
sb
```

Out[30]:

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

In [16]:

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

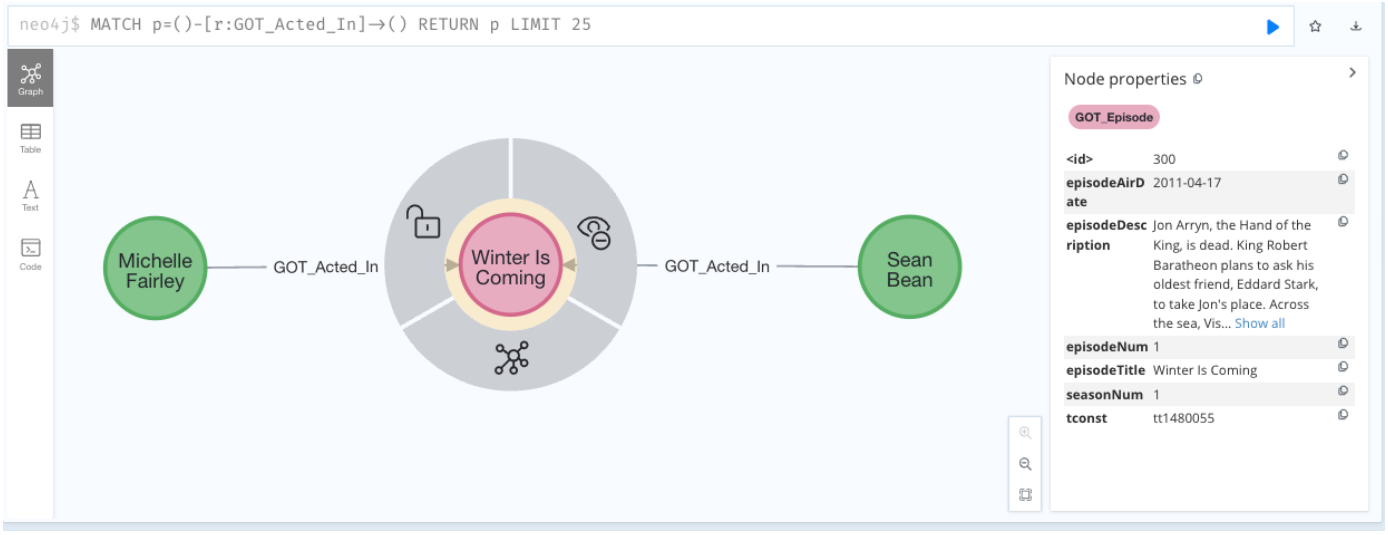
In [32]:

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

In [33]:

```
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 [195]:

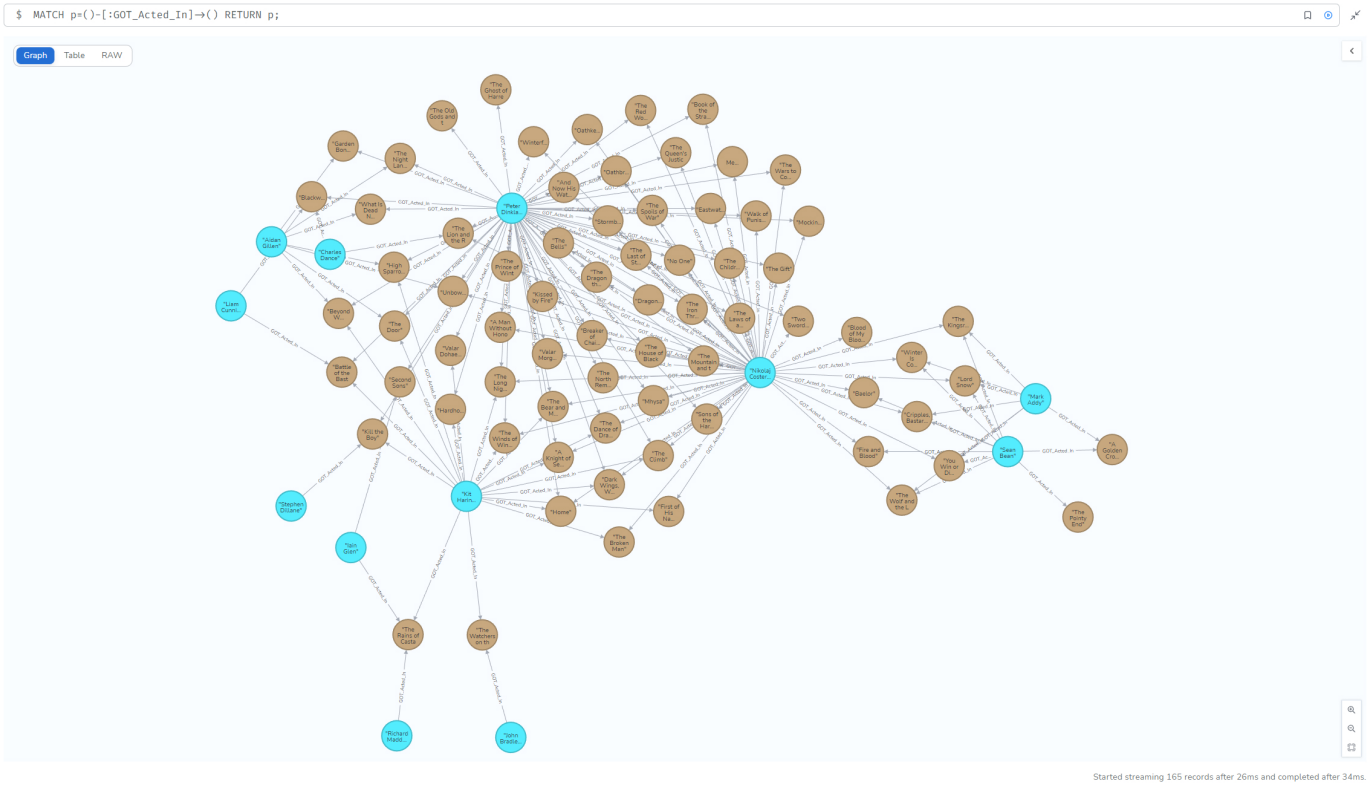
```
# Import data
for i in range(len(cast_df)):
    # Read name, nconst and born
    name_sql = cast_df.values[i][1]
    nconst_sql = cast_df.values[i][0]
    born_sql = cast_df.values[i][2]
    # Create node for each actor
    sb = Node('GOT_Actor', name=name_sql, nconst=nconst_sql, born=born_sql)
    create_node(sb)
for i in range(len(episodes_df)):
    episode_data = episodes_df.to_dict('records')[i]
    episode_node = Node("GOT_Episode", seasonNum=episode_data["seasonNum"], episodeNum=episode_data["episodeNum"],\
                        episodeTitle=episode_data["episodeTitle"], tconst=episode_data["tconst"], \
                        episodeAirDate=episode_data["episodeAirDate"], episodeDescription=episode_data["episodeDescription"])
    create_node(episode_node)
```

In [196]:

```
import json
# Create relationship
ep = get_nodes("GOT_Episode", {})
for i in range(len(ep)):
    # Find all actor and actress from each episode
    tconst = ep[i]["tconst"]
    query_str = "select nconst from w4111_f22_hw4.got_title_cast where tconst='tconst_tmp'"
    query_str = query_str.replace("tconst_tmp", tconst)
    nconst = %sql $query_str
    # Dig all actors' detail from Neo4j
    for j in range(len(nconst)):
        neo_query = '{ "nconst": "nconst_tmp"}'
        neo_query = json.loads(neo_query.replace("nconst_tmp", nconst[j]['nconst']))
        actor_detail = get_nodes("GOT_Actor", neo_query)
        ## print(actor_detail)
        # Find character for this actor/actress
        query2_str = "select characters from w4111_f22_hw4.got_title_cast where nconst='nconst_tmp' and tconst='tconst_tmp'"
        query2_str = query2_str.replace("nconst_tmp", actor_detail[0]['nconst']).replace("tconst_tmp", tconst)
        character = %sql $query2_str
        ## print(character[0]['characters'])
        # Add relationship to Neo4j
        neo_query2 =  '{ "roles": "roles_tmp"}'
        character_tmp = character[0]['characters'].replace('["','').replace('"]','')
        neo_query2 = json.loads(neo_query2.replace("roles_tmp", character_tmp))
        r = create_relationship(actor_detail[0], "GOT_Acted_In", ep[i], neo_query2)
```

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

The screenshot after creating relationships:



Query Result for Relationships

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 [ ]: