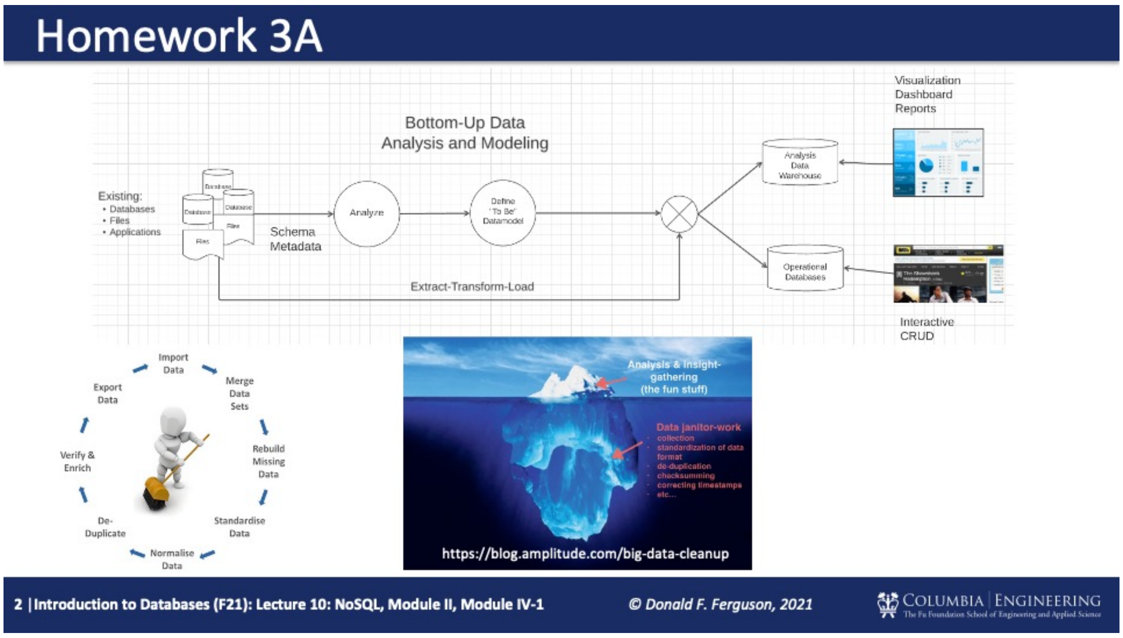
# COMS W4111: Introduction to Databases Section 002, Fall 2021

*Homework 3A*

## Overview

- To smooth the time students spend on homework per week, we split each of HW 3 and HW 4 into two parts: A, B.
- HW 3A is worth 8 points out of the semesters 100 total possible points.
- HW 3A is common to both the programming and non-programming tracks. HW 3A requires importing and transforming data for MySQL, MongoDB and Neo4j databases. Subsequent HW projects will use the processed data.



**HW 3A Concept**

- HW 3A has two sources of raw data input files:
  - CSV data downloaded from [IMDB. (https://www.imdb.com/interfaces/)](https://www.imdb.com/interfaces/)
  - JSON data files from Jeffrey Lancaster's Game-of-Thrones [visualization project. (https://jeffreylancaster.github.io/game-of-thrones/)](https://jeffreylancaster.github.io/game-of-thrones/)
- We have downloaded, simplified and reduced the size and complexity of some of the

data to make the assignment easier and to require less powerful computing resources.

- In HW 3A, you will process the raw data to produce well-design data models and data in MySQL, Neo4j and MongoDB. The final data model:
  - Contains core information in MySQL.
  - Document and hierarchical information in MongoDB.
  - Graph data describing relationships between characters and actors in IMDB.

- The HW 3A submission format is a copy of this notebook with each of the tasks completed. Completing a specific task involves:
  - Creating a "to be" schema.
  - Populating with data by extract-transform-load of the raw data.
  - Providing the queries and code you use to perform the schema creation and transformation.
  - Providing test queries that show the structure of the resulting data and schema.

This homework will be due **Monday, November 22, 2021 at midnight**.

# Environment Setup

## Installation

- You must install and set up.
  - Neo4j Desktop (https://neo4j.com/download-neo4j-now/): This includes configuring and using the sample movie graph to test your configuration: `:play movie graph` . (https://neo4j.com/developer/neo4j-browser/ (https://neo4j.com/developer/neo4j-browser/))
  - MongoDB Community Edition (https://docs.mongodb.com/manual/installation/)
  - MongoDB Compass (https://docs.mongodb.com/compass/current/install/)

- Create two new MySQL schema/databases: `HW3_IMDBRaw and HW3_IMDBFixed.`

## Test Setup

### Neo4j

- Using Neo4j, create a new project `HW3` and create a graph in the project. **Remember the DB password you choose.**

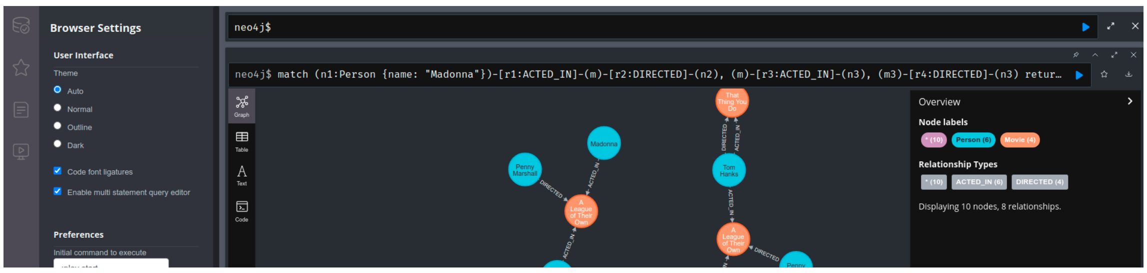
- Start and connect to the graph using the Neo4j browser (launch-able from `Open` on the desktop after you create the graph).

- Enter `:play movie graph` in the Cypher command area in the UI and follow the tutorial instructions.

- After completion, run the query

```
match (n1:Person {name: "Madonna"})-[r1:ACTED_IN]-(m)-[r2:DI
RECTED]-(n2), (m)-[r3:ACTED_IN]-(n3), (m3)-[r4:DIRECTED]-(n
3) return n1,r1,m,r2,n2,r3,n3,r4,m3
```

- Capture the result, save to a file and embed the file below. You answer should be:



```
neo4j$ match (n1:Person {name: "Madonna"})-[r1:ACTED_IN]-(m)-[r2:DIRECTED]-(n2), (m)-[r3:ACTED_IN]-(n3), (m3)-[r4:DIRECTED]-(n3) retur…
```

Browser Settings

User Interface

Theme

- (●) Auto
- ( ) Normal
- ( ) Outline
- ( ) Dark

☑ Code font ligatures

☑ Enable multi statement query editor

Preferences

Initial command to execute

Overview

Node labels: * (10) Person (6) Movie (4)

Relationship Types: * (10) ACTED_IN (6) DIRECTED (4)

Displaying 10 nodes, 8 relationships.

- Install the Neo4j python client library `py2neo` (**Note:** Your output might be different).

In [1]:
```
!pip install py2neo
```

```
Collecting py2neo
  Downloading py2neo-2021.2.3-py2.py3-none-any.whl (177 kB)
     |████████████████████████████████| 177 kB 3.2 MB/s eta 0:00:01
Requirement already satisfied: packaging in /home/adityasidharta/an
aconda3/lib/python3.8/site-packages (from py2neo) (20.9)
Collecting monotonic
  Downloading monotonic-1.6-py2.py3-none-any.whl (8.2 kB)
Requirement already satisfied: pygments>=2.0.0 in /home/adityasidha
rta/anaconda3/lib/python3.8/site-packages (from py2neo) (2.8.1)
Requirement already satisfied: six>=1.15.0 in /home/adityasidharta/
anaconda3/lib/python3.8/site-packages (from py2neo) (1.15.0)
Requirement already satisfied: urllib3 in /home/adityasidharta/anac
onda3/lib/python3.8/site-packages (from py2neo) (1.26.4)
Collecting pansi>=2020.7.3
  Downloading pansi-2020.7.3-py2.py3-none-any.whl (10 kB)
Requirement already satisfied: certifi in /home/adityasidharta/anac
onda3/lib/python3.8/site-packages (from py2neo) (2020.12.5)
Collecting interchange~=2021.0.4
  Downloading interchange-2021.0.4-py2.py3-none-any.whl (28 kB)
Requirement already satisfied: pytz in /home/adityasidharta/anacond
a3/lib/python3.8/site-packages (from interchange~=2021.0.4->py2neo)
(2021.1)
Requirement already satisfied: pyparsing>=2.0.2 in /home/adityasidh
arta/anaconda3/lib/python3.8/site-packages (from packaging->py2neo)
(2.4.7)
Installing collected packages: pansi, monotonic, interchange, py2ne
o
Successfully installed interchange-2021.0.4 monotonic-1.6 pansi-202
0.7.3 py2neo-2021.2.3
```

- Using the credentials you defined when creating the Neo4j project and graph, test your ability to connect to the graph.
- There is an [on-line tutorial (https://medium.com/@technologydata25/connect-neo4j-to-jupyter-notebook-c178f716d6d5)](https://medium.com/@technologydata25/connect-neo4j-to-jupyter-notebook-c178f716d6d5) that may help.

In [14]:
```
from py2neo import Graph,Node,Relationship
```

In [15]:
```
#
# The bolt URL and neo4j should be the same for everyone.
# Replace dbuserdbuser with the passsword you set when creating the g
#
graph = Graph("bolt://localhost:7687", auth=("neo4j", "password"))
```

In [21]:
```
graph
```

Out[21]:
```
Graph('bolt://localhost:7687')
```

In [22]:
```
#
# The following is the query you entered above.
#
q = """match (n1:Person {name: "Madonna"})-[r1:ACTED_IN]-(m)-[r2:DIRE
        (m)-[r3:ACTED_IN]-(n3), (m3)-[r4:DIRECTED]-(n3)
        return n1,r1,m,r2,n2,r3,n3,r4,m3"""
```

In [23]:
```
#
# Run the query.
#
result=graph.run(q)
```

In [24]:
```
for r in result:
    for x in r:
        print(type(x), ":", dict(x))
```

```
<class 'py2neo.data.Node'> : {'name': 'Madonna', 'born': 1954}
<class 'py2neo.data.ACTED_IN'> : {'roles': ['"All the Way" Mae Mord
abito']}
<class 'py2neo.data.Node'> : {'tagline': 'Once in a lifetime you ge
t a chance to do something different.', 'title': 'A League of Their
Own', 'released': 1992}
<class 'py2neo.data.DIRECTED'> : {}
<class 'py2neo.data.Node'> : {'name': 'Penny Marshall', 'born': 194
3}
<class 'py2neo.data.ACTED_IN'> : {'roles': ['Jimmy Dugan']}
<class 'py2neo.data.Node'> : {'name': 'Tom Hanks', 'born': 1956}
<class 'py2neo.data.DIRECTED'> : {}
<class 'py2neo.data.Node'> : {'tagline': 'In every life there comes
a time when that thing you dream becomes that thing you do', 'title
': 'That Thing You Do', 'released': 1996}
```

## MongoDB and Compass

- Run the code snippet below to load the raw information about characters in Game of

Thrones.

In [25]:
```python
import json
```

In [26]:
```python
with open('./characters.json', "r") as in_file:
    c_data = json.load(in_file)
c_data = c_data['characters']
```

In [27]:
```python
c_data[1]
```

Out[27]:
```
{'characterName': 'Aegon Targaryen',
 'houseName': 'Targaryen',
 'royal': True,
 'parents': ['Elia Martell', 'Rhaegar Targaryen'],
 'siblings': ['Rhaenys Targaryen', 'Jon Snow'],
 'killedBy': ['Gregor Clegane']}
```

In [29]:
```
!pip install pymongo
```

```
Collecting pymongo
  Downloading pymongo-3.12.1-cp38-cp38-manylinux_2_17_x86_64.manyli
nux2014_x86_64.whl (527 kB)
     |████████████████████████████████| 527 kB 4.6 MB/s eta 0:00:01
Installing collected packages: pymongo
Successfully installed pymongo-3.12.1
```

In [30]:
```python
#
# Connect to MongoDB
#
from pymongo import MongoClient
client = MongoClient(
                host="localhost",
                port=27017
            )
client
```

Out[30]:
```
MongoClient(host=['localhost:27017'], document_class=dict, tz_aware
=False, connect=True)
```

In [31]:
```python
#
# Load the character information into the HW3 MongoDB and collection
#
for c in c_data:
    client.HW3.GOT_Characters.insert_one(c)
```

In [32]:
```python
#
# Now, test for correct loading.
#
f = {"siblings": "Sansa Stark"}
p = {
    "_id": 0,
    "characterName": 1,
    "characterImageFull": 1,
    "actorName": 1
}
```

In [33]:
```
result = client.HW3.GOT_Characters.find(f, p)
result = list(result)
```

In [34]:
```
for r in result:
    print(json.dumps(r, indent=2))
```

```
{
  "characterName": "Arya Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/im
ages/M/MV5BMTk5MTYwNDc0OF5BMl5BanBnXkFtZTcwOTg2NDg1Nw@@._V1_SY1000_
CR0,0,665,1000_AL_.jpg",
  "actorName": "Maisie Williams"
}
{
  "characterName": "Bran Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/im
ages/M/MV5BMTA1NTg0NTI3MTBeQTJeQWpwZ15BbWU3MDEyNjg4OTQ@._V1_SX1500_
CR0,0,1500,999_AL_.jpg",
  "actorName": "Isaac Hempstead Wright"
}
{
  "characterName": "Rickon Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/im
ages/M/MV5BMWZiOGNjMDAtOTRlNi00MDJmLWEyMTMtOGEwZTM5ODJlNDAyXkEyXkFq
cGdeQXVyMjk3NTUyOTc@._V1_.jpg",
  "actorName": "Art Parkinson"
}
{
  "characterName": "Robb Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/im
ages/M/MV5BMjI2NDE1NzczNF5BMl5BanBnXkFtZTcwNjcwODg4OQ@@._V1_SY1000_
CR0,0,845,1000_AL_.jpg",
  "actorName": "Richard Madden"
}
```

In [35]:
```
#
# And, just for the heck of it ...
#
from IPython import display
display.Image(result[0]["characterImageFull"], width="300px")
```

Out[35]:

In [37]:
```
!pip install nameparser
```

```
Collecting nameparser
  Downloading nameparser-1.0.6-py2.py3-none-any.whl (23 kB)
Installing collected packages: nameparser
Successfully installed nameparser-1.0.6
```

In [38]:
```
from nameparser import HumanName
```

In [39]:
```
from pymongo import MongoClient
import json
import pandas as pd
```

In [40]:
```
from sqlalchemy import create_engine
```

In [56]:
```
engine = create_engine("mysql+pymysql://root:password@127.0.0.1/HW3_(
```

In [54]:
```
client = MongoClient(
                host="localhost",
                port=27017
            )
```

In [55]:
```
client.list_database_names()
```

Out[55]:
```
['HW3', 'admin', 'config', 'local']
```

# Task I: Essential Game of Thrones Character and Actor Information

## Task I-a: Load Raw Information

- Character documents in the collection `GOT_Characters` have several fields.
- The first task is to get the essential fields and then load info a core MySQL table.
- The core fields are:
  - actorLink
  - actorName
  - characterName
  - characterLink
  - characterImageFull
  - characterImageThumb
  - houseName
  - kingsguard
  - nickname
  - royal
- This requires a simple `find` call to MongoDB.

- **Question:** Put your code here.

In [47]:
```
p = {
    "_id": 1,
    "actorLink": 1,
    "actorName": 1,
    "characterName": 1,
    "characterLink": 1,
    "characterImageFull": 1,
    "characterImageThumb": 1,
    "houseName": 1,
    "kingsguard": 1,
    "nickname": 1,
    "royal": 1
}
```

- Execute the following test.

In [48]:
```
result = client.HW3.GOT_Characters.find({}, p)
result = list(result)
```

In [49]:
```
result = list(result)
for r in result:
    r["id"] = str(r["_id"])
    del r["_id"]
result[10]
```

Out[49]:
```
{'characterName': 'Archmaester Marwyn',
 'characterLink': '/character/ch0578265/',
 'actorName': 'Jim Broadbent',
 'actorLink': '/name/nm0000980/',
 'id': '619471811c6ae3633b2297e8'}
```

- **Question:** Create a table in `HW3_IMDBRaw` to hold the `characters` information. Show you create table statement, your code for loading the table and a test query below. You may use the `%sql` extension. You may also use `pandas`.

In [94]:
```
df = pd.DataFrame(result)
```

In [95]:
```
df = df.astype(str)
```

In [96]:
```
import numpy as np
```

In [97]:
```
df = df.replace('nan', np.nan)
```

In [98]:
```
df
```

Out[98]:

| | characterName | characterLink | actorName | actorLink | id |
|---|---|---|---|---|---|
| **0** | Addam Marbrand | /character /ch0305333/ | B.J. Hogg | /name/nm0389698/ | 619471801c6ae3633b2297de |
| **1** | Aegon Targaryen | NaN | NaN | NaN | 619471811c6ae3633b2297df |

| | characterName | characterLink | actorName | actorLink | id |
|---|---|---|---|---|---|
| **2** | Aeron Greyjoy | /character /ch0540081/ | Michael Feast | /name/nm0269923/ | 619471811c6ae3633b2297e0 |
| **3** | Aerys II Targaryen | /character /ch0541362/ | David Rintoul | /name/nm0727778/ | 619471811c6ae3633b2297e1 |
| **4** | Akho | /character /ch0544520/ | Chuku Modu | /name/nm6729880/ | 619471811c6ae3633b2297e2 |
| **...** | ... | ... | ... | ... | ... |
| **384** | Young Nan | /character /ch0305018/ | Annette Tierney | /name/nm1519719/ | 619471811c6ae3633b22995e |
| **385** | Young Ned | /character /ch0154681/ | Robert Aramayo | /name/nm7075019/ | 619471811c6ae3633b22995f |
| **386** | Young Ned Stark | /character /ch0154681/ | Sebastian Croft | /name/nm7509185/ | 619471811c6ae3633b229960 |
| **387** | Young Rodrik Cassel | /character /ch0171391/ | Fergus Leathem | /name/nm7509186/ | 619471811c6ae3633b229961 |
| **388** | Zanrush | /character /ch0540870/ | Gerald Lepkowski | /name/nm0503319/ | 619471811c6ae3633b229962 |

In [73]:
```
df.to_sql('characters', engine)
```

- Test your result with the query below.

In [77]:
```
%reload_ext sql
%sql mysql+pymysql://root:password@127.0.0.1/HW3_GOT_Raw
```

In [78]:
```
%sql select * from HW3_GOT_Raw.characters limit 10;
```

```
 * mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Raw
10 rows affected.
```

Out[78]:

| index | characterName | characterLink | actorName | actorLink | |
|---|---|---|---|---|---|
| 0 | Addam Marbrand | /character /ch0305333/ | B.J. Hogg | /name/nm0389698/ | 619471801c6ae3633b2297c |
| 1 | Aegon Targaryen | None | None | None | 619471811c6ae3633b2297 |
| 2 | Aeron Greyjoy | /character /ch0540081/ | Michael Feast | /name/nm0269923/ | 619471811c6ae3633b2297e |
| 3 | Aerys II Targaryen | /character /ch0541362/ | David Rintoul | /name/nm0727778/ | 619471811c6ae3633b2297e |

| | | | | | |
|---|---|---|---|---|---|
| 4 | Akho | /character /ch0544520/ | Chuku Modu | /name/nm6729880/ | 619471811c6ae3633b2297e |
| 5 | Alliser Thorne | /character /ch0246938/ | Owen Teale | /name/nm0853583/ | 619471811c6ae3633b2297e |
| 6 | Alton Lannister | /character /ch0305012/ | Karl Davies | /name/nm0203801/ | 619471811c6ae3633b2297e |
| 7 | Alys Karstark | /character /ch0576836/ | Megan Parkinson | /name/nm8257864/ | 619471811c6ae3633b2297e |
| 8 | Amory Lorch | /character /ch0305002/ | Fintan McKeown | /name/nm0571654/ | 619471811c6ae3633b2297e |
| 9 | Anguy | /character /ch0316930/ | Philip McGinley | /name/nm1528121/ | 619471811c6ae3633b2297e |

## Task I-b: Improve Schema

- There are several problems with the raw characters and actors information. Some obvious examples are:
  - There are two entity types in one table: `characters and actors.`
  - The columns are not typed.
  - There are no keys or constraints.
  - Repeating prefixes like `/name/` is a poor design.
- Create a schema `HW3_GOT_Fixed` that has an improved schema and data model. Show your create and alter table, and data loading statements below. Also, run a query against your tables to show the data.

In [99]:
```
df = df.fillna('nan')
df['characterLink'] = df['characterLink'].apply(lambda x : x.replace(
df['actorLink'] = df['actorLink'].apply(lambda x : x.replace('/name',
```

In [104]:
```
df['kingsguard'].value_counts()
```

Out[104]:
```
nan     384
True      5
Name: kingsguard, dtype: int64
```

In [175]:
```
chardf = df[['id', 'characterName', 'characterLink', 'houseName', 'ro
actordf = df[['actorName', 'actorLink']].copy()
```

In [176]:
```
actordf = actordf.replace('nan', np.nan).dropna().drop_duplicates().r
actordf['index'] = actordf.index.astype(str)
```

In [177]:
```
actordf
```

Out[177]:

|  | index | actorName | actorLink |
|---|---|---|---|
| **0** | 0 | B.J. Hogg | nm0389698 |
| **1** | 1 | Michael Feast | nm0269923 |
| **2** | 2 | David Rintoul | nm0727778 |
| **3** | 3 | Chuku Modu | nm6729880 |
| **4** | 4 | Owen Teale | nm0853583 |
| **...** | ... | ... | ... |
| **345** | 345 | Annette Tierney | nm1519719 |
| **346** | 346 | Robert Aramayo | nm7075019 |
| **347** | 347 | Sebastian Croft | nm7509185 |
| **348** | 348 | Fergus Leathem | nm7509186 |
| **349** | 349 | Gerald Lepkowski | nm0503319 |

350 rows × 3 columns

```
In [178]: chardf['actorId'] = chardf.merge(actordf, how='left', on='actorName'
```

```
In [179]: chardf = chardf.replace('nan', np.nan)
```

```
In [180]: chardf = chardf.drop(columns=['actorName'])
```

```
In [202]: actordf = actordf.rename(columns = {'index' : 'id'})
```

```
In [183]: chardf
```

Out[183]:

|  | id | characterName | characterLink | houseName | royal | characterI |
|---|---|---|---|---|---|---|
| **0** | 619471801c6ae3633b2297de | Addam Marbrand | ch0305333 | NaN | NaN |  |
| **1** | 619471811c6ae3633b2297df | Aegon Targaryen | NaN | Targaryen | True |  |
| **2** | 619471811c6ae3633b2297e0 | Aeron Greyjoy | ch0540081 | Greyjoy | NaN | https://ir images- |
| **3** | 619471811c6ae3633b2297e1 | Aerys II Targaryen | ch0541362 | Targaryen | True | https://ir images- |
| **4** | 619471811c6ae3633b2297e2 | Akho | ch0544520 | NaN | NaN | https://ir images- |

In [184]:
```
engine = create_engine("mysql+pymysql://root:password@127.0.0.1/HW3_(
```

In [204]:
```
actordf.to_sql('actor', engine, if_exists='replace')
```

In [186]:
```
chardf.to_sql('char', engine, if_exists='replace
```

In [187]:
```
%reload_ext sql
%sql mysql+pymysql://root:password@127.0.0.1/HW3_GOT_Fixed
```

In [213]:
```
actordf
```

Out[213]:

| | id | actorName | actorLink |
|---|---|---|---|
| **0** | 0 | B.J. Hogg | nm0389698 |
| **1** | 1 | Michael Feast | nm0269923 |
| **2** | 2 | David Rintoul | nm0727778 |
| **3** | 3 | Chuku Modu | nm6729880 |
| **4** | 4 | Owen Teale | nm0853583 |
| **...** | ... | ... | ... |
| **345** | 345 | Annette Tierney | nm1519719 |
| **346** | 346 | Robert Aramayo | nm7075019 |
| **347** | 347 | Sebastian Croft | nm7509185 |
| **348** | 348 | Fergus Leathem | nm7509186 |
| **349** | 349 | Gerald Lepkowski | nm0503319 |

350 rows × 3 columns

In [217]:
```
%%sql
ALTER TABLE HW3_GOT_Fixed.actor
MODIFY COLUMN id varchar(256);
ALTER TABLE HW3_GOT_Fixed.actor
MODIFY COLUMN actorName varchar(256);
ALTER TABLE HW3_GOT_Fixed.actor
MODIFY COLUMN actorLink varchar(256);
```

```
 * mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Fixed
   mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Raw
0 rows affected.
350 rows affected.
350 rows affected.
```

Out[217]:
```
[]
```

In [218]:
```
%%sql

ALTER TABLE HW3_GOT_Fixed.actor
ADD PRIMARY KEY (id)
```

```
 * mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Fixed
   mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Raw
0 rows affected.
```

Out[218]:
```
[]
```

In [219]:
```
chardf
```

Out[219]:

| | id | characterName | characterLink | houseName | royal | characterI |
|---|---|---|---|---|---|---|
| **0** | 619471801c6ae3633b2297de | Addam Marbrand | ch0305333 | NaN | NaN | |
| **1** | 619471811c6ae3633b2297df | Aegon Targaryen | NaN | Targaryen | True | |
| **2** | 619471811c6ae3633b2297e0 | Aeron Greyjoy | ch0540081 | Greyjoy | NaN | https://ir images- |
| **3** | 619471811c6ae3633b2297e1 | Aerys II Targaryen | ch0541362 | Targaryen | True | https://ir images- |
| **4** | 619471811c6ae3633b2297e2 | Akho | ch0544520 | NaN | NaN | https://ir images- |
| **...** | ... | ... | ... | ... | ... | |
| **384** | 619471811c6ae3633b22995e | Young Nan | ch0305018 | NaN | NaN | |
| **385** | 619471811c6ae3633b22995f | Young Ned | ch0154681 | Stark | NaN | |
| **386** | 619471811c6ae3633b229960 | Young Ned Stark | ch0154681 | Stark | NaN | |
| **387** | 619471811c6ae3633b229961 | Young Rodrik Cassel | ch0171391 | NaN | NaN | |
| **388** | 619471811c6ae3633b229962 | Zanrush | ch0540870 | NaN | NaN | https://ir images- |

389 rows × 10 columns

In [225]:
```
%%sql
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN id varchar(256);
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN characterName varchar(256);
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN characterLink varchar(256);
```

```
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN houseName varchar(256);
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN royal varchar(256);
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN characterImageThumb varchar(256);
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN characterImageFull varchar(256);
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN nickname varchar(256);
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN kingsguard varchar(256);
ALTER TABLE HW3_GOT_Fixed.char
MODIFY COLUMN actorId INT;
```

```
 * mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Fixed
   mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Raw
0 rows affected.
0 rows affected.
0 rows affected.
0 rows affected.
389 rows affected.
389 rows affected.
389 rows affected.
389 rows affected.
389 rows affected.
389 rows affected.
```

Out[225]: []

In [227]:
```
%%sql

ALTER TABLE HW3_GOT_Fixed.char
ADD PRIMARY KEY (id);
```

```
 * mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Fixed
   mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Raw
(pymysql.err.OperationalError) (1068, 'Multiple primary key defined
')
[SQL: ALTER TABLE HW3_GOT_Fixed.char ADD PRIMARY KEY (id);]
(Background on this error at: http://sqlalche.me/e/14/e3q8) (htt
p://sqlalche.me/e/14/e3q8))
```

In [229]:
```
%sql select * from HW3_GOT_Fixed.actor limit 10;
```

```
 * mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Fixed
   mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Raw
10 rows affected.
```

Out[229]:

| index | id | actorName | actorLink |
|---|---|---|---|
| 0 | 0 | B.J. Hogg | nm0389698 |
| 1 | 1 | Michael Feast | nm0269923 |
| 10 | 10 | Deobia Oparei | nm0649046 |
| 100 | 100 | Dominic Carter | nm0141582 |

| | | | |
|---|---|---|---|
| 101 | 101 | Tom Wlaschiha | nm0937239 |
| 102 | 102 | Patrick O'Kane | nm0641433 |
| 103 | 103 | Jeffrey O'Brien | nm4475335 |
| 104 | 104 | James Cosmo | nm0181920 |
| 105 | 105 | Sarita Piotrowski | nm4424689 |

In [266]:
```
%sql select * from HW3_GOT_Fixed.char limit 10;
```

```
 * mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Fixed
   mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Raw
10 rows affected.
```

Out[266]:

| index | id | characterName | characterLink | houseName | royal | |
|---|---|---|---|---|---|---|
| 0 | 619471801c6ae3633b2297de | Addam Marbrand | ch0305333 | None | None | |
| 1 | 619471811c6ae3633b2297df | Aegon Targaryen | None | Targaryen | True | |
| 2 | 619471811c6ae3633b2297e0 | Aeron Greyjoy | ch0540081 | Greyjoy | None | |
| 3 | 619471811c6ae3633b2297e1 | Aerys II Targaryen | ch0541362 | Targaryen | True | /M/MV5E |
| 4 | 619471811c6ae3633b2297e2 | Akho | ch0544520 | None | None | |
| 5 | 619471811c6ae3633b2297e3 | Alliser Thorne | ch0246938 | None | None | |
| 6 | 619471811c6ae3633b2297e4 | Alton Lannister | ch0305012 | Lannister | None | |
| 7 | 619471811c6ae3633b2297e5 | Alys Karstark | ch0576836 | None | None | |
| 8 | 619471811c6ae3633b2297e6 | Amory Lorch | ch0305002 | None | None | |
| 9 | 619471811c6ae3633b2297e7 | Anguy | ch0316930 | None | None | |

## Task II: Relationships

### Task II-a: Getting Relationship Data

- The MongoDB collection for `characters` has fields representing one-to-many relationships between characters.
- The fields are in the list below.

In [231]:
```
relationship_names = [
 'abducted',
 'abductedBy',
 #'actors',
 'allies',
 'guardedBy',
 'guardianOf',
 'killed',
 'killedBy',
 'marriedEngaged',
 'parentOf',
 'parents',
 'servedBy',
 'serves',
 'sibling',
 'siblings'
]
```

- The Task II-a objective is to produce a table `HW3_GOT_Raw.character_relationships` of the form:

`character_relationships(sourceCharacterName, relationship, targetCharacterName)`

- Producing this information requires some pretty tricky MongoDB aggregate pipeline development. The critical hint is to realize that:
  - You can write a function that implements a generic pipeline to produce the information given a specific relationship name.
  - Write a python function that saves the information produced by the function in the SQL table.
  - Write a python loop that calls the function to produce the information for each of the relationships in the list above and saves/appends the information to the relationship table.

In [240]:
```
p = {
    "_id": 1,
    "characterName": 1,
}

for r in relationship_names:
    p[r] = 1
```

In [241]:
```
p
```

Out[241]:

```
{'_id': 1,
 'characterName': 1,
```

In [242]:
```
result = client.HW3.GOT_Characters.find({}, p)
result = list(result)
```

In [243]:
```
df = pd.DataFrame(result)
```

In [251]:
```
final_result = []
for idx, row in df.iterrows():
    for relationship in relationship_names:
        targets = row[relationship]
        if not np.any(pd.isnull(targets)):
            for target in targets:
                final_result.append({
                    'sourceCharacterName' : row['characterName'],
                    'relationship': relationship,
                    'targetCharacterName': target
                })
```

In [253]:
```
final_result_df = pd.DataFrame(final_result)
```

In [254]:
```
final_result_df
```

Out[254]:

| | sourceCharacterName | relationship | targetCharacterName |
|---|---|---|---|
| **0** | Aegon Targaryen | killedBy | Gregor Clegane |
| **1** | Aegon Targaryen | parents | Elia Martell |
| **2** | Aegon Targaryen | parents | Rhaegar Targaryen |
| **3** | Aegon Targaryen | siblings | Rhaenys Targaryen |
| **4** | Aegon Targaryen | siblings | Jon Snow |
| **...** | ... | ... | ... |
| **842** | Ygritte | killed | Pypar |
| **843** | Ygritte | killedBy | Olly |
| **844** | Ygritte | marriedEngaged | Jon Snow |
| **845** | Yohn Royce | parentOf | Waymar Royce |
| **846** | Yoren | killedBy | Amory Lorch |

847 rows × 3 columns

In [255]:
```
final_result_df.to_sql('character_relationships', engine, if_exists=
```

In [257]:
```
%%sql
```

```
ALTER TABLE HW3_GOT_Fixed.character_relationships
MODIFY COLUMN sourceCharacterName varchar(256);
ALTER TABLE HW3_GOT_Fixed.character_relationships
MODIFY COLUMN relationship varchar(256);
ALTER TABLE HW3_GOT_Fixed.character_relationships
MODIFY COLUMN targetCharacterName varchar(256);
```

```
 * mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Fixed
   mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Raw
847 rows affected.
847 rows affected.
847 rows affected.
```

Out[257]: `[]`

In [267]:

```
%sql select * from HW3_GOT_Fixed.character_relationships limit 10;
```

```
 * mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Fixed
   mysql+pymysql://root:***@127.0.0.1/HW3_GOT_Raw
10 rows affected.
```

Out[267]:

| index | sourceCharacterName | relationship | targetCharacterName |
|---|---|---|---|
| 0 | Aegon Targaryen | killedBy | Gregor Clegane |
| 1 | Aegon Targaryen | parents | Elia Martell |
| 2 | Aegon Targaryen | parents | Rhaegar Targaryen |
| 3 | Aegon Targaryen | siblings | Rhaenys Targaryen |
| 4 | Aegon Targaryen | siblings | Jon Snow |
| 5 | Aeron Greyjoy | siblings | Balon Greyjoy |
| 6 | Aeron Greyjoy | siblings | Euron Greyjoy |
| 7 | Aerys II Targaryen | killed | Brandon Stark |
| 8 | Aerys II Targaryen | killed | Rickard Stark |
| 9 | Aerys II Targaryen | killedBy | Jaime Lannister |

## Task II-b: Load Neo4j

- At this point, you should have the following tables in `HW3_GOT_Fixed`:
  - `characters`
  - `character_relationships`
- You will now load this information into Neo4j. The following code shows you some simple steps to create nodes and relationships.

In [271]:

```
characters = pd.read_sql("SELECT * FROM HW3_GOT_Fixed.char", engine)
character_relationships = pd.read_sql("SELECT * FROM HW3_GOT_Fixed.ch
```

In [297]:

```
for idx, row in characters.iterrows():
    n = Node("Character", **row)
    graph.create(n)
```

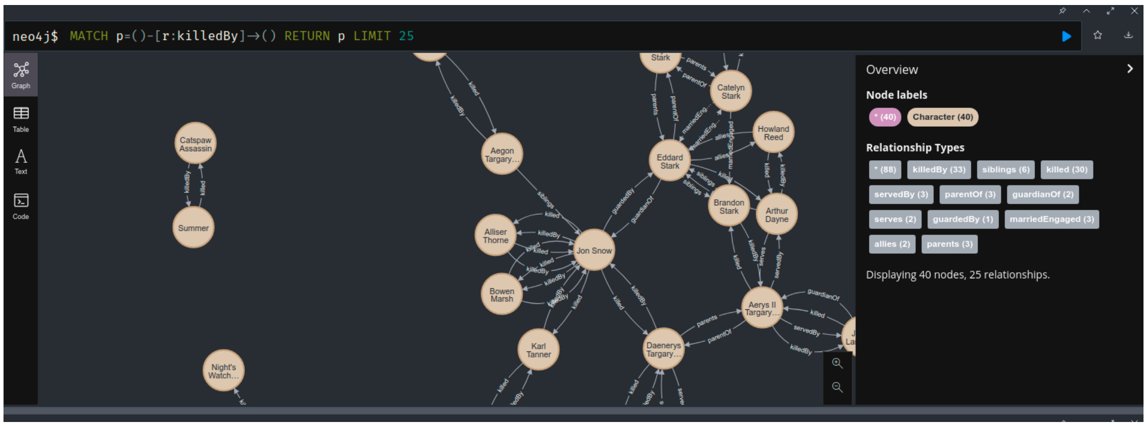
In [298]:
```
character_relationships
```

Out[298]:

| | index | sourceCharacterName | relationship | targetCharacterName |
|---|---|---|---|---|
| **0** | 0 | Aegon Targaryen | killedBy | Gregor Clegane |
| **1** | 1 | Aegon Targaryen | parents | Elia Martell |
| **2** | 2 | Aegon Targaryen | parents | Rhaegar Targaryen |
| **3** | 3 | Aegon Targaryen | siblings | Rhaenys Targaryen |
| **4** | 4 | Aegon Targaryen | siblings | Jon Snow |
| **...** | ... | ... | ... | ... |
| **842** | 842 | Ygritte | killed | Pypar |
| **843** | 843 | Ygritte | killedBy | Olly |
| **844** | 844 | Ygritte | marriedEngaged | Jon Snow |
| **845** | 845 | Yohn Royce | parentOf | Waymar Royce |
| **846** | 846 | Yoren | killedBy | Amory Lorch |

847 rows × 4 columns

In [299]:
```
for idx, row in character_relationships.iterrows():
    q = """
            match (n1:Character {{characterName: "{a}"}}), (n2:Charac
            create (n1)-[:{c}]->(n2)
    """.format(a=row['sourceCharacterName'], b=row['targetCharacterNa
    graph.run(q)
```



```
neo4j$ MATCH p=()-[r:killedBy]→() RETURN p LIMIT 25
```

Overview

Node labels

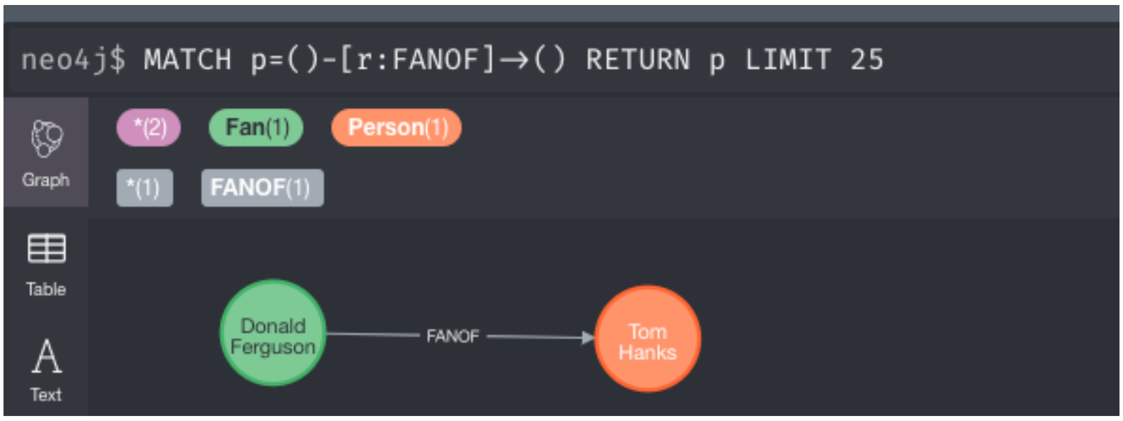
* (40) Character (40)

Relationship Types

* (88) killedBy (33) siblings (6) killed (30) servedBy (3) parentOf (3) guardianOf (2) serves (2) guardedBy (1) marriedEngaged (3) allies (2) parents (3)

Displaying 40 nodes, 25 relationships.

- Now we can do a verification test ... ...



```
neo4j$ MATCH p=()-[r:FANOF]→() RETURN p LIMIT 25
```

- So, your task is the following:
  - Create a `Node` for each character.
  - Create a relationship connecting characters based on their relationships.

- Show you code to create and some verification tests below.