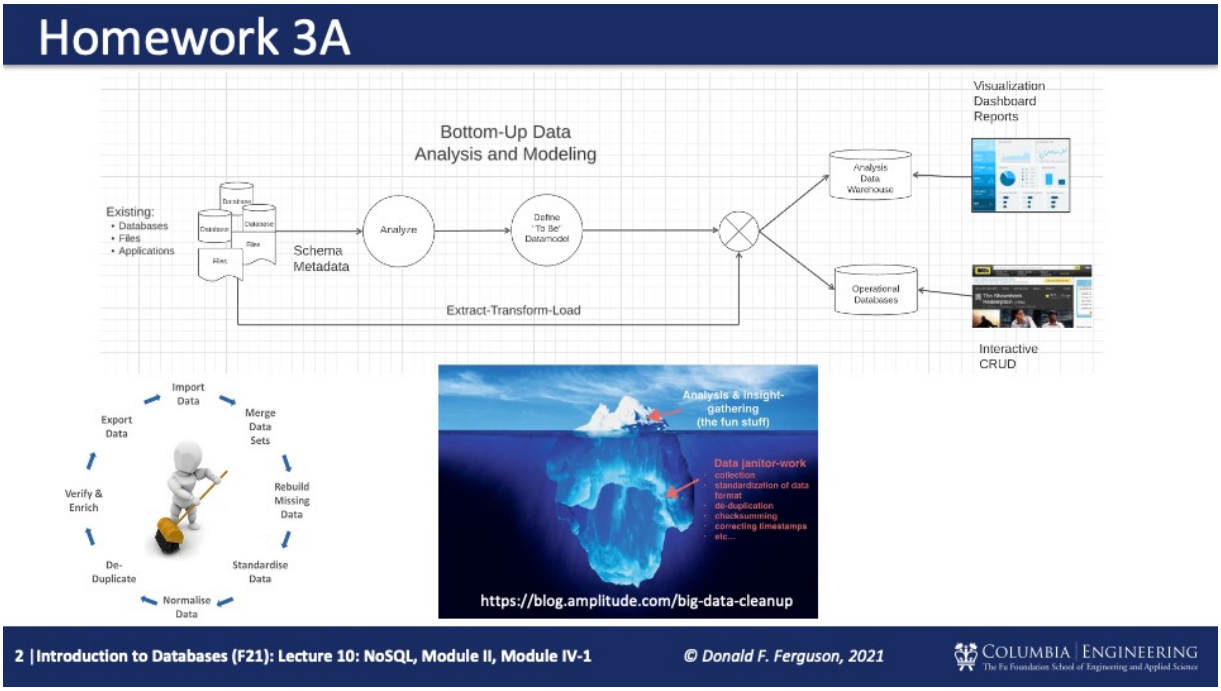
**COMS W4111: Introduction to Databases**
**Section 002, Fall 2021**

*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/)
  - JSON data files from Jeffrey Lancaster's Game-of-Thrones visualization project. (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`.

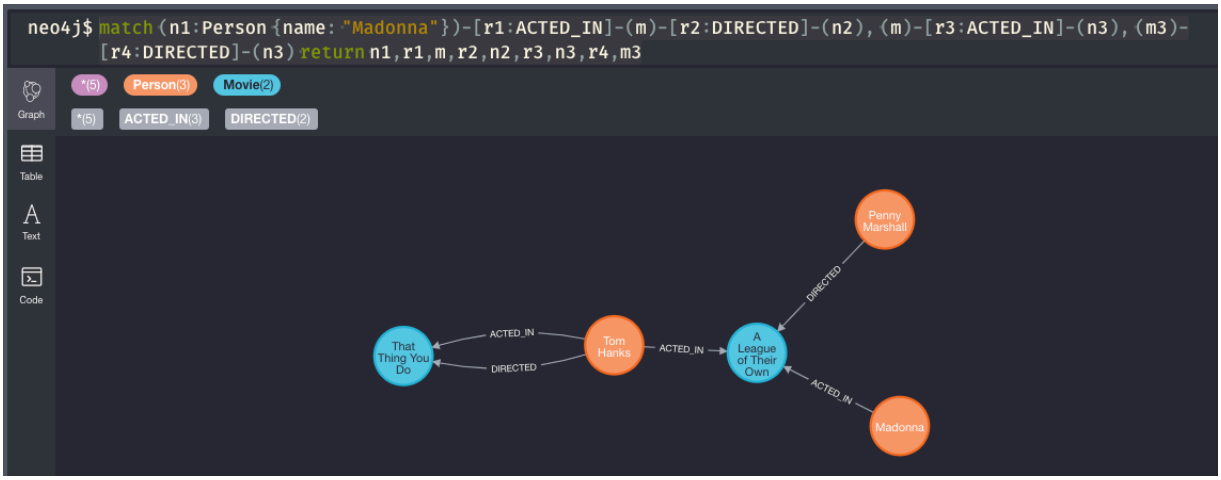
## Test Setup

# 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

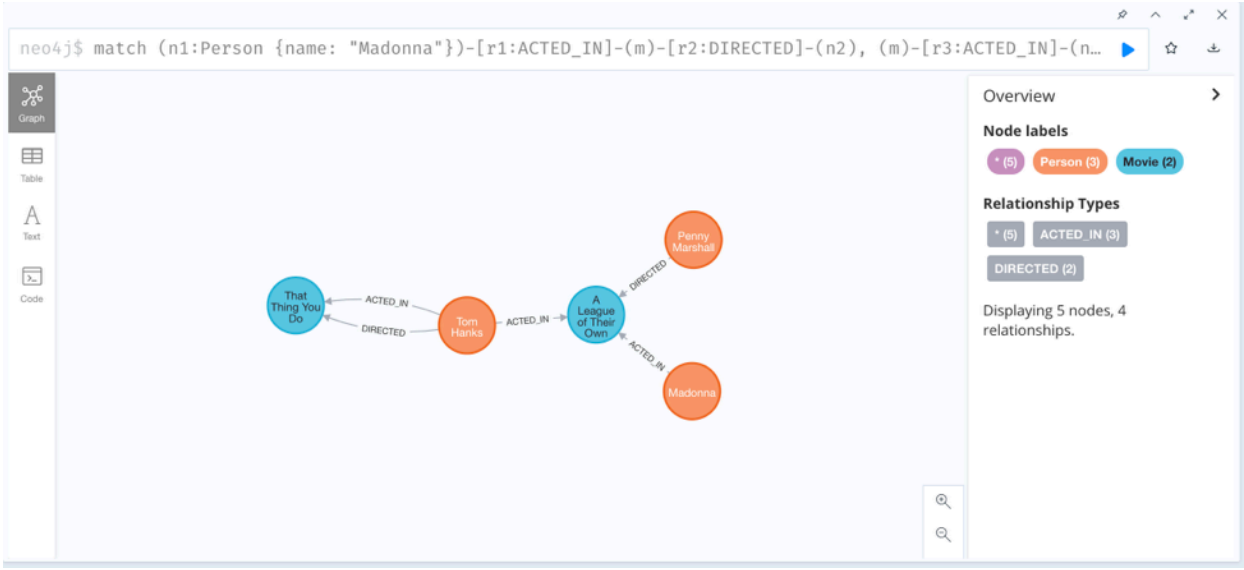
```
match (n1:Person {name: "Madonna"})-[r1:ACTED_IN]-(m)-[r2:DIRECTED]-(n2), (m)-[r3:ACTED_IN]-(n3), (m3)-[r4:DIRECTED]-(n3) return n1,r1,m,r2,n2,r3,n3,r4,m3
```

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




**Neo4j Setup Test**

# Result




- 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 3.4 MB/s eta 0:00:01
Requirement already satisfied: urllib3 in /Users/herman/opt/anaconda3
/envs/herman/lib/python3.8/site-packages (from py2neo) (1.26.6)
Requirement already satisfied: certifi in /Users/herman/opt/anaconda3
/envs/herman/lib/python3.8/site-packages (from py2neo) (2021.5.30)
Requirement already satisfied: six>=1.15.0 in /Users/herman/opt/anaco
nda3/envs/herman/lib/python3.8/site-packages (from py2neo) (1.16.0)
Requirement already satisfied: pygments>=2.0.0 in /Users/herman/opt/a
naconda3/envs/herman/lib/python3.8/site-packages (from py2neo) (2.10.
0)
Collecting interchange~=2021.0.4
  Downloading interchange-2021.0.4-py2.py3-none-any.whl (28 kB)
Requirement already satisfied: packaging in /Users/herman/opt/anacond
a3/envs/herman/lib/python3.8/site-packages (from py2neo) (21.0)
Collecting monotonic
  Downloading monotonic-1.6-py2.py3-none-any.whl (8.2 kB)
Collecting pansi>=2020.7.3
  Downloading pansi-2020.7.3-py2.py3-none-any.whl (10 kB)
Requirement already satisfied: pytz in /Users/herman/opt/anaconda3/en
vs/herman/lib/python3.8/site-packages (from interchange~=2021.0.4->py
2neo) (2021.1)
Requirement already satisfied: pyparsing>=2.0.2 in /Users/herman/opt/
anaconda3/envs/herman/lib/python3.8/site-packages (from packaging->py
2neo) (2.4.7)
Installing collected packages: pansi, monotonic, interchange, py2neo
Successfully installed interchange-2021.0.4 monotonic-1.6 pansi-2020.
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) that may help.

In [59]:
```
from py2neo import Graph,Node,Relationship
```

In [61]:
```
#
# The bolt URL and neo4j should be the same for everyone.
# Replace dbuserdbuser with the passsword you set when creating the gr
#
graph = Graph("bolt://localhost:7687", auth=("neo4j", "7Senses_kiki"))
```

In [4]:
```
#
# The following is the query you entered above.
#
q = """match (n1:Person {name: "Madonna"})-[r1:ACTED_IN]-(m)-[r2:DIREC
        (m)-[r3:ACTED_IN]-(n3), (m3)-[r4:DIRECTED]-(n3)
        return n1,r1,m,r2,n2,r3,n3,r4,m3"""
```

In [5]:
```
#
# Run the query.
#
result=graph.run(q)
```

In [6]:
```
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 Mordab
ito']}
<class 'py2neo.data.Node'> : {'tagline': 'Once in a lifetime you get
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': 1943}
<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 [7]:
```
import json
```

In [8]:
```
with open('./characters.json', "r") as in_file:
    c_data = json.load(in_file)
c_data = c_data['characters']
```

In [9]:
```
c_data[1]
```

Out[9]:
```
{'characterName': 'Aegon Targaryen',
 'houseName': 'Targaryen',
 'royal': True,
 'parents': ['Elia Martell', 'Rhaegar Targaryen'],
 'siblings': ['Rhaenys Targaryen', 'Jon Snow'],
 'killedBy': ['Gregor Clegane']}
```

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

```
Collecting pymongo
  Downloading pymongo-3.12.1-cp38-cp38-macosx_10_9_x86_64.whl (395 kB
)
     |████████████████████████████████| 395 kB 2.8 MB/s eta 0:00:01
Installing collected packages: pymongo
Successfully installed pymongo-3.12.1
```

In [13]:
```
#
# Connect to MongoDB
#
from pymongo import MongoClient
client = MongoClient(
                host="localhost",
                port=27017
            )
client
```

Out[13]:
```
MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=F
alse, connect=True)
```

In [14]:
```
#
# Load the character information into the HW3 MongoDB and collection
#
for c in c_data:
    client.HW3.GOT_Characters.insert_one(c)
```

In [15]:
```
#
# Now, test for correct loading.
#
f = {"siblings": "Sansa Stark"}
p = {
    "_id": 0,
    "characterName": 1,
    "characterImageFull": 1,
    "actorName": 1
}
```

In [16]:
```
result = client.HW3.GOT_Characters.find(f, p)
result = list(result)
```

In [17]:
```
for r in result:
    print(json.dumps(r, indent=2))
```


```
{
  "characterName": "Arya Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/imag
es/M/MV5BMTk5MTYwNDc0OF5BMl5BanBnXkFtZTcwOTg2NDg1Nw@@._V1_SY1000_CR0,
0,665,1000_AL_.jpg",
  "actorName": "Maisie Williams"
}
{
  "characterName": "Bran Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/imag
es/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/imag
es/M/MV5BMWZiOGNjMDAtOTRlNi00MDJmLWEyMTMtOGEwZTM5ODJlNDAyXkEyXkFqcGde
QXVyMjk3NTUyOTc@._V1_.jpg",
  "actorName": "Art Parkinson"
}
{
  "characterName": "Robb Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/imag
es/M/MV5BMjI2NDE1NzczNF5BMl5BanBnXkFtZTcwNjcwODg4OQ@@._V1_SY1000_CR0,
0,845,1000_AL_.jpg",
  "actorName": "Richard Madden"
}
```

In [18]:
```
#
# And, just for the heck of it ...
#
from IPython import display
display.Image(result[0]["characterImageFull"], width="300px")
```

In [20]:
```
!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 [21]:
```
from nameparser import HumanName
```

In [1]:
```
from pymongo import MongoClient
import json
import pandas as pd
```

In [2]:
```
from sqlalchemy import create_engine
```

In [3]:
```
engine = create_engine("mysql+pymysql://admin:7Senses_kiki@tutorialdb.
```

In [4]:
```
client = MongoClient(
                host="localhost",
                port=27017
            )
```

In [5]:
```
client.list_database_names()
```

Out[5]:
```
['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 [63]:
```
# Based on Ed # 902, ignore housename column
f = {}
p = {
    "actorLink": 1,
    "actorName": 1,
    "characterName": 1,
    "characterLink": 1,
    "characterImageFull": 1,
    "characterImageThumb": 1,
    "kingsguard": 1,
    "nickname": 1,
    "royal": 1
}
```

In [64]:
```
result = client.HW3.GOT_Characters.find(f, p)
```

- Execute the following test.

In [65]:
```
result = list(result)
for r in result:
    r["id"] = str(r["_id"])
    del r["_id"]
result[10]
```

Out[65]:
```
{'characterName': 'Archmaester Marwyn',
 'characterLink': '/character/ch0578265/',
 'actorName': 'Jim Broadbent',
 'actorLink': '/name/nm0000980/',
 'id': '619aea27198c09d7fbb5ad21'}
```

- **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 [66]:
```
df = pd.DataFrame(result)
df.to_sql('characters', con=engine, if_exists='replace', index=False)
```

- Test your result with the query below.

In [6]:
```
%load_ext sql
```

In [7]:
```
%sql mysql+pymysql://admin:7Senses_kiki@tutorialdb.cbezzskgwcl3.us-eas
```

In [3]:
```
%sql select * from HW3_IMDBRaw.characters limit 10;
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
10 rows affected.
```

Out[3]:

| characterName | characterLink | actorName | actorLink | id |
|---|---|---|---|---|
| Addam Marbrand | /character/ch0305333/ | B.J. Hogg | /name/nm0389698/ | 619aea27198c09d7fbb5ad1 |
| Aegon Targaryen | None | None | None | 619aea27198c09d7fbb5ad1 |
| Aeron Greyjoy | /character/ch0540081/ | Michael Feast | /name/nm0269923/ | 619aea27198c09d7fbb5ad1 |
| Aerys II Targaryen | /character/ch0541362/ | David Rintoul | /name/nm0727778/ | 619aea27198c09d7fbb5ad1 |
| Akho | /character/ch0544520/ | Chuku Modu | /name/nm6729880/ | 619aea27198c09d7fbb5ad1 |
| Alliser Thorne | /character/ch0246938/ | Owen Teale | /name/nm0853583/ | 619aea27198c09d7fbb5ad1 |
| Alton Lannister | /character/ch0305012/ | Karl Davies | /name/nm0203801/ | 619aea27198c09d7fbb5ad1 |
| Alys Karstark | /character/ch0576836/ | Megan Parkinson | /name/nm8257864/ | 619aea27198c09d7fbb5ad1 |
| Amory Lorch | /character/ch0305002/ | Fintan McKeown | /name/nm0571654/ | 619aea27198c09d7fbb5ad1 |
| Anguy | /character/ch0316930/ | Philip McGinley | /name/nm1528121/ | 619aea27198c09d7fbb5ad2 |

# 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 [69]:
```
%sql mysql+pymysql://admin:7Senses_kiki@tutorialdb.cbezzskgwcl3.us-eas
```

In [70]:
```
%%sql

DROP table if exists characters;
DROP table if exists actors;
DROP table if exists chars_acts;
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.am
azonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.am
azonaws.com/HW3_IMDBRaw
0 rows affected.
0 rows affected.
0 rows affected.
```

Out[70]: []

In [71]:
```
%%sql

CREATE table HW3_IMDBFixed.characters as
SELECT characterName, characterLink, id, royal, characterImageThumb, c
FROM HW3_IMDBRaw.characters
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.am
azonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.am
azonaws.com/HW3_IMDBRaw
389 rows affected.
```

Out[71]: []

In [72]:
```
%%sql

ALTER table characters MODIFY characterName varchar(256) null;
ALTER table characters MODIFY characterLink varchar(256) null;
ALTER table characters MODIFY id varchar(256) null;
ALTER table characters MODIFY royal int null;
ALTER table characters MODIFY characterImageThumb varchar(256) null;
ALTER table characters MODIFY characterImageFull varchar(256) null;
ALTER table characters MODIFY nickname varchar(256) null;
ALTER table characters MODIFY kingsguard varchar(256) null;
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
389 rows affected.
389 rows affected.
389 rows affected.
389 rows affected.
389 rows affected.
389 rows affected.
389 rows affected.
389 rows affected.
```

Out[72]: []

In [73]:
```
%%sql

UPDATE characters
SET characterLink = substr(characterLink, 12);
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
389 rows affected.
```

Out[73]: []

In [74]:
```
%%sql

ALTER table characters
ADD constraint characters_pk
primary key (id);
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
389 rows affected.
```

Out[74]: []

In [75]:
```
%sql SELECT * from characters limit 10
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
10 rows affected.
```

Out[75]:

| characterName | characterLink | id | royal | |
|---|---|---|---|---|
| Addam Marbrand | ch0305333/ | 619aea27198c09d7fbb5ad17 | None | |
| Aegon Targaryen | None | 619aea27198c09d7fbb5ad18 | 1 | |
| Aeron Greyjoy | ch0540081/ | 619aea27198c09d7fbb5ad19 | None | amazon.c |
| Aerys II Targaryen | ch0541362/ | 619aea27198c09d7fbb5ad1a | 1 | amazon.com/images/M/MV5BMV |
| Akho | ch0544520/ | 619aea27198c09d7fbb5ad1b | None | amazon.com/i |
| Alliser Thorne | ch0246938/ | 619aea27198c09d7fbb5ad1c | None | |
| Alton Lannister | ch0305012/ | 619aea27198c09d7fbb5ad1d | None | h |
| Alys Karstark | ch0576836/ | 619aea27198c09d7fbb5ad1e | None | |
| Amory Lorch | ch0305002/ | 619aea27198c09d7fbb5ad1f | None | amazon.com/ |
| Anguy | ch0316930/ | 619aea27198c09d7fbb5ad20 | None | amazon.com/ |

In [ ]:

In [76]:
```
%%sql

CREATE table actors
SELECT actorName, actorLink, id
FROM HW3_IMDBRaw.characters
WHERE actorName is not NULL
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
354 rows affected.
```

Out[76]: []

In [78]:
```
%%sql

ALTER table actors MODIFY actorName varchar(256) null;
ALTER table actors MODIFY actorLink varchar(256) null;
ALTER table actors MODIFY id varchar(256) null;
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
354 rows affected.
354 rows affected.
354 rows affected.
```

Out[78]: []

In [79]:
```
%%sql

UPDATE actors
SET actorLink = substr(actorLink, 7);
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
354 rows affected.
```

Out[79]: []

In [80]:
```
%%sql

ALTER table actors
ADD constraint actors_pk
primary key (id);
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
354 rows affected.
```

Out[80]: []

In [82]:
```
%sql SELECT * from actors limit 10
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
10 rows affected.
```

Out[82]:

| actorName | actorLink | id |
|---|---|---|
| B.J. Hogg | nm0389698/ | 619aea27198c09d7fbb5ad17 |
| Michael Feast | nm0269923/ | 619aea27198c09d7fbb5ad19 |
| David Rintoul | nm0727778/ | 619aea27198c09d7fbb5ad1a |
| Chuku Modu | nm6729880/ | 619aea27198c09d7fbb5ad1b |
| Owen Teale | nm0853583/ | 619aea27198c09d7fbb5ad1c |
| Karl Davies | nm0203801/ | 619aea27198c09d7fbb5ad1d |
| Megan Parkinson | nm8257864/ | 619aea27198c09d7fbb5ad1e |
| Fintan McKeown | nm0571654/ | 619aea27198c09d7fbb5ad1f |
| Philip McGinley | nm1528121/ | 619aea27198c09d7fbb5ad20 |
| Jim Broadbent | nm0000980/ | 619aea27198c09d7fbb5ad21 |

In [90]:
```
%%sql

CREATE table chars_acts as
SELECT characters.id from
characters JOIN actors ON characters.id = actors.id;
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
354 rows affected.
```

Out[90]: []

In [91]:
```
%%sql

ALTER table chars_acts
ADD constraint chars_acts_pk
primary key (id);
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
0 rows affected.
```

Out[91]: []

In [92]:
```
%%sql

ALTER table chars_acts
ADD constraint chars_acts_fk_1
foreign key (id) references characters (id);

ALTER table chars_acts
ADD constraint chars_acts_fk_2
foreign key (id) references actors (id);
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
354 rows affected.
354 rows affected.
```

Out[92]: []

In [93]:
```
%sql SELECT * from chars_acts limit 10
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
10 rows affected.
```

Out[93]:

| id |
|---|
| 619aea27198c09d7fbb5ad17 |
| 619aea27198c09d7fbb5ad19 |
| 619aea27198c09d7fbb5ad1a |
| 619aea27198c09d7fbb5ad1b |
| 619aea27198c09d7fbb5ad1c |
| 619aea27198c09d7fbb5ad1d |
| 619aea27198c09d7fbb5ad1e |
| 619aea27198c09d7fbb5ad1f |
| 619aea27198c09d7fbb5ad20 |
| 619aea27198c09d7fbb5ad21 |

In [95]:
```
%%sql

SELECT *
from characters JOIN actors ON characters.id = actors.id
limit 10
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
10 rows affected.
```

Out[95]:

| characterName | characterLink | id | royal | |
|---|---|---|---|---|
| Addam Marbrand | ch0305333/ | 619aea27198c09d7fbb5ad17 | None | |
| Aeron Greyjoy | ch0540081/ | 619aea27198c09d7fbb5ad19 | None | amazon.c |
| Aerys II Targaryen | ch0541362/ | 619aea27198c09d7fbb5ad1a | 1 | amazon.com/images/M/MV5BMV |
| Akho | ch0544520/ | 619aea27198c09d7fbb5ad1b | None | amazon.com/i |
| Alliser Thorne | ch0246938/ | 619aea27198c09d7fbb5ad1c | None | |
| Alton Lannister | ch0305012/ | 619aea27198c09d7fbb5ad1d | None | h |
| Alys Karstark | ch0576836/ | 619aea27198c09d7fbb5ad1e | None | |
| Amory Lorch | ch0305002/ | 619aea27198c09d7fbb5ad1f | None | amazon.com |
| Anguy | ch0316930/ | 619aea27198c09d7fbb5ad20 | None | amazon.com |
| Archmaester Marwyn | ch0578265/ | 619aea27198c09d7fbb5ad21 | 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 [8]:
```
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 [36]:
```
def lookup(relationship, client):
    f = {relationship:{"$exists":True}}
    p = {
        "_id": 0,
        "characterName": 1,
        relationship: 1
    }
    result = client.HW3.GOT_Characters.find(f, p)
    result = list(result)
    result = [(r['characterName'], relationship, r[relationship][0]) f
    return result
```

In [38]:
```
def insert_to_sql(result):
    df = pd.DataFrame(result, columns = ['sourceCharacterName', 'relat
    df.to_sql('character_relationships', con=engine, if_exists='replac
```

In [40]:
```
res = []
for rel in relationship_names:
    res.extend(lookup(rel, client))

insert_to_sql(res)
```

In [41]:
```
%sql select * from character_relationships limit 10
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.am
azonaws.com/HW3_IMDBRaw
10 rows affected.
```

Out[41]:

| sourceCharacterName | relationship | targetCharacterName |
|---|---|---|
| Rhaegar Targaryen | abducted | Lyanna Stark |
| Lyanna Stark | abductedBy | Rhaegar Targaryen |
| Eddard Stark | allies | Howland Reed |
| Howland Reed | allies | Eddard Stark |
| Jon Arryn | allies | Robert Baratheon |
| Robert Baratheon | allies | Jon Arryn |
| Tywin Lannister | allies | Robert Baratheon |
| Arya Stark | guardedBy | Nymeria |
| Bran Stark | guardedBy | Summer |
| Daenerys Targaryen | guardedBy | Drogon |

In [42]:
```
%sql mysql+pymysql://admin:7Senses_kiki@tutorialdb.cbezzskgwcl3.us-eas
```

In [48]:
```
%%sql

DROP table if exists character_relationships;
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
0 rows affected.
```

Out[48]: []

In [50]:
```
%%sql

CREATE table character_relationships as
SELECT *
FROM HW3_IMDBRaw.character_relationships
WHERE HW3_IMDBRaw.character_relationships.sourceCharacterName in
(SELECT characterName FROM characters) and
HW3_IMDBRaw.character_relationships.targetCharacterName in
(SELECT characterName FROM characters)
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
472 rows affected.
```

Out[50]: []

In [51]:
```
%%sql

ALTER table character_relationships MODIFY sourceCharacterName varchar
ALTER table character_relationships MODIFY relationship varchar(256) r
ALTER table character_relationships MODIFY targetCharacterName varchar
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
472 rows affected.
472 rows affected.
472 rows affected.
```

Out[51]: []

In [52]:
```
%%sql

ALTER table character_relationships
ADD constraint character_relationships_pk
primary key (sourceCharacterName, relationship, targetCharacterName);
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
472 rows affected.
```

Out[52]: []

In [58]:
```
%%sql

SELECT * from character_relationships limit 10;
```

```
 * mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBFixed
   mysql+pymysql://admin:***@tutorialdb.cbezzskgwcl3.us-east-2.rds.amazonaws.com/HW3_IMDBRaw
10 rows affected.
```

Out[58]:

| sourceCharacterName | relationship | targetCharacterName |
|---|---|---|
| Aegon Targaryen | killedBy | Gregor Clegane |
| Aegon Targaryen | parents | Elia Martell |
| Aegon Targaryen | siblings | Rhaenys Targaryen |
| Aeron Greyjoy | siblings | Balon Greyjoy |
| Aerys II Targaryen | killed | Brandon Stark |
| Aerys II Targaryen | killedBy | Jaime Lannister |
| Aerys II Targaryen | marriedEngaged | Rhaella Targaryen |
| Aerys II Targaryen | parentOf | Daenerys Targaryen |
| Aerys II Targaryen | servedBy | Arthur Dayne |
| Aerys II Targaryen | siblings | Rhaella Targaryen |

# Task II-b: Load Neo4j

- At this point, you should have the following tables in `HW3_GOT_Fixed`:
  - `characters`
  - `character_relationships`

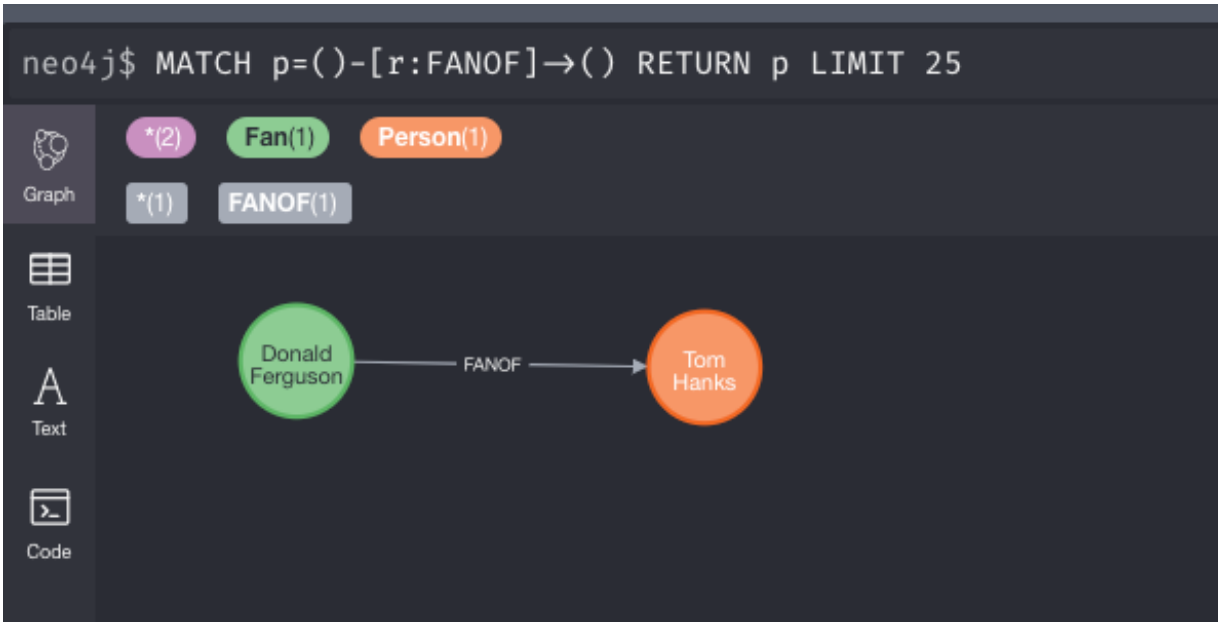
- You will now load this information into Neo4j. The following code shows you some simple steps to create nodes and relationships.

In [225]:
```
n = Node("Fan", uni='dff9', name='Donald Ferguson')
graph.create(n)
```

In [231]:
```
q = """
        match (n1:Fan {uni: 'dff9'}), (n2:Person {name: $name})
            create (n1)-[:FANOF]->(n2)
    """
graph.run(q, name='Tom Hanks')
```

Out[231]: (No data)

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




**Result of Create**

- 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.

In [62]:
```
import pymysql

conn = pymysql.connect(host="tutorialdb.cbezzskgwcl3.us-east-2.rds.ama
        port=3306,
        user="admin",
        password="7Senses_kiki",
        db="HW3_IMDBFixed",
        cursorclass=pymysql.cursors.DictCursor)
```

In [64]:
```
q = "select characterName from characters"
cursor = conn.cursor()
cursor.execute(q, args=None)
result = cursor.fetchall()
conn.commit()
```

In [66]:
```
all_characters = []
for pair in result:
    all_characters.append(pair['characterName'])
```

In [67]:
```
q = "select * from character_relationships"
cursor = conn.cursor()
cursor.execute(q, args=None)
result = cursor.fetchall()
conn.commit()
```

In [70]:
```
all_relationships = []
for pair in result:
    all_relationships.append((pair['sourceCharacterName'], pair['relat
```

In [105]:
```
for char_name in all_characters:
    n = Node('Character', name=char_name)
    graph.create(n)
```

In [106]:
```
for rel in all_relationships:

    q = """
        match (n1:Character {name: $name1}), (n2:Character {name: $nam
            create (n1)-[r:RELTYPE {name: $rel_name}]->(n2)
        """
    graph.run(q, name1 = rel[0], name2 = rel[2], rel_name = rel[1])
```

In [107]:
```
q = """match (n1:Character)-[r:RELTYPE {name:"killed"}]-(n2:Character)
        return n1,r,n2"""
```

In [108]:
```
result = graph.run(q)
```

In [109]:
```
for r in result:
    for x in r:
        print(type(x), ":", dict(x))
    print()
```

```
<class 'py2neo.data.Node'> : {'name': 'Locke'}
<class 'py2neo.data.RELTYPE'> : {'name': 'killed'}
<class 'py2neo.data.Node'> : {'name': 'Hodor'}

<class 'py2neo.data.Node'> : {'name': 'Lommy Greenhands'}
<class 'py2neo.data.RELTYPE'> : {'name': 'killed'}
<class 'py2neo.data.Node'> : {'name': 'Polliver'}

<class 'py2neo.data.Node'> : {'name': 'Lothar Frey'}
<class 'py2neo.data.RELTYPE'> : {'name': 'killed'}
<class 'py2neo.data.Node'> : {'name': 'Talisa Maegyr'}

<class 'py2neo.data.Node'> : {'name': 'Lyanna Mormont'}
<class 'py2neo.data.RELTYPE'> : {'name': 'killed'}
<class 'py2neo.data.Node'> : {'name': 'Wun Wun'}

<class 'py2neo.data.Node'> : {'name': 'Lyanna Mormont'}
<class 'py2neo.data.RELTYPE'> : {'name': 'killed'}
<class 'py2neo.data.Node'> : {'name': 'Wun Wun'}
```