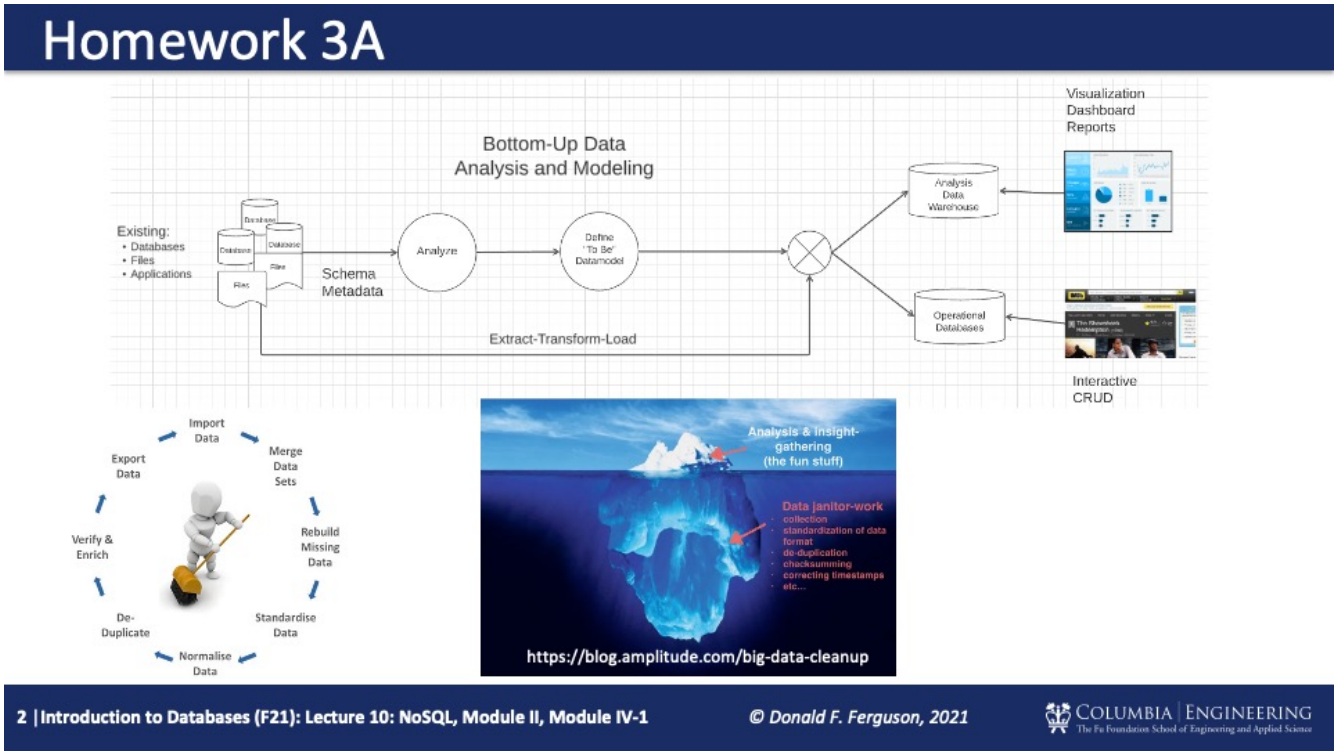
COMS W4111: Introduction to Databases Section 002, Fall 2021

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

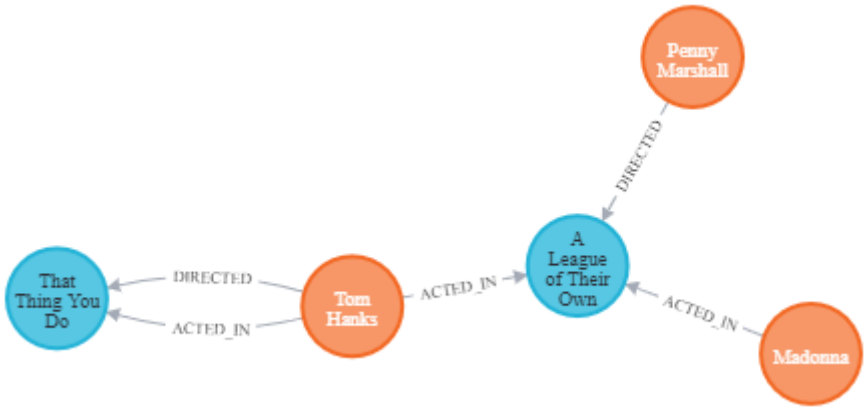
Neo4j

- *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*** |

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

In [1]:

```
!pip install py2neo
```

```
Collecting py2neo
  Downloading https://files.pythonhosted.org/packages/c7/76/8615d736888db5
793cef66165ca45807188c41031bfbc2533f84eab74e20/py2neo-2021.2.3-py2.py3-non
e-any.whl (177kB)
Collecting interchange~=2021.0.4 (from py2neo)
  Downloading https://files.pythonhosted.org/packages/88/bd/abc58e5a36a28e
0e55501f4bc15df74e430f399375b14b83f4ce22a257b4/interchange-2021.0.4-py2.py
3-none-any.whl
Requirement already satisfied: urllib3 in c:\users\tushar\anaconda3\lib\si
te-packages (from py2neo) (1.26.7)
Requirement already satisfied: certifi in c:\users\tushar\anaconda3\lib\si
te-packages (from py2neo) (2021.10.8)
Requirement already satisfied: six>=1.15.0 in c:\users\tushar\appdata\roam
ing\python\python37\site-packages (from py2neo) (1.15.0)
Requirement already satisfied: packaging in c:\users\tushar\anaconda3\lib
\site-packages (from py2neo) (21.2)
Collecting monotonic (from py2neo)
  Downloading https://files.pythonhosted.org/packages/9a/67/7e8406a29b6c45
be7af7740456f7f37025f0506ae2e05fb9009a53946860/monotonic-1.6-py2.py3-none-
any.whl
Requirement already satisfied: pygments>=2.0.0 in c:\users\tushar\anaconda
3\lib\site-packages (from py2neo) (2.3.1)
Collecting pansi>=2020.7.3 (from py2neo)
  Downloading https://files.pythonhosted.org/packages/0b/15/7972e08b7ec14a
8b10d5ff206c644d4478906c909c134123ed7e6bd16724/pansi-2020.7.3-py2.py3-none
-any.whl
Requirement already satisfied: pytz in c:\users\tushar\anaconda3\lib\site-
packages (from interchange~=2021.0.4->py2neo) (2021.3)
Requirement already satisfied: pyparsing<3,>=2.0.2 in c:\users\tushar\anac
onda3\lib\site-packages (from packaging->py2neo) (2.4.7)
Installing collected packages: interchange, monotonic, pansi, py2neo
Successfully installed interchange-2021.0.4 monotonic-1.6 pansi-2020.7.3 p
y2neo-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 [2]:

```
from py2neo import Graph,Node,Relationship
```

In [3]:

```
#
# The bolt URL and neo4j should be the same for everyone.
# Replace dbuserdbuser with the passsword you set when creating the graph.
#
graph = Graph("bolt://localhost:7687", auth=("neo4j", "admin123"))
```

In [4]:

```
#
# The following is the query you entered above.
#
q = """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"""
```

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 Mordabit
o']}
<class 'py2neo.data.Node'> : {'tagline': 'Once in a lifetime you get a cha
nce to do something different.', 'title': 'A League of Their Own', 'releas
ed': 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 [10]:

```
#
# Connect to MongoDB
#
from pymongo import MongoClient
client = MongoClient(
                host="localhost",
                port=27017
            )
client
```

Out[10]:

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

In [11]:

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

In [12]:

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

In [13]:

```
result = client.HW3.GOT_Characters.find(f, p)
result = list(result)
```

In [14]:

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


```
{
  "characterName": "Arya Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/
MV5BMTk5MTYwNDc0OF5BMl5BanBnXkFtZTcwOTg2NDg1Nw@@._V1_SY1000_CR0,0,665,1000
_AL_.jpg",
  "actorName": "Maisie Williams"
}
{
  "characterName": "Bran Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/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/images/M/
MV5BMWZiOGNjMDAtOTRlNi00MDJmLWEyMTMtOGEwZTM5ODJlNDAyXkEyXkFqcGdeQXVyMjk3NT
UyOTc@._V1_.jpg",
  "actorName": "Art Parkinson"
}
{
  "characterName": "Robb Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/
MV5BMjI2NDE1NzczNF5BMl5BanBnXkFtZTcwNjcwODg4OQ@@._V1_SY1000_CR0,0,845,1000
_AL_.jpg",
  "actorName": "Richard Madden"
}
```

In [15]:

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

Out[15]:

In [17]:

```
#!pip install nameparser
```

```
Collecting nameparser
  Downloading https://files.pythonhosted.org/packages/3d/70/8cc66ac7d01118
aa7a2ca938915ce4d622c5d73bb9f08058c7dce4ea9853/nameparser-1.0.6-py2.py3-no
ne-any.whl
Installing collected packages: nameparser
Successfully installed nameparser-1.0.6
```

In [18]:

```
from nameparser import HumanName
```

In [19]:

```
from pymongo import MongoClient
import json
import pandas as pd
```

```
C:\Users\tushar\Anaconda3\lib\site-packages\pandas\compat\_optional.py:13
8: UserWarning: Pandas requires version '2.7.0' or newer of 'numexpr' (ver
sion '2.6.9' currently installed).
  warnings.warn(msg, UserWarning)
```

In [35]:

```
from sqlalchemy import create_engine
```

In [36]:

```
engine = create_engine("mysql+pymysql://root:admin123@localhost")
```

In [20]:

```
client = MongoClient(
                host="localhost",
                port=27017
            )
```

In [21]:

```
client.list_database_names()
```

Out[21]:

```
['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 [29]:

```
result = client.HW3.GOT_Characters.find()
```

- *Execute the following test.*

In [30]:

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

Out[30]:

```
{'characterName': 'Archmaester Marwyn',
 'characterLink': '/character/ch0578265/',
 'actorName': 'Jim Broadbent',
 'actorLink': '/name/nm0000980/',
 'id': '619acf930551f2986f847123'}
```

- ***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 [31]:

```
data = pd.DataFrame.from_dict(result)
```

In [44]:

```
data = data[['id', 'characterName', 'characterLink','actorName', 'actorLink', 'characte
rImageFull', 'characterImageThumb', 'kingsguard', 'nickname', 'royal']]
```

In [52]:

```
data = data.drop('houseName', axis=1)
```

In [53]:

```
data.head()
```

Out[53]:

	id	characterName	characterLink	actorName	actorLi
0	619acf930551f2986f847119	Addam Marbrand	/character/ch0305333/	B.J. Hogg	/name/nm03896
1	619acf930551f2986f84711a	Aegon Targaryen	NaN	NaN	N
2	619acf930551f2986f84711b	Aeron Greyjoy	/character/ch0540081/	Michael Feast	/name/nm02699
3	619acf930551f2986f84711c	Aerys II Targaryen	/character/ch0541362/	David Rintoul	/name/nm07277
4	619acf930551f2986f84711d	Akho	/character/ch0544520/	Chuku Modu	/name/nm67298

In [54]:

```
from sqlalchemy import create_engine
db_data = 'mysql+pymysql://' + 'root' + ':' + 'admin123' + '@' + 'localhost' + ':'+ str
(3306)
engine = create_engine(db_data)
#engine = create_engine("mysql+pymysql://root:admin123@localhost")
data.to_sql("characters", engine, schema="HW3_GOT_Raw", if_exists='replace', index=Fals
e)
```

- *Test your result with the query below.*

In [55]:

```
pd.read_sql_query("select * from HW3_GOT_Raw.characters limit 10;", engine)
```

Out[55]:

	id	characterName	characterLink	actorName	actorLi
0	619acf930551f2986f847119	Addam Marbrand	/character/ch0305333/	B.J. Hogg	/name/nm03896
1	619acf930551f2986f84711a	Aegon Targaryen	None	None	Nc
2	619acf930551f2986f84711b	Aeron Greyjoy	/character/ch0540081/	Michael Feast	/name/nm02699
3	619acf930551f2986f84711c	Aerys II Targaryen	/character/ch0541362/	David Rintoul	/name/nm07277
4	619acf930551f2986f84711d	Akho	/character/ch0544520/	Chuku Modu	/name/nm67298
5	619acf930551f2986f84711e	Alliser Thorne	/character/ch0246938/	Owen Teale	/name/nm08535
6	619acf930551f2986f84711f	Alton Lannister	/character/ch0305012/	Karl Davies	/name/nm02038
7	619acf930551f2986f847120	Alys Karstark	/character/ch0576836/	Megan Parkinson	/name/nm82578
8	619acf930551f2986f847121	Amory Lorch	/character/ch0305002/	Fintan McKeown	/name/nm05716
9	619acf930551f2986f847122	Anguy	/character/ch0316930/	Philip McGinley	/name/nm15281

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.

Cleaning the data

The following DDL create a new schema and characters table for entering cleaned data

```
USE hw3_got_fixed;

create table characters
(
    id                  text       null,
    characterName       text       null,
    characterLink       text       null,
    actorName           text       null,
    actorLink           text       null,
    characterImageFull  text       null,
    characterImageThumb text       null,
    kingsguard          tinyint(1) null,
    nickname            text       null,
    royal               tinyint(1) null
);

-- Copy Data from raw
insert into characters
select * from hw3_got_raw.characters;

alter table characters modify id VARCHAR(50) not null;

alter table characters modify characterName VARCHAR(30) null;

alter table characters modify characterImageFull VARCHAR(200) null;

alter table characters modify characterImageThumb VARCHAR(200) null;

alter table characters modify kingsguard bool default FALSE null;

alter table characters modify royal bool default FALSE null;

alter table characters modify nickname VARCHAR(50) null;


-- Create an actor_id
alter table characters
    add actor_id VARCHAR(20) null;

UPDATE characters SET actor_id =
substr(actorLink, length('name')+3, length(actorLink) - length('name')-3)
WHERE 1;

-- Create a character_id
alter table characters
    add character_id VARCHAR(30) null;

UPDATE characters SET character_id =
substr(characterLink, length('character')+3, length(characterLink) - length('cha
racter')-3)
```

```
WHERE 1;

-- Update the kingsguard and royal columns
UPDATE characters set kingsguard = 0
where kingsguard IS NULL;

UPDATE characters set royal = 0
where royal IS NULL;
```

Ids received from the Mongo DB are used as a primary key in the characters table as the data is not cleaned and many different characters have the same name. Eg: White Walker. These should instead be White Walker #1, White Walker #2 etc. Therefore to model this data we use id.

```
alter table characters
    add constraint characters_pk
        primary key (id);
```

Currently the characters table contains actor infomration as well. We will drop actor info later. We need the information to populate data into the actors table

In [ ]:

```
character_data = pd.read_sql_query("select * from hw3_got_fixed.characters;", engine)
```

In [141]:

```
character_data.head()
```

Out[141]:

	id	characterName	characterLink	actorName	actorLi
0	619acf930551f2986f847119	Addam Marbrand	/character/ch0305333/	B.J. Hogg	/name/nm03896
1	619acf930551f2986f84711a	Aegon Targaryen	None	None	No
2	619acf930551f2986f84711b	Aeron Greyjoy	/character/ch0540081/	Michael Feast	/name/nm02699
3	619acf930551f2986f84711c	Aerys II Targaryen	/character/ch0541362/	David Rintoul	/name/nm07277
4	619acf930551f2986f84711d	Akho	/character/ch0544520/	Chuku Modu	/name/nm67298

Now we create an actors table. It has an autoincrement column (a_id) which acts as a unique value for an actor. We also create a joint character_actors table as characters and actors follow a many-to-many relationship.

```
-- Create an actors table
drop table if exists actors;
create table actors
(
    actor_id VARCHAR(20) null,
    actorName VARCHAR(100) null,
    a_id int NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (a_id)
);

-- Create a character_actors table
create table characters_actors
(
    c_id VARCHAR(50) not null,
    a_id INT not null
);
```

Load data into actors using pandas. We utilise the new characters table in hw3_got_fixed as it contains the cleaned column `actor_id`

In [ ]:

```
actors_data = character_data[['actor_id', 'actorName']].drop_duplicates()

actors_data = actors_data.dropna()

actors_data.to_sql('actors', engine, schema="HW3_GOT_Fixed", if_exists='append', index=False)
```

In [142]:

```
pd.read_sql_query("select * from hw3_got_fixed.actors LIMIT 10;", engine)
```

Out[142]:

	actor_id	actorName	a_id
0	nm0389698	B.J. Hogg	1
1	nm0269923	Michael Feast	2
2	nm0727778	David Rintoul	3
3	nm6729880	Chuku Modu	4
4	nm0853583	Owen Teale	5
5	nm0203801	Karl Davies	6
6	nm8257864	Megan Parkinson	7
7	nm0571654	Fintan McKeown	8
8	nm1528121	Philip McGinley	9
9	nm0000980	Jim Broadbent	10

We now insert data (id, actor_id) into the character_actors table as this would serve the many-to-many relationship between the character and actors table

```
INSERT INTO characters_actors
select t1.id as c_id, t2.a_id as a_id from
    hw3_got_fixed.characters as t1 join hw3_got_fixed.actors as t2
where
    exists
        (select * from hw3_got_fixed.characters as t3 where
            t1.id=t3.id and t2.actorName=t3.actorName);
```

In [144]:

```
pd.read_sql("select * from hw3_got_fixed.characters_actors LIMIT 10;", engine)
```

Out[144]:

	c_id	a_id
0	619acf930551f2986f847119	1
1	619acf930551f2986f84711b	2
2	619acf930551f2986f84711c	3
3	619acf930551f2986f84711d	4
4	619acf930551f2986f84711e	5
5	619acf930551f2986f84711f	6
6	619acf930551f2986f847120	7
7	619acf930551f2986f847121	8
8	619acf930551f2986f847122	9
9	619acf930551f2986f847123	10

We now add foreign key relationships from the characters_actors tables to the respective tables.

```
alter table characters_actors
    add constraint characters_actors_actors_actor_id_fk
        foreign key (a_id) references actors (a_id);

alter table characters_actors
    add constraint characters_actors_characters_id_fk
        foreign key (c_id) references characters (id);
```

We now drop the extra information in characters table

```
alter table characters drop column characterLink;

alter table characters drop column actorName;

alter table characters drop column actorLink;

alter table characters drop column actor_id;
```

The table now looks like:

In [155]:

```
pd.read_sql("select * from hw3_got_fixed.characters LIMIT 10;", engine)
```

Out[155]:

	id	characterName	characterImageFull	characterImageThumb	king
0	619acf930551f2986f847119	Addam Marbrand	None	None	
1	619acf930551f2986f84711a	Aegon Targaryen	None	None	
2	619acf930551f2986f84711b	Aeron Greyjoy	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	
3	619acf930551f2986f84711c	Aerys II Targaryen	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	
4	619acf930551f2986f84711d	Akho	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	
5	619acf930551f2986f84711e	Alliser Thorne	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	
6	619acf930551f2986f84711f	Alton Lannister	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	
7	619acf930551f2986f847120	Alys Karstark	None	None	
8	619acf930551f2986f847121	Amory Lorch	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	
9	619acf930551f2986f847122	Anguy	https://images-na.ssl-images-amazon.com/images...	https://images-na.ssl-images-amazon.com/images...	

Thus, we have added a many to many relationship between charaters and actors table.

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 [69]:

```
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 [85]:

```
def get_relation_data(rel_type):
    print(rel_type)
    data = []
    result = client['HW3']['GOT_Characters'].aggregate([
        {
            '$unwind': {
                'path': '$' + rel_type
            }
        }, {
            '$project': {
                'sourceCharacterName': '$characterName',
                'relation': rel_type,
                'targetCharacterName': '$' + rel_type
            }
        }
    ])
    result = list(result)

    return pd.DataFrame(result)
```

In [86]:

```
for rel_type in relationship_names:
    data = get_relation_data(rel_type)
    data.to_sql("character_relationships", engine, schema="HW3_GOT_Raw", if_exists='app
end', index=False)
```

```
abducted
abductedBy
allies
guardedBy
guardianOf
killed
killedBy
marriedEngaged
parentOf
parents
servedBy
serves
sibling
siblings
```

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 [157]:

```
n = Node("Fan", uni='tg2749', name='Tushar Gupta')
graph.create(n)
```

In [158]:

```
q = """
        match (n1:Fan {uni: 'tg2749'}), (n2:Person {name: $name})
            create (n1)-[:FANOF]->(n2)
    """
graph.run(q, name='Tom Hanks')
```

Out[158]:

(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 [128]:

```
character_data = pd.read_sql_query("select * from HW3_GOT_Fixed.characters;", engine)
```

In [101]:

```
def create_graph(row):
    id_num = row['id']
    c_name = row['characterName']
    n = Node("character", id = id_num , name=c_name)
    graph.create(n)
```

In [102]:

```
res = character_data.apply(create_graph, axis=1)
```

In [108]:

```
#test
# q = """
#         match (c1:character), (c2:character) where c1.name = $sname and c2.name=$tnam
e
#             create (c1)-[:{rel_type}]->(c2)
#     """
# fq = q.format(rel_type='killedBy')
# graph.run(fq, sname="Lady Crane", tname="Goldcloak")
```

In [137]:

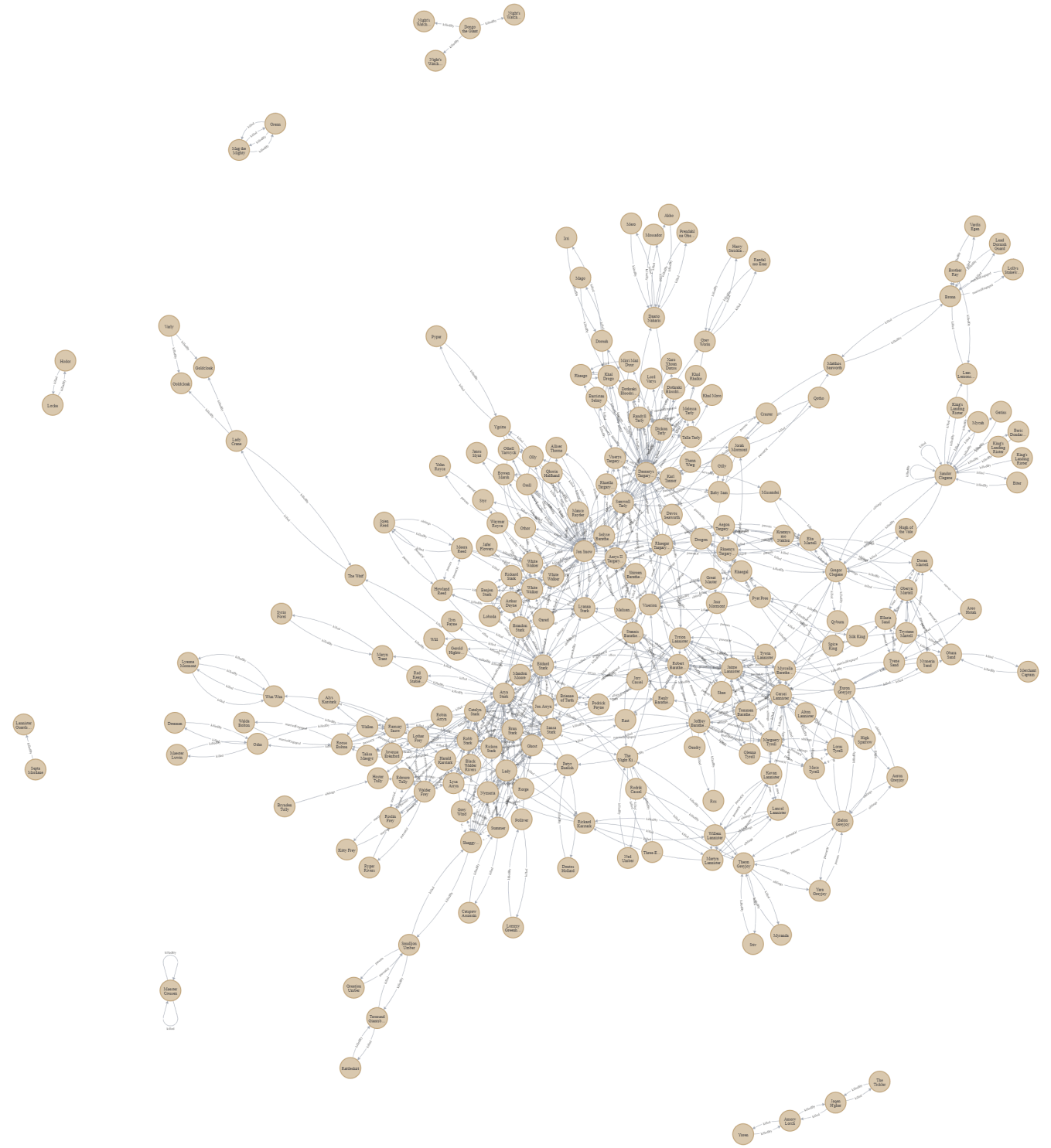
```
relationship_data = pd.read_sql_query("select * from HW3_GOT_Raw.character_relationship
s;", engine)
```

In [156]:

```
def create_edge(row):
    q = """
        match (c1:character), (c2:character) where c1.name = $sname and c2.name=$tname
            create (c1)-[:{rel_type}]->(c2)
    """
    fq = q.format(rel_type=row['relation'])
    s_name = row['sourceCharacterName']
    t_name = row['targetCharacterName']
    graph.run(fq, sname=s_name, tname=t_name)
```

In [ ]:

```
relationship_data.apply(create_edge, axis=1)
```



Result of Create

In [159]:

```
q = """match (n1:character)-[m]-(n2:character) return n1, n2, m LIMIT 25"""
```

In [160]:

```
result = graph.run(q)
```

In [161]:

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

```
Node('character', id='619acf930551f2986f84711a', name='Aegon Targaryen')
Node('character', id='619acf930551f2986f847194', name='Jon Snow')       si
blings(Node('character', id='619acf930551f2986f84711a', name='Aegon Targar
yen'), Node('character', id='619acf930551f2986f847194', name='Jon Snow'))
Node('character', id='619acf930551f2986f84711a', name='Aegon Targaryen')
Node('character', id='619acf930551f2986f847233', name='Rhaenys Targaryen')
siblings(Node('character', id='619acf930551f2986f84711a', name='Aegon Targ
aryen'), Node('character', id='619acf930551f2986f847233', name='Rhaenys Ta
rgaryen'))
Node('character', id='619acf930551f2986f84711a', name='Aegon Targaryen')
Node('character', id='619acf930551f2986f847233', name='Rhaenys Targaryen')
sibling(Node('character', id='619acf930551f2986f847233', name='Rhaenys Tar
garyen'), Node('character', id='619acf930551f2986f84711a', name='Aegon Tar
garyen'))
Node('character', id='619acf930551f2986f84711a', name='Aegon Targaryen')
Node('character', id='619acf930551f2986f847230', name='Rhaegar Targaryen')
parents(Node('character', id='619acf930551f2986f84711a', name='Aegon Targa
ryen'), Node('character', id='619acf930551f2986f847230', name='Rhaegar Tar
garyen'))
Node('character', id='619acf930551f2986f84711a', name='Aegon Targaryen')
Node('character', id='619acf930551f2986f847158', name='Elia Martell')   pa
rents(Node('character', id='619acf930551f2986f84711a', name='Aegon Targary
en'), Node('character', id='619acf930551f2986f847158', name='Elia Martel
l'))
Node('character', id='619acf930551f2986f84711a', name='Aegon Targaryen')
Node('character', id='619acf930551f2986f847230', name='Rhaegar Targaryen')
parentOf(Node('character', id='619acf930551f2986f847230', name='Rhaegar Ta
rgaryen'), Node('character', id='619acf930551f2986f84711a', name='Aegon Ta
rgaryen'))
Node('character', id='619acf930551f2986f84711a', name='Aegon Targaryen')
Node('character', id='619acf930551f2986f847158', name='Elia Martell')   pa
rentOf(Node('character', id='619acf930551f2986f847158', name='Elia Martel
l'), Node('character', id='619acf930551f2986f84711a', name='Aegon Targarye
n'))
Node('character', id='619acf930551f2986f84711a', name='Aegon Targaryen')
Node('character', id='619acf930551f2986f847170', name='Gregor Clegane') ki
lledBy(Node('character', id='619acf930551f2986f84711a', name='Aegon Targar
yen'), Node('character', id='619acf930551f2986f847170', name='Gregor Clega
ne'))
Node('character', id='619acf930551f2986f84711a', name='Aegon Targaryen')
Node('character', id='619acf930551f2986f847170', name='Gregor Clegane') ki
lled(Node('character', id='619acf930551f2986f847170', name='Gregor Clegan
e'), Node('character', id='619acf930551f2986f84711a', name='Aegon Targarye
n'))
Node('character', id='619acf930551f2986f84711b', name='Aeron Greyjoy')  No
de('character', id='619acf930551f2986f84715a', name='Euron Greyjoy')    si
blings(Node('character', id='619acf930551f2986f84715a', name='Euron Greyjo
y'), Node('character', id='619acf930551f2986f84711b', name='Aeron Greyjo
y'))
Node('character', id='619acf930551f2986f84711b', name='Aeron Greyjoy')  No
de('character', id='619acf930551f2986f84712a', name='Balon Greyjoy')    si
blings(Node('character', id='619acf930551f2986f84712a', name='Balon Greyjo
y'), Node('character', id='619acf930551f2986f84711b', name='Aeron Greyjo
y'))
Node('character', id='619acf930551f2986f84711b', name='Aeron Greyjoy')  No
de('character', id='619acf930551f2986f84715a', name='Euron Greyjoy')    si
blings(Node('character', id='619acf930551f2986f84711b', name='Aeron Greyjo
y'), Node('character', id='619acf930551f2986f84715a', name='Euron Greyjo
y'))
Node('character', id='619acf930551f2986f84711b', name='Aeron Greyjoy')  No
de('character', id='619acf930551f2986f84712a', name='Balon Greyjoy')    si
```

```
blings(Node('character', id='619acf930551f2986f84711b', name='Aeron Greyjo
y'), Node('character', id='619acf930551f2986f84712a', name='Balon Greyjo
y'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f847232', name='Rhaella Tar
garyen')        siblings(Node('character', id='619acf930551f2986f847232',
name='Rhaella Targaryen'), Node('character', id='619acf930551f2986f84711
c', name='Aerys II Targaryen'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f847232', name='Rhaella Tar
garyen')        siblings(Node('character', id='619acf930551f2986f84711c',
name='Aerys II Targaryen'), Node('character', id='619acf930551f2986f84723
2', name='Rhaella Targaryen'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f847127', name='Arthur Dayn
e')     serves(Node('character', id='619acf930551f2986f847127', name='Arth
ur Dayne'), Node('character', id='619acf930551f2986f84711c', name='Aerys I
I Targaryen'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f84718a', name='Jaime Lanni
ster')  servedBy(Node('character', id='619acf930551f2986f84711c', name='Ae
rys II Targaryen'), Node('character', id='619acf930551f2986f84718a', name
='Jaime Lannister'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f847127', name='Arthur Dayn
e')     servedBy(Node('character', id='619acf930551f2986f84711c', name='Ae
rys II Targaryen'), Node('character', id='619acf930551f2986f847127', name
='Arthur Dayne'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f84727c', name='Viserys Tar
garyen')        parents(Node('character', id='619acf930551f2986f84727c', n
ame='Viserys Targaryen'), Node('character', id='619acf930551f2986f84711c',
name='Aerys II Targaryen'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f847230', name='Rhaegar Tar
garyen')        parents(Node('character', id='619acf930551f2986f847230', n
ame='Rhaegar Targaryen'), Node('character', id='619acf930551f2986f84711c',
name='Aerys II Targaryen'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f847145', name='Daenerys Ta
rgaryen')       parents(Node('character', id='619acf930551f2986f847145', n
ame='Daenerys Targaryen'), Node('character', id='619acf930551f2986f84711
c', name='Aerys II Targaryen'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f84727c', name='Viserys Tar
garyen')        parentOf(Node('character', id='619acf930551f2986f84711c',
name='Aerys II Targaryen'), Node('character', id='619acf930551f2986f84727
c', name='Viserys Targaryen'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f847230', name='Rhaegar Tar
garyen')        parentOf(Node('character', id='619acf930551f2986f84711c',
name='Aerys II Targaryen'), Node('character', id='619acf930551f2986f84723
0', name='Rhaegar Targaryen'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f847145', name='Daenerys Ta
rgaryen')       parentOf(Node('character', id='619acf930551f2986f84711c',
name='Aerys II Targaryen'), Node('character', id='619acf930551f2986f84714
5', name='Daenerys Targaryen'))
Node('character', id='619acf930551f2986f84711c', name='Aerys II Targarye
n')     Node('character', id='619acf930551f2986f847232', name='Rhaella Tar
garyen')        marriedEngaged(Node('character', id='619acf930551f2986f847
```

```
232', name='Rhaella Targaryen'), Node('character', id='619acf930551f2986f8
4711c', name='Aerys II Targaryen'))
```