COMS W4111: Introduction to Databases
Section 002, Fall 2021

Homework 3A

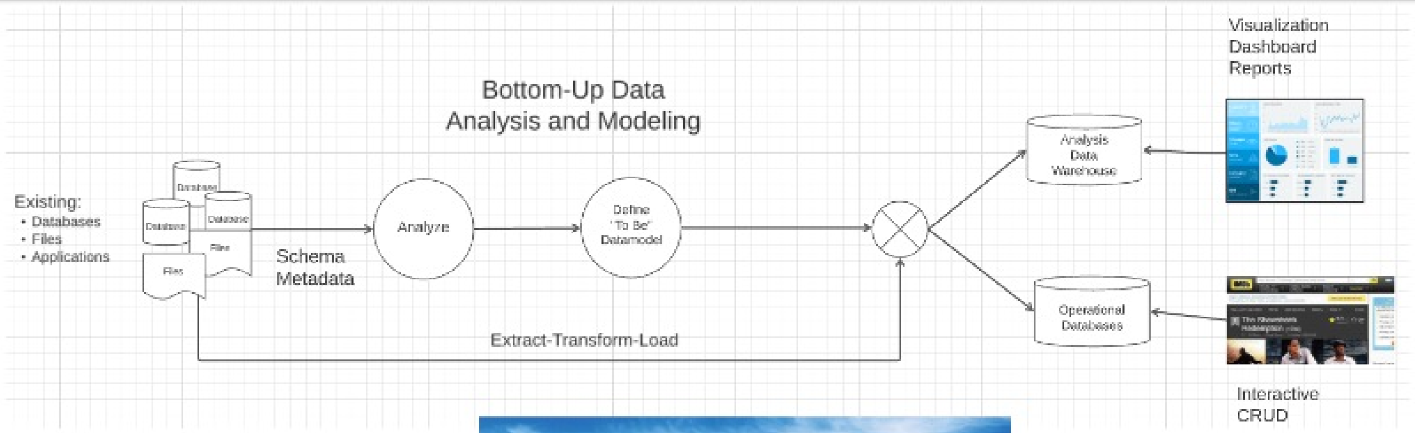
Student Name: Dana AlShehri

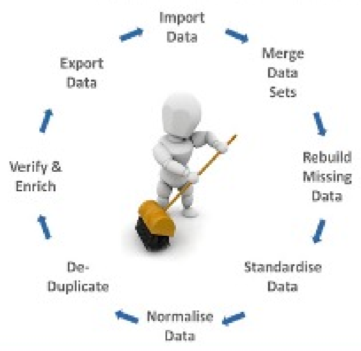
Student ID: da2975

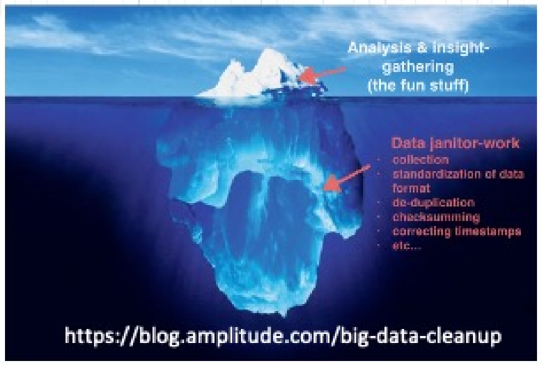
Overview

- 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.

Homework 3A






Analysis & insight-gathering (the fun stuff)
Data janitor-work
collection
standardization of data format
de-duplication
checksumming
correcting timestamps
etc...
https://blog.amplitude.com/big-data-cleanup

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.**

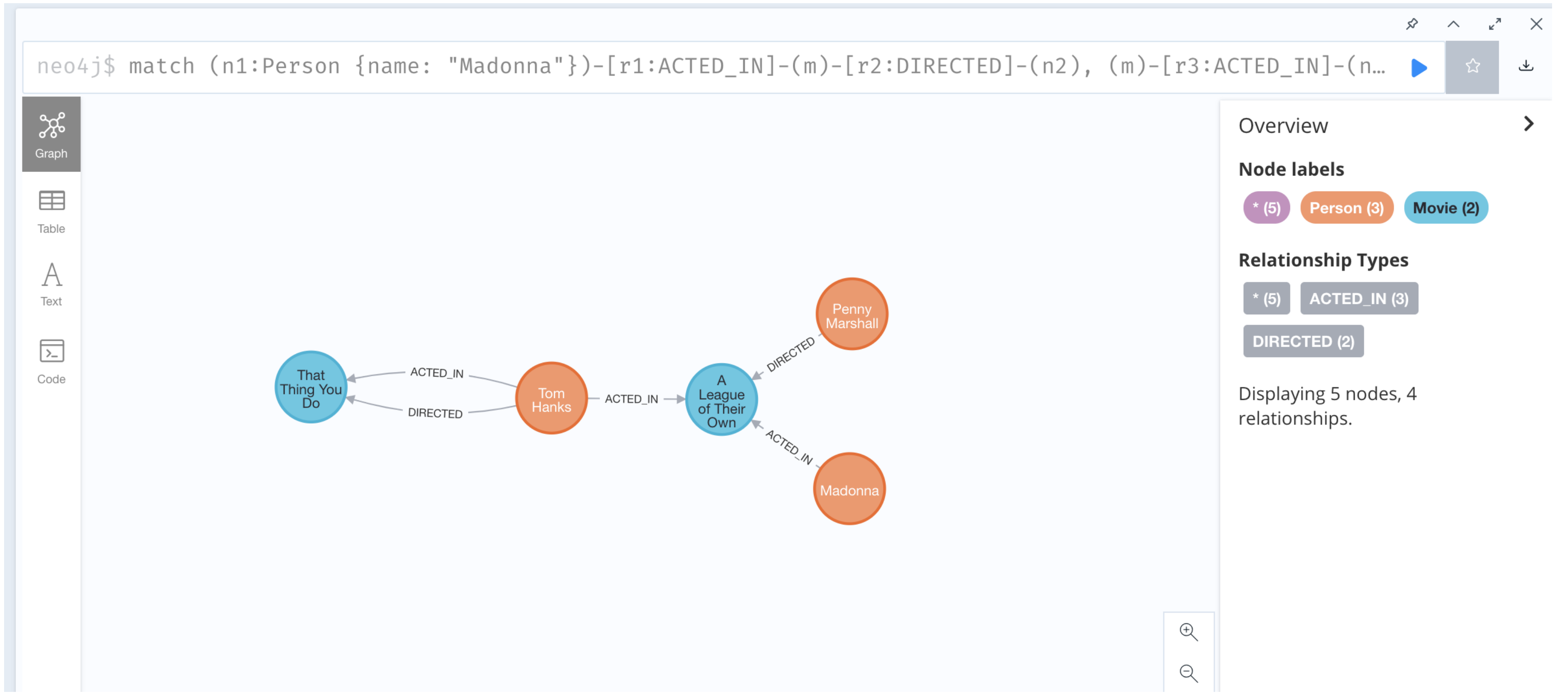
- Start and connect to the graph using the Neo4j browser (launch-able from `Open` on the desktop after you create the graph).

- 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:DIRECTED]-(n2), (m)-[r3:ACTED_IN]-(n
3), (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$ match (n1:Person {name: "Madonna"})-[r1:ACTED_IN]-(m)-[r2:DIRECTED]-(n2), (m)-[r3:ACTED_IN]-(n…
```

Overview

Node labels

* (5) Person (3) Movie (2)

Relationship Types

* (5) ACTED_IN (3) DIRECTED (2)

Displaying 5 nodes, 4 relationships.

Neo4j Setup Test

- 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 6.0 MB/s eta 0:00:01
Requirement already satisfied: pygments>=2.0.0 in /Users/danaalshehri/opt/anaconda3/lib/python3.8/site
-packages (from py2neo) (2.8.1)
Collecting monotonic
  Downloading monotonic-1.6-py2.py3-none-any.whl (8.2 kB)
Requirement already satisfied: certifi in /Users/danaalshehri/opt/anaconda3/lib/python3.8/site-package
s (from py2neo) (2020.12.5)
Requirement already satisfied: urllib3 in /Users/danaalshehri/opt/anaconda3/lib/python3.8/site-package
s (from py2neo) (1.26.4)
Requirement already satisfied: six>=1.15.0 in /Users/danaalshehri/opt/anaconda3/lib/python3.8/site-pac
kages (from py2neo) (1.15.0)
Collecting pansi>=2020.7.3
  Downloading pansi-2020.7.3-py2.py3-none-any.whl (10 kB)
Collecting interchange~=2021.0.4
  Downloading interchange-2021.0.4-py2.py3-none-any.whl (28 kB)
Requirement already satisfied: packaging in /Users/danaalshehri/opt/anaconda3/lib/python3.8/site-packa
ges (from py2neo) (20.9)
Requirement already satisfied: pytz in /Users/danaalshehri/opt/anaconda3/lib/python3.8/site-packages
(from interchange~=2021.0.4->py2neo) (2021.1)
Requirement already satisfied: pyparsing>=2.0.2 in /Users/danaalshehri/opt/anaconda3/lib/python3.8/sit
e-packages (from packaging->py2neo) (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 [1]:
```
from py2neo import Graph,Node,Relationship
```

```
In [2]: #
        # 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", "87651234"))
```

```
In [3]: #
        # 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 [4]: #
        # Run the query.
        #
        result=graph.run(q)
```

```
In [5]: 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 Mordabito']}
<class 'py2neo.data.Node'> : {'tagline': 'Once in a lifetime you get a chance to do something differen
t.', '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 b
ecomes 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 [6]: import json
```

```
In [7]: with open('./characters.json', "r") as in_file:
            c_data = json.load(in_file)
        c_data = c_data['characters']
```

```
In [8]: c_data[1]

Out[8]: {'characterName': 'Aegon Targaryen',
         'houseName': 'Targaryen',
         'royal': True,
         'parents': ['Elia Martell', 'Rhaegar Targaryen'],
         'siblings': ['Rhaenys Targaryen', 'Jon Snow'],
         'killedBy': ['Gregor Clegane']}
```

```
In [9]: #
        # Connect to MongoDB
        #
        from pymongo import MongoClient
        client = MongoClient(
                        host="localhost",
                        port=27017
                    )
        client

Out[9]: MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True)
```

```
In [10]: #
         # Load the character information into the HW3 MongoDB and collection
         #
         for c in c_data:
             client.HW3.GOT_Characters.insert_one(c)
```

In [11]:
```
#
# Now, test for correct loading.
#
f = {"siblings": "Sansa Stark"}
p = {
    "_id": 0,
    "characterName": 1,
    "characterImageFull": 1,
    "actorName": 1
}
```

In [12]:
```
result = client.HW3.GOT_Characters.find(f, p)
result = list(result)
```

```
In [13]: for r in result:
             print(json.dumps(r, indent=2))
```

```
{
  "characterName": "Arya Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTk5MTYwNDc0OF5BMl5BanBn
XkFtZTcwOTg2NDg1Nw@@._V1_SY1000_CR0,0,665,1000_AL_.jpg",
  "actorName": "Maisie Williams"
}
{
  "characterName": "Bran Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTA1NTg0NTI3MTBeQTJeQWpw
Zl5BbWU3MDEyNjg4OTQ@._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/MV5BMWZiOGNjMDAtOTRlNi00MDJm
LWEyMTMtOGEwZTM5ODJlNDAyXkEyXkFqcGdeQXVyMjk3NTUyOTc@._V1_.jpg",
  "actorName": "Art Parkinson"
}
{
  "characterName": "Robb Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/MV5BMjI2NDE1NzczNF5BMl5BanBn
XkFtZTcwNjcwODg4OQ@@._V1_SY1000_CR0,0,845,1000_AL_.jpg",
  "actorName": "Richard Madden"
}
{
  "characterName": "Arya Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTk5MTYwNDc0OF5BMl5BanBn
XkFtZTcwOTg2NDg1Nw@@._V1_SY1000_CR0,0,665,1000_AL_.jpg",
  "actorName": "Maisie Williams"
}
{
  "characterName": "Bran Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTA1NTg0NTI3MTBeQTJeQWpw
Zl5BbWU3MDEyNjg4OTQ@._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/MV5BMWZiOGNjMDAtOTRlNi00MDJm
LWEyMTMtOGEwZTM5ODJlNDAyXkEyXkFqcGdeQXVyMjk3NTUyOTc@._V1_.jpg",
  "actorName": "Art Parkinson"
}
{
  "characterName": "Robb Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/MV5BMjI2NDE1NzczNF5BMl5BanBn
XkFtZTcwNjcwODg4OQ@@._V1_SY1000_CR0,0,845,1000_AL_.jpg",
  "actorName": "Richard Madden"
}
{
  "characterName": "Arya Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTk5MTYwNDc0OF5BMl5BanBn
XkFtZTcwOTg2NDg1Nw@@._V1_SY1000_CR0,0,665,1000_AL_.jpg",
  "actorName": "Maisie Williams"
}
{
  "characterName": "Bran Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTA1NTg0NTI3MTBeQTJeQWpw
Zl5BbWU3MDEyNjg4OTQ@._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/MV5BMWZiOGNjMDAtOTRlNi00MDJm
LWEyMTMtOGEwZTM5ODJlNDAyXkEyXkFqcGdeQXVyMjk3NTUyOTc@._V1_.jpg",
  "actorName": "Art Parkinson"
}
{
  "characterName": "Robb Stark",
  "characterImageFull": "https://images-na.ssl-images-amazon.com/images/M/MV5BMjI2NDE1NzczNF5BMl5BanBn
XkFtZTcwNjcwODg4OQ@@._V1_SY1000_CR0,0,845,1000_AL_.jpg",
  "actorName": "Richard Madden"
}
```

In [14]:
```python
#
# And, just for the heck of it ...
#
from IPython import display
display.Image(result[0]["characterImageFull"], width="300px")
```

Out[14]:

```
In [15]: from nameparser import HumanName
```

```
In [16]: from pymongo import MongoClient
         import json
         import pandas as pd
```

```
In [17]: from sqlalchemy import create_engine
```

```
In [18]: engine = create_engine("mysql+pymysql://root:87651234@localhost/")
```

```
In [19]: client = MongoClient(
                             host="localhost",
                             port=27017
                         )
```

```
In [20]: client.list_database_names()
```

```
Out[20]: ['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 [103]:
```
p = {
    "_id": 1,
    "actorLink": 1,
    "actorName": 1,
    "characterName": 1,
    "characterLink": 1,
    "characterImageFull": 1,
    "characterImageThumb": 1,
    "houseName": 1,
    "kingsguard": 1,
    "nickname": 1,
    "royal": 1
}
result = client.HW3.GOT_Characters.find({}, p)
```

- Execute the following test.

In [104]:
```
result = list(result)
for r in result:
    r["id"] = str(r["_id"])
    del r["_id"]
result[10]
```

Out[104]:
```
{'characterName': 'Archmaester Marwyn',
 'characterLink': '/character/ch0578265/',
 'actorName': 'Jim Broadbent',
 'actorLink': '/name/nm0000980/',
 'id': '619961ea551d060dfac11d3e'}
```

- **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 [105]:
```
sql_engine = create_engine("mysql+pymysql://root:56781234@localhost")
```

In [106]:
```
%load_ext sql
```

In [107]:
```
%sql mysql+pymysql://root:56781234@localhost/
```

In [105]:
```
cols = ['actorLink', 'characterName', 'characterLink','actorName', '_id','royal','characterImageThumb',
        'characterImageFull','nickname','kingsguard','houseName']
data = pd.read_csv('GOT_Characters.csv',  usecols = cols)
df = pd.DataFrame(data)
df.to_sql('characters', con = sql_engine, schema= 'HW3_IMDBRaw')
```

- Test your result with the query below.

```
In [155]: %sql select * from HW3_IMDBRaw.characters limit 10;
```

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

Out[155]:

id	actorLink	actorName	
61997c2bdee7e981f7970f30	/name/nm0389698/	B.J. Hogg	
61997c2bdee7e981f7970f31	None	None	
61997c2bdee7e981f7970f32	/name/nm0269923/	Michael Feast	amazon.com/images/M/MV5BNzI5MDg0ZDAtN2Y2ZC00MzU1LTgyY
61997c2bdee7e981f7970f33	/name/nm0727778/	David Rintoul	amazon.com/images/M/MV5BMWQzOWViN2ItNDZhOS00MmZlLTkxZTYtZDg5NGUwMGRm
61997c2bdee7e981f7970f34	/name/nm6729880/	Chuku Modu	amazon.com/images/M/MV5BOGE4ZDZmOGUtNGE4Ny00Y2VmLThiOG
61997c2bdee7e981f7970f35	/name/nm0853583/	Owen Teale	https://images-na.ssl-images-amazon.com/images/M/MV5BMjAxMjExMjA3M15BMl5Ba
61997c2bdee7e981f7970f36	/name/nm0203801/	Karl Davies	https://images-na.ssl-images-amazon.com/images/M/MV5B
61997c2bdee7e981f7970f37	/name/nm8257864/	Megan Parkinson	
61997c2bdee7e981f7970f38	/name/nm0571654/	Fintan McKeown	amazon.com/images/M/MV5BOTVmY2M2YmUtY2JkYS00NjIyLWFhYTA
61997c2bdee7e981f7970f39	/name/nm1528121/	Philip McGinley	amazon.com/images/M/MV5BNmRhY2M4YmItNjc2Yi00ZDc0LWE5NmU

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.

Data loading statements:

```
In [109]: char_cols = ['characterName', 'characterLink', '_id','royal','characterImageThumb',
                  'characterImageFull','nickname','kingsguard','houseName']
          data = pd.read_csv('GOT_Characters.csv',  usecols = char_cols)
          df = pd.DataFrame(data)
          df.to_sql('characters', con = sql_engine, schema= 'HW3_IMDBFixed')
```

```
In [110]: actors_cols = ['characterName','actorLink', 'actorName', '_id']
          data = pd.read_csv('GOT_Characters.csv',  usecols = actors_cols)
          df = pd.DataFrame(data)
          df.to_sql('actors', con = sql_engine, schema= 'HW3_IMDBFixed')
```

Alter statements for both tables:

```
alter table actors change _id id varchar(100) not null;

alter table actors modify actorLink varchar(50) null;

alter table actors modify actorName varchar(30) null;

alter table actors modify characterName varchar(30) null;

alter table actors
    add constraint actors_pk
        primary key (id);

alter table characters change _id id varchar(100) not null;

alter table characters modify characterImageFull varchar(200) null;

alter table characters modify characterImageThumb varchar(200) null;

alter table characters modify characterLink varchar(200) null;
alter table characters modify characterName varchar(30) null;

alter table characters modify houseName varchar(100) null;

alter table characters modify kingsguard boolean null;

alter table characters modify nickname varchar(30) null;

alter table characters modify royal boolean null;

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

create index character_index
    on actors (characterName);

alter table characters change characterImageFull imageFull varchar(200) null;

alter table characters change characterImageThumb imageThumb varchar(200) null;

alter table characters change characterLink link varchar(200) null;

alter table characters change characterName name varchar(30) null;

alter table characters change houseName house varchar(100) null;

alter table actors change actorName name varchar(30) null;

alter table actors change actorLink link varchar(50) null after name;

alter table characters
    add constraint characters_fk
        foreign key (name) references actors (characterName);
```

Final create table statement for characters table:

```
create table if not exists HW3_IMDBFixed.characters
(
    `index` bigint null,
    id varchar(100) not null
        primary key,
    imageFull varchar(200) null,
    imageThumb varchar(200) null,
    link varchar(200) null,
    name varchar(30) null,
    house varchar(100) null,
    kingsguard tinyint(1) null,
    nickname varchar(30) null,
    royal tinyint(1) null,
    constraint characters_fk
        foreign key (name) references HW3_IMDBFixed.actors (characterName)
);

create index ix_HW3_IMDBFixed_characters_index
    on HW3_IMDBFixed.characters (`index`);
```

Final create table statement for actors table:

```
create table if not exists HW3_IMDBFixed.actors
(
    `index` bigint null,
    id varchar(100) not null
        primary key,
    name varchar(30) null,
    link varchar(50) null,
    characterName varchar(30) null
);
```

```
create index character_index
    on HW3_IMDBFixed.actors (characterName);

create index ix_HW3_IMDBFixed_actors_index
    on HW3_IMDBFixed.actors (`index`);
```

Verification Tests:

In [112]:
```
%sql select * from HW3_IMDBFixed.actors limit 10
```

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

Out[112]:

index	id	name	link	characterName
0	61997c2bdee7e981f7970f30	B.J. Hogg	/name/nm0389698/	Addam Marbrand
1	61997c2bdee7e981f7970f31	None	None	Aegon Targaryen
2	61997c2bdee7e981f7970f32	Michael Feast	/name/nm0269923/	Aeron Greyjoy
3	61997c2bdee7e981f7970f33	David Rintoul	/name/nm0727778/	Aerys II Targaryen
4	61997c2bdee7e981f7970f34	Chuku Modu	/name/nm6729880/	Akho
5	61997c2bdee7e981f7970f35	Owen Teale	/name/nm0853583/	Alliser Thorne
6	61997c2bdee7e981f7970f36	Karl Davies	/name/nm0203801/	Alton Lannister
7	61997c2bdee7e981f7970f37	Megan Parkinson	/name/nm8257864/	Alys Karstark
8	61997c2bdee7e981f7970f38	Fintan McKeown	/name/nm0571654/	Amory Lorch
9	61997c2bdee7e981f7970f39	Philip McGinley	/name/nm1528121/	Anguy

```
In [113]: %sql select * from HW3_IMDBFixed.characters limit 10
```

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

Out[113]:

imageFull	
None	
None	
https://images-na.ssl-images- YjEzODczZDVhXkEyXkFqcGdeQXVyNTg0Nzg4NTE@._V1_.jpg	amazon.com/images/M/MV5BNzI5MDg0ZDAtN2Y2ZC00MzU1LTgyYjQtNT
https://images-na.ssl-images- ltYWdlL2ltYWdlXkEyXkFqcGdeQXVyMjk3NTUyOTc@._V1_.jpg	amazon.com/images/M/MV5BMWQzOWViN2ItNDZhOS00MmZlLTkxZTYtZDg5NGUwMGRmYWZ
https://images-na.ssl-images- U0ZDY3OWQxXkEyXkFqcGdeQXVyMjk3NTUyOTc@._V1_.jpg	amazon.com/images/M/MV5BOGE4ZDZmOGUtNGE4Ny00Y2VmLThiOGItMjk
tZTcwMjI1ODg5NA@@._V1_SY1000_CR0,0,666,1000_AL_.jpg	https://images-na.ssl-images-amazon.com/images/M/MV5BMj
TU1NTAzOF5BMl5BanBnXkFtZTcwNzA2NDk4OA@@._V1_.jpg	https://images-na.ssl-images-amazon.com/images/M/MV5BMTU
None	
https://images-na.ssl-images- nI4ODdiNmE5XkEyXkFqcGdeQXVyMjg2MTMyNTM@._V1_.jpg	amazon.com/images/M/MV5BOTVmY2M2YmUtY2JkYS00NjIyLWFhYTAtNTN
https://images-na.ssl-images- WE0YTQ2YjY3XkEyXkFqcGdeQXVyMjk3NTUyOTc@._V1_.jpg	amazon.com/images/M/MV5BNmRhY2M4YmItNjc2Yi00ZDc0LWE5NmUtNGE

```
In [115]: %%sql select a.name as Actor_Name, a.characterName as Character_Name, b.house as House_Name
          from HW3_IMDBFixed.actors a join HW3_IMDBFixed.characters b on a.characterName = b.name
          where a.name is not null and b.house is not null limit 5;
```

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

Out[115]:

Actor_Name	Character_Name	House_Name
Michael Feast	Aeron Greyjoy	Greyjoy
David Rintoul	Aerys II Targaryen	Targaryen
Karl Davies	Alton Lannister	Lannister
Maisie Williams	Arya Stark	Stark
Patrick Malahide	Balon Greyjoy	Greyjoy

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 [22]:
```
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 [126]:
```
import pymysql
conn = pymysql.connect(host="localhost",
                           port=3306,
                           user="root",
                           password="56781234",
                           db="HW3_IMDBFixed")
```

In [134]:
```
def run_q(cnx, q, args, fetch=False): #From HW2 Skeleton Code

    cursor = cnx.cursor()
    cursor.execute(q, args)
    if fetch:
        result = cursor.fetchall()
    else:
        result = None
    cnx.commit()
    return result

def load_SQL_table(records):

    total = len(records)

    for i in range(0, total):
        source = records[i][0]
        relation = records[i][1]
        target = records[i][2]
        q = "insert into HW3_IMDBFixed.character_relationships values(%s, %s, %s)"
        result = run_q(conn, q, (source,relation,target))
    return


def load_info(relation, projection, filter = {}):

    records = []

    result = client.HW3.GOT_Characters.find(filter, projection)
    result = list(result)

    for r in result:
        if len(r.keys()) == 2:  #we have source character, relation, target character
            record = (r[list(projection)[1]], relation, r[relation][0],)
            if record in records:
                continue
            records.append(record)

    return records #source, relationship, target


for name in relationship_names:
    p = { "_id": 0, "characterName": 1, str(name): 1}
    records = load_info(relation=str(name), projection=p)
    load_SQL_table(records)
```

NOTE: Imported a copy of 'character_relationships' table from HW3_IMDBFIXED to HW3_GOT_RAW in DataGrip

Verifcation Test

In [237]:
```
%sql select * from HW3_GOT_RAW.character_relationships limit 10
```

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

Out[237]:

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

Task II-b: Load Neo4j

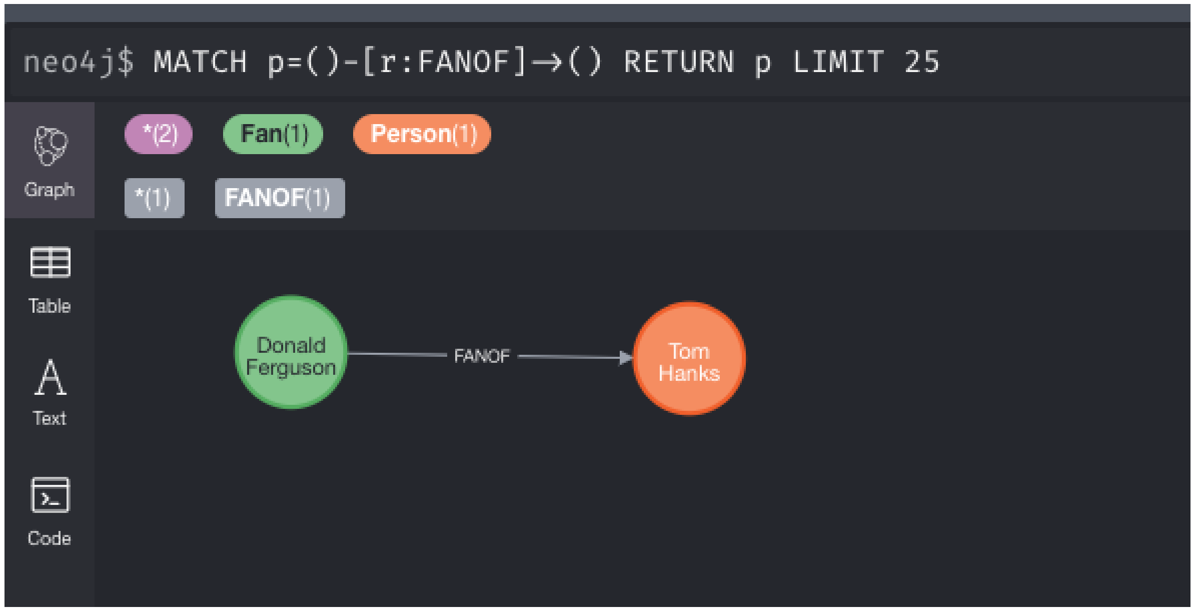
- At this point, you should have the following tables in `HW3_GOT_Fixed:`
  - `characters`
  - `character_relationships`
- You will now load this information into Neo4j. The following code shows you some simple steps to create nodes and relationships.

```
In [208]: n = Node("Fan", uni='dff9', name='Donald Ferguson')
          graph.create(n)
```

```
In [209]: q = """
                  match (n1:Fan {uni: 'dff9'}), (n2:Person {name: $name})
                      create (n1)-[:FANOF]->(n2)
              """
          graph.run(q, name='Tom Hanks')
```

```
Out[209]: (No data)
```

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



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

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.

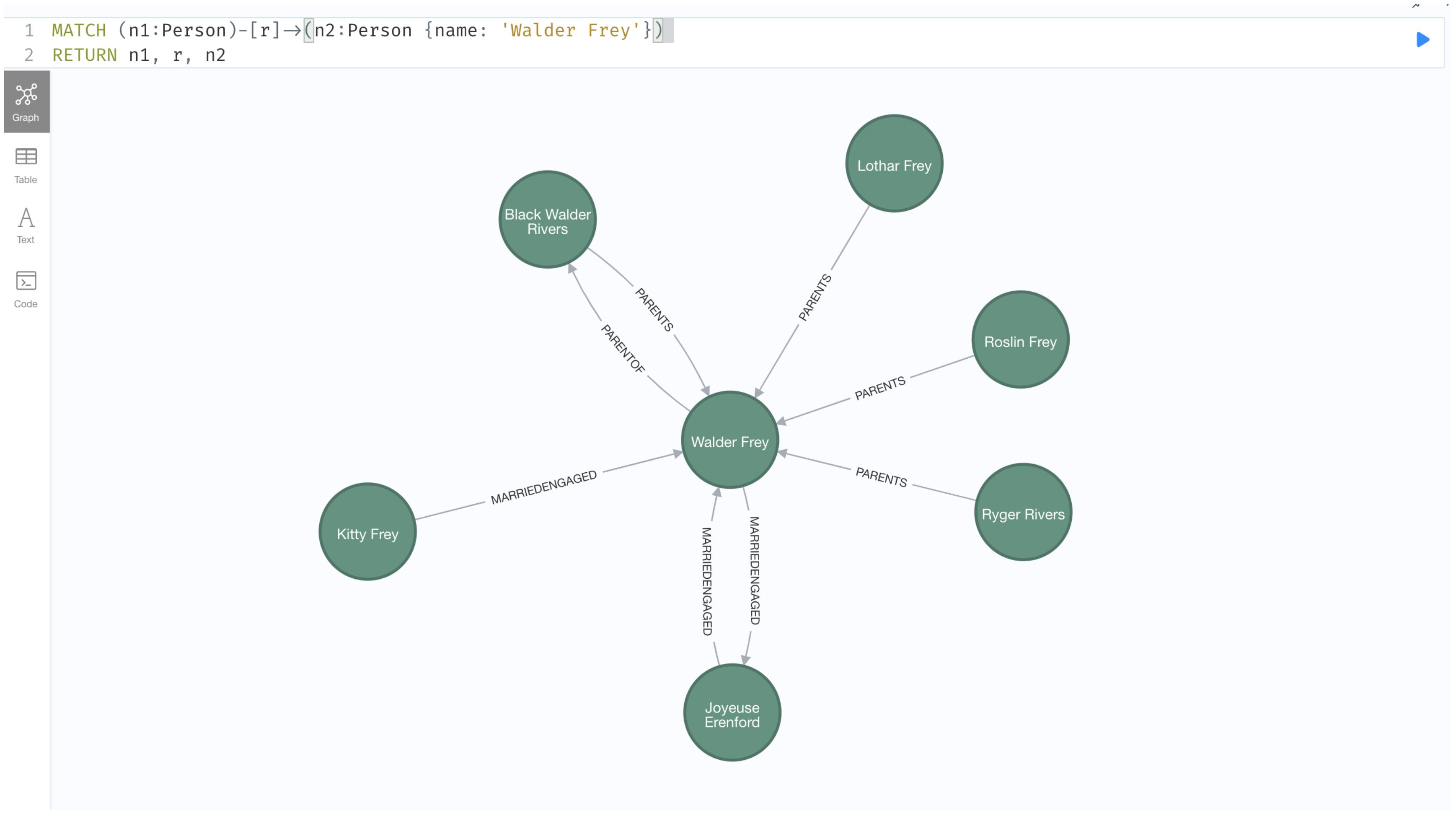
```
In [232]: #Load character names and ids from characters table into a list
          characters = []
          total_characters = 389

          for i in range(0, total_characters):
              query = "select  id, name from HW3_IMDBFixed.characters"
              df = pd.read_sql(query, conn)
              characters = df.to_records(index=False)

          characters[0]

Out[232]: ('61997c2bdee7e981f7970f30', 'Addam Marbrand')
```

```
In [198]: #Load info from characters_relationships table into a list
          characters_relationships = []
          total_characters = 510

          for i in range(0, total_characters):
              query = "select sourceCharacterName, relationship, targetCharacterName from HW3_IMDBFixed.character_
              df = pd.read_sql(query, conn)
              characters_relationships = df.to_records(index=False)

          characters_relationships[0]

Out[198]: ('Rhaegar Targaryen', 'abducted', 'Lyanna Stark')
```

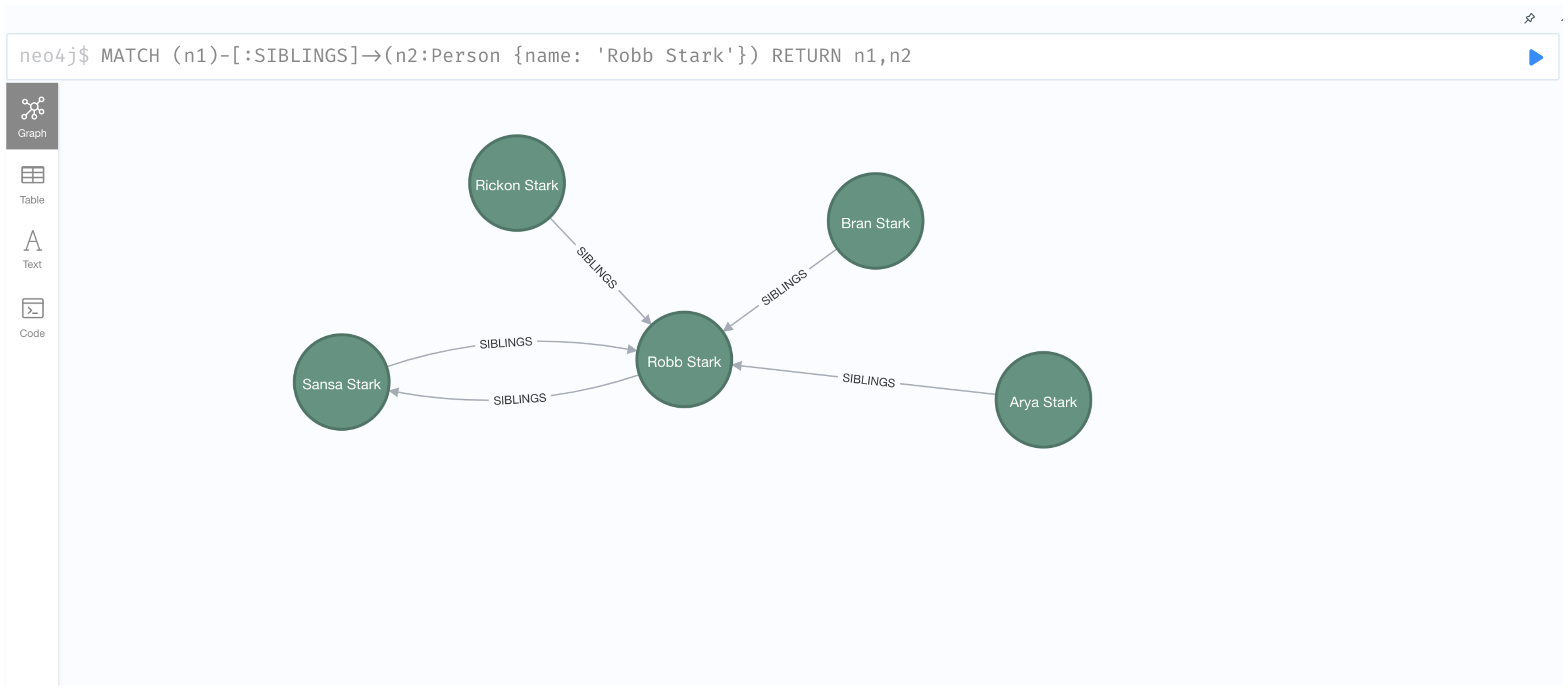
```
In [230]: #Cypher Qeury to create a Node for each character in neo4j
          for tuple in characters:
              id = tuple[0]
              name = tuple[1]
              n = Node("Person", id= str(id), name=str(name))
              graph.create(n)
```

In [231]:
```python
#Creating the relationship between the characters based on characters_relationships table info.
for record in characters_relationships:
    source_name = record[0]
    rel = record[1]
    target_name = record[2]

    source_id = None
    target_id = None
    for tuple in characters:

        if source_id is not None and target_id is not None:
            break
        elif tuple[1] == source_name:
            source_id = tuple[0]

        elif tuple[1] == target_name:
            target_id = tuple[0]

    if target_id is None:
        continue
    cq1 = "MATCH (n:Person) WHERE n.id = '" + str(source_id) + "' return n"
    cq2 = "MATCH (n:Person) WHERE n.id = '" + str(target_id) + "' return n"

    #Evaluate the Cypher query
    source_node = graph.evaluate(cq1)
    target_node = graph.evaluate(cq2)

    graph.create(Relationship(source_node, rel.upper(), target_node))
```

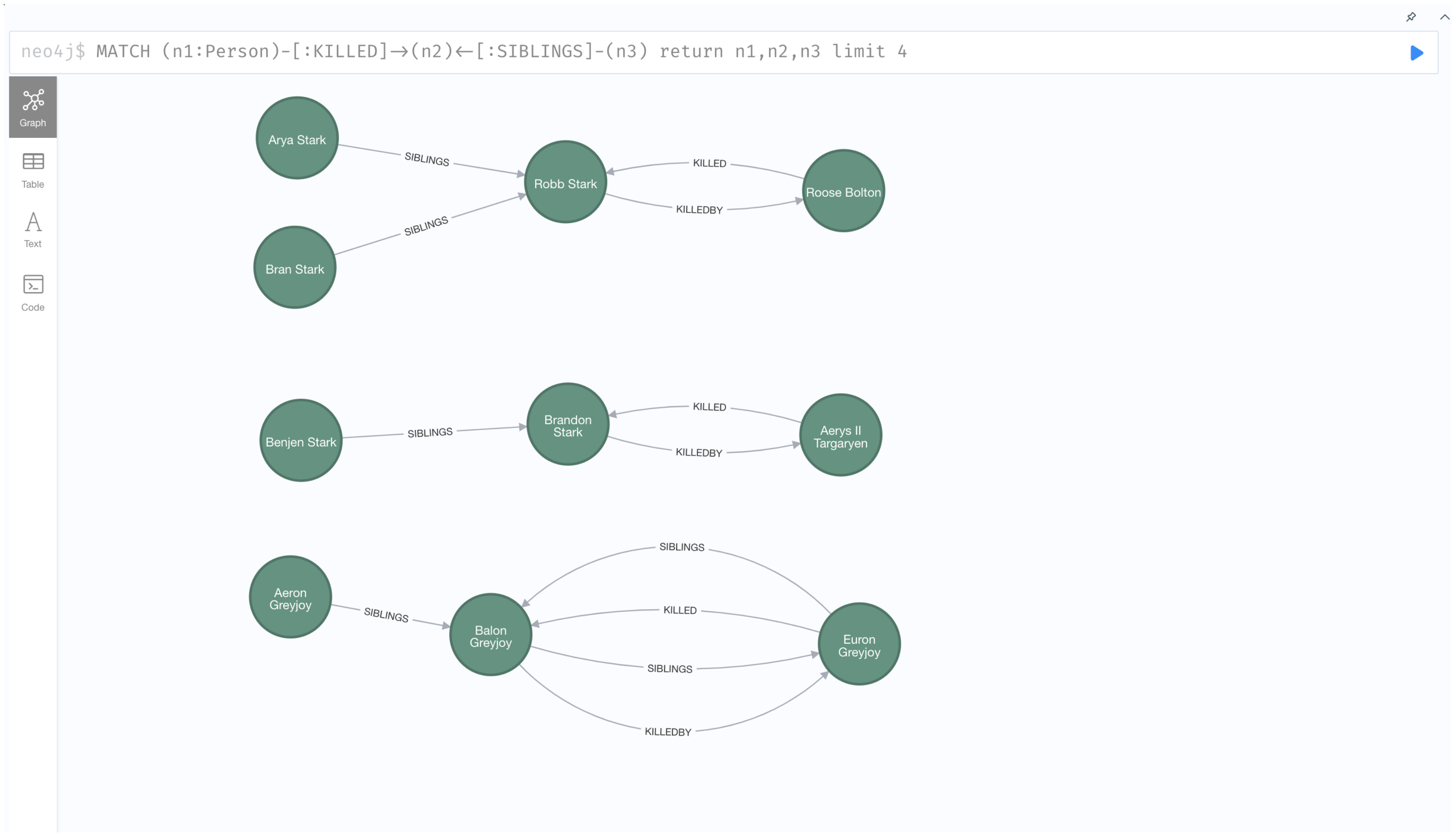
Verifications Tests



```
MATCH (n1:Person)-[r]→(n2:Person {name: 'Walder Frey'})
RETURN n1, r, n2
```



```
neo4j$ MATCH (n1)-[:SIBLINGS]→(n2:Person {name: 'Robb Stark'}) RETURN n1,n2
```



```
neo4j$ MATCH (n1:Person)-[:KILLED]→(n2)←[:SIBLINGS]-(n3) return n1,n2,n3 limit 4
```