

COMS W4111-003/V03 (Fall 2022)

Introduction to Databases

Homework 1, Part 2

Note:

- Please replace the information below with your last name, first name and UNI.
- Please delete the track that you are not taking from "Programming, Non-Programming."

Student Information: LastName, FirstName, UNI

Track: Programming, Non-Programming

Introduction

Overview and Objectives

HW 1 is the first step in the process of incrementally implementing a small project. You will have an executable, demoable project by the end of the semester. We build the project one homework assignment at a time. The non-programming track develops a simple data engineering and data science Jupyter notebook. The programming track builds a simple full stack web application.

There are two sections to HW 1, part 2. There is one section for each track. You only need to complete the section for the track you have chosen.

Submission

1. Remove `dff9` from the file name and replace with your UNI.
2. File > Print Preview > Download as PDF

3. Upload .pdf and .ipynb to GradeScope

This assignment is due 12-October-2022 at 11:59PM EDT.

Collaboration

- You may use any information found in TA or Prof. Ferguson's office hours, class recordings, slides,
- You may use information you find on the web, but must provide a link to the information and cite.
- You may not copy code or answers verbatim. To can use the web to find information, but must provide your own answers.
- You are not allowed to collaborate outside of office hours
- You are NOT allowed to collaborate with other students outside of office hours.

Non-Programming Section

Data Loading

The following sections load the data from files into MySQL. The HW task uses the MySQL tables.

Step 1: Read Episode Information

The zip file for the homework contains a JSON file with information about episodes in Game of Thrones. The following code loads the file into a Pandas data frame.

```
In [1]: import pandas as pd
```

```
In [20]: file_name = "./flattened_episodes.json"
df = pd.read_json(file_name)
df
```

Out[20]:

	seasonNum	episodeNum	episodeTitle	episodeLink	episodeAirDate	episodeDescrip
0	1	1	Winter Is Coming	/title/tt1480055/	2011-04-17	Jon Arryn, Hand of the King, dead. King
1	1	1	Winter Is Coming	/title/tt1480055/	2011-04-17	Jon Arryn, Hand of the King, dead. King
2	1	1	Winter Is Coming	/title/tt1480055/	2011-04-17	Jon Arryn, Hand of the King, dead. King
3	1	1	Winter Is Coming	/title/tt1480055/	2011-04-17	Jon Arryn, Hand of the King, dead. King
4	1	1	Winter Is Coming	/title/tt1480055/	2011-04-17	Jon Arryn, Hand of the King, dead. King
...
4160	8	6	The Iron Throne	/title/tt6027920/	2019-05-19	In the aftermath of the devastating attack on King's Landing
4161	8	6	The Iron Throne	/title/tt6027920/	2019-05-19	In the aftermath of the devastating attack on King's Landing
4162	8	6	The Iron Throne	/title/tt6027920/	2019-05-19	In the aftermath of the devastating attack on King's Landing
4163	8	6	The Iron Throne	/title/tt6027920/	2019-05-19	In the aftermath of the devastating attack on King's Landing
4164	8	6	The Iron Throne	/title/tt6027920/	2019-05-19	In the aftermath of the devastating attack on King's Landing

4165 rows × 13 columns

Step 2: Save the Episode Information

The following code saves the episode information to a relational database table. You must change the user ID and password for the MySQL database.

```
In [3]: %load_ext sql
```

```
In [4]: %sql mysql+pymysql://root:dbuserdbuser@localhost
```

Danger: The following code will delete any previous work in the database you have done.

```
In [5]: %sql drop database if exists f22_hw1_got
```

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

Out[5]: []

```
In [6]: %sql create database f22_hw1_got
```

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

Out[6]: []

Pandas needs a SQLAlchemy engine to interact with a relational database.

```
In [8]: from sqlalchemy import create_engine
```

```
In [9]: engine = create_engine("mysql+pymysql://root:dbuserdbuser@localhost")
```

```
In [21]: df.to_sql("episodes_scenes", schema="f22_hw1_got", con=engine, index=False, if_
```

Out[21]: 4165

The following code is a simple test to see if you have written the data.

```
In [12]: %sql select seasonNum, episodeNum, count(scene_no) as no_of_scenes from \
          f22_hw1_got.episodes_scenes group by seasonNum, episodeNum \
          order by seasonNum, episodeNum
```

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

Out[12]:

seasonNum	episodeNum	no_of_scenes
1	1	36
1	2	31
1	3	25
1	4	28
1	5	28
1	6	19
1	7	25
1	8	37
1	9	25
1	10	32
2	1	30
2	2	31
2	3	30
2	4	33
2	5	38
2	6	47
2	7	38
2	8	43
2	9	133
2	10	45
3	1	50
3	2	49
3	3	42
3	4	50
3	5	37
3	6	26
3	7	48
3	8	50
3	9	71
3	10	47
4	1	56
4	2	82
4	3	55
4	4	44
4	5	50

4	6	38
4	7	27
4	8	34
4	9	86
4	10	45
5	1	47
5	2	46
5	3	55
5	4	51
5	5	48
5	6	39
5	7	46
5	8	59
5	9	53
5	10	64
6	1	44
6	2	45
6	3	37
6	4	46
6	5	85
6	6	60
6	7	47
6	8	53
6	9	71
6	10	89
7	1	40
7	2	59
7	3	50
7	4	86
7	5	54
7	6	75
7	7	104
8	1	86
8	2	69
8	3	292
8	4	113

8	5	220
8	6	91

Step 3: Load the Character Information

```
In [14]: # This logic is basically the same as above.
file_name = "./flattened_characters.json"
df = pd.read_json(file_name)
df
```

Out[14]:

	characterName	characterLink	actorName	actorLink	houseName	roy
0	Addam Marbrand	/character/ch0305333/	B.J. Hogg	/name/nm0389698/	NaN	Na
1	Aegon Targaryen	NaN	NaN	NaN	Targaryen	1.
2	Aeron Greyjoy	/character/ch0540081/	Michael Feast	/name/nm0269923/	Greyjoy	Na
3	Aerys II Targaryen	/character/ch0541362/	David Rintoul	/name/nm0727778/	Targaryen	1.
4	Akho	/character/ch0544520/	Chuku Modu	/name/nm6729880/	NaN	Na
...
384	Young Nan	/character/ch0305018/	Annette Tierney	/name/nm1519719/	NaN	Na
385	Young Ned	/character/ch0154681/	Robert Aramayo	/name/nm7075019/	Stark	Na
386	Young Ned Stark	/character/ch0154681/	Sebastian Croft	/name/nm7509185/	Stark	Na
387	Young Rodrik Cassel	/character/ch0171391/	Fergus Leatham	/name/nm7509186/	NaN	Na
388	Zanrush	/character/ch0540870/	Gerald Lepkowski	/name/nm0503319/	NaN	Na

389 rows x 25 columns

Step 4: Save the Data

```
In [15]: df.to_sql("characters", schema="f22_hw1_got", con=engine, index=False, if_exists="replace")
```

Out[15]: 389

```
In [17]: # Test the load.
%sql select characterName, actorName, actorLink from f22_hw1_got.characters where
```

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

Out[17]:

characterName	actorName	actorLink
Arthur Dayne	Luke Roberts	/name/nm1074361/
Brienne of Tarth	Gwendoline Christie	/name/nm3729225/
Jaime Lannister	Nikolaj Coster-Waldau	/name/nm0182666/
Mandon Moore	James Doran	/name/nm0243696/
Podrick Payne	Daniel Portman	/name/nm4535552/

Once More with Feeling

We are going to do the same thing with locations and subLocations. But this, time we are really going to get excited about data processing. So, "Once More with Feeling!"

In [24]:

```
# This logic is basically the same as above.
file_name = "./flattened_locations.json"
df = pd.read_json(file_name)
df
```

Out[24]:

	location	subLocation
0	North of the Wall	The Lands of Always Winter
1	North of the Wall	Cave Outside Wildling Camp
2	North of the Wall	Wildling Camp
3	North of the Wall	Frostfang Mountains
4	North of the Wall	The Three-Eyed Raven
...
115	The Red Waste	The Desert
116	Qarth	
117	Qarth	King's Landing
118	Qarth	The Wall
119	Qarth	Vaes Dothrak

120 rows x 2 columns

In [25]:

```
df.to_sql("locations", schema="f22_hw1_got", con=engine, index=False, if_exists=
```

Out[25]: 120



Non-Programming Tasks

Complete the tasks in this section if you are on the Non-Programming Track



The basic idea is the following:

- You have three tables in your database:
 1. `episodes_scenes`
 2. `characters`
 3. `locations`
- The raw data we loaded is kind of "icky," which is a highly technical data engineering term.
- We are going to going to restructure and de-icky the data a little bit, and then do some queries.
- So, you want to have a cool job in data science, AI/ML, IEOR, ... that involves getting insight from data ... I have some bad news.

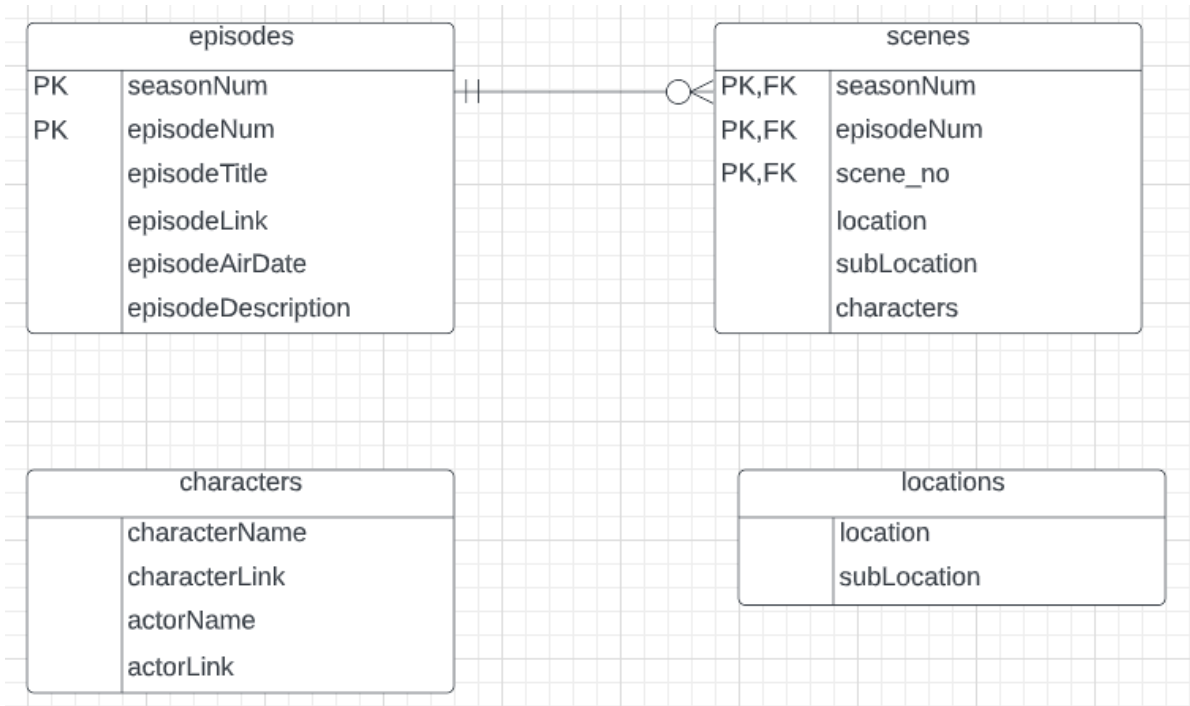
Data Janitor



"While it is a commonly held belief that data janitor work is fully automated, many data scientists are employed primarily as data janitors. The Information technology industry has been increasingly turning towards new sources of data gathered on consumers, so data janitors have become more commonplace in recent years." (https://en.wikipedia.org/wiki/Data_janitor)

Task 1: Copy the Data and Create Some Keys

- We are going to keep the original tables and make some copies that we will clean up.
- Your first task is create a new database `f22_hw1_got_clean` that has the following structure.



- Put and execute your SQL statements in the cells below. Note: You have to create the primary keys and foreign keys from the ER diagram.
- You can use the `create table xxx as select * from` version of select to create the tables. We provide one example.

```
In [ ]: ## These two cells are the examples - go and run these cells in order!
%%sql create database f22_hw1_got_clean
```

```
In [ ]: %%sql

create table f22_hw1_got_clean.episodes as
select distinct seasonNum, episodeNum, episodeTitle, episodeLink, episodeAirDate, episodeDescription
from f22_hw1_got.episodes_scenes
```

- Put the rest of your SQL below, which will be `create table` and `alter table` statements. You must execute your statements.

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

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

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

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

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

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

Task 2: Convert to NULL

[Ted Codd](#), who pioneered relational databases, defined 12 rules for RDBs.

A critical rule is [Rule 3: Systematic Treatment of NULL Values](#)

The NULL values in a database must be given a systematic and uniform treatment.

This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.

There are columns that are effectively NULL but have some other marker, e.g. "", ";". Your task is to identify these columns and covert the symbol indicating NULL to the value NULL.

Put and execute your SQL below.

In []: %%sql

In []: %%sql

Task 3: Some not so Simple Queries

- We saw JOIN statements in class. We also saw the = comparison operator in class.
- Finding out which characters were in which scenes is a little more complicated, however. We have incompletely cleaned up the data. We will do a better job in the future.
- In the short term, we can use the LIKE from SQL. The following query shows how to use the operator to find out (approximately) in which scenes a character appeared.

In [42]: %%sql

```
use f22_hw1_got_clean;
```

```
select characterName, seasonNum, episodeNum, scene_no, location, subLocation fr
    scenes.characters like concat("%", characters.characterName, "%");
where characterName="Nymeria";
```

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

Out[42]:

characterName	seasonNum	episodeNum	scene_no	location	subLocation
Nymeria	1	1	15	The North	Outside Winterfell
Nymeria	1	2	5	The North	Winterfell
Nymeria	1	2	21	The Riverlands	Crossroads Inn
Nymeria	1	2	22	The Riverlands	Crossroads Inn
Nymeria	5	4	31	Dorne	None
Nymeria	5	4	32	Dorne	None
Nymeria	5	6	19	Dorne	The Water Gardens
Nymeria	5	6	20	Dorne	The Water Gardens
Nymeria	5	6	22	Dorne	The Water Gardens
Nymeria	5	6	23	Dorne	The Water Gardens
Nymeria	5	7	30	Dorne	The Water Gardens
Nymeria	5	9	22	Dorne	The Water Gardens
Nymeria	5	9	23	Dorne	The Water Gardens
Nymeria	5	9	35	Dorne	The Water Gardens
Nymeria	5	10	38	Dorne	None
Nymeria	5	10	40	Dorne	None
Nymeria	6	1	26	The Crownlands	Blackwater Bay
Nymeria	6	10	70	Dorne	The Water Gardens
Nymeria	6	10	71	Dorne	The Water Gardens
Nymeria	7	2	33	The Riverlands	To The Twins
Nymeria	7	2	36	The Narrow Sea	None
Nymeria	7	2	45	The Narrow Sea	None
Nymeria	7	2	47	The Narrow Sea	None
Nymeria	7	2	48	The Narrow Sea	None
Nymeria	7	2	55	The Narrow Sea	None
Nymeria	7	2	57	The Narrow Sea	None

Task 3.1: Find the Starks

- Write a query that returns the `characters` whose last name is Stark. The basic form of a `characterName` in `characters` is `"firstName lastName"`.

In []: `%%sql`

Task 3.2: An Aggregations

- Using the hint on how to **JOIN** the tables **characters** and **scenes**, Produce a table that returns:
 - **characterName**
 - **location**
 - **subLocation**
 - **no_of_scenes**, which is the count of the number of **scenes** in which the character appeared in the **location**, **subLocation**
 - sorted by **no_of_scenes** descending.
 - Only include results with **no_of_scenes** **>= 100**

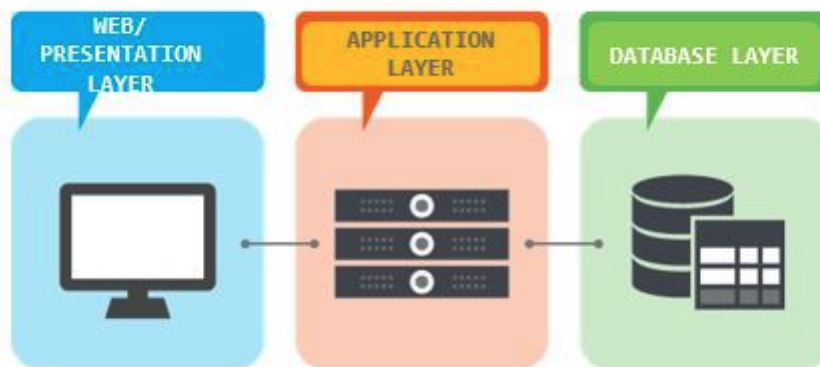
In []: `%%sql`

Programming Track

Concept

- Most "databases" have a common core set of operations: Create, Retrieve, Update, Delete.
- In the relational model, the matching operations are: **INSERT, SELECT, UPDATE, DELETE.**
- Full stack web applications are typically a **3-tier application architecture**.

Let us walk through a three tier architecture :



A typical representation of three tier architecture

- There interface/protocol between the presentation layer and application later is typically **REST**.

- To get started with our application, we are going to focus on just some code that reads the database and returns information. Professor Ferguson will provide code that completes the stack to implement your first web application.
- The following "get started" code will help with some of your work.

```
In [63]: import pymysql
import pandas as pd
import numpy as np

def get_connection():
    """
    This function connects to a database and returns the connection.
    :return: The connection
    """

    # TODO Replace the user and password with the information for your MySQL in
    conn = pymysql.connect(
        user="root",
        password="dbuserdbuser",
        host="localhost",
        autocommit=True,
        cursorclass=pymysql.cursors.DictCursor
    )

    return conn

def run_query(sql, args, fetch=True):
    """
    Runs a query. The SQL contains "%s" placeholders for parameters for the query
    result set.

    :param sql: An SQL string with "%s" placeholders for parameters.
    :param args: A list of values to insert into the query for the parameters.
    :param fetch: If true, return the result set.
    :return: The result set or the number of rows affected.
    """

    result = None

    conn = get_connection()
    cursor = conn.cursor()

    result = cursor.execute(sql, args)
    if fetch:
        result = cursor.fetchall()

    return result
```

- And this is a simple test.

```
In [50]: sql = "select characterName, actorName from f22_hw1_got.characters where charac
res = run_query(sql, ("Arya Stark"))
res
```



```
Out[50]: [{'characterName': 'Arya Stark', 'actorName': 'Maisie Williams'}]
```

Tasks

Task 1: Load the Data

- The following statements create a schema and some tables.

```
In [51]: %sql create database f22_hw1_got_programming
```

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

```
Out[51]: []
```

```
In [52]: %%sql
```

```
create table if not exists f22_hw1_got_programming.characters
(
    characterName      text      null,
    characterLink      text      null,
    actorName          text      null,
    actorLink          text      null,
    houseName          text      null,
    royal              double    null,
    parents            text      null,
    siblings           text      null,
    killedBy          text      null,
    characterImageThumb text      null,
    characterImageFull text      null,
    nickname           text      null,
    killed            text      null,
    servedBy           text      null,
    parentOf           text      null,
    marriedEngaged     text      null,
    serves             text      null,
    kingsguard         double    null,
    guardedBy          text      null,
    actors             text      null,
    guardianOf         text      null,
    allies             text      null,
    abductedBy         text      null,
    abducted           text      null,
    sibling             text      null
);

create table if not exists f22_hw1_got_programming.episodes_scenes
(
    seasonNum          bigint    null,
    episodeNum         bigint    null,
    episodeTitle       text      null,
    episodeLink        text      null,
    episodeAirDate     text      null,
    episodeDescription text      null,
    openingSequenceLocations text  null,
    sceneStart         text      null,
    sceneEnd           text      null,
```



```

location      text    null,
subLocation   text    null,
characters     text    null,
scene_no      bigint  null
);

```

```

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

```

Out[52]: []

- You can load information from JSON files using `pandas`. I like lists, so I convert to a list.

In [55]: `df = pd.read_json('flattened_characters.json')`
`df`

Out[55]:

	characterName	characterLink	actorName	actorLink	houseName	roy
0	Addam Marbrand	/character/ch0305333/	B.J. Hogg	/name/nm0389698/	NaN	Na
1	Aegon Targaryen	NaN	NaN	NaN	Targaryen	1.
2	Aeron Greyjoy	/character/ch0540081/	Michael Feast	/name/nm0269923/	Greyjoy	Na
3	Aerys II Targaryen	/character/ch0541362/	David Rintoul	/name/nm0727778/	Targaryen	1.
4	Akho	/character/ch0544520/	Chuku Modu	/name/nm6729880/	NaN	Na
...
384	Young Nan	/character/ch0305018/	Annette Tierney	/name/nm1519719/	NaN	Na
385	Young Ned	/character/ch0154681/	Robert Aramayo	/name/nm7075019/	Stark	Na
386	Young Ned Stark	/character/ch0154681/	Sebastian Croft	/name/nm7509185/	Stark	Na
387	Young Rodrik Cassel	/character/ch0171391/	Fergus Leatham	/name/nm7509186/	NaN	Na
388	Zanrush	/character/ch0540870/	Gerald Lepkowski	/name/nm0503319/	NaN	Na

389 rows x 25 columns

In [57]: `character_list = df.to_dict('records')`
`character_list[0:4]`

```

Out[57]: [{ 'characterName': 'Addam Marbrand',
  'characterLink': '/character/ch0305333/',
  'actorName': 'B.J. Hogg',
  'actorLink': '/name/nm0389698/',
  'houseName': nan,
  'royal': nan,
  'parents': nan,
  'siblings': nan,
  'killedBy': nan,
  'characterImageThumb': nan,
  'characterImageFull': nan,
  'nickname': nan,
  'killed': nan,
  'servedBy': nan,
  'parentOf': nan,
  'marriedEngaged': nan,
  'serves': nan,
  'kingsguard': nan,
  'guardedBy': nan,
  'actors': nan,
  'guardianOf': nan,
  'allies': nan,
  'abductedBy': nan,
  'abducted': nan,
  'sibling': nan},
{ 'characterName': 'Aegon Targaryen',
  'characterLink': nan,
  'actorName': nan,
  'actorLink': nan,
  'houseName': 'Targaryen',
  'royal': 1.0,
  'parents': 'Elia Martell;Rhaegar Targaryen',
  'siblings': 'Rhaenys Targaryen;Jon Snow',
  'killedBy': 'Gregor Clegane',
  'characterImageThumb': nan,
  'characterImageFull': nan,
  'nickname': nan,
  'killed': nan,
  'servedBy': nan,
  'parentOf': nan,
  'marriedEngaged': nan,
  'serves': nan,
  'kingsguard': nan,
  'guardedBy': nan,
  'actors': nan,
  'guardianOf': nan,
  'allies': nan,
  'abductedBy': nan,
  'abducted': nan,
  'sibling': nan},
{ 'characterName': 'Aeron Greyjoy',
  'characterLink': '/character/ch0540081/',
  'actorName': 'Michael Feast',
  'actorLink': '/name/nm0269923/',
  'houseName': 'Greyjoy',
  'royal': nan,
  'parents': nan,
  'siblings': 'Balon Greyjoy;Euron Greyjoy',
  'killedBy': nan,
  'characterImageThumb': 'https://images-na.ssl-images-amazon.com/images/M/MV5

```

```

BNzI5MDg0ZDAtn2Y2ZC00MzU1LTgyYjQtnTBjYjEzODczZDVhXkEyXkFqcGdeQXVyNTg0Nzg4NTE@.
_v1._SX100_SY140_.jpg',
  'characterImageFull': 'https://images-na.ssl-images-amazon.com/images/M/MV5B
NzI5MDg0ZDAtn2Y2ZC00MzU1LTgyYjQtnTBjYjEzODczZDVhXkEyXkFqcGdeQXVyNTg0Nzg4NTE@._
v1_.jpg',
  'nickname': 'Damphair',
  'killed': nan,
  'servedBy': nan,
  'parentOf': nan,
  'marriedEngaged': nan,
  'serves': nan,
  'kingsguard': nan,
  'guardedBy': nan,
  'actors': nan,
  'guardianOf': nan,
  'allies': nan,
  'abductedBy': nan,
  'abducted': nan,
  'sibling': nan},
{ 'characterName': 'Aerys II Targaryen',
  'characterLink': '/character/ch0541362/',
  'actorName': 'David Rintoul',
  'actorLink': '/name/nm0727778/',
  'houseName': 'Targaryen',
  'royal': 1.0,
  'parents': nan,
  'siblings': 'Rhaella Targaryen',
  'killedBy': 'Jaime Lannister',
  'characterImageThumb': 'https://images-na.ssl-images-amazon.com/images/M/MV5
BMWQzOWViN2ItNDZhOS00MmZlLTkxZTYtZDg5NGUwMGRmYWZjL2ltYWdlL2ltYWdlXkEyXkFqcGdeQ
XVyMjk3NTUyOTc@._v1._SX100_SY140_.jpg',
  'characterImageFull': 'https://images-na.ssl-images-amazon.com/images/M/MV5B
MWQzOWViN2ItNDZhOS00MmZlLTkxZTYtZDg5NGUwMGRmYWZjL2ltYWdlL2ltYWdlXkEyXkFqcGdeQ
XVyMjk3NTUyOTc@._v1_.jpg',
  'nickname': 'The Mad King',
  'killed': 'Brandon Stark;Rickard Stark',
  'servedBy': 'Arthur Dayne;Jaime Lannister',
  'parentOf': 'Daenerys Targaryen;Rhaegar Targaryen;Viserys Targaryen',
  'marriedEngaged': 'Rhaella Targaryen',
  'serves': nan,
  'kingsguard': nan,
  'guardedBy': nan,
  'actors': nan,
  'guardianOf': nan,
  'allies': nan,
  'abductedBy': nan,
  'abducted': nan,
  'sibling': nan}]

```

- The task is to:

1. Write a function that will insert a dictionary into a table.
2. Use the function to load the `characters` and `episodes_scenes` tables.
3. The data is in the files `flattened_characters.json` and `flattened_episodes.json`

- Implement the functions below.

```
In [61]: def insert_row_table(database_name, table_name, row_dict):
        """
        Insert a dictionary into a table.
        :param database_name: Name of the database.
        :param table_name: Name of the table.
        :param row_dict: A dictionary of column names and values.
        :return: 1 if the insert occurred and 0 otherwise.
        """

        # your code goes here
        pass

def load_table_programming(list_of_dicts, database_name, table_name):
    """
    :param list_of_dicts: List of dictionaries to insert
    :param database_name: Database name
    :param table_name: Table name
    :return: No of rows inserted
    """

    # your code goes here
    pass
```

- You can test your functions with the following cells.

```
In [64]: %sql delete from f22_hw1_got_programming.characters
%sql delete from f22_hw1_got_programming.episodes_scenes

* mysql+pymysql://root:***@localhost
0 rows affected.
* mysql+pymysql://root:***@localhost
4165 rows affected.
```

Out[64]: []

```
In [65]: df = pd.read_json('flattened_episodes.json')
episodes_list = df.to_dict('records')
load_table_programming(episodes_list, "f22_hw1_got_programming", "episodes_scenes")

df = pd.read_json('flattened_characters.json')
df = df.replace({np.nan: None})
episodes_list = df.to_dict('records')
load_table_programming(episodes_list, "f22_hw1_got_programming", "characters")
```

Out[65]: 1

```
In [68]: %sql select distinct seasonNum, episodeNum, episodeTitle, episodeAirDate from f22_hw1_got_programming

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

Out[68]:	seasonNum	episodeNum	episodeTitle	episodeAirDate
	1	1	Winter Is Coming	2011-04-17
	1	2	The Kingsroad	2011-04-24
	1	3	Lord Snow	2011-05-01
	1	4	Cripples, Bastards, and Broken Things	2011-05-08
	1	5	The Wolf and the Lion	2011-05-15
	1	6	A Golden Crown	2011-05-22
	1	7	You Win or You Die	2011-05-29
	1	8	The Pointy End	2011-06-05
	1	9	Baelor	2011-06-12
	1	10	Fire and Blood	2011-06-19
	2	1	The North Remembers	2012-04-01
	2	2	The Night Lands	2012-04-08
	2	3	What Is Dead May Never Die	2012-04-15
	2	4	Garden of Bones	2012-04-22
	2	5	The Ghost of Harrenhal	2012-04-29
	2	6	The Old Gods and the New	2012-05-06
	2	7	A Man Without Honor	2012-05-13
	2	8	The Prince of Winterfell	2012-05-20
	2	9	Blackwater	2012-05-27
	2	10	Valar Morghulis	2012-06-03
	3	1	Valar Dohaeris	2013-03-31
	3	2	Dark Wings, Dark Words	2013-04-07
	3	3	Walk of Punishment	2013-04-14
	3	4	And Now His Watch Is Ended	2013-04-21
	3	5	Kissed by Fire	2013-04-28
	3	6	The Climb	2013-05-05
	3	7	The Bear and the Maiden Fair	2013-05-12
	3	8	Second Sons	2013-05-19
	3	9	The Rains of Castamere	2013-06-02
	3	10	Mhysa	2013-06-09
	4	1	Two Swords	2014-04-06
	4	2	The Lion and the Rose	2014-04-13
	4	3	Breaker of Chains	2014-04-20
	4	4	Oathkeeper	2014-04-27
	4	5	First of His Name	2014-05-04

4	6	The Laws of Gods and Men	2014-05-11
4	7	Mockingbird	2014-05-18
4	8	The Mountain and the Viper	2014-06-01
4	9	The Watchers on the Wall	2014-06-08
4	10	The Children	2014-06-15
5	1	The Wars to Come	2015-03-29
5	2	The House of Black and White	2015-04-19
5	3	High Sparrow	2015-04-26
5	4	Sons of the Harpy	2015-05-03
5	5	Kill the Boy	2015-05-10
5	6	Unbowed, Unbent, Unbroken	2015-05-17
5	7	The Gift	2015-05-24
5	8	Hardhome	2015-05-31
5	9	The Dance of Dragons	2015-06-07
5	10	Mother's Mercy	2015-06-14
6	1	The Red Woman	2016-04-24
6	2	Home	2016-05-01
6	3	Oathbreaker	2016-05-08
6	4	Book of the Stranger	2016-05-15
6	5	The Door	2016-05-22
6	6	Blood of My Blood	2016-05-29
6	7	The Broken Man	2016-06-05
6	8	No One	2016-06-12
6	9	Battle of the Bastards	2016-06-19
6	10	The Winds of Winter	2016-06-26
7	1	Dragonstone	2017-07-16
7	2	Stormborn	2017-07-23
7	3	The Queen's Justice	2017-07-30
7	4	The Spoils of War	2017-08-06
7	5	Eastwatch	2017-08-13
7	6	Beyond the Wall	2017-08-20
7	7	The Dragon and the Wolf	2017-08-27
8	1	Winterfell	2019-04-14
8	2	A Knight of the Seven Kingdoms	2019-04-21
8	3	The Long Night	2019-04-28
8	4	The Last of the Starks	2019-05-05

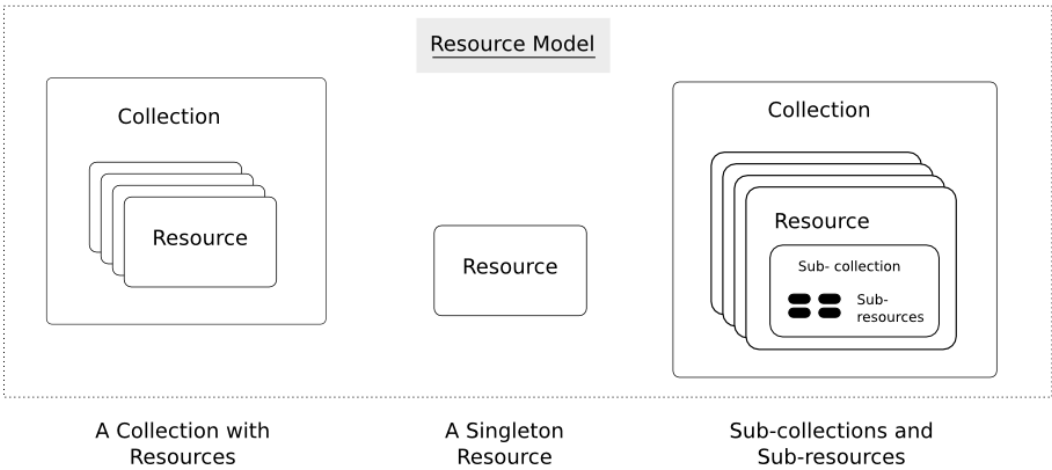
8	5	The Bells	2019-05-12
8	6	The Iron Throne	2019-05-19

```
In [70]: %sql select characterName, actorName from f22_hw1_got_programming.characters where
* mysql+pymysql://root:***@localhost
14 rows affected.
```

Out[70]:

characterName	actorName
Arya Stark	Maisie Williams
Benjen Stark	Joseph Mawle
Brandon Stark	None
Bran Stark	Isaac Hempstead Wright
Catelyn Stark	Michelle Fairley
Eddard Stark	Sean Bean
Lyanna Stark	Aisling Franciosi
Rickard Stark	None
Rickon Stark	Art Parkinson
Robb Stark	Richard Madden
Sansa Stark	Sophie Turner
Young Benjen Stark	Matteo Elezi
Young Lyanna Stark	Cordelia Hill
Young Ned Stark	Sebastian Croft

Query the Data



- REST is by definition resource oriented. A core concept is that there are resources that are collections containing other resources.
- A "path" identifies a resource. In our model/data,
 - The path `/characters` would represent all characters in the `characters` table.
 - The path `/characters/Arya Stark` would represent the character named "Arya Stark," assuming that `characterName` is the primary key for the table.
- REST and URLs also define the concept of a **query string**. The query string is similar to a `WHERE` clause in SQL.
- A `GET` on the path `/episodes_scenes?seasonNum=1&location=The Wall` is logically equivalent to:

```
select * from f22_got_hw1_programming.episodes_scenes where
seasonNum='1' and location='The Wall'
```

- A simple way to represent a query string in Python is a dictionary. In the example, the corresponding dictionary would be:

```
{
    "seasonNum": "1",
    "location": "The Wall"
}
```

- The final task is to write a function `retrieve` that we can later use to implement queries on REST collections.
- The template for the functions is:

```
In [71]: def retrieve(database_name, table_name, field_list, query_dict):
        """
        Maps a query on a resource collection to an SQL statement and returns the result.

        :param database_name: Name of the database.
        :param table_name: Name of the table.
        :param field_list: List of columns to return.
        :param query_dict: Dictionary of name, value pairs to form a where clause.
        :return: The result set as a list of dictionaries.

        Calling this function with

        retrieve(
            'f22_hw1_got_programming', 'episodes_scenes',
            ['seasonNum', 'episodeNum', 'episodeTitle', 'scene_no', 'location'],
            {
                'seasonNum': '1',
                'subLocation': 'The Wall'
```



```
        }  
    )  
  
    would map to the SQL statement  
  
    select seasonNum, episodeNum, episodeTitle, scene_no, location  
        from f22_hwl_got_programming.episodes_scenes where  
            seasonNum='1' and subLocation='The Wall'  
    ""  
  
    # Your code goes here  
    pass
```

- Write a couple of tests for your functions below.

In []: