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.

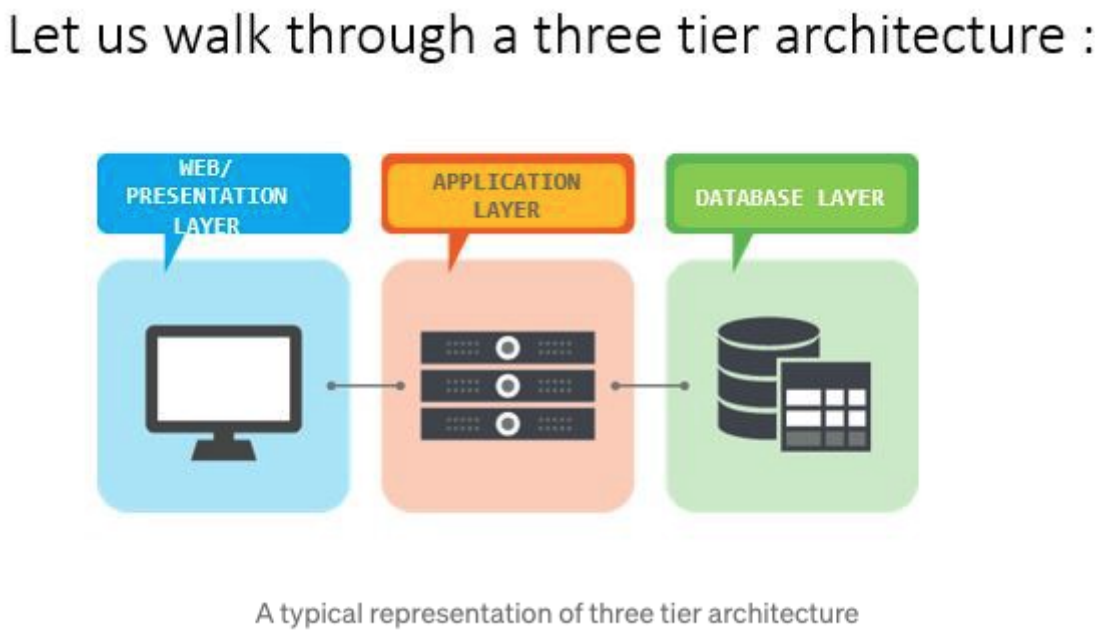
Collaboration

- **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.**

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.**



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 [140…
```
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="1168622jin",
        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 que
    result set.

    :param sql: An SQL string with "%s" please holders 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 [141…
```
sql = "select characterName, actorName from f22_hw1_got.characters where charac
```

```
res = run_query(sql, ("Arya Stark"))
res
```

Out[141]:
```
[{'characterName': 'Arya Stark', 'actorName': 'Maisie Williams'}]
```

Tasks

Task 1: Load the Data

- The following statements create a schema and some tables.

In [142…
```
%sql create database f22_hw1_got_programming
```

```
 * mysql+pymysql://root:***@localhost
(pymysql.err.ProgrammingError) (1007, "Can't create database 'f22_hw1_got_prog
ramming'; database exists")
[SQL: create database f22_hw1_got_programming]
(Background on this error at: https://sqlalche.me/e/14/f405)
```

In [143…
```
%%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[143]: []

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

In [144…

```
df = pd.read_json('flattened_characters.json')
df
```

Out[144]:

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

389 rows × 25 columns

In [145…

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

Out[145]:

```
[{'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/MV
```

```
5BNzI5MDg0ZDAtN2Y2ZC00MzU1LTgyYjQtNTBjYjEzODczZDVhXkEyXkFqcGdeQXVyNTg0Nzg4NTE
@._V1._SX100_SY140_.jpg',
  'characterImageFull': 'https://images-na.ssl-images-amazon.com/images/M/MV5
BNzI5MDg0ZDAtN2Y2ZC00MzU1LTgyYjQtNTBjYjEzODczZDVhXkEyXkFqcGdeQXVyNTg0Nzg4NTE
@._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/MV
5BMWQzOWViN2ItNDZhOS00MmZlLTkxZTYtZDg5NGUwMGRmYWZjL2ltYWdlL2ltYWdlXkEyXkFqcGd
eQXVyMjk3NTUyOTc@._V1._SX100_SY140_.jpg',
  'characterImageFull': 'https://images-na.ssl-images-amazon.com/images/M/MV5
BMWQzOWViN2ItNDZhOS00MmZlLTkxZTYtZDg5NGUwMGRmYWZjL2ltYWdlL2ltYWdlXkEyXkFqcGde
QXVyMjk3NTUyOTc@._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 [146…

```
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 of the insert occurred and 0 otherwise.
    """

    db_name = database_name[:]
    tb_name = table_name[:]
    sql = "insert into " + db_name + "." + table_name + " "

    cols = []
    slots = []
    vals = []

    for k, v in row_dict.items():
        cols.append(k)
        slots.append('%s')
        vals.append(v)

    cols_clause = "(" + ",".join(cols) + ")"
    slots_clause = "values(" + ",".join(slots) + ")"
    sql = sql + cols_clause + " " + slots_clause

    # your code goes here
    conn = get_connection()
    cursor = conn.cursor()
    real_sql = cursor.mogrify(sql, args=vals)
    res = cursor.execute(sql, args=vals)
    print("No of rows affected = ", res)

    return 0



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
    """
    db_name = database_name[:]
    tb_name = table_name[:]
    sql = "insert into " + db_name + "." + table_name + " "

    cols_dups = set()
    cols = []
    vals = []
    slots = []

    for item in list_of_dicts:
        for k,v in item.items():
            if (v is not None) and (k not in cols_dups):
                cols.append(k)
                vals.append(v)
                slots.append('%s')
```

```
                cols_dups.add(k)
            else:
                continue

    cols_clause = "(" + ",".join(cols) + ")"
    slots_clause = "values(" + ",".join(slots) + ")"
    sql = sql + cols_clause + " " + slots_clause

    # your code goes here
    conn = pymysql.connect(
        host="localhost",
        port=3306,
        user="root",
        password="1168622jin",
        autocommit=True,
        cursorclass=pymysql.cursors.DictCursor)
    cursor = conn.cursor()
    real_sql = cursor.mogrify(sql, args=vals)
    print(real_sql)
    res = cursor.execute(sql, args=vals)
    print("No of rows affected = ", res)
```

- You can test your functions with the following cells.

In [147…
```
%sql delete from f22_hw1_got_programming.characters
%sql delete from f22_hw1_got_programming.episodes_scenes
```

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

Out[147]:
```
[]
```

In [148…
```
df = pd.read_json('flattened_episodes.json')
episodes_list = df.to_dict('records')
load_table_programming(episodes_list, "f22_hw1_got_programming", "episodes_scer

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")
```

```
insert into f22_hw1_got_programming.episodes_scenes (seasonNum,episodeNum,episodeTitle,episodeLink,episodeAirDate,episodeDescription,openingSequenceLocations,sceneStart,sceneEnd,location,subLocation,characters,scene_no) values(1,1,'Winter Is Coming','/title/tt1480055/','2011-04-17','Jon Arryn, the Hand of the King, is dead. King Robert Baratheon plans to ask his oldest friend, Eddard Stark, to take Jon\'s place. Across the sea, Viserys Targaryen plans to wed his sister to a nomadic warlord in exchange for an army.','King\'s Landing;Winterfell;The Wall;Pentos','0:00:40','0:01:45','The Wall','Castle Black','Waymar Royce;Will;Gared;',0)
No of rows affected =  1
insert into f22_hw1_got_programming.characters (characterName,characterLink,actorName,actorLink,houseName,royal,parents,siblings,killedBy,characterImageThumb,characterImageFull,nickname,killed,servedBy,parentOf,marriedEngaged,serves,kingsguard,guardedBy,actors,guardianOf,allies,abductedBy,abducted,sibling) values('Addam Marbrand','/character/ch0305333/','B.J. Hogg','/name/nm0389698/','Targaryen',1.0e0,'Elia Martell;Rhaegar Targaryen','Rhaenys Targaryen;Jon Snow','Gregor Clegane','https://images-na.ssl-images-amazon.com/images/M/MV5BNzI5MDg0ZDAtN2Y2ZC00MzU1LTgyYjQtNTBjYjEzODczZDVhXkEyXkFqcGdeQXVyNTg0Nzg4NTE@._V1._SX100_SY140_.jpg','https://images-na.ssl-images-amazon.com/images/M/MV5BNzI5MDg0ZDAtN2Y2ZC00MzU1LTgyYjQtNTBjYjEzODczZDVhXkEyXkFqcGdeQXVyNTg0Nzg4NTE@._V1_.jpg','Damphair','Brandon Stark;Rickard Stark','Arthur Dayne;Jaime Lannister','Daenerys Targaryen;Rhaegar Targaryen;Viserys Targaryen','Rhaella Targaryen','Doran Martell',1.0e0,'Nymeria','{\'actorName\': \'William Wilson\', \'actorLink\': \'/name/nm8251159/\', \'seasonsActive\': [7]};{\'actorName\': \'James Wilson\', \'actorLink\': \'/name/nm8251160/\', \'seasonsActive\': [7]}','Daenerys Targaryen','Howland Reed;Robert Baratheon','Rhaegar Targaryen','Lyanna Stark','Aegon Targaryen;Jon Snow')
No of rows affected =  1
```

In [149…
```
%sql select distinct seasonNum, episodeNum, episodeTitle, episodeAirDate from f
```

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

Out[149]:

seasonNum	episodeNum	episodeTitle	episodeAirDate
1	1	Winter Is Coming	2011-04-17

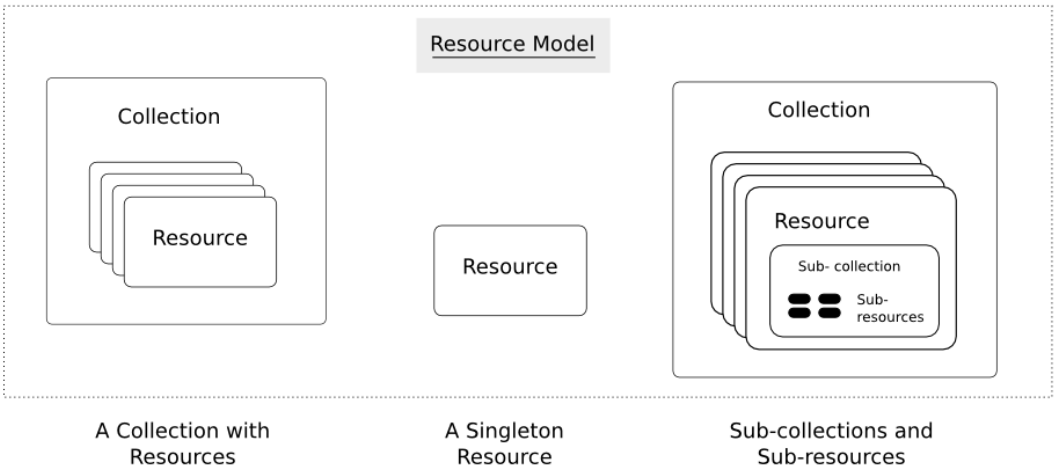
In [150…
```
%sql select characterName, actorName from f22_hw1_got_programming.characters wh
```

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

Out[150]:

characterName	actorName

Query the Data



REST Collections and Resources

- 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 "Ary 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 [151…

```
def retrieve(database_name, table_name, field_list, query_dict):
    """
    Maps a query on a resource collection to an SQL statement and returns the r

    :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_hw1_got_programming.episodes_scenes where
                seasonNum='1' and subLocation='The Wall'
    """

    # Your code goes here
    db_name = database_name[:]
    tb_name = table_name[:]
    cols_list = field_list[:]
    dict_list = query_dict

    sql = "from " + db_name + "." + table_name + " where"

    cols = []
    cols_slots = []
    vals_slots = []

    for k,v in dict_list.items():
        cols_slots.append(k)
        vals_slots.append(v)


    cols_clause = ", ".join(cols_list)
    sql = "select " + cols_clause + " " + sql + " seasonNum="+'%s' + "and subLo

    # your code goes here
    conn = pymysql.connect(
        host="localhost",
        port=3306,
        user="root",
        password="1168622jin",
        autocommit=True,
        cursorclass=pymysql.cursors.DictCursor)
```

```
    cursor = conn.cursor()

    real_sql = cursor.mogrify(sql, (vals_slots[0], vals_slots[1]))
    print(real_sql)
    res = cursor.execute(sql, (vals_slots[0], vals_slots[1]))

    print("No of rows affected = ", res)
```

- **Write a couple of tests for your functions below.**

In [152…

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

```
select seasonNum, episodeNum, episodeTitle, scene_no, location from f22_hw1_go
t_programming.episodes_scenes where seasonNum='1'and subLocation='The Wall'
No of rows affected =  0
```