ipython-sql Magic

```
In [1]: import json
```

```
In [2]: import pandas
```

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

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

```
In [5]: %sql select * from db_book.student limit 10
```

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

Out[5]:

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32
19991	Brandt	History	80
23121	Chavez	Finance	110
44553	Peltier	Physics	56
45678	Levy	Physics	46
54321	Williams	Comp. Sci.	54
55739	Sanchez	Music	38
70557	Snow	Physics	0
76543	Brown	Comp. Sci.	58

PyMySQL

```
In [6]: import pymysql
```

In [7]:
```
conn = pymysql.connect(
    host="localhost",
    user="root",
    password="dbuserbdbuser",
    cursorclass=pymysql.cursors.DictCursor,
    autocommit=True)
```

In [8]:
```
cur = conn.cursor()
res = cur.execute("select * from db_book.student limit 10")
res = cur.fetchall()
```

In [9]:
```
res
```

```
Out[9]: [{'ID': '00128',
          'name': 'Zhang',
          'dept_name': 'Comp. Sci.',
          'tot_cred': Decimal('102')},
         {'ID': '12345',
          'name': 'Shankar',
          'dept_name': 'Comp. Sci.',
          'tot_cred': Decimal('32')},
         {'ID': '19991',
          'name': 'Brandt',
          'dept_name': 'History',
          'tot_cred': Decimal('80')},
         {'ID': '23121',
          'name': 'Chavez',
          'dept_name': 'Finance',
          'tot_cred': Decimal('110')},
         {'ID': '44553',
          'name': 'Peltier',
          'dept_name': 'Physics',
          'tot_cred': Decimal('56')},
         {'ID': '45678',
          'name': 'Levy',
          'dept_name': 'Physics',
          'tot_cred': Decimal('46')},
         {'ID': '54321',
          'name': 'Williams',
          'dept_name': 'Comp. Sci.',
          'tot_cred': Decimal('54')},
         {'ID': '55739',
          'name': 'Sanchez',
          'dept_name': 'Music',
          'tot_cred': Decimal('38')},
         {'ID': '70557',
          'name': 'Snow',
          'dept_name': 'Physics',
          'tot_cred': Decimal('0')},
         {'ID': '76543',
          'name': 'Brown',
          'dept_name': 'Comp. Sci.',
          'tot_cred': Decimal('58')}]
```

Pandas and SQLAlchemy

In [16]:
```
import numpy as np
```

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

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

In [19]:
```
df = pandas.read_sql("select * from db_book.student limit 10", con=engine)
```

In [20]:
```
df
```

Out[20]:

	ID	name	dept_name	tot_cred
0	00128	Zhang	Comp. Sci.	102.0
1	12345	Shankar	Comp. Sci.	32.0
2	19991	Brandt	History	80.0
3	23121	Chavez	Finance	110.0
4	44553	Peltier	Physics	56.0
5	45678	Levy	Physics	46.0
6	54321	Williams	Comp. Sci.	54.0
7	55739	Sanchez	Music	38.0
8	70557	Snow	Physics	0.0
9	76543	Brown	Comp. Sci.	58.0

MongoDB

Note: The following cell only works for me. I use this approach to avoid putting passwords in publicly accessible documents,

In [153...
```
import sys
import pymongo


# sys.path.append(
#     "/Users/donaldferguson/Dropbox/00Spring2023/Intro_to_Databases_S23/DONOTSHARE"
# )
```

In [154…

```
# import mongo_secrets

# mongo_url = mongo_secrets.mongo_atlas_url
password = "bq2150"
url = f"mongodb+srv://bq2150:{password}@s23-w4111.ovdrkzr.mongodb.net/?retryWrites=true&w=majority"
```

In [155…

```
mongo_client = pymongo.MongoClient(url)
# db = client.test
```

In [156…

```
list(mongo_client.list_databases())
```

Out[156]:

```
[{'name': 'S23_GoT', 'sizeOnDisk': 405504, 'empty': False},
 {'name': 's23_hw4', 'sizeOnDisk': 835584, 'empty': False},
 {'name': 'sample_airbnb', 'sizeOnDisk': 55152640, 'empty': False},
 {'name': 'sample_analytics', 'sizeOnDisk': 9674752, 'empty': False},
 {'name': 'sample_geospatial', 'sizeOnDisk': 1425408, 'empty': False},
 {'name': 'sample_guides', 'sizeOnDisk': 40960, 'empty': False},
 {'name': 'sample_mflix', 'sizeOnDisk': 49238016, 'empty': False},
 {'name': 'sample_restaurants', 'sizeOnDisk': 6946816, 'empty': False},
 {'name': 'sample_supplies', 'sizeOnDisk': 1196032, 'empty': False},
 {'name': 'sample_training', 'sizeOnDisk': 52195328, 'empty': False},
 {'name': 'sample_weatherdata', 'sizeOnDisk': 2932736, 'empty': False},
 {'name': 'admin', 'sizeOnDisk': 344064, 'empty': False},
 {'name': 'local', 'sizeOnDisk': 22173671424, 'empty': False}]
```

Neo4j

Question 8: Neo4j

I scoped my query to released dates after 2008. I will accept answers that are not scoped. I am not looking for perfection and am focusing on understanding the concepts.

Note: The following cells only work for me.

In [145…

```
# import neo4j_secrets
```

In [21]:

```
# aura_url = neo4j_secrets.aura_url
# aura_user = neo4j_secrets.aura_user
# aura_pw = neo4j_secrets.aura_pw
```

In [1]:
```
import py2neo
```

- Uncomment and set the Aura information, then run the test.

In [18]:
```
from py2neo import Graph

# aura_url =
# aura_user =
# aura_pw =

aura_url = 'neo4j+s://ea59b107.databases.neo4j.io'
aura_user = 'neo4j'
aura_pw = 'IspP3KUjmy_QSIZHnQg2eh5VRvDdSpeHjGGEEK7R8CY'

def t1():
    graph = Graph(aura_url, auth=(aura_user, aura_pw))
    q = "match (r:Person) where r.name='Tom Hanks' return r"
    res = graph.run(q)

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

In [19]:
```
t1()
```

```
Node('Person', born=1956, name='Tom Hanks')
```

Relational Algebra

- Just kidding.
- I think we all have had as much fun as we can stand using relational algebra and the RelaX calculator.
- You're welcome.

Entity Relationship Modeling

Definition to Model

- The model you will diagram has four entity types:
    1. Faculty has the properties:
        - UNI
        - last_name
        - first_name
        - job_title
    2. Department has two properties:
        - department_code
        - department_name
    3. Student has the properties:
        - UNI
        - last_name
        - first_name
        - enrollment_year
    4. Section:
        - section_id
        - semester
        - year
        - credits

- The model has the following relationships:
    1. Faculty_Department:
        - A faculty may be associated with one or more departments.
        - The association has a type: member, chair, emeritus.
    2. Student_Department:
        - A student has exactly one department that is the major_department.

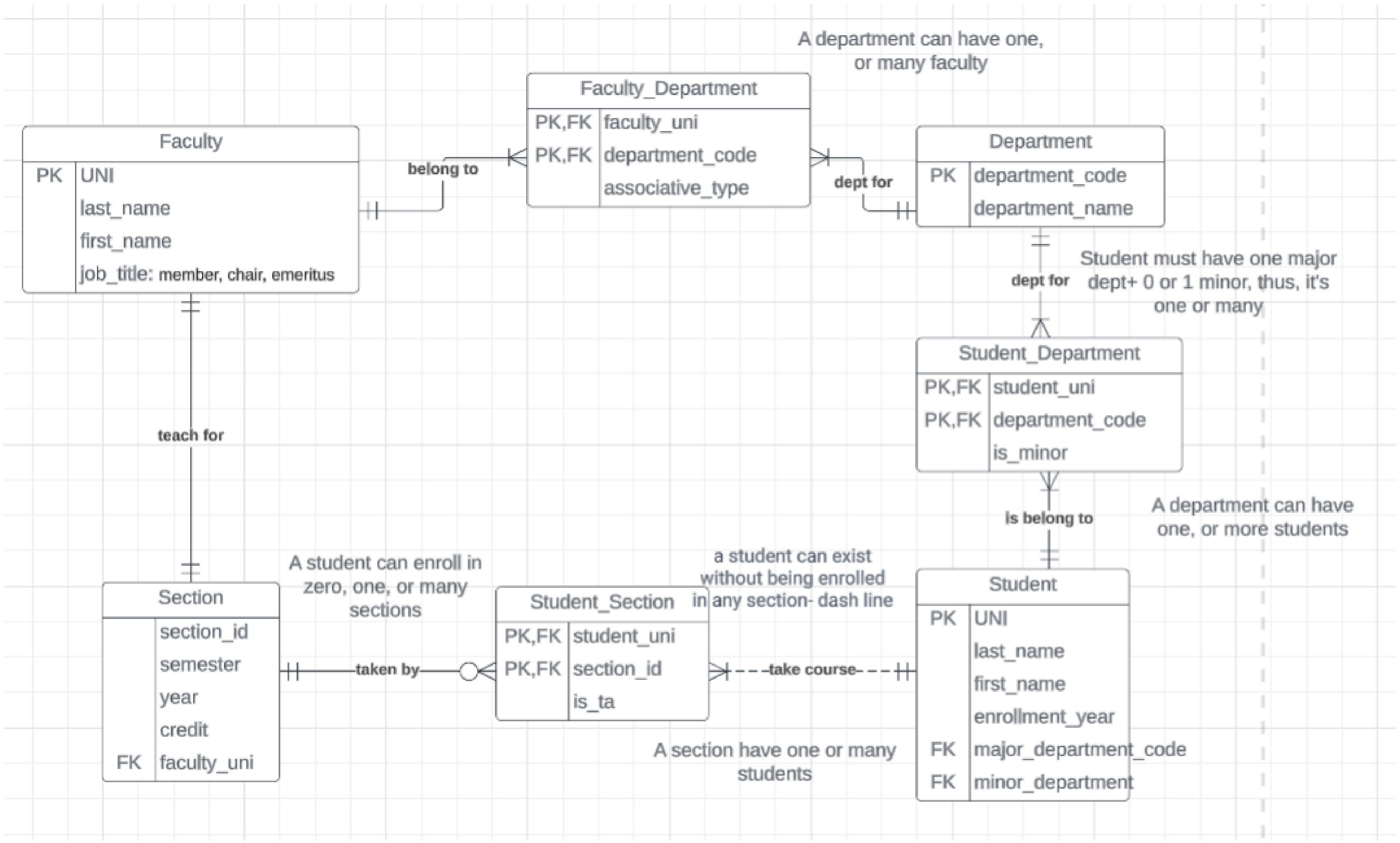
    - The student may have 0 or one minor_department.
3. Student_Section:
    - A student has a relationship to 0, 1 or many sections.
    - The student may be enrolled_in_ the section or a ta_for the section.
    - A section may have many enrolled students and many TAs.
4. Faculty_Section:
    - A faculty member teaches exactly one section per semester.
    - A section has exactly one instructor per semester.

- Use Lucidchart to draw a Crow's Foot Notation ER diagram for the logical model. You may add notes to explain any reasonable assumptions you make.

ER Diagram

```
In [20]: from IPython.display import Image
         Image(filename='atlas-2.png')
```

Out[20]:



Model to Schema

- Create a new schema `s23_final_exam`.
- Implement and execute the DDL statements to implement your ER diagram.
- The university is extremely large. So, you should define indexes that you think appropriate.

SQL DDL Statements

In [40]:
```
#create schema
%sql create schema s23_final_exam
```
```
 * mysql+pymysql://root:***@localhost
1 rows affected.
```
Out[40]:
```
[]
```

In [41]:
```
%%sql
use s23_final_exam;

drop table if exists Faculty;
create table Faculty (
  UNI VARCHAR(10) PRIMARY KEY,
  last_name VARCHAR(50) not null,
  first_name VARCHAR(50) not null,
  job_title VARCHAR(50)
);

drop table if exists Department;
create table Department (
  department_code VARCHAR(10) PRIMARY KEY,
  department_name VARCHAR(50) not null
);

drop table if exists Student;
create table  Student (
  UNI VARCHAR(10) PRIMARY KEY,
  last_name VARCHAR(50) not null,
  first_name VARCHAR(50) not null,
  enrollment_year INT not null,
  major_department_code VARCHAR(10) not null,
  minor_department_code VARCHAR(10),
  FOREIGN KEY (major_department_code) REFERENCES Department (department_code),
  FOREIGN KEY (minor_department_code) REFERENCES Department (department_code)
);

drop table if exists Section;
create table Section (
  section_id INT PRIMARY KEY,
  semester VARCHAR(20) not null,
  year INT not null,
  credits INT not null,
  faculty_UNI VARCHAR(10) not null,
  FOREIGN KEY (faculty_UNI) REFERENCES Faculty (UNI)
);
```

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

Out[41]:

```
[]
```

In [42]:

```
%%sql
use s23_final_exam;
drop table if exists Student_Section;
create table Student_Section (
  student_UNI VARCHAR(10) not null,
  section_id INT not null,
  is_ta BOOL not null,
  PRIMARY KEY (student_UNI, section_id),
  FOREIGN KEY (student_UNI) REFERENCES Student (UNI),
  FOREIGN KEY (section_id) REFERENCES Section (section_id)
);

drop table if exists Faculty_Department;
create table Faculty_Department (
  faculty_UNI VARCHAR(10) not null,
  department_code VARCHAR(10) not null,
  association_type enum('member', 'chair', 'emeritus') not null,
  PRIMARY KEY (faculty_UNI, department_code),
  FOREIGN KEY (faculty_UNI) REFERENCES Faculty (UNI),
  FOREIGN KEY (department_code) REFERENCES Department (department_code)
);

drop table if exists Student_Department;
create table Student_Department (
  student_UNI VARCHAR(10) not null,
  department_code VARCHAR(10) not null,
  is_minor BOOL not null,
  PRIMARY KEY (student_UNI, department_code),
  FOREIGN KEY (student_UNI) REFERENCES Student (UNI),
  FOREIGN KEY (department_code) REFERENCES Department (department_code)
);
```

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

Out[42]:
```
[]
```

In [43]:
```
%%sql
use s23_final_exam;
create index idx_section_instructor_UNI ON Section (faculty_UNI);
Ccreate index idx_student_section_section_id ON Student_Section (section_id);
create index idx_faculty_department_department_code ON Faculty_Department (department_code);
create index idx_student_department_department_code ON Student_Department (department_code);
```

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

Out[43]:
```
[]
```

SQL

Customer Summary

- The following is a view that is a summary of customers and orders from Classic Models.

In [107…
```
# #load in csv, just for take a look in datagrip

customer_order = pandas.read_csv("./customer_order_summary.csv")
customer_order.to_sql("customer_a", schema="s23_final_exam", con=engine,
                      index=False, if_exists="replace")
```

Out[107]:
```
326
```

In [108…
```
# create table customer_order_summary
%%sql
```

```
use classicmodels;
drop table if exists customer_order_summary;
create table customer_order_summary (
  customerName VARCHAR(68) not null,
  customerNumber INT not null,
  orderNumber INT not null,
  orderDate DATE,
  shippedDate DATE,
  orderTotal DECIMAL(10,2)
);
```

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

Out[108]:
```
[]
```

In [109…
```
%%sql
use classicmodels;
INSERT INTO customer_order_summary (customerName,customerNumber, orderNumber,
                                    orderDate, shippedDate, orderTotal)
select
  cust.customerName,
  cust.customerNumber,
  orders.orderNumber,
  orders.orderDate,
  orders.shippedDate,
  SUM(orderdetails.quantityOrdered * orderdetails.priceEach) AS orderTotal
FROM
  classicmodels.customers as cust
  JOIN classicmodels.orders ON cust.customerNumber = orders.customerNumber
  JOIN classicmodels.orderdetails ON orders.orderNumber = orderdetails.orderNumber
GROUP BY
  cust.customerNumber,
  orders.orderNumber
ORDER BY
  orders.orderNumber;
```

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

Out[109]:
```
[]
```

In [110…
```
%sql select * from classicmodels.customer_order_summary limit 20;
```

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

Out[110]:

customerName	customerNumber	orderNumber	orderDate	shippedDate	orderTotal
Online Diecast Creations Co.	363	10100	2003-01-06	2003-01-10	10223.83
Blauer See Auto, Co.	128	10101	2003-01-09	2003-01-11	10549.01
Vitachrome Inc.	181	10102	2003-01-10	2003-01-14	5494.78
Baane Mini Imports	121	10103	2003-01-29	2003-02-02	50218.95
Euro+ Shopping Channel	141	10104	2003-01-31	2003-02-01	40206.20
Danish Wholesale Imports	145	10105	2003-02-11	2003-02-12	53959.21
Rovelli Gifts	278	10106	2003-02-17	2003-02-21	52151.81
Land of Toys Inc.	131	10107	2003-02-24	2003-02-26	22292.62
Cruz & Sons Co.	385	10108	2003-03-03	2003-03-08	51001.22
Motor Mint Distributors Inc.	486	10109	2003-03-10	2003-03-11	25833.14
AV Stores, Co.	187	10110	2003-03-18	2003-03-20	48425.69
Mini Wheels Co.	129	10111	2003-03-25	2003-03-30	16537.85
Volvo Model Replicas, Co	144	10112	2003-03-24	2003-03-29	7674.94
Mini Gifts Distributors Ltd.	124	10113	2003-03-26	2003-03-27	11044.30
La Corne D'abondance, Co.	172	10114	2003-04-01	2003-04-02	33383.14
Classic Legends Inc.	424	10115	2003-04-04	2003-04-07	21665.98
Royale Belge	381	10116	2003-04-11	2003-04-13	1627.56
Dragon Souveniers, Ltd.	148	10117	2003-04-16	2003-04-17	44380.15
Enaco Distributors	216	10118	2003-04-21	2003-04-26	3101.40
Salzburg Collectables	382	10119	2003-04-28	2003-05-02	35826.33

- There is a CSV file in the final exam zipfile that contains the data.
- `orderTotal` is the sum of `quantityOrdered*priceEach` over all `orderdetails` in the order.

Task 1

- Create a view that produces the information. Put your SQL below.

In [111…

```
# I put the code to create table customer_order_summary in the above, please take a look
# only create view here
%%sql
create or replace view customers_summary as
 select
 *
 from
 classicmodels.customer_order_summary;
```

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

Out[111]:

```
[]
```

In [112…

```
%sql select * from customers_summary limit 20;
```

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

Out[112]:

customerName	customerNumber	orderNumber	orderDate	shippedDate	orderTotal
Online Diecast Creations Co.	363	10100	2003-01-06	2003-01-10	10223.83
Blauer See Auto, Co.	128	10101	2003-01-09	2003-01-11	10549.01
Vitachrome Inc.	181	10102	2003-01-10	2003-01-14	5494.78
Baane Mini Imports	121	10103	2003-01-29	2003-02-02	50218.95
Euro+ Shopping Channel	141	10104	2003-01-31	2003-02-01	40206.20
Danish Wholesale Imports	145	10105	2003-02-11	2003-02-12	53959.21
Rovelli Gifts	278	10106	2003-02-17	2003-02-21	52151.81
Land of Toys Inc.	131	10107	2003-02-24	2003-02-26	22292.62
Cruz & Sons Co.	385	10108	2003-03-03	2003-03-08	51001.22
Motor Mint Distributors Inc.	486	10109	2003-03-10	2003-03-11	25833.14
AV Stores, Co.	187	10110	2003-03-18	2003-03-20	48425.69
Mini Wheels Co.	129	10111	2003-03-25	2003-03-30	16537.85
Volvo Model Replicas, Co	144	10112	2003-03-24	2003-03-29	7674.94
Mini Gifts Distributors Ltd.	124	10113	2003-03-26	2003-03-27	11044.30
La Corne D'abondance, Co.	172	10114	2003-04-01	2003-04-02	33383.14
Classic Legends Inc.	424	10115	2003-04-04	2003-04-07	21665.98
Royale Belge	381	10116	2003-04-11	2003-04-13	1627.56
Dragon Souveniers, Ltd.	148	10117	2003-04-16	2003-04-17	44380.15
Enaco Distributors	216	10118	2003-04-21	2003-04-26	3101.40
Salzburg Collectables	382	10119	2003-04-28	2003-05-02	35826.33

Task 2

- Manually logically create a materialized view `customer_order_copy` by creating a table that is a copy of the data in the view.

- You must add some SQL to your model that automatically updates the materialized view/copy table whenever the `orderdetails` table has a new row inserted.
- Enter and test your DDL below.

```
In [113…  %%sql #create materialized view
          drop table if exists customer_order_copy;
          create table customer_order_copy
          select * from customers_summary;

           * mysql+pymysql://root:***@localhost
          0 rows affected.
          326 rows affected.
Out[113]: []
```

```
In [114…  %%sql

drop trigger if exists update_customer_order_copy;

create trigger update_customer_order_copy
    after insert ON orderdetails
    for each row
begin
    if new.orderNumber is not null THEN
        update customer_order_copy
        set orderTotal = (
            SELECT SUM(quantityOrdered*priceEach)
            from orderdetails
            WHERE orderNumber = new.orderNumber
        )
        where orderNumber = new.orderNumber;
    else
        insert into customer_order_copy (
            customerName, customerNumber, orderNumber, orderDate, shippedDate, orderTotal
        )
        select cust.customerName, orders.customerNumber, orders.orderNumber,
        orders.orderDate, orders.shippedDate, (
            SELECT SUM(od.quantityOrdered*od.priceEach)
            from orderdetails as od
            WHERE od.orderNumber = orders.orderNumber
        )
        from customers as cust
        JOIN orders ON cust.customerNumber = orders.customerNumber
        where orders.orderNumber = new.orderNumber;
    end if;
end;
```

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

Out[114]:
```
[]
```

In [122…
```
%%sql #test
use classicmodels;
insert into orderdetails (orderNumber, productCode, quantityOrdered, priceEach, orderLineNumber)
values (10100,'S10_1678', 1, 20, 1);
```

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

Out[122]:
```
[]
```

In [123…
```
%%sql
select * from customer_order_copy where orderNumber=10100;
#we can notice from previous view, we have 10223.83. Add 20, it becomes 10243.83
```

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

Out[123]:

customerName	customerNumber	orderNumber	orderDate	shippedDate	orderTotal
Online Diecast Creations Co.	363	10100	2003-01-06	2003-01-10	10243.83

Task 3

- Only certain people should be able to see order details.
- Create a new user in your database `general_user`. Configure security so that `general_user` can only query (read) `customer_order_copy` and perform no other operations.
- Put your DDL below.

In [21]:
```
%%sql
DROP USER 'general_user'@'%';
```

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

Out[21]:
```
[]
```

In [22]:
```
%%sql

/*
    '%' means can log in from any host.
*/

create user 'general_user'@'%'
    identified by 'dbuserdbuser';
```

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

Out[22]:
```
[]
```

In [25]:
```
%%sql
GRANT select ON classicmodels.customer_order_copy TO 'general_user';
```

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

Out[25]:
```
[]
```

Identifying Traitors

- You use the Lahman's Baseball Database for this problem.
- You need the `People` table and the `Appearances` table.
- A perfidious, modern day traitor is:
  - A player who appeared/played for the Red Sox before ever playing for the Yankees.
  - Subsequently (later) appeared/played for the Yankees.
  - Played/appeared in their first game for any team after the year 2000.
- Write a query that produces the perfidious, modern day traitors. Your table should have the following columns:
  - `playerID`
  - `nameFirst`

        - `nameLast`
        - `firstRedSoxGameYear` is the year of the first appearance for the Red Sox.
        - `firstYankeeGameYear` is the year of the first appearance for the Yankees.

- Order the result by `nameLast, nameFirst`.

- The `teamID` for the Yankees is `NYA` and the `teamID` for the Red Sox is `BOS`.

- Put your SQL below.

In [124…

```
# #load in csv

app = pandas.read_csv("./Appearances.csv")
app.to_sql("appearances", schema="s23_final_exam", con=engine,
                      index=False, if_exists="replace")
peo = pandas.read_csv("./People.csv")
peo.to_sql("people", schema="s23_final_exam", con=engine,
                      index=False, if_exists="replace")
```

Out[124]:

```
20370
```

In [125…

```
%%sql
use s23_final_exam;
select people.playerID, people.nameFirst, people.nameLast,
    min(IF(appearances.teamID='BOS', appearances.yearID, NULL)) as firstRedSoxGameYear,
    min(IF(appearances.teamID='NYA', appearances.yearID, NULL)) as firstYankeeGameYear
    from people left join appearances ON people.playerID = appearances.playerID
    group by people.playerID, people.nameFirst, people.nameLast
    having
        (firstRedSoxGameYear>2000 or firstYankeeGameYear>2000)
        and firstRedSoxGameYear is not null
        and firstYankeeGameYear is not null
        and firstRedSoxGameYear<firstYankeeGameYear
    order by people.nameLast, people.nameFirst;
```

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

Out[125]:

playerID	nameFirst	nameLast	firstRedSoxGameYear	firstYankeeGameYear
aardsda01	David	Aardsma	2008	2012
bailean01	Andrew	Bailey	2012	2015
bellhma01	Mark	Bellhorn	2004	2005
braggda01	Darren	Bragg	1996	2001
cashke01	Kevin	Cash	2007	2009
clarkto02	Tony	Clark	2002	2004
colemmi01	Michael	Coleman	1997	2001
colonba01	Bartolo	Colon	2008	2011
damonjo01	Johnny	Damon	2002	2006
drewst01	Stephen	Drew	2013	2014
ellsbja01	Jacoby	Ellsbury	2007	2014
embreal01	Alan	Embree	2002	2005
flahejo01	John	Flaherty	1992	2003
gordoto01	Tom	Gordon	1996	2004
hammoch01	Chris	Hammond	1997	2003
hillri01	Rich	Hill	2010	2014
hinsker01	Eric	Hinske	2006	2009
lamarry01	Ryan	LaMarre	2016	2021
layneto01	Tommy	Layne	2014	2016
lillibr01	Brent	Lillibridge	2012	2013
lowede01	Derek	Lowe	1997	2012
mcdonda02	Darnell	McDonald	2010	2012
mientdo01	Doug	Mientkiewicz	2004	2007
millean01	Andrew	Miller	2011	2015
molingu01	Gustavo	Molina	2010	2011

myersmi01	Mike	Myers	2004	2006
pridecu01	Curtis	Pride	1997	2003
quantpa01	Paul	Quantrill	1992	2004
thornma01	Matt	Thornton	2013	2014
youklke01	Kevin	Youkilis	2004	2013

MongoDB

- Use the `episodes` collection you have previously loaded into MongoDB Atlas.
- An episode has an array `openingSequenceLocations`.
- Write an aggregation that produces a Pandas data frame of the form:
  - `openingSequenceLocation`
  - `numberOfEpisodes` is the number of episodes that have the location in the opening sequence.
  - `firstAirDate` is the air date of the first episode in which the location appears in the opening.
  - `lastAirDate` is the air date of the last episode in which the location appears.
- The zipfile for the final exam contains a CSV file with the result of the aggregation. You must sort your result by `numOfEpisodes.`
- The data is the following. **Do not worry about the leading index column.**

In [157…
```
opening_sequences_df = pandas.read_csv("opening_sequence_info.csv")
```

In [158…
```
opening_sequences_df
```

Out[158]:

	location	numOfEpisodes	firstAirDate	lastAirDate
0	Winterfell	73	4/17/11	5/19/19
1	King's Landing	73	4/17/11	5/19/19
2	The Wall	67	4/17/11	8/27/17
3	Meereen	30	4/6/14	6/26/16
4	Braavos	21	5/11/14	6/19/16
5	Dragonstone	19	4/1/12	8/27/17
6	Harrenhal	15	4/22/12	5/19/13
7	Pyke	15	4/8/12	8/6/17
8	Vaes Dothrak	15	4/24/11	5/29/16
9	Dorne	9	5/3/15	6/26/16
10	Riverrun	9	4/14/13	6/19/16
11	Dreadfort	7	4/6/14	5/18/14
12	Oldtown	7	7/16/17	8/27/17
13	The Eyrie	7	5/15/11	5/15/16
14	Qarth	7	4/22/12	6/3/12
15	Last Hearth	6	4/14/19	5/19/19
16	Yunkai	6	4/28/13	6/9/13
17	The Twins	5	6/12/11	7/16/17
18	Moat Cailin	4	6/1/14	4/26/15
19	Astapor	4	3/31/13	4/21/13
20	Eastwatch	3	8/13/17	8/27/17
21	Pentos	2	4/17/11	3/29/15

```
In [ ]:
```

- Write an execute your aggregation below.

In [159…

```
password = "bq2150"
url = f"mongodb+srv://bq2150:{password}@s23-w4111.ovdrkzr.mongodb.net/?retryWrites=true&w=majority"
```

In [160…

```
import pymongo
```

In [161…

```
#test it
client = pymongo.MongoClient(
    url
)
db = client.test
```

In [162…

```
mongo_client = pymongo.MongoClient(url)
result = mongo_client['s23_hw4']['episodes'].aggregate(
[
    {
        '$unwind': {
            'path': '$openingSequenceLocations',
            'includeArrayIndex': 'openinglocation',
            'preserveNullAndEmptyArrays': False
        }
    }, {
        '$project': {
            'openingSequenceLocations': 1,
            'episodeNum': 1,
            'episodeAirDate': 1
        }
    }, {
        '$group': {
            '_id': {
                'location': '$openingSequenceLocations'
            },
            'count': {
                '$count': {}
            },
            'sum': {
                '$sum': '$episodeNum'
            },
            'min': {
                '$min': '$episodeAirDate'
            },
            'max': {
                '$max': '$episodeAirDate'
            }
        }
    }, {
        '$project': {
            'location': '$_id.location',
            'numOfEpisodes': '$count',
            'firstAirDate': '$min',
            'lastAirDate': '$max',
            '_id': 0
        }
    },
    {'$sort': {'numOfEpisodes': -1}}
]
)
```

In [164…

```
characters_location = pandas.DataFrame(result)
```

In [165…

```
characters_location
```

Out[165]:

	location	numOfEpisodes	firstAirDate	lastAirDate
0	Winterfell	73	2011-04-17	2019-05-19
1	King's Landing	73	2011-04-17	2019-05-19
2	The Wall	67	2011-04-17	2017-08-27
3	Meereen	30	2014-04-06	2016-06-26
4	Braavos	21	2014-05-11	2016-06-19
5	Dragonstone	19	2012-04-01	2017-08-27
6	Harrenhal	15	2012-04-22	2013-05-19
7	Vaes Dothrak	15	2011-04-24	2016-05-29
8	Pyke	15	2012-04-08	2017-08-06
9	Dorne	9	2015-05-03	2016-06-26
10	Riverrun	9	2013-04-14	2016-06-19
11	Oldtown	7	2017-07-16	2017-08-27
12	Qarth	7	2012-04-22	2012-06-03
13	Dreadfort	7	2014-04-06	2014-05-18
14	The Eyrie	7	2011-05-15	2016-05-15
15	Last Hearth	6	2019-04-14	2019-05-19
16	Yunkai	6	2013-04-28	2013-06-09
17	The Twins	5	2011-06-12	2017-07-16
18	Astapor	4	2013-03-31	2013-04-21
19	Moat Cailin	4	2014-06-01	2015-04-26
20	Eastwatch	3	2017-08-13	2017-08-27
21	Pentos	2	2011-04-17	2015-03-29

Neo4j

- Use the sample movie data for these questions.

- Write a Cyper query that returns a table with the following information:
  - director
  - movie_title
  - actor
  - movie_released_year

- The zipfile for the final exam contains a CSV file with a sample result. **Do not worry about the leading index column.**

In [128…

```
neo4j_result = pandas.read_csv('neo4j_result.csv')
neo4j_result
```

Out[128]:

	director	movie_title	actor	movie_release_year
0	Lana Wachowski	Speed Racer	Emile Hirsch	2008
1	Lana Wachowski	Speed Racer	Rain	2008
2	Lana Wachowski	Speed Racer	Christina Ricci	2008
3	Lana Wachowski	Speed Racer	Ben Miles	2008
4	Lana Wachowski	Speed Racer	Susan Sarandon	2008
5	Lana Wachowski	Speed Racer	John Goodman	2008
6	Lana Wachowski	Speed Racer	Matthew Fox	2008
7	Lilly Wachowski	Speed Racer	Emile Hirsch	2008
8	Lilly Wachowski	Speed Racer	Rain	2008
9	Lilly Wachowski	Speed Racer	Christina Ricci	2008
10	Lilly Wachowski	Speed Racer	Ben Miles	2008
11	Lilly Wachowski	Speed Racer	Susan Sarandon	2008
12	Lilly Wachowski	Speed Racer	John Goodman	2008
13	Lilly Wachowski	Speed Racer	Matthew Fox	2008
14	Ron Howard	Frost/Nixon	Sam Rockwell	2008
15	Ron Howard	Frost/Nixon	Michael Sheen	2008
16	Ron Howard	Frost/Nixon	Frank Langella	2008
17	Ron Howard	Frost/Nixon	Oliver Platt	2008
18	Ron Howard	Frost/Nixon	Kevin Bacon	2008
19	James Marshall	Ninja Assassin	Rain	2009
20	James Marshall	Ninja Assassin	Ben Miles	2009
21	James Marshall	Ninja Assassin	Rick Yune	2009
22	James Marshall	Ninja Assassin	Naomie Harris	2009
23	Tom Tykwer	Cloud Atlas	Tom Hanks	2012
24	Tom Tykwer	Cloud Atlas	Jim Broadbent	2012

	director	movie_title	actor	movie_release_year
25	Tom Tykwer	Cloud Atlas	Halle Berry	2012
26	Tom Tykwer	Cloud Atlas	Hugo Weaving	2012
27	Lana Wachowski	Cloud Atlas	Tom Hanks	2012
28	Lana Wachowski	Cloud Atlas	Jim Broadbent	2012
29	Lana Wachowski	Cloud Atlas	Halle Berry	2012
30	Lana Wachowski	Cloud Atlas	Hugo Weaving	2012
31	Lilly Wachowski	Cloud Atlas	Tom Hanks	2012
32	Lilly Wachowski	Cloud Atlas	Jim Broadbent	2012
33	Lilly Wachowski	Cloud Atlas	Halle Berry	2012
34	Lilly Wachowski	Cloud Atlas	Hugo Weaving	2012

- Execute your Neo4j query below.

In [4]:
```
%pip install py2neo
```

```
Requirement already satisfied: py2neo in c:\users\11139\anaconda3\lib\site-packages (2021.2.3)
Requirement already satisfied: pygments>=2.0.0 in c:\users\11139\anaconda3\lib\site-packages (from py2neo) (2.11.2)
Requirement already satisfied: pansi>=2020.7.3 in c:\users\11139\anaconda3\lib\site-packages (from py2neo) (2020.7.3)
Requirement already satisfied: certifi in c:\users\11139\anaconda3\lib\site-packages (from py2neo) (2021.10.8)
Requirement already satisfied: six>=1.15.0 in c:\users\11139\anaconda3\lib\site-packages (from py2neo) (1.16.0)
Requirement already satisfied: urllib3 in c:\users\11139\anaconda3\lib\site-packages (from py2neo) (1.26.9)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: packaging in c:\users\11139\anaconda3\lib\site-packages (from py2neo) (21.3)
Requirement already satisfied: interchange~=2021.0.4 in c:\users\11139\anaconda3\lib\site-packages (from py2neo) (2021.
0.4)
Requirement already satisfied: monotonic in c:\users\11139\anaconda3\lib\site-packages (from py2neo) (1.6)
Requirement already satisfied: pytz in c:\users\11139\anaconda3\lib\site-packages (from interchange~=2021.0.4->py2neo)
(2021.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\11139\anaconda3\lib\site-packages (from packaging->
py2neo) (3.0.4)
```

In [ ]:
```
#
# This is similar to setting up the MongoDB client or pymysql client.
#
# I keep my passwords etc. in a file because the notebooks are public on github.
```

```
#
# When you create you database, Neo4j generates a password. There is an option to download
# the information into a text file. You can past the information below. This info is from a
# prior version of the database.
#
#
# Wait 60 seconds before connecting using these details, or login to https://console.neo4j.io to validate the Aura Inst
# NEO4J_URI=neo4j+s://5c70a2c4.databases.neo4j.io
# NEO4J_USERNAME=neo4j
# NEO4J_PASSWORD=IwJpGansNq1EqLoeMjJMwbbuSumOGOUu8C7XqNiR-9g
# AURA_INSTANCEID=5c70a2c4
# AURA_INSTANCENAME=Instance01
```

```
In [9]: # I am going to use the real information
        #
        #
        NEO4J_URI = 'neo4j+s://ea59b107.databases.neo4j.io'
        NEO4J_USERNAME = 'neo4j'
        NEO4J_PASSWORD = 'IspP3KUjmy_QSIZHnQg2eh5VRvDdSpeHjGGEEK7R8CY'
```

```
In [10]: from py2neo import Graph

         my_graph =  Graph(NEO4J_URI, auth=(NEO4J_USERNAME, NEO4J_PASSWORD))
```

```
In [11]: cypher_query = """
         MATCH (d:Person)-[:DIRECTED]->(m:Movie)<-[:ACTED_IN]-(a:Person)

         WHERE m.released >= 2008

         RETURN d.name AS director, m.title AS movie_title, a.name AS actor, m.released AS movie_released_year

         """
```

```
In [15]: result = my_graph.run(cypher_query)
         df = pandas.DataFrame([dict(i) for i in result])
         df
```

Out[15]:

	director	movie_title	actor	movie_released_year
0	Lana Wachowski	Speed Racer	Emile Hirsch	2008
1	Lana Wachowski	Speed Racer	Rain	2008
2	Lana Wachowski	Speed Racer	Christina Ricci	2008
3	Lana Wachowski	Speed Racer	Ben Miles	2008
4	Lana Wachowski	Speed Racer	Susan Sarandon	2008
5	Lana Wachowski	Speed Racer	John Goodman	2008
6	Lana Wachowski	Speed Racer	Matthew Fox	2008
7	Lilly Wachowski	Speed Racer	Emile Hirsch	2008
8	Lilly Wachowski	Speed Racer	Rain	2008
9	Lilly Wachowski	Speed Racer	Christina Ricci	2008
10	Lilly Wachowski	Speed Racer	Ben Miles	2008
11	Lilly Wachowski	Speed Racer	Susan Sarandon	2008
12	Lilly Wachowski	Speed Racer	John Goodman	2008
13	Lilly Wachowski	Speed Racer	Matthew Fox	2008
14	Ron Howard	Frost/Nixon	Sam Rockwell	2008
15	Ron Howard	Frost/Nixon	Michael Sheen	2008
16	Ron Howard	Frost/Nixon	Frank Langella	2008
17	Ron Howard	Frost/Nixon	Oliver Platt	2008
18	Ron Howard	Frost/Nixon	Kevin Bacon	2008
19	James Marshall	Ninja Assassin	Rain	2009
20	James Marshall	Ninja Assassin	Ben Miles	2009
21	James Marshall	Ninja Assassin	Rick Yune	2009
22	James Marshall	Ninja Assassin	Naomie Harris	2009
23	Tom Tykwer	Cloud Atlas	Tom Hanks	2012
24	Tom Tykwer	Cloud Atlas	Jim Broadbent	2012

	director	movie_title	actor	movie_released_year
25	Tom Tykwer	Cloud Atlas	Halle Berry	2012
26	Tom Tykwer	Cloud Atlas	Hugo Weaving	2012
27	Lana Wachowski	Cloud Atlas	Tom Hanks	2012
28	Lana Wachowski	Cloud Atlas	Jim Broadbent	2012
29	Lana Wachowski	Cloud Atlas	Halle Berry	2012
30	Lana Wachowski	Cloud Atlas	Hugo Weaving	2012
31	Lilly Wachowski	Cloud Atlas	Tom Hanks	2012
32	Lilly Wachowski	Cloud Atlas	Jim Broadbent	2012
33	Lilly Wachowski	Cloud Atlas	Halle Berry	2012
34	Lilly Wachowski	Cloud Atlas	Hugo Weaving	2012