COMS W4111: Introduction to Databases Section 002/V02, Spring, 2022

HW 1 Notebook

Introduction

This notebook has three top level sections:

1. *Setup* tests the environment setup, and should work assuming you completed HW0.
2. *Common Tasks* are the HW1 tasks for both the programming and non-programming track. All students complete this section.
3. *Non-Programing Track* contains the tasks that students in the non-programming track must complete.
4. *Programming Track* contains the tasks that students in the programming track must complete.

Submission format:

- All students (both tracks) submit a completed version of this notebook. Students need to complete the setup section, the common section, and the section specific to their track. The submission format is a PDF generated from the notebook. Students can generate the PDF by:
  - Choosing `File->Print Preview` in the notebook's menu bar. This will open a new browser tab.
  - In the new browser tab, select `File->Print` and choose to save as PDF.
  - **Make sure that everything renders properly in the generated PDF.** Troubleshoot/reach out if you have issues. Images/outputs that render incorrectly will not be graded.

- All students submit a zip file containing their cloned HW0/1 project, which they got by cloning the GitHub repository. Students can:
  - Open a command/terminal window in the root directory where they cloned the project.
  - Enter `git pull` to retrieve any updates to the project, including required data files.

- Students can edit the notebook using Anaconda Navigator to open Jupyter Notebook.

- Students on the programming track also create and modify Python files in the sub-folder `<UNI>_web_src` . Remember, you should be using a folder with your UNI. In my case, the folder would be `dff9_web_src.`

- The zip file you submit should contain **only** the following sub-folders/files:
  - `<UNI>_src.` (All students) This folder must container your version of this notebook.
  - `<UNI>_web_src.` (Only programming track)

    - To be clear: the zipped directory for non-programming track submissions should contain **one** file. The corresponding `zip` for the programming track should contain **two** files.
- Make sure to submit your notebook in the PDF format separately from the zip file, based on your track as well. That is, you need to make **two** submissions in total like below:
    - Submit your notebook file in PDF format to `Homework 1: Non-programming or Programming` **(Make sure that you assigned pages properly).**
    - Submit your zip file to `Homework 1: Zip File Submission`

Setup

Note: You will have to put the correct user ID and password in the connection strings below, e.g. replace `dbuser and dbuserdbuser.`

iPython-SQL

In [2]:
```
%load_ext sql
```

```
https://www.columbia.edu/
```

In [3]:
```
%sql mysql+pymysql://root:Edy990127@localhost
```

Out[3]:
```
'Connected: root@None'
```

In [4]:
```
%sql select * from db_book.student where name like "z%" or name like "sh%"
```

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

Out[4]:

ID	name	dept_name	tot_cred
00128	Zhang	Comp. Sci.	102
12345	Shankar	Comp. Sci.	32

PyMySQL

In [5]:
```
import pymysql
```

In [6]:
```
conn = pymysql.connect(host="localhost", user="root", password="Edy990127")
```

In [7]:
```
conn
```

Out[7]:
```
<pymysql.connections.Connection at 0x1f0b74644f0>
```

In [8]:

```
sql = """
    select * from db_book.student where
        name like %s or name like %s
"""
```

In [9]:
```
pattern_1 = "z%"
pattern_2 = "sh%"
```

In [10]:
```
cur = conn.cursor()
res = cur.execute(
    sql, args=(pattern_1, pattern_2)
)
res
```

Out[10]:
```
2
```

In [11]:
```
res = cur.fetchall()
```

In [ ]:
```

```

In [12]:
```
res
```

Out[12]:
```
(('00128', 'Zhang', 'Comp. Sci.', Decimal('102')),
 ('12345', 'Shankar', 'Comp. Sci.', Decimal('32')))
```

Pandas

In [13]:
```
import pandas as pd
```

In [14]:
```
#
#  Replace the path below with the path of your project directory.
#  Use // instead of / if you're on Windows.
#
project_root = "E:\Github\spring-2022-COMS4111\S22-W4111-HW-1-0"
```

In [15]:
```
people_df = pd.read_csv(project_root + "/data/People.csv")
```

In [16]:
```
people_df
```

Out[16]:

	playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear	dea
0	aardsda01	1981.0	12.0	27.0	USA	CO	Denver	NaN	
1	aaronha01	1934.0	2.0	5.0	USA	AL	Mobile	2021.0	

	playerID	birthYear	birthMonth	birthDay	birthCountry	birthState	birthCity	deathYear	dea
2	aaronto01	1939.0	8.0	5.0	USA	AL	Mobile	1984.0	
3	aasedo01	1954.0	9.0	8.0	USA	CA	Orange	NaN	
4	abadan01	1972.0	8.0	25.0	USA	FL	Palm Beach	NaN	
...	...	...	...	...	...	...	...	...	
20353	zupofr01	1939.0	8.0	29.0	USA	CA	San Francisco	2005.0	
20354	zuvelpa01	1958.0	10.0	31.0	USA	CA	San Mateo	NaN	
20355	zuverge01	1924.0	8.0	20.0	USA	MI	Holland	2014.0	
20356	zwilldu01	1888.0	11.0	2.0	USA	MO	St. Louis	1978.0	
20357	zychto01	1990.0	8.0	7.0	USA	IL	Monee	NaN	

20358 rows × 24 columns

In [17]:
```
people_df.loc[
    (people_df['nameLast'] == "Williams") & (people_df['birthCity'] == 'San Diego'),
    ["playerID", "nameLast", "nameFirst", "birthYear", 'birthCity', 'bats', 'throws']
]
```

Out[17]:

	playerID	nameLast	nameFirst	birthYear	birthCity	bats	throws
19773	willite01	Williams	Ted	1918.0	San Diego	L	R
19776	willitr01	Williams	Trevor	1992.0	San Diego	R	R

SQLAlchemy

In [18]:
```
from sqlalchemy import create_engine
```

In [19]:
```
engine = create_engine("mysql+pymysql://root:Edy990127@localhost")
```

In [20]:
```
sql = """
    select * from db_book.student where
        name like %s or name like %s
"""
pattern_1 = "z%"
pattern_2 = "sh%"
```

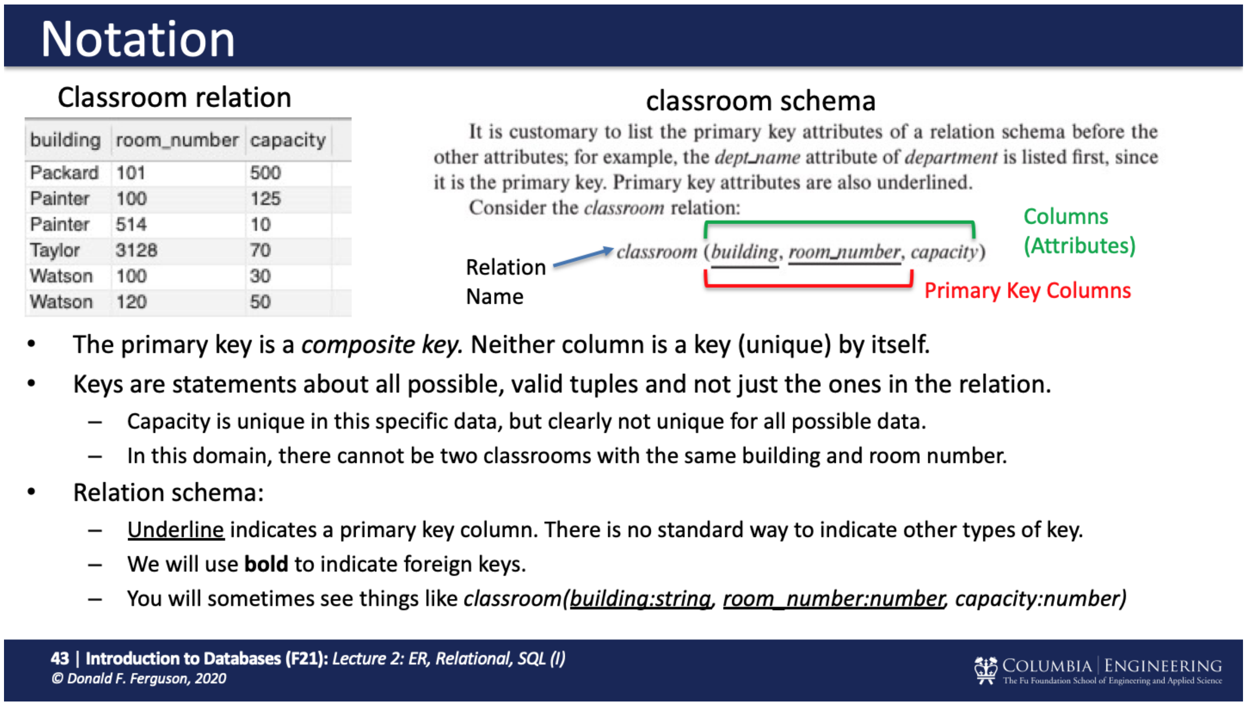
In [21]:
```python
another_df = pd.read_sql(sql, params=(pattern_1, pattern_2), con=engine)
another_df
```

Out[21]:
```
      ID     name   dept_name  tot_cred
0  00128    Zhang  Comp. Sci.     102.0
1  12345  Shankar  Comp. Sci.      32.0
```

Common Tasks

Schema and Data Modeling

- There are three entity types:
  1. Employee with attributes:
     - employee_no
     - last_name
     - first_name
  2. Department with attributes
     - department_id
     - department_name
  3. Applicant with attributes:
     - email
     - last_name
     - first_name



Notation

Classroom relation

building	room_number	capacity
Packard	101	500
Painter	100	125
Painter	514	10
Taylor	3128	70
Watson	100	30
Watson	120	50

classroom schema

It is customary to list the primary key attributes of a relation schema before the other attributes; for example, the *dept_name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined.

Consider the *classroom* relation:

classroom (<u>*building*</u>, <u>*room_number*</u>, *capacity*)

- The primary key is a *composite key.* Neither column is a key (unique) by itself.
- Keys are statements about all possible, valid tuples and not just the ones in the relation.
  - Capacity is unique in this specific data, but clearly not unique for all possible data.
  - In this domain, there cannot be two classrooms with the same building and room number.
- Relation schema:
  - <u>Underline</u> indicates a primary key column. There is no standard way to indicate other types of key.
  - We will use **bold** to indicate foreign keys.
  - You will sometimes see things like *classroom(<u>building:string</u>, <u>room_number:number</u>, capacity:number)*

Relational Schema

- Using the notation from the textbook slides and lecture notes, define the relation definitions for each of the entity types. That is, the schema definition for the relations. You will need to choose a primary key.
- The snippet below shows how to use under-bar.

$$This\ is\ a\ sentence\ with\ someting\_in\_parentheses(\underline{something}, another\_thing)\ and\ som$$

You can double click on the cell above to see the source, which is

```
\begin{equation}
This\ is\ a\ sentence\ with\ someting\_in\_parentheses(
    \underline{something}, another\_thing)\  and\ something\ with\
underbar.
\end{equation}
```

Put your relation definitions below between the horizontal lines.

<hr style="height: 1px";>

$$Employee(\underline{employee\_no}, last\_name, first\_name) \tag{2}$$

$$Department(\underline{department\_id}, department\_name) \tag{3}$$

$$Applicant(\underline{applicant\_id}, email, last\_name, first\_name) \tag{4}$$

<hr style="height: 1px";>

ER Modeling

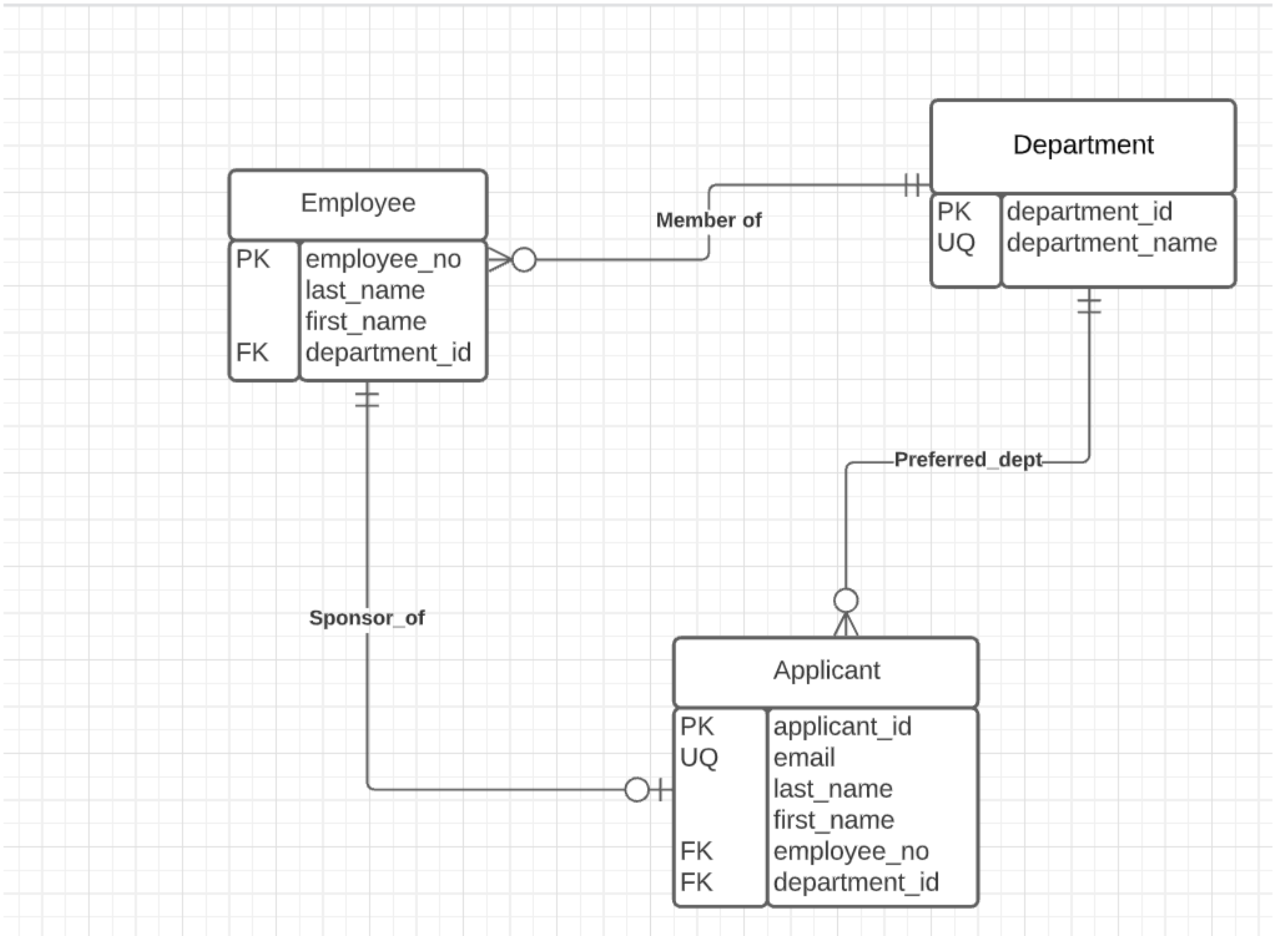
- Continuing the example above:
  - An *employee* is a _member_of_ exactly one *department*.
  - An *applicant* has exactly one *employee* who is _sponsor_of_ of the applicant.
  - An *applicant* may have specified a *department* that is the *applicant's* _preferred_dept._
- Use Lucidchart to draw the logical diagram.
- **Note:** You may have to add columns/attributes to some tables to implement the relationships.
- To submit the diagram, take a screen capture and modify the cell below to load your diagram from the file system. The following is an example for how to include the screenshot.

In [22]:

```
er_model_file_name = 'Screenshot_lucidchart.png'

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name)
```

Out[22]:



Relational Algebra

Instructions

- You will use the RelaX online relational algebra calculator.
- You must use the dataset `Silberschatz - UniversityDB.` I demonstrated how to select a dataset during a lecture.
- For submitting your answer, you must:
  - Cut and paste your relational expression in text.
  - Take a screenshot and include the image.
- The following is an example question and answer.

Example

Question: Produce a table of the form `(course_id, title, prereq_id, preqreq_title)` that lists courses and their prereqs.

```
π course_id, title, prereq_id, prereq_title
    (
        (π course_id, title, prereq_id (course ⋈ prereq))
```

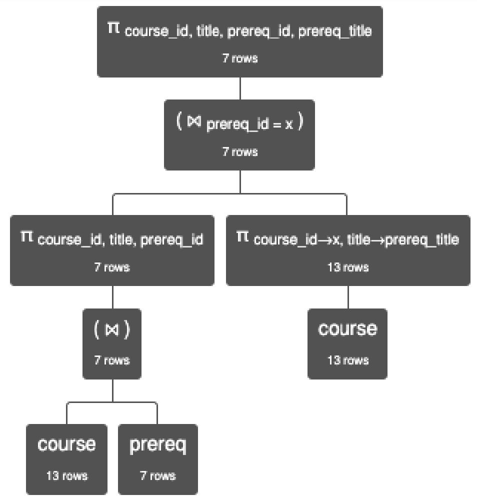
```
⋈ prereq_id=x
(π x←course_id, prereq_title←title (course))
```

In [23]:

```python
er_model_file_name = 'Screen Shot 2022-02-06 at 3.04.39 PM.png'

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name)
```

Out[23]:


π course_id, title, prereq_id, prereq_title
7 rows
(⋈ prereq_id = x)
7 rows
π course_id, title, prereq_id
7 rows
π course_id→x, title→prereq_title
13 rows
(⋈)
7 rows
course
13 rows
course
13 rows
prereq
7 rows

π course_id, title, prereq_id, prereq_title ((π course_id, title, prereq_id (course ⋈ prereq)) ⋈ prereq_id = x (π course_id→x, title→prereq_title (course)))

course.course_id	course.title	prereq.prereq_id	prereq_title
'BIO-301'	'Genetics'	'BIO-101'	'Intro. to Biology'
'BIO-399'	'Computational Biology'	'BIO-101'	'Intro. to Biology'
'CS-190'	'Game Design'	'CS-101'	'Intro. to Computer Science'
'CS-315'	'Robotics'	'CS-101'	'Intro. to Computer Science'
'CS-319'	'Image Processing'	'CS-101'	'Intro. to Computer Science'
'CS-347'	'Database System Concepts'	'CS-101'	'Intro. to Computer Science'
'EE-181'	'Intro. to Digital Systems'	'PHY-101'	'Physical Principles'

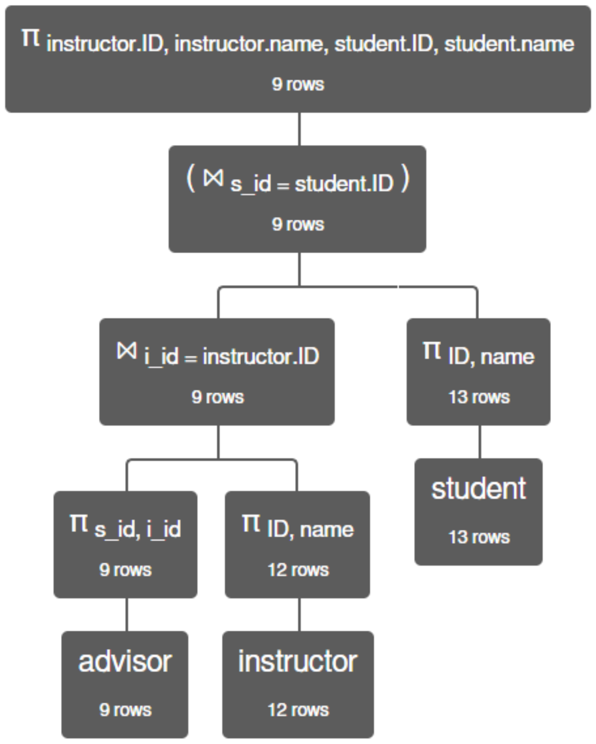
Relational Algebra Q1

- Use `student, advisor and instructor` for this question.
- Produce a table of the form `(student.ID, student.name, instructor.ID, instructor.name)` that shows students and their advisors.

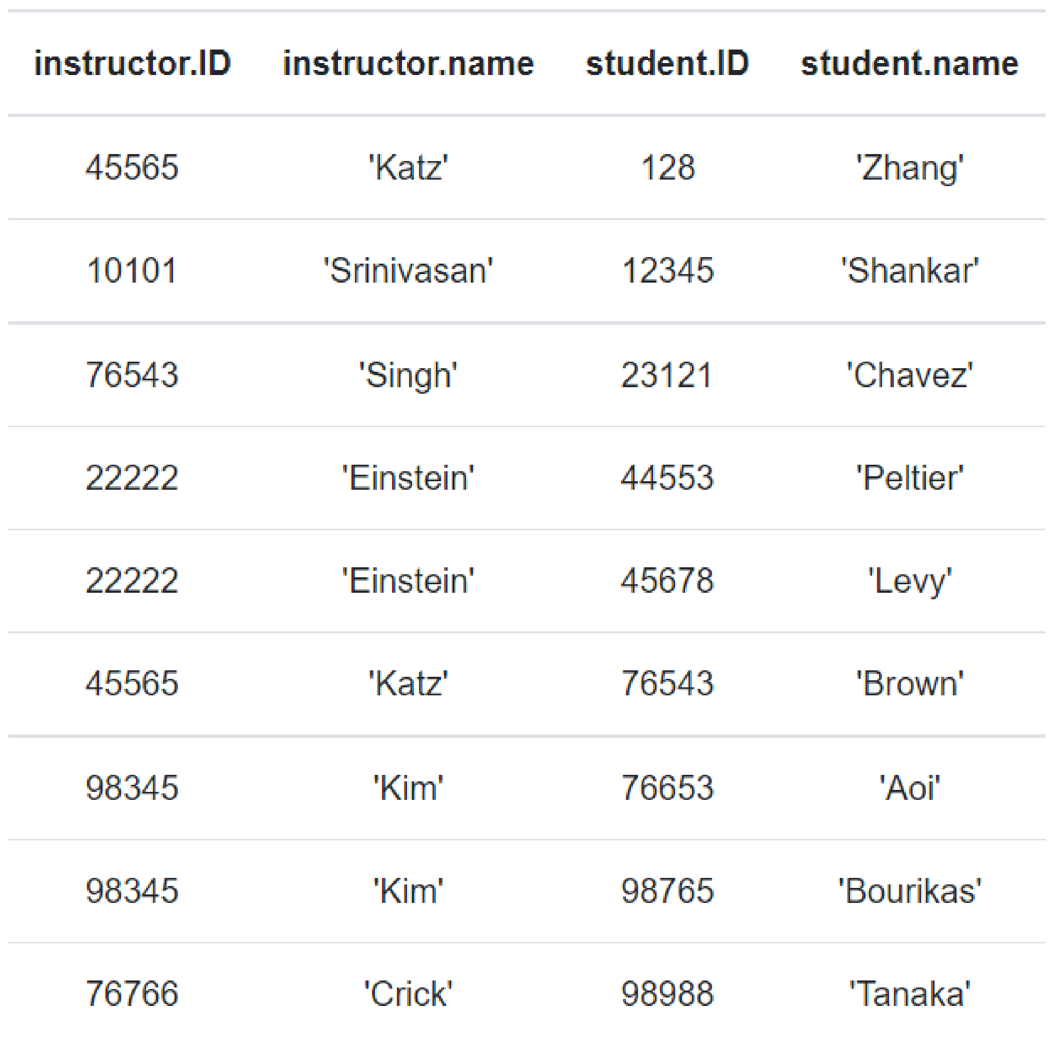
π instructor.ID, instructor.name, student.ID, student.name ((π s_id, i_id (advisor)) ⋈ i_id =

instructor.ID (π ID, name (instructor)) ⋈ s_id = student.ID (π ID, name (student)))

In [24]:
```
er_model_file_name1 = 'Screenshot (20).png'
er_model_file_name2 = 'Screenshot (21).png'
print("\n")
from IPython.display import Image
from IPython.display import display
x = Image(filename=er_model_file_name1)
y = Image(filename=er_model_file_name2)
display(x, y)
```



π instructor.ID, instructor.name, student.ID, student.name (((π s_id, i_id (advisor)) ⋈ i_id = instructor.ID (π ID, name (instructor))) ⋈ s_id = student.ID (π ID, name (student)))



instructor.ID	instructor.name	student.ID	student.name
45565	'Katz'	128	'Zhang'
10101	'Srinivasan'	12345	'Shankar'
76543	'Singh'	23121	'Chavez'
22222	'Einstein'	44553	'Peltier'
22222	'Einstein'	45678	'Levy'
45565	'Katz'	76543	'Brown'
98345	'Kim'	76653	'Aoi'
98345	'Kim'	98765	'Bourikas'
76766	'Crick'	98988	'Tanaka'

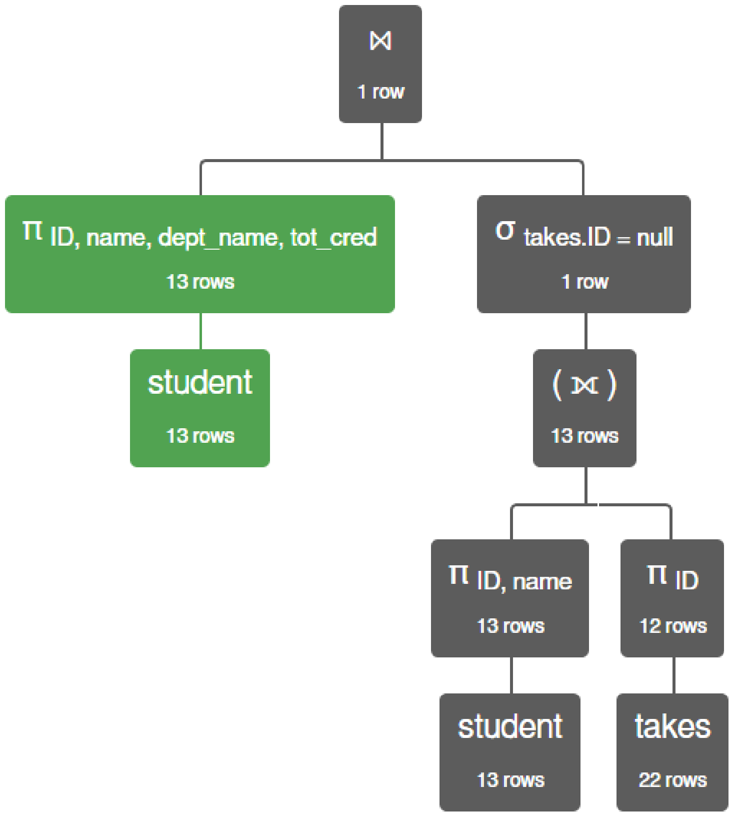
Relational Algebra Q2

- Use `student` and `takes` for this question.
- Produce a table of the form `(student.ID, student.name, student.tot_cred, student_dept_name)` for students that have not taken any course/section.

π ID, name, dept_name, tot_cred (student) ⋈ σ takes.ID = null (π ID, name (student) ⟕ π ID (takes))

In [25]:

```python
er_model_file_name1 = 'Screenshot (18).png'
er_model_file_name2 = 'Screenshot (19).png'
print("\n")
from IPython.display import Image
from IPython.display import display
x = Image(filename=er_model_file_name1)
y = Image(filename=er_model_file_name2)
display(x, y)
```



$\pi_{\text{ID, name, dept\_name, tot\_cred}}(\text{student}) \bowtie \sigma_{\text{takes.ID = null}}(\pi_{\text{ID, name}}(\text{student}) ⟕ \pi_{\text{ID}}(\text{takes}))$

student.ID	student.name	student.dept_name	student.tot_cred
70557	'Snow'	'Physics'	0

SQL

Instructions

- The questions in this section ask you to write and execute SQL statements.
- Your answer should be a code cell with `%sql` and your query.
- You must execute the query.

Example

- This is the SQL version of the query from the relational algebra section above.

In [26]:
```
%%sql
use db_book;

select a.course_id as course_id,
       a.title as title,
       prereq_id,
       b.title as prereq_tiles
from
              (select course_id, title, prereq_id from course join prereq using(course_
join
    course as b on a.prereq_id=b.course_id
```

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

Out[26]:

course_id	title	prereq_id	prereq_tiles
BIO-301	Genetics	BIO-101	Intro. to Biology
BIO-399	Computational Biology	BIO-101	Intro. to Biology
CS-190	Game Design	CS-101	Intro. to Computer Science
CS-315	Robotics	CS-101	Intro. to Computer Science
CS-319	Image Processing	CS-101	Intro. to Computer Science
CS-347	Database System Concepts	CS-101	Intro. to Computer Science
EE-181	Intro. to Digital Systems	PHY-101	Physical Principles

SQL Question 1

- Translate your answer from Relational Algebra Q1 into SQL.
- Do not worry about correctly naming the columns.

In [27]:
```
%%sql

use db_book;

select instructor.ID as instrctor_ID,
        instructor.name as instructor_name,
        student.ID as student_ID,
        student.name as student_name
from
        advisor
        join instructor
        on advisor.i_id = instructor.ID
        join student
        on advisor.s_id = student.ID
```

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

Out[27]:

instrctor_ID	instructor_name	student_ID	student_name
10101	Srinivasan	12345	Shankar
22222	Einstein	44553	Peltier
22222	Einstein	45678	Levy
45565	Katz	00128	Zhang
45565	Katz	76543	Brown
76543	Singh	23121	Chavez
76766	Crick	98988	Tanaka
98345	Kim	76653	Aoi
98345	Kim	98765	Bourikas

SQL Question 2

- You guessed it.
- Translate your answer from Relational Algebra Q2 into SQL.
- Do not worry about correctly naming the columns.

S

Y is students that have not taken a section Y = S JOIN takes

In [28]:

```
%%sql

select * from student join takes using(ID)
```

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

Out[28]:

ID	name	dept_name	tot_cred	course_id	sec_id	semester	year	grade
00128	Zhang	Comp. Sci.	102	CS-101	1	Fall	2017	A
00128	Zhang	Comp. Sci.	102	CS-347	1	Fall	2017	A-
12345	Shankar	Comp. Sci.	32	CS-101	1	Fall	2017	C
12345	Shankar	Comp. Sci.	32	CS-190	2	Spring	2017	A
12345	Shankar	Comp. Sci.	32	CS-315	1	Spring	2018	A
12345	Shankar	Comp. Sci.	32	CS-347	1	Fall	2017	A
19991	Brandt	History	80	HIS-351	1	Spring	2018	B

ID	name	dept_name	tot_cred	course_id	sec_id	semester	year	grade
23121	Chavez	Finance	110	FIN-201	1	Spring	2018	C+
44553	Peltier	Physics	56	PHY-101	1	Fall	2017	B-
45678	Levy	Physics	46	CS-101	1	Fall	2017	F
45678	Levy	Physics	46	CS-101	1	Spring	2018	B+
45678	Levy	Physics	46	CS-319	1	Spring	2018	B
54321	Williams	Comp. Sci.	54	CS-101	1	Fall	2017	A-
54321	Williams	Comp. Sci.	54	CS-190	2	Spring	2017	B+
55739	Sanchez	Music	38	MU-199	1	Spring	2018	A-
76543	Brown	Comp. Sci.	58	CS-101	1	Fall	2017	A
76543	Brown	Comp. Sci.	58	CS-319	2	Spring	2018	A
76653	Aoi	Elec. Eng.	60	EE-181	1	Spring	2017	C
98765	Bourikas	Elec. Eng.	98	CS-101	1	Fall	2017	C-
98765	Bourikas	Elec. Eng.	98	CS-315	1	Spring	2018	B
98988	Tanaka	Biology	120	BIO-101	1	Summer	2017	A
98988	Tanaka	Biology	120	BIO-301	1	Summer	2018	None

In [29]:
```
%sql select * from department
```

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

Out[29]:

dept_name	building	budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

In [30]:
```
%sql select building from department where budget > 100000
```

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

Out[30]:

building
Painter

In [31]:
```
%%sql select * from classroom where
```

```
    not building in (select building from department where budget > 100000)
```

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

Out[31]:

building	room_number	capacity
Packard	101	500
Taylor	3128	70
Watson	100	30
Watson	120	50

In [32]:

```
%%sql
select * from student where
not ID in (select ID from student join takes using(ID))
```

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

Out[32]:

ID	name	dept_name	tot_cred
70557	Snow	Physics	0

SQL Question 3

- The following query makes a copy of the `department` table.

In [33]:

```
%%sql

drop table if exists hw1_department;
create table hw1_department as select * from department
```

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

Out[33]:

```
[]
```

- The next query shows the content.

In [34]:

```
%sql select * from db_book.hw1_department
```

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

Out[34]:

dept_name	building	budget
Biology	Watson	90000.00
Comp. Sci.	Taylor	100000.00
Elec. Eng.	Taylor	85000.00
Finance	Painter	120000.00

dept_name	building	budget
History	Painter	50000.00
Music	Packard	80000.00
Physics	Watson	70000.00

- You have two tasks for this question.
    1. Create a new table `db_book.hw1_schools` that has columns `school_id` and `school_name`.
    2. Modify table `db_book.hw1_department` to contain a columns `school_id`.
- **Notes:**
    - You do not have to worry about foreign keys.
    - You do not need to populate any data or link `school_id` to the `hw1_schools`.
    - You can use DataGrip or another tool to produce the SQL DDL, but you must show successful execution on the code cells below.

In [35]:
```
%%sql

use db_book;

drop table if exists hw1_schools;

create table hw1_schools
(
    school_id varchar(4) null,
    school_name varchar(64) null
);
```

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

Out[35]:
```
[]
```

In [36]:
```
%%sql
select * from db_book.hw1_schools
```

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

Out[36]:

school_id	school_name

In [37]:
```
%%sql

alter table hw1_department
add school_id varchar(4);
```

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

Out[37]: []

In [38]:
```
%%sql
select * from db_book.hw1_department
```

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

Out[38]:

dept_name	building	budget	school_id
Biology	Watson	90000.00	None
Comp. Sci.	Taylor	100000.00	None
Elec. Eng.	Taylor	85000.00	None
Finance	Painter	120000.00	None
History	Painter	50000.00	None
Music	Packard	80000.00	None
Physics	Watson	70000.00	None

Non-Programming Track

Tasks

- There is a subdirectory in the project `data/GoT` that contains three CSV files:
  - `characters.csv`
  - `episodes.csv`
  - `character_relationships.csv`
- Your first task is to create tables to hold the data.
  - This means you must create three tables. Use a new schema and create the three tables:
    - `S22_W4111_HW1.characters`
    - `S22_W4111_HW1.episodes`
    - `S22_W4111_HW1.character_relationships.`
  - The table must have a column for each of the columns in the CSV.
  - You can use DataGrip or another tool to produce the create table statements, but you must execute the DDL statements in the code cells.
- Your second task is to load the data from the CSV files into the newly created tables. Do do this, you use a `LOAD` statement.
- Finally, you should examine the data and change column types to better reflect the actual values in the columns.
- To make the instruction more clear, I do an example of the tasks for another table. This is `got_imdb_names.csv.` You will do similar steps for the files above.

Example

- Manual examining the CSV file shows that the data has the following attributes.
  - `nconst`
  - `primaryName`
  - `birthYear`
  - `deathYear`
  - `primaryProfession`
  - `knownForTitles`
- So, my first step is to create a table to hold the information.
- **Note:** I have dozens of schema. So, I am prefixing this one with `aaaa_` to make it easy for me to find. You can drop this prefix.
- The following are the statements for creating the schema and table.

```
nconst    primaryName birthYear    deathYear    primaryProfession    knownForTitles
```

In [39]:

```
# Create the schema if it does not exist.
#%sql create schema if not exists aaaa_S22_W4111_HW1;
```

In [40]:

```
# Drop the table if it exists.
#%sql drop table if exists aaaa_S22_W4111_HW1.got_imdb_actors;
```

- Now create the table.

In [41]:

```
#%%sql
#create table if not exists aaaa_S22_W4111_HW1.got_imdb_actors
#(
#       nconst text null,
#       primaryName text null,
#       birthYear text null,
#       deathYear text null,
#       primaryProfession text null,
#       knownForTitles text null
#);
```

- This is where it gets real and you do some wizard stuff.

In [42]:

```
# This command allows loading CSV files from the local disk.
# This is set of OFF by default.
# You should only have to run this once, that is if you execute the example, you do not
#
#%sql SET GLOBAL local_infile = 'ON';
```

In [43]:

```
# This is creating a connection to the database.
# You need to replace the user and passsword with your values for your installation of 
# Do not ask about the local_infile. That is Voldemort stuff.
#
#con = pymysql.connect(host="localhost",
 #                       user="root",
#                        password="Edy990127",
 #                       autocommit=True,
  #                      local_infile=1)
```

In [44]:
```
# This statement performs the load.
# You will need to change the TABLE name and the INFILE to the correct values.
#
#sql = """
#LOAD DATA LOCAL INFILE
#'/Users/donaldferguson/Dropbox/Columbia/W4111-Intro-to-DB-S22/HWs/S22-W4111-HW-1-0/dat
#INTO TABLE aaaa_S22_W4111_HW1.got_imdb_actors
 #   FIELDS TERMINATED BY ','
 #   ENCLOSED BY '"'
 #   LINES TERMINATED BY '\n'
  #  IGNORE 1 LINES;
#"""
```

In [45]:
```
# Create a cursor. Again. Voldemort stuff, or maybe Sauron stuff.
#
#cur = con.cursor()
```

In [46]:
```
# Run the sql
#cur.execute(sql)
```

In [47]:
```
# Close the cursor. Sort of like the opposite of alohomora
#cur.close()
```

In [48]:
```
# Now test that your loading worked.
#%sql select * from aaaa_S22_W4111_HW1.got_imdb_actors;
```

In [49]:
```
#%sql describe aaaa_S22_W4111_HW1.got_imdb_actors;
```

- The final part of the task for each of the tables will be making some corrections.
- We would only ask you to do two or three corrections per table.
- Mine for this example would be in the following.

In [50]:
```
#%%sql

#use aaaa_S22_W4111_HW1;

#alter table got_imdb_actors modify nconst varchar(12) null;
```

```
#alter table got_imdb_actors modify primaryName varchar(256) null;

#alter table got_imdb_actors modify birthYear char(4) null;

#alter table got_imdb_actors modify deathYear char(4) null;
```

Characters

- Perform the tasks for characters.

In [51]:
```
# Create the schema if it does not exist.
%sql create schema if not exists S22_W4111_HW1;
```

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

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

In [52]:
```
# Drop the table if it exists.
%sql drop table if exists S22_W4111_HW1.characters;
```

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

Out[52]:
```
[]
```

In [53]:
```
%%sql
create table if not exists S22_W4111_HW1.characters
(
        characterName text null,
        characterLink text null,
        actorName text null,
        actorLink text null,
        id varchar(128) null,
        royal varchar(128) null,
    characterImageThumb text null,
    characterImageFull text null,
    nickname text null,
    kingsguard text null
);
```

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

Out[53]:
```
[]
```

In [54]:
```
con = pymysql.connect(host="localhost",
                        user="root",
                        password="Edy990127",
                        autocommit=True,
                        local_infile=1)
```

In [55]:
```
sql = """
LOAD DATA LOCAL INFILE
```

```
'E:/Github/spring-2022-COMS4111/S22-W4111-HW-1-0/data/GoT/characters.csv'
INTO TABLE S22_W4111_HW1.characters
    FIELDS TERMINATED BY ','
    ENCLOSED BY '"'
    LINES TERMINATED BY '\n'
    IGNORE 1 LINES;
"""
```

In [56]:
```
cur = con.cursor()
```

In [57]:
```
cur.execute(sql)
```

Out[57]:
```
389
```

In [58]:
```
cur.close()
```

In [60]:
```
%sql select * from S22_W4111_HW1.characters LIMIT 20;
```

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

Out[60]:

characterName	characterLink	actorName	actorLink	id	royal
Addam Marbrand	/character/ch0305333/	B.J. Hogg	/name/nm0389698/	6191091c06029e3acded09e1	
Aegon Targaryen				6191091c06029e3acded09e2	1
Aeron Greyjoy	/character/ch0540081/	Michael Feast	/name/nm0269923/	6191091c06029e3acded09e3	
Aerys II Targaryen	/character/ch0541362/	David Rintoul	/name/nm0727778/	6191091c06029e3acded09e4	1
Akho	/character/ch0544520/	Chuku Modu	/name/nm6729880/	6191091c06029e3acded09e5	
Alliser Thorne	/character/ch0246938/	Owen Teale	/name/nm0853583/	6191091c06029e3acded09e6	
Alton Lannister	/character/ch0305012/	Karl Davies	/name/nm0203801/	6191091c06029e3acded09e7	
Alys Karstark	/character/ch0576836/	Megan Parkinson	/name/nm8257864/	6191091c06029e3acded09e8	
Amory Lorch	/character/ch0305002/	Fintan McKeown	/name/nm0571654/	6191091c06029e3acded09e9	
Anguy	/character/ch0316930/	Philip McGinley	/name/nm1528121/	6191091c06029e3acded09ea	
Archmaester Marwyn	/character/ch0578265/	Jim Broadbent	/name/nm0000980/	6191091c06029e3acded09eb	
Areo Hotah	/character/ch0507107/	Deobia Oparei	/name/nm0649046/	6191091c06029e3acded09ec	

characterName	characterLink	actorName	actorLink	id	royal
Armeca	/character/ch0305014/	Sahara Knite	/name/nm1783582/	6191091c06029e3acded09ed	
Arthur	/character/ch0305326/	Nathanael Saleh	/name/nm8127149/	6191091c06029e3acded09ee	
Arthur Dayne	/character/ch0540097/	Luke Roberts	/name/nm1074361/	6191091c06029e3acded09ef	
Arya Stark	/character/ch0158604/	Maisie Williams	/name/nm3586035/	6191091c06029e3acded09f0	
Baby Sam	/character/ch0547881/			6191091c06029e3acded09f1	
Balon Greyjoy	/character/ch0292152/	Patrick Malahide	/name/nm0538869/	6191091c06029e3acded09f2	
Baratheon Guard	/character/ch0350989/	Phil Barnhill	/name/nm4207240/	6191091c06029e3acded09f3	
Barristan Selmy	/character/ch0241346/	Ian McElhinney	/name/nm0568400/	6191091c06029e3acded09f4	

In [61]:

```
%sql describe S22_W4111_HW1.characters;
```

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

Out[61]:

Field	Type	Null	Key	Default	Extra
characterName	text	YES		None	
characterLink	text	YES		None	
actorName	text	YES		None	
actorLink	text	YES		None	
id	varchar(128)	YES		None	
royal	varchar(128)	YES		None	
characterImageThumb	text	YES		None	
characterImageFull	text	YES		None	
nickname	text	YES		None	
kingsguard	text	YES		None	

In [62]:

```
%%sql

use S22_W4111_HW1;

alter table characters modify characterName varchar(256) null;

alter table characters modify actorName varchar(256) null;

alter table characters modify id varchar(24) null;
```

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

Out[62]:
```
[]
```

Episodes

- Perform the tasks for episodes.

In [63]:
```
%sql drop table if exists S22_W4111_HW1.episodes;
```

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

Out[63]:
```
[]
```

In [64]:
```
%%sql
create table if not exists S22_W4111_HW1.episodes
(
        seasonNum text null,
        episodeNum text null,
        sceneNum text null,
        sceneLocation text null,
        sceneSubLocation text null,
        sceneStartTime text null,
    sceneEndTime text null
);
```

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

Out[64]:
```
[]
```

In [65]:
```
con = pymysql.connect(host="localhost",
                      user="root",
                      password="Edy990127",
                      autocommit=True,
                      local_infile=1)
```

In [66]:
```
sql = """
LOAD DATA LOCAL INFILE
'E:/Github/spring-2022-COMS4111/S22-W4111-HW-1-0/data/GoT/episodes.csv'
INTO TABLE S22_W4111_HW1.episodes
    FIELDS TERMINATED BY ','
    ENCLOSED BY '"'
    LINES TERMINATED BY '\n'
    IGNORE 1 LINES;
"""
```

In [67]:
```
cur = con.cursor()
```

In [68]:
```
cur.execute(sql)
```
Out[68]:
```
4165
```
In [69]:
```
cur.close()
```
In [70]:
```
%sql select * from S22_W4111_HW1.episodes LIMIT 20;
```
```
 * mysql+pymysql://root:***@localhost
20 rows affected.
```
Out[70]:

seasonNum	episodeNum	sceneNum	sceneLocation	sceneSubLocation	sceneStartTime	sceneEndTime
1	1	0	The Wall	Castle Black	0:00:40	0:01:45
1	1	1	North of the Wall	The Haunted Forest	0:01:45	0:03:24
1	1	2	North of the Wall	The Haunted Forest	0:03:24	0:03:31
1	1	3	North of the Wall	The Haunted Forest	0:03:31	0:03:38
1	1	4	North of the Wall	The Haunted Forest	0:03:38	0:03:44
1	1	5	North of the Wall	The Haunted Forest	0:03:44	0:05:36
1	1	6	North of the Wall	The Haunted Forest	0:05:36	0:05:41
1	1	7	North of the Wall	The Haunted Forest	0:05:41	0:05:48
1	1	8	North of the Wall	The Haunted Forest	0:05:48	0:05:58
1	1	9	North of the Wall	The Haunted Forest	0:05:58	0:06:21
1	1	10	North of the Wall	The Haunted Forest	0:06:21	0:06:39
1	1	11	North of the Wall	The Haunted Forest	0:06:39	0:06:49
1	1	12	North of the Wall	The Haunted Forest	0:06:49	0:07:45
1	1	13	The North	Winterfell	0:09:27	0:12:38
1	1	14	The North	Outside Winterfell	0:12:38	0:15:41
1	1	15	The North	Outside Winterfell	0:15:41	0:18:44
1	1	16	The Crownlands	King's Landing	0:18:44	0:20:45
1	1	17	The North	Winterfell	0:20:45	0:22:43
1	1	18	The North	Winterfell	0:22:43	0:23:09

seasonNum	episodeNum	sceneNum	sceneLocation	sceneSubLocation	sceneStartTime	sceneEndTime
1	1	19	The North	Winterfell	0:23:09	0:23:39

In [71]:

```
%%sql

use S22_W4111_HW1;

alter table episodes modify seasonNum char(4) null;

alter table episodes modify episodeNum char(4) null;

alter table episodes modify sceneNum char(4) null;

alter table episodes modify sceneLocation varchar(256) null;

alter table episodes modify sceneSubLocation varchar(256) null;

alter table episodes modify sceneStartTime time null;

alter table episodes modify sceneEndTime time null;
```

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

Out[71]:

```
[]
```

In [72]:

```
%sql describe S22_W4111_HW1.episodes;
```

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

Out[72]:

Field	Type	Null	Key	Default	Extra
seasonNum	char(4)	YES		None	
episodeNum	char(4)	YES		None	
sceneNum	char(4)	YES		None	
sceneLocation	varchar(256)	YES		None	
sceneSubLocation	varchar(256)	YES		None	
sceneStartTime	time	YES		None	
sceneEndTime	time	YES		None	

Characters Relationships

- Perform the tasks for character_relationships.

In [73]:
```
%sql drop table if exists S22_W4111_HW1.character_relationships;
```
```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```
Out[73]:
```
[]
```

In [74]:
```
%%sql
create table if not exists S22_W4111_HW1.character_relationships
(
        source_character_id text null,
        sourceCharacterName text null,
        relationship text null,
        target_character_id text null,
        targetCharacterName text null
);
```
```
 * mysql+pymysql://root:***@localhost
0 rows affected.
```
Out[74]:
```
[]
```

In [75]:
```
con = pymysql.connect(host="localhost",
                      user="root",
                      password="Edy990127",
                      autocommit=True,
                      local_infile=1)
```

In [76]:
```
cur = con.cursor()
```

In [77]:
```
sql = """
LOAD DATA LOCAL INFILE
'E:/Github/spring-2022-COMS4111/S22-W4111-HW-1-0/data/GoT/character_relationships.csv'
INTO TABLE S22_W4111_HW1.character_relationships
    FIELDS TERMINATED BY ','
    ENCLOSED BY '"'
    LINES TERMINATED BY '\n'
    IGNORE 1 LINES;
"""
```

In [78]:
```
cur.execute(sql)
```
Out[78]:
```
785
```

In [79]:
```
cur.close()
```

In [80]:
```
%sql select * from S22_W4111_HW1.character_relationships LIMIT 20;
```
```
 * mysql+pymysql://root:***@localhost
20 rows affected.
```
Out[80]:

source_character_id	sourceCharacterName	relationship	target_character_id	targetChai

source_character_id	sourceCharacterName	relationship	target_character_id	targetChar
6191091c06029e3acded09e2	Aegon Targaryen	parents	6191091c06029e3acded0a20	
6191091c06029e3acded09e2	Aegon Targaryen	killedBy	6191091c06029e3acded0a38	Gre
6191091c06029e3acded09e2	Aegon Targaryen	siblings	6191091c06029e3acded0a5c	
6191091c06029e3acded09e2	Aegon Targaryen	parents	6191091c06029e3acded0af8	Rhaeg
6191091c06029e3acded09e2	Aegon Targaryen	siblings	6191091c06029e3acded0afb	Rhaen
6191091c06029e3acded09e3	Aeron Greyjoy	siblings	6191091c06029e3acded09f2	Ba
6191091c06029e3acded09e3	Aeron Greyjoy	siblings	6191091c06029e3acded0a22	Eu
6191091c06029e3acded09e4	Aerys II Targaryen	servedBy	6191091c06029e3acded09ef	A
6191091c06029e3acded09e4	Aerys II Targaryen	killed	6191091c06029e3acded09fd	Br
6191091c06029e3acded09e4	Aerys II Targaryen	parentOf	6191091c06029e3acded0a0d	Daener
6191091c06029e3acded09e4	Aerys II Targaryen	killedBy	6191091c06029e3acded0a52	Jair
6191091c06029e3acded09e4	Aerys II Targaryen	servedBy	6191091c06029e3acded0a52	Jair
6191091c06029e3acded09e4	Aerys II Targaryen	parentOf	6191091c06029e3acded0af8	Rhaeg
6191091c06029e3acded09e4	Aerys II Targaryen	marriedEngaged	6191091c06029e3acded0afa	Rhael
6191091c06029e3acded09e4	Aerys II Targaryen	siblings	6191091c06029e3acded0afa	Rhael
6191091c06029e3acded09e4	Aerys II Targaryen	killed	6191091c06029e3acded0afd	F
6191091c06029e3acded09e4	Aerys II Targaryen	parentOf	6191091c06029e3acded0b44	Visery
6191091c06029e3acded09e5	Akho	killedBy	6191091c06029e3acded0a0c	Da
6191091c06029e3acded09e6	Alliser Thorne	killed	6191091c06029e3acded0a5c	
6191091c06029e3acded09e6	Alliser Thorne	killedBy	6191091c06029e3acded0a5c	

In [81]:

```
%%sql

use S22_W4111_HW1;

alter table character_relationships modify source_character_id varchar(256) null;

alter table character_relationships modify sourceCharacterName varchar(256) null;

alter table character_relationships modify relationship char(56) null;

alter table character_relationships modify target_character_id varchar(256) null;

alter table character_relationships modify targetCharacterName char(56) null;
```

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

```
785 rows affected.
785 rows affected.
```

Out[81]:
```
[]
```

In [82]:
```
%sql describe S22_W4111_HW1.character_relationships;
```

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

Out[82]:

Field	Type	Null	Key	Default	Extra
source_character_id	varchar(256)	YES		None	
sourceCharacterName	varchar(256)	YES		None	
relationship	char(56)	YES		None	
target_character_id	varchar(256)	YES		None	
targetCharacterName	char(56)	YES		None	

In [83]:
```
import os
os.listdir("../data/GoT")
```

Out[83]:
```
['characters.csv',
 'character_relationships.csv',
 'episodes.csv',
 'got_actors.csv',
 'got_imdb_actors.csv']
```

Programming Track

Note: If you have activated student license when installing Datagrip, you can also use Pycharm Professional version instead of Community edition.

Tasks

- You will create and modify files in the directory `<uni>_web_src.`
- You will use the database that comes with the book, e.g. `db_book,` that you previously installed.
- Your web application will support `GET` on the path `/api/db_book/students/<ID>.` This means you have to implement two things:
  1. A function in `application.py` that implements the path endpoint.
  2. A method on a class `Student` that connects to the database, runs the SQL and returns the result. The project has been updated to have implementation templates for where your code goes.
- For submission, you must copy your code from the Python file below to show your code.
- You must include a screenshot of calling your application from a browser.

Modified application.py

```
from flask import Flask, Response, request
import json
from datetime import datetime
import rest_utils

app = Flask(__name__)


###############################################################################


# DFF TODO A real service would have more robust health check methods.
# This path simply echoes to check that the app is working.
# The path is /health and the only method is GETs
@app.route("/health", methods=["GET"])
def health_check():
    rsp_data = {"status": "healthy", "time": str(datetime.now())}
    rsp_str = json.dumps(rsp_data)
    rsp = Response(rsp_str, status=200, content_type="application/json")
    return rsp


# TODO Remove later. Solely for explanatory purposes.
# The method take any REST request, and produces a response indicating
what
# the parameters, headers, etc. are. This is simply for education
purposes.
#
@app.route("/api/demo/<parameter1>", methods=["GET", "POST", "PUT",
"DELETE"])
@app.route("/api/demo/", methods=["GET", "POST", "PUT", "DELETE"])
def demo(parameter1=None):
    """
    Returns a JSON object containing a description of the received
request.

    :param parameter1: The first path parameter.
    :return: JSON document containing information about the request.
    """

    # DFF TODO -- We should wrap with an exception pattern.
    #

    # Mostly for isolation. The rest of the method is isolated from the
specifics of Flask.
    inputs = rest_utils.RESTContext(request, {"parameter1": parameter1})

    # DFF TODO -- We should replace with logging.
    r_json = inputs.to_json()
    msg = {
        "/demo received the following inputs": inputs.to_json()
    }
    print("/api/demo/<parameter> received/returned:\n", msg)

    rsp = Response(json.dumps(msg), status=200,
content_type="application/json")
    return rsp

################################################################################



@app.route("/api/db_book/students/<ID>", methods=["GET"])
def get_student_by_id(ID):
    #
    # Your code goes here.
    #
    pass


if __name__ == '__main__':
    app.run(host="0.0.0.0", port=5000)
```

Modified student_resource.py

```
class Student:

    def __init__(self):
        # You may have to put code here.
        pass

    def get_by_id(self, ID):
        # Connect to DB.
        # Form SQL
        # Run query
        # return result
        pass
```

Screen Capture of Calling from Browser