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 [1]:
```
%load_ext sql
```

In [2]:
```
%sql mysql+pymysql://root:Xcz990208!@localhost
```

Out[2]:
```
'Connected: root@None'
```

In [3]:
```
%sql select * from db_book.student where name like "z%" or name like "sh%"
```

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

Out[3]:

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

PyMySQL

In [4]:
```
import pymysql
```

In [5]:
```
conn = pymysql.connect(host="localhost", user="root", password="Xcz990208!")
```

In [6]:
```
sql = """
    select * from db_book.student where
        name like %s or name like %s
"""
```

In [7]:
```
pattern_1 = "z%"
pattern_2 = "sh%"
```

In [8]:
```
cur = conn.cursor()
res = cur.execute(
    sql, args=(pattern_1, pattern_2)
)
res = cur.fetchall()
```

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

Out[9]:
```
(('00128', 'Zhang', 'Comp. Sci.', Decimal('102')),
 ('12345', 'Shankar', 'Comp. Sci.', Decimal('32')))
```

Pandas

In [10]:
```
import pandas as pd
```

In [11]:
```
pwd
```

Out[11]:
```
'C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\Data_1\\S22-W4111-HW-1-0\\cx2273_
src'
```

In [12]:
```
#
#  Replace the path below with the path of your project directory.
#  Use // instead of / if you're on Windows.
#
project_root = "C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\Data_1\\S22-W4111-
```

In [13]:
```
people_df = pd.read_csv(project_root + "/data/People.csv")
```

In [14]:
```
people_df.loc[
    (people_df['nameLast'] == "Williams") & (people_df['birthCity'] == 'San Diego'),
    ["playerID", "nameLast", "nameFirst", "birthYear", 'birthCity', 'bats', 'throws']
]
```

Out[14]:

	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 [15]:
```
from sqlalchemy import create_engine
```

In [16]:
```
engine = create_engine("mysql+pymysql://root:Xcz990208!@localhost")
```

In [17]:
```
sql = """
    select * from db_book.student where
        name like %s or name like %s
"""
```

```
pattern_1 = "z%"
pattern_2 = "sh%"
```

In [18]:

```
another_df = pd.read_sql(sql, params=(pattern_1, pattern_2), con=engine)
another_df
```

Out[18]:

	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

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.

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

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";>

$$\begin{aligned} &Employee\ (\underline{employee\_no}, last\_name, first\_name, \textbf{department\_no}) \\ &Department\ (\underline{department\_id}, department\_name) \\ &Applicant\ (\underline{email}, last\_name, first\_name, \textbf{sponsor\_no, preferred\_dept\_no}) \end{aligned} \tag{2}$$

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

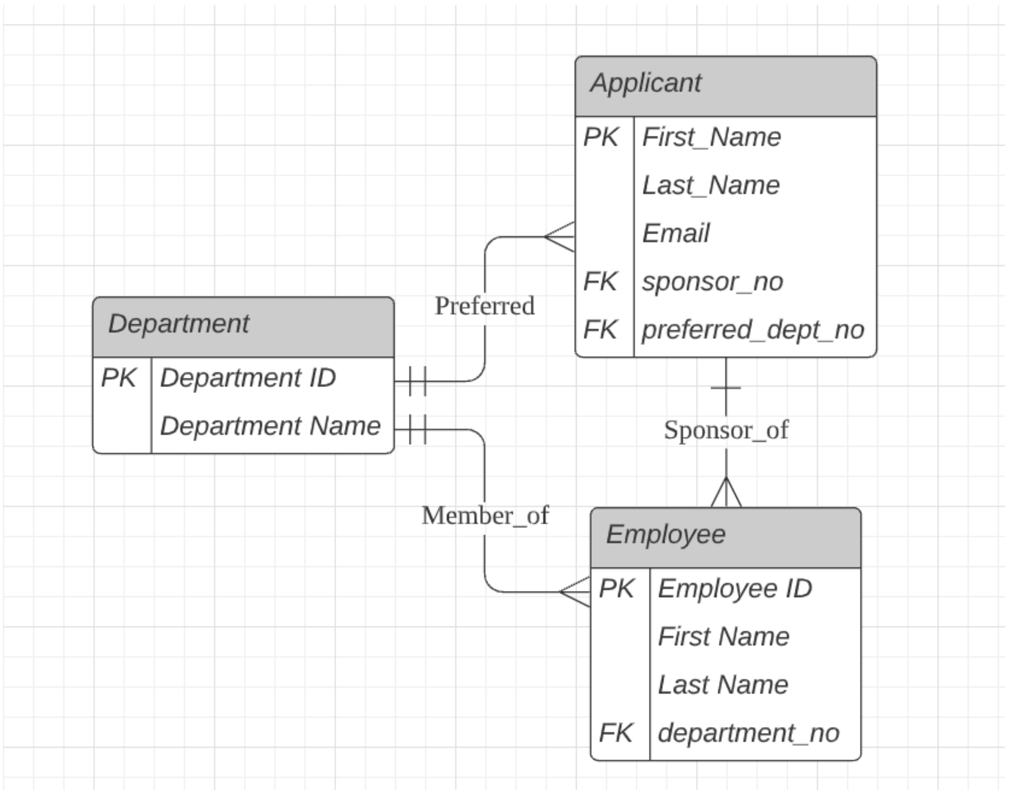
ER Modeling

- 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 [1]:

```python
er_model_file_name = 'ER_diagram.png'

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name, width = 500, height = 500)
```

Out[1]:



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

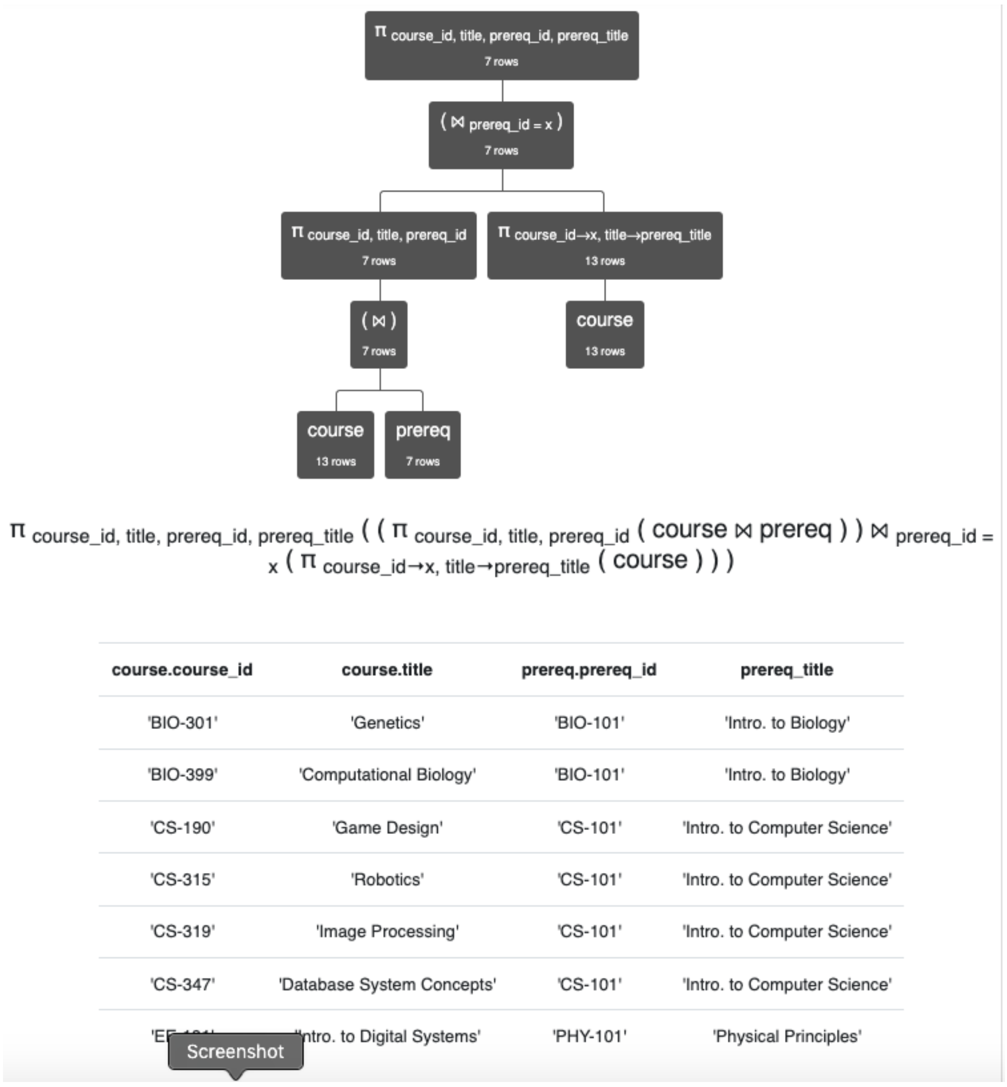
Question: Produce a table of the form `(course_id, title, prereq_id, preqreq_title)` that lists courses and their prereqs.

```
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 [22]:
```
er_model_file_name = 'sample_answer_q.png'

print("\n")
```

```python
from IPython.display import Image
Image(filename=er_model_file_name, width = 500, height = 500)
```

Out[22]:


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
Screenshot

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	ntro. to Digital Systems'	'PHY-101'	'Physical Principles'

Relational Algebra Q1

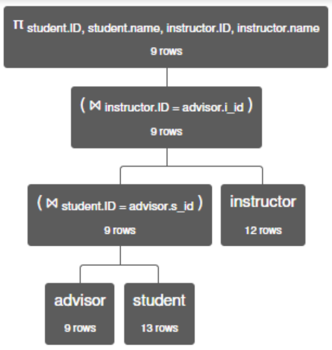
- Use `student, advisor and instructor` for this question.
- Produce a table of the form `(student.ID, student.name, instructor.ID, instructor.name)` that shows students and their advisors.

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

In [23]:

```python
er_model_file_name = 'R_question_1.png'

print("\n")
from IPython.display import Image
Image(filename=er_model_file_name, width = 500, height = 500)
```

Out[23]:


π student.ID, student.name, instructor.ID, instructor.name
9 rows
(⋈ instructor.ID = advisor.i_id)
9 rows
(⋈ student.ID = advisor.s_id)
9 rows
instructor
12 rows
advisor
9 rows
student
13 rows

$\pi_{\text{student.ID, student.name, instructor.ID, instructor.name}}$ ((advisor $\bowtie_{\text{student.ID = advisor.s\_id}}$ student) $\bowtie_{\text{instructor.ID = advisor.i\_id}}$ instructor)

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

Relational Algebra Q2

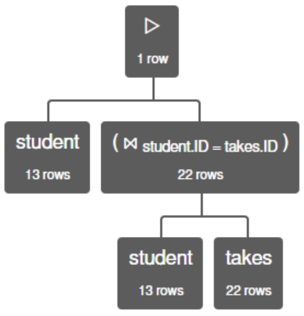
- Use `student` and `takes` for this question.
- Produce a table of the form `(student.ID, student.name, student_tot_cred, student_dept_name)` for students that have not taken any course/section.

`student ▷ (student ⋈ student.ID = takes.ID takes)`

In [44]:

```
er_model_file_name = ’R_question_2.png’

print(”\n”)
from IPython.display import Image
Image(filename=er_model_file_name, width = 500, height = 500)
```

Out[44]:



student ▷ (student $\bowtie_{\text{student.ID = takes.ID}}$ 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 [65]:

```
%%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(cou
join
    course as b on a.prereq_id=b.course_id
```

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

Out[65]:

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

course_id	title	prereq_id	prereq_tiles
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 [97]:

```
%%sql
use db_book;
select student.ID as student_ID, student.name as student_name, instructor.ID as inst
from advisor
join student ON advisor.s_id = student.ID join instructor on instructor.ID = advisor
```

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

Out[97]:

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

SQL Question 2

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

In [114…

```
%%sql
select student.ID as student_id, student.name as student_name, student.dept_name as
from student
left join takes ON student.ID = takes.ID
where takes.ID is null
```

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

Out[114...

student_id	student_name	student_dept_name	student_tot_cred
70557	Snow	Physics	0

SQL Question 3

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

In [22]:

```
%%sql

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

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

Out[22]:

```
[]
```

- The next query shows the content.

In [23]:

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

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

Out[23]:

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

- 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_boot.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 [102…

```
%%sql
create table hw1_schools
(
    school_id int null,
    school_name varchar(255) null
);

alter table hw1_schools
    add constraint hw1_schools_pk
        primary key (school_id);

alter table department
    add school_id int null;
```

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

Out[102…

```
[]
```

In [113…

```
%%sql
alter table department
    add constraint department_hw1_schools__fk
        foreign key (school_id) references hw1_schools (school_id);
        '''
        Because I execute the code in DataGrip, so it's duplicate, LOL.
        '''
```

```
 * mysql+pymysql://root:***@localhost
(pymysql.err.OperationalError) (1826, "Duplicate foreign key constraint name 'departme
nt_hw1_schools__fk'")
[SQL: alter table department
    add constraint department_hw1_schools__fk
        foreign key (school_id) references hw1_schools (school_id);]
(Background on this error at: https://sqlalche.me/e/14/e3q8)
```

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.

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

In [ ]:
```
# Drop the table if it exists.
%sql drop table if exists aaaa_S22_W4111_HW1.got_imdb_actors;
```

- Now create the table.

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

```
# 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 no
#
%sql SET GLOBAL local_infile = 'ON';
```

In [ ]:
```
# 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="dbuser",
                      password="dbuserdbuser",
                      autocommit=True,
                      local_infile=1)
```

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

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

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

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

In [ ]:
```
%sql select * from aaaa_S22_W4111_HW1.characters;
```

- 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 [ ]:
```
%%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.

Episodes

- Perform the tasks for episodes.

Characters Relatrionships

- Perform the tasks for character_relationships.

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
import pymysql

from student_resource import Student

student_resource = Student()

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/student/<ID>", methods=["GET"])
def get_student_by_id(ID):
    msg = student_resource.get_by_id(ID)
    rsp = Response(msg, status = 200, content_type="text/plain")
    return rsp

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

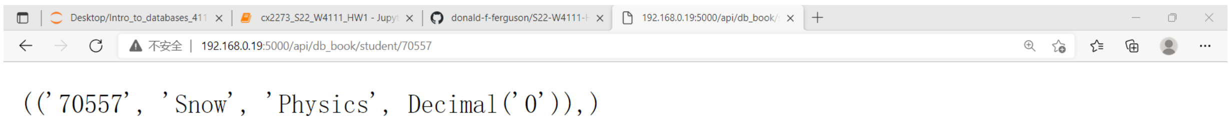
Modified student_resource.py

```
import pymysql
class Student:

    def __init__(self):
        self.host = 'localhost'
        self.user = 'root'
        self.password = 'Xcz990208!'
        self.database = 'db_book'

    def get_by_id(self, ID):
        # Connect to DB.
        connection = pymysql.connect(
                                host=self.host,
                                user=self.user,
                                password=self.password,
                                database=self.database
                               )
        # Form SQL
        sql = "SELECT * FROM student WHERE student.id = '%s'" %(ID)
        # Run query
        cursor = connection.cursor()
        cursor.execute(sql)
        result = cursor.fetchall()
        # return result
        return str(result)
```

Screen Capture of Calling from Browser

In [25]:
```
er_model_file_name = 'C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\Data_1\\S22-

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

Out[25]:

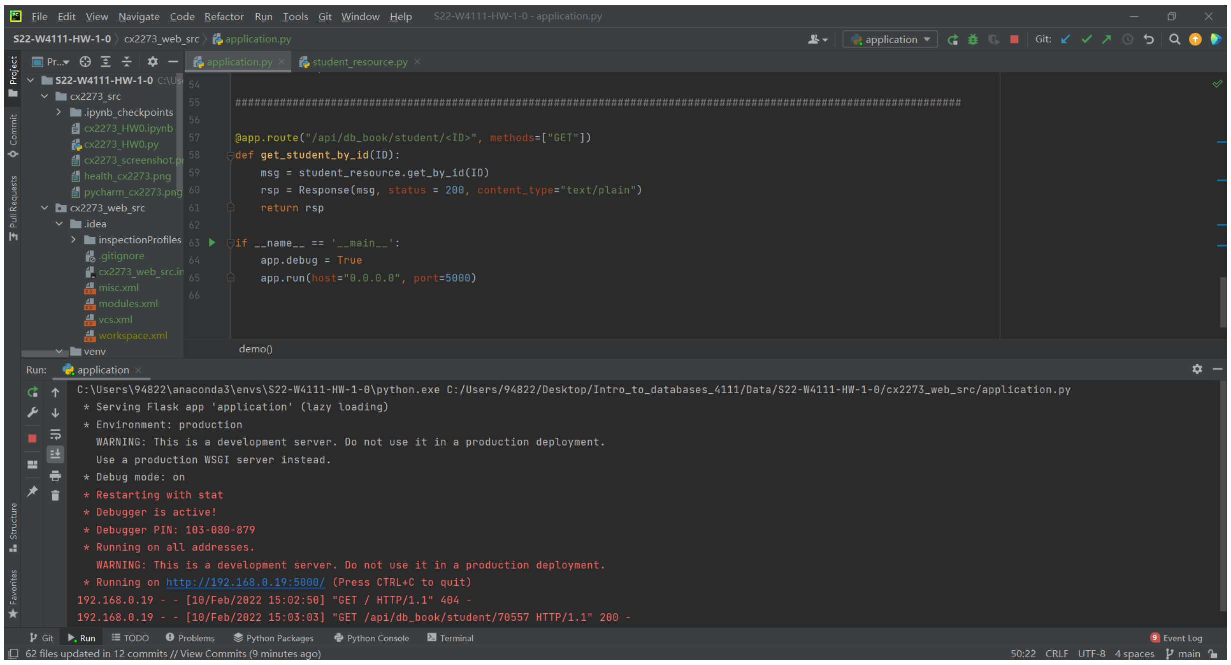

Desktop/Intro_to_databases_411
cx2273_S22_W4111_HW1 - Jupy
donald-f-ferguson/S22-W4111-
192.168.0.19:5000/api/db_book/
不安全 | 192.168.0.19:5000/api/db_book/student/70557

```
(('70557', 'Snow', 'Physics', Decimal('0')),)
```

In [27]:

```
er_model_file_name = 'C:\\Users\\94822\\Desktop\\Intro_to_databases_4111\\Data_1\\S22-

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

Out[27]:


File Edit View Navigate Code Refactor Run Tools Git Window Help S22-W4111-HW-1-0 - application.py
S22-W4111-HW-1-0 〉 cx2273_web_src 〉 application.py
application
application.py
student_resource.py
S22-W4111-HW-1-0
cx2273_src
.ipynb_checkpoints
cx2273_HW0.ipynb
cx2273_HW0.py
cx2273_screenshot.p
health_cx2273.png
pycharm_cx2273.png
cx2273_web_src
.idea
inspectionProfiles
.gitignore
cx2273_web_src.in
misc.xml
modules.xml
vcs.xml
workspace.xml
venv
demo()
Run: application
Git Run TODO Problems Python Packages Python Console Terminal Event Log
62 files updated in 12 commits // View Commits (9 minutes ago)
50:22 CRLF UTF-8 4 spaces main

```
####################################################################################################

@app.route("/api/db_book/student/<ID>", methods=["GET"])
def get_student_by_id(ID):
    msg = student_resource.get_by_id(ID)
    rsp = Response(msg, status = 200, content_type="text/plain")
    return rsp

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

```
C:\Users\94822\anaconda3\envs\S22-W4111-HW-1-0\python.exe C:/Users/94822/Desktop/Intro_to_databases_4111/Data/S22-W4111-HW-1-0/cx2273_web_src/application.py
 * Serving Flask app 'application' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 103-080-879
 * Running on all addresses.
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://192.168.0.19:5000/ (Press CTRL+C to quit)
192.168.0.19 - - [10/Feb/2022 15:02:50] "GET / HTTP/1.1" 404 -
192.168.0.19 - - [10/Feb/2022 15:03:03] "GET /api/db_book/student/70557 HTTP/1.1" 200 -
```