# Lab Assignment 6: Creating and Connecting to Databases

## DS 6001: Practice and Application of Data Science

### Instructions

Please answer the following questions as completely as possible using text, code, and the results of code as needed. Format your answers in a Jupyter notebook. To receive full credit, make sure you address every part of the problem, and make sure your document is formatted in a clean and professional way.

**Please note: you will not be able to use Rivanna for this lab as Rivanna is not set up to work with Docker or with Databases. If you need help getting your local system running, please let me know.**

### Problem 0 [No points, no need to write anything here for any of the following parts, but do it anyway!]

Databases require a lot of external software. The good news is that there are excellent free and open source options to do very advanced work with databases. The bad news is that each piece of additional software comes with its own complications. This problem will guide you through the installation steps for the software you need to run database systems on your computer, document those databases, and connect to them through Python with (fingers crossed) as few problems as possible.

#### Part a

Use `pip` to install the following Python packages on your system:

```
mysql-connector-python
psycopg
pymongo
sqlalchemy
wget
```

#### Part b

With the exception of SQlite, database systems run as external software that must be installed and run on your computer. To make the installation steps easier, you will need some configuration files that I wrote and saved in a GitHub repository. Open your terminal and use the `cd` command to navigate to the folder in your computer where you want to work. Then type

```
git clone https://github.com/jkropko/ds6001databases
```

If this command works, it will create a new directory within your current folder called "ds6001databases".

- Check that this folder exists and contains the following files: LICENSE, README.md, compose.yaml, db_tests.ipynb, and requirements.txt
- Save the notebook file you will be using for your Lab 6 work inside the "ds6001databases" folder

#### Part c

We will be using a system called Docker to work with databases. Docker is the most commonly used platform for working with **containers**. While we will not be delving into the topic of containerization in this course, a container is space in your computer's memory that is set apart from the rest of your computer. We can act as if the container is an entirely new computer, and inside the container we can change the operating system and install other software external to Python, such as database management systems. We can use a container to run Windows on a Mac, or vice versa, or Linux on any system. By far the easiest way to run MySQL, PostgreSQL, and MongoDB is through Docker containers.

You will need to install Docker Desktop on your computer. Go to https://www.docker.com/products/docker-desktop/ and click on the Download button, making sure the operating system listed matches the operating system of your computer.

Once Docker Desktop is installed, find the Docker Desktop program on your computer and run it.

To confirm that Docker Desktop is running, open a terminal and type `docker help`. If you see documentation that begins

```
Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers
```

then you are all set. If you see an error that Docker is not found, then the Docker Desktop client was not installed properly, so you should try downloading and installing it from the website again. If you receive a message that the Docker daemon is not running, then Docker is installed but is not running. Find the Docker Desktop executable on your computer and click it to get Docker running.

### Part d

Inside your "ds6001databases" folder, create a .env file. On a Mac, type `touch .env` then `open .env` to create and open the file. On Windows, open a new file on Notepad and go to "Save As", then save it in your "ds6001databases" folder -- make sure to set "File As Type" to "All files" and name the file ".env".

Inside the .env file you need to choose passwords for the MySQL, PostgreSQL, and MongoDB databases, so type

```
MYSQL_ROOT_PASSWORD=redlobstercheddarbiscuits
POSTGRES_PASSWORD=outbackbloominonion
MONGO_INITDB_ROOT_PASSWORD=olivegardenunlimitedbreadsticks
MONGO_INITDB_ROOT_USERNAME=mongo
mongo_init_db = mongodb
MYSQL_DATABASE=mysql
```

Change the passwords on the first three lines to whatever you want, but DON'T USE THE @ SYMBOL as that will cause problems. Leave the fourth, fifth, and sixth lines alone, as well as the names of each environmental variable.

### Part e

In the terminal, make sure you are in the "ds6001databases" folder (you can check by typing `pwd`. If not, then use `cd` to navigate to the "ds6001databases" folder). Then type

```
docker compose up
```

This command launches all of the databases. If successful, you will see a long stream of output with messages that begin `ds6001databases-postgres-1`, `ds6001databases-mysql-1`, and `ds6001databases-mongo-1`. If not, we will need to debug together, but the issue likely has to do with something preventing parts a, b, c, or d from being completed successfully.

### Part f

To confirm that the databases are running on your system, open the "db_tests.ipynb" notebook file, which should be saved in you "ds6001databases" folder. Run everything in this notebook and make sure there are no errors.

### Part g

In addition to the databases, we will be using dbdocs.io to create documentation for our databases and post them online with a stable URL. But to get dbdocs running, you first need to install NodeJS on your computer: https://nodejs.org/en

Then to install dbdocs, follow the instructions here: https://dbdocs.io/docs

### Part h

Finally, create a notebook inside your "ds6001databases" folder for your work on this lab. Import the following libraries, and load the `.env` file where you store your passwords.

In [1]:
```
import dotenv.main
import numpy as np
import pandas as pd
import wget
import sqlite3
import requests
import json
import os
import sys
import dotenv
from dotenv.main import load_dotenv
import mysql.connector
import pymongo
from sqlalchemy import create_engine



dotenv.main.load_dotenv()
POSTGRES_PASSWORD = os.getenv('POSTGRES_PASSWORD')
MONGO_INITDB_ROOT_USERNAME = os.getenv('MONGO_INITDB_ROOT_USERNAME')
MONGO_INITDB_ROOT_PASSWORD = os.getenv('MONGO_INITDB_ROOT_PASSWORD')
mongo_init_db = os.getenv('mongo_init_db')
MYSQL_ROOT_PASSWORD = os.getenv('MYSQL_ROOT_PASSWORD')
```

## Problem 1

**This problem requires you to create Markdown tables**

To create a table in a markdown cell, I recommend using the markdown table generator here: https://www.tablesgenerator.com/markdown_tables. This interface allows you to choose the number of rows and columns, fill in those rows and colums, and push the "generate" button. The website will display markdown table code that looks like:

```
| Day       | Temp | Rain |
|-----------|------|------|
| Monday    | 74   | No   |
| Tuesday   | 58   | Yes  |
| Wednesday | 76   | No   |
```

Copy the markdown code and paste it into a markdown cell in your notebook. Markdown will read the code and display a table that looks like this:

| Day | Temp | Rain |
|---|---|---|
| Monday | 74 | No |
| Tuesday | 58 | Yes |
| Wednesday | 76 | No |

Suppose that we have (fake) data on people who were hospitalized and received at least one prescription for a medication. Here are ten records in the data:

(If this table gets cut off in the PDF, please look at the .ipynb notebook file on the module 6 page on Canvas)

| patient_name | date_of_birth | prescribed_drug | prior_conditions | patient_sex | patient_insurance | drug_maker | drug_cos |
|---|---|---|---|---|---|---|---|
| Nkemdilim Arendonk | 2/21/1962 | Amoxil | [Pneumonia, Diabetes] | M | Aetna | USAntibiotics | 14.62 |
| Nkemdilim Arendonk | 2/21/1962 | Micronase | [Pneumonia, Diabetes] | M | Aetna | Pfizer | 20.55 |

| patient_name | date_of_birth | prescribed_drug | prior_conditions | patient_sex | patient_insurance | drug_maker | drug_cos |
|---|---|---|---|---|---|---|---|
| Raniero Coumans | 8/15/1990 | Zosyn | [Appendicitis, Crohn's disease] | M | Cigna | Baxter International Inc | 394.00 |
| Raniero Coumans | 8/15/1990 | Humira | [Appendicitis, Crohn's disease] | M | Cigna | Abbvie | 7000.00 |
| Mizuki Debenham | 3/12/1977 | Inlyta | [Kidney Cancer] | F | Kaiser Permanente | Pfizer | 21644.00 |
| Zoë De Witt | 11/23/1947 | Atenolol | [Cardiomyopathy, Diabetes, Sciatica] | F | Medicare | Mylan Pharmaceuticals | 10.58 |
| Zoë De Witt | 11/23/1947 | Micronase | [Cardiomyopathy, Diabetes, Sciatica] | F | Medicare | Pfizer | 20.55 |
| Zoë De Witt | 11/23/1947 | Demerol | [Cardiomyopathy, Diabetes, Sciatica] | F | Medicare | Pfizer | 37.50 |
| Bonnie Hooper | 7/4/1951 | Xeloda | [Pancreatic Cancer, Sciatica] | F | Blue Cross Blue Shield | Genentech | 860.00 |
| Bonnie Hooper | 7/4/1951 | Demerol | [Pancreatic Cancer, Sciatica] | F | Blue Cross Blue Shield | Pfizer | 37.50 |

The columns in this dataset are:

- **patient_name**: The patient's name
- **date_of_birth**: The patient's date of birth
- **prescribed_drug**: The brand name of the medication that patient has been prescribed
- **prior_conditions**: A list of the conditions that the patient had been diagnosed with prior to the patient's hospitalization
- **patient_sex**: The patient's sex
- **patient_insurance**: The company responsible for the patient's health insurance coverage
- **drug_maker**: The company that manufactures the prescribed drug
- **drug_cost**: The cost of the prescribed drug
- **attending_physician**: The name of the attending physician for the patient
- **AP_medschool**: The name of the school where the attending physician got a medical degree
- **AP_years_experience**: The attending physician's number of years of experience post-residency
- **hospital**: The hospital where the attending physicial is employed
- **hospital_location**: The location of the hospital

For this problem, assume that

1. No two rows in this table share both the same patient and the same prescribed drug.
2. Some patients in the data share the same name, but no two patients in the data share the same name and date of birth.
3. No two different drugs share the same brand name.
4. No two attending physicians have the same name, and every attending physician is employed at only one hospital.
5. No two hospitals share the same name, and every hospital exists at only one location.
6. Each patient has only one attending physician. (In real-world applications we may want to design a database that allows for multiple hospitalizations for some patients, but here we'll keep it simpler by assuming each patient has one hospitalization with one attending physician.)

In [2]:
```
data = {
    'patient_name': ['Nkemdilim Arendonk', 'Nkemdilim Arendonk', 'Raniero Coumans', 'Raniero Coumans', 'Mizu
    'date_of_birth': ['2/21/1962', '2/21/1962', '8/15/1990', '8/15/1990', '3/12/1977', '11/23/1947', '11/23/
    'prescribed_drug': ['Amoxil', 'Micronase', 'Zosyn', 'Humira', 'Inlyta', 'Atenolol', 'Micronase', 'Demero
    'prior_conditions': [['Pneumonia', 'Diabetes'], ['Pneumonia', 'Diabetes'], ['Appendicitis', 'Crohn\'s di
    'patient_sex': ['M', 'M', 'M', 'M', 'F', 'F', 'F', 'F', 'F', 'F'],
    'patient_insurance': ['Aetna', 'Aetna', 'Cigna', 'Cigna', 'Kaiser Permanente', 'Medicare', 'Medicare', '
    'drug_maker': ['USAntibiotics', 'Pfizer', 'Baxter International Inc', 'Abbvie', 'Pfizer', 'Mylan Pharmac
    'drug_cost': [14.62, 20.55, 394.00, 7000.00, 21644.00, 10.58, 20.55, 37.50, 860.00, 37.50],
    'attending_physician': ['Earnest Caro', 'Earnest Caro', 'Pamela English', 'Pamela English', 'Lewis Conti
    'AP_medschool': ['University of California (Irvine)', 'University of California (Irvine)', 'University o
    'AP_years_experience': [14, 14, 29, 29, 8, 17, 17, 17, 36, 36],
    'hospital': ['UPMC Presbyterian Shadyside', 'UPMC Presbyterian Shadyside', 'Northwestern Memorial Hospit
    'hospital_location': ['Pittsburgh, PA', 'Pittsburgh, PA', 'Chicago, IL', 'Chicago, IL', 'Houston, TX', '
}
```

In [3]:
```
data_df = pd.DataFrame(data)
data_df
```

Out[3]:

| | patient_name | date_of_birth | prescribed_drug | prior_conditions | patient_sex | patient_insurance | drug_maker | drug_c |
|---|---|---|---|---|---|---|---|---|
| **0** | Nkemdilim Arendonk | 2/21/1962 | Amoxil | [Pneumonia, Diabetes] | M | Aetna | USAntibiotics | 14 |
| **1** | Nkemdilim Arendonk | 2/21/1962 | Micronase | [Pneumonia, Diabetes] | M | Aetna | Pfizer | 20 |
| **2** | Raniero Coumans | 8/15/1990 | Zosyn | [Appendicitis, Crohn's disease] | M | Cigna | Baxter International Inc | 394 |
| **3** | Raniero Coumans | 8/15/1990 | Humira | [Appendicitis, Crohn's disease] | M | Cigna | Abbvie | 7000 |
| **4** | Mizuki Debenham | 3/12/1977 | Inlyta | [Kidney Cancer] | F | Kaiser Permanente | Pfizer | 21644 |
| **5** | Zoë De Witt | 11/23/1947 | Atenolol | [Cardiomyopathy, Diabetes, Sciatica] | F | Medicare | Mylan Pharmaceuticals | 10 |
| **6** | Zoë De Witt | 11/23/1947 | Micronase | [Cardiomyopathy, Diabetes, Sciatica] | F | Medicare | Pfizer | 20 |
| **7** | Zoë De Witt | 11/23/1947 | Demerol | [Cardiomyopathy, Diabetes, Sciatica] | F | Medicare | Pfizer | 37 |
| **8** | Bonnie Hooper | 7/4/1951 | Xeloda | [Pancreatic Cancer, Sciatica] | F | Blue Cross Blue Shield | Genentech | 860 |
| **9** | Bonnie Hooper | 7/4/1951 | Demerol | [Pancreatic Cancer, Sciatica] | F | Blue Cross Blue Shield | Pfizer | 37 |

## Part a

Rearrange the data into a group of data tables that together meet the requirements of first normal form. [2 points]

In [4]:
```
patients = data_df.loc[:,('patient_name', 'date_of_birth', 'patient_sex', 'prior_conditions','patient_insura
patients['patient_id'] = patients.index + 1
patients
```

Out[4]:

| | patient_name | date_of_birth | patient_sex | prior_conditions | patient_insurance | prescribed_drug | patient_id |
|---|---|---|---|---|---|---|---|
| **0** | Nkemdilim Arendonk | 2/21/1962 | M | [Pneumonia, Diabetes] | Aetna | Amoxil | 1 |
| **1** | Nkemdilim Arendonk | 2/21/1962 | M | [Pneumonia, Diabetes] | Aetna | Micronase | 2 |
| **2** | Raniero Coumans | 8/15/1990 | M | [Appendicitis, Crohn's disease] | Cigna | Zosyn | 3 |
| **3** | Raniero Coumans | 8/15/1990 | M | [Appendicitis, Crohn's disease] | Cigna | Humira | 4 |
| **4** | Mizuki Debenham | 3/12/1977 | F | [Kidney Cancer] | Kaiser Permanente | Inlyta | 5 |
| **5** | Zoë De Witt | 11/23/1947 | F | [Cardiomyopathy, Diabetes, Sciatica] | Medicare | Atenolol | 6 |
| **6** | Zoë De Witt | 11/23/1947 | F | [Cardiomyopathy, Diabetes, Sciatica] | Medicare | Micronase | 7 |
| **7** | Zoë De Witt | 11/23/1947 | F | [Cardiomyopathy, Diabetes, Sciatica] | Medicare | Demerol | 8 |
| **8** | Bonnie Hooper | 7/4/1951 | F | [Pancreatic Cancer, Sciatica] | Blue Cross Blue Shield | Xeloda | 9 |
| **9** | Bonnie Hooper | 7/4/1951 | F | [Pancreatic Cancer, Sciatica] | Blue Cross Blue Shield | Demerol | 10 |

In [5]:
```
drugs = data_df.loc[:,('prescribed_drug', 'drug_maker', 'drug_cost')]
drugs['drug_id'] = drugs.index + 1
drugs
```

Out[5]:

| | prescribed_drug | drug_maker | drug_cost | drug_id |
|---|---|---|---|---|
| **0** | Amoxil | USAntibiotics | 14.62 | 1 |
| **1** | Micronase | Pfizer | 20.55 | 2 |
| **2** | Zosyn | Baxter International Inc | 394.00 | 3 |
| **3** | Humira | Abbvie | 7000.00 | 4 |
| **4** | Inlyta | Pfizer | 21644.00 | 5 |
| **5** | Atenolol | Mylan Pharmaceuticals | 10.58 | 6 |
| **6** | Micronase | Pfizer | 20.55 | 7 |
| **7** | Demerol | Pfizer | 37.50 | 8 |
| **8** | Xeloda | Genentech | 860.00 | 9 |
| **9** | Demerol | Pfizer | 37.50 | 10 |

In [6]:
```
doctors = data_df.loc[:,('attending_physician', 'AP_medschool', 'AP_years_experience')]
doctors['physician_id'] = doctors.index + 1
doctors
```

Out[6]:

| | attending_physician | AP_medschool | AP_years_experience | physician_id |
|---|---|---|---|---|
| **0** | Earnest Caro | University of California (Irvine) | 14 | 1 |
| **1** | Earnest Caro | University of California (Irvine) | 14 | 2 |
| **2** | Pamela English | University of Michigan | 29 | 3 |
| **3** | Pamela English | University of Michigan | 29 | 4 |
| **4** | Lewis Conti | North Carolina State University | 8 | 5 |
| **5** | Theresa Dahlmans | Lake Erie College of Medicine | 17 | 6 |
| **6** | Theresa Dahlmans | Lake Erie College of Medicine | 17 | 7 |
| **7** | Theresa Dahlmans | Lake Erie College of Medicine | 17 | 8 |
| **8** | Steven Garbutt | Ohio State University | 36 | 9 |
| **9** | Steven Garbutt | Ohio State University | 36 | 10 |

In [7]:
```
hospitals = data_df.loc[:,('hospital', 'hospital_location')]
hospitals['hospital_id'] = hospitals.index + 1
hospitals
```

Out[7]:

| | hospital | hospital_location | hospital_id |
|---|---|---|---|
| **0** | UPMC Presbyterian Shadyside | Pittsburgh, PA | 1 |
| **1** | UPMC Presbyterian Shadyside | Pittsburgh, PA | 2 |
| **2** | Northwestern Memorial Hospital | Chicago, IL | 3 |
| **3** | Northwestern Memorial Hospital | Chicago, IL | 4 |
| **4** | Houston Methodist Hospital | Houston, TX | 5 |
| **5** | Mount Sinai Hospital | New York, NY | 6 |
| **6** | Mount Sinai Hospital | New York, NY | 7 |
| **7** | Mount Sinai Hospital | New York, NY | 8 |
| **8** | UCSF Medical Center | San Francisco, CA | 9 |
| **9** | UCSF Medical Center | San Francisco, CA | 10 |

In [8]:
```
prescriptions = data_df[['patient_name', 'prescribed_drug', 'attending_physician', 'hospital']]
```

In [9]:
```
prescriptions = prescriptions.merge(patients[['patient_name', 'patient_id']], on='patient_name')
prescriptions = prescriptions.merge(drugs[['prescribed_drug', 'drug_id']], on='prescribed_drug')
prescriptions = prescriptions.merge(doctors[['attending_physician', 'physician_id']], on='attending_physicia
prescriptions = prescriptions.merge(hospitals[['hospital', 'hospital_id']], on='hospital')

prescriptions
```

Out[9]:

| | patient_name | prescribed_drug | attending_physician | hospital | patient_id | drug_id | physician_id | hospital_id |
|---|---|---|---|---|---|---|---|---|
| **0** | Nkemdilim Arendonk | Amoxil | Earnest Caro | UPMC Presbyterian Shadyside | 1 | 1 | 1 | 1 |
| **1** | Nkemdilim Arendonk | Amoxil | Earnest Caro | UPMC Presbyterian Shadyside | 1 | 1 | 1 | 2 |
| **2** | Nkemdilim Arendonk | Amoxil | Earnest Caro | UPMC Presbyterian Shadyside | 1 | 1 | 2 | 1 |
| **3** | Nkemdilim Arendonk | Amoxil | Earnest Caro | UPMC Presbyterian Shadyside | 1 | 1 | 2 | 2 |
| **4** | Nkemdilim Arendonk | Amoxil | Earnest Caro | UPMC Presbyterian Shadyside | 2 | 1 | 1 | 1 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **195** | Bonnie Hooper | Demerol | Steven Garbutt | UCSF Medical Center | 10 | 8 | 10 | 10 |
| **196** | Bonnie Hooper | Demerol | Steven Garbutt | UCSF Medical Center | 10 | 10 | 9 | 9 |
| **197** | Bonnie Hooper | Demerol | Steven Garbutt | UCSF Medical Center | 10 | 10 | 9 | 10 |
| **198** | Bonnie Hooper | Demerol | Steven Garbutt | UCSF Medical Center | 10 | 10 | 10 | 9 |
| **199** | Bonnie Hooper | Demerol | Steven Garbutt | UCSF Medical Center | 10 | 10 | 10 | 10 |

200 rows × 8 columns

### Part b

Rearrange the data on the five patients into a group of data tables that together meet the requirements of second normal form. [2 points]

In [10]:
```
second_nf_df = prescriptions[['patient_id', 'drug_id', 'physician_id', 'hospital_id']]
second_nf_df
```

Out[10]:

| | patient_id | drug_id | physician_id | hospital_id |
|---|---|---|---|---|
| **0** | 1 | 1 | 1 | 1 |
| **1** | 1 | 1 | 1 | 2 |
| **2** | 1 | 1 | 2 | 1 |
| **3** | 1 | 1 | 2 | 2 |
| **4** | 2 | 1 | 1 | 1 |
| **...** | ... | ... | ... | ... |
| **195** | 10 | 8 | 10 | 10 |
| **196** | 10 | 10 | 9 | 9 |
| **197** | 10 | 10 | 9 | 10 |
| **198** | 10 | 10 | 10 | 9 |
| **199** | 10 | 10 | 10 | 10 |

200 rows × 4 columns

### Part c

Rearrange the data into a group of data tables that together meet the requirements of third normal form. [2 points]

In [11]:
```
insurance = data_df[['patient_insurance']].drop_duplicates().reset_index(drop=True)
insurance['insurance_id'] = insurance.index + 1
insurance
```

Out[11]:

| | patient_insurance | insurance_id |
|---|---|---|
| **0** | Aetna | 1 |
| **1** | Cigna | 2 |
| **2** | Kaiser Permanente | 3 |
| **3** | Medicare | 4 |
| **4** | Blue Cross Blue Shield | 5 |

In [12]:
```
patients = patients.merge(insurance, on='patient_insurance')
patients = patients[['patient_id', 'patient_name', 'date_of_birth', 'prior_conditions', 'patient_sex', 'insu
```

In [13]:
```
prescriptions = data_df.loc[:,('patient_name', 'prescribed_drug', 'attending_physician', 'hospital')]
```

In [14]:
```
prescriptions = prescriptions.merge(patients[['patient_name', 'patient_id']], on='patient_name')
prescriptions = prescriptions.merge(drugs[['prescribed_drug', 'drug_id']], on='prescribed_drug')
prescriptions = prescriptions.merge(doctors[['attending_physician', 'physician_id']], on='attending_physicia
prescriptions = prescriptions.merge(hospitals[['hospital', 'hospital_id']], on='hospital')
```

In [15]:
```
third_nf_df = prescriptions.loc[:,('patient_id', 'drug_id', 'physician_id', 'hospital_id')]
third_nf_df
```

Out[15]:

| | patient_id | drug_id | physician_id | hospital_id |
|---|---|---|---|---|
| **0** | 1 | 1 | 1 | 1 |
| **1** | 1 | 1 | 1 | 2 |
| **2** | 1 | 1 | 2 | 1 |
| **3** | 1 | 1 | 2 | 2 |
| **4** | 2 | 1 | 1 | 1 |
| **...** | ... | ... | ... | ... |
| **195** | 10 | 8 | 10 | 10 |
| **196** | 10 | 10 | 9 | 9 |
| **197** | 10 | 10 | 9 | 10 |
| **198** | 10 | 10 | 10 | 9 |
| **199** | 10 | 10 | 10 | 10 |

200 rows × 4 columns

## Problem 2

For this problem, create ER diagrams of the database you created in problem 1, part c using https://dbdocs.io/. Make sure you install DBDocs on your system by following these instructions: https://dbdocs.io/docs

### Part a

Write code using the database markup language (DBML) that represents all of the tables in this database and the connections between the tables. Paste your DBML code in a markdown cell in your notebook, contained within three backticks to begin and end the code snippet, as shown in the cell below.

Two good resources to help you:

1. The example on the Getting Started page on dbdocs.io: https://dbdocs.io/docs
2. The full syntax guide for DBML: https://dbml.dbdiagram.io/docs/#project-definition

A few notes:

- Make sure to specify the data type for each column in each table. Use varchar for strings/text, int for integers, and float for numeric data with decimals.
- You will probably find it useful to alias each table with one or two letters, such as: Table PRESCRIPTIONS as PR. That will allow you to use PR to refer to the PRESCRIPTIONS table, for example, in the Reference statements to link tables together.
- Use the syntax [pk] after a column name and data type to designate the columns that are primary keys in each table.
- To draw the lines linking one table to another, use the Ref: syntax.
  - If many rows from the left table match to one row in the right table, use the "many to one" symbol >
  - If one row from the left table matches to many rows in the right table, use the "one to many" symbol <
  - If one row from the left table matches to one row in the right table, use the "one to one" symbol -
  - If many rows from the left table match to many rows in the right table, use the "many to many" symbol <>

[2 points]

```
Project hospitalizations {
  database_type: 'PostgreSQL'
  Note: '''
    # fake data on people who were hospitalized and received at least one prescription for
a medication.
    **Database created for DS6001 July 2024**
  '''
```

```
}
Table patients {
    patient_id int [pk, increment]
    patient_name varchar
    date_of_birth date
    prior_conditions varchar
    patient_sex char
    insurance_id int
}

Table drugs {
    drug_id int [pk, increment]
    prescribed_drug varchar
    drug_maker varchar
    drug_cost float
}

Table doctors {
    physician_id int [pk, increment]
    attending_physician varchar
    AP_medschool varchar
    AP_years_experience int
}

Table hospitals {
    hospital_id int [pk, increment]
    hospital varchar
    hospital_location varchar
}

Table insurance {
    insurance_id int [pk, increment]
    patient_insurance varchar
}

Table prescriptions {
    prescription_id int [pk, increment]
    patient_id int
    drug_id int
    physician_id int
    hospital_id int
}


Ref: prescriptions.patient_id > patients.patient_id
Ref: prescriptions.drug_id > drugs.drug_id
Ref: prescriptions.physician_id > doctors.physician_id
Ref: prescriptions.hospital_id > hospitals.hospital_id
Ref: patients.insurance_id > insurance.insurance_id
```

## Part b

Use the instructions on DBDocs.io (https://dbdocs.io/docs) to create a website for your ER diagram. Type the URL for your website in a markdown cell here. [1 point]

https://dbdocs.io/kianadane/hospitalizations

## Problem 3

For this problem, you will download the individual CSV files that comprise a relational database on album reviews from Pitchfork Magazine, collected via webscraping by Nolan B. Conaway, and use them to initialize local databases using SQlite, MySQL, and PostgreSQL.

The following code of code will download the CSV files. Please run this as is:

In [16]:
```
url = "https://github.com/nolanbconaway/pitchfork-data/raw/master/pitchfork.db"
pfork = wget.download(url)
pitchfork = sqlite3.connect(pfork)
for t in ['artists','content','genres','labels','reviews','years']:
    datatable = pd.read_sql_query("SELECT * FROM {tab}".format(tab=t), pitchfork)
    datatable.to_csv("{tab}.csv".format(tab=t))
```

Note: this code downloaded a SQlite database and extracted the tables, saving each one as a CSV. That seems backwards, as the purpose of this exercise is to create databases. But the point here is to practice creating databases from individual data frames. Next we load the CSVs to create the data frames in Python:

In [17]:
```
reviews = pd.read_csv("reviews.csv")
artists = pd.read_csv("artists.csv")
content = pd.read_csv("content.csv")
genres = pd.read_csv("genres.csv")
labels = pd.read_csv("labels.csv")
years = pd.read_csv("years.csv")
```

### Part a

Initialize a new database using SQlite and the `sqlite3` library. Add the six dataframes to this database. Then issue the following query to the database

```
SELECT title, artist, score FROM reviews WHERE score=10
```

using two methods: first, using the `.cursor()` method, and second using `pd.read_sql_query()`. Finally, commit your changes to the database and close the database. (If you get a warning about spaces in the column names, feel free to ignore it this time.) [2 points]

In [18]:
```
reviews.to_sql('reviews', pitchfork, if_exists='replace', index=False)
artists.to_sql('artists', pitchfork, if_exists='replace', index=False)
content.to_sql('content', pitchfork, if_exists='replace', index=False)
genres.to_sql('genres', pitchfork, if_exists='replace', index=False)
labels.to_sql('labels', pitchfork, if_exists='replace', index=False)
years.to_sql('years', pitchfork, if_exists='replace', index=False)
```

Out[18]:
```
19108
```

## Issuing the query using the .cursor() method

In [19]:
```
cursor = pitchfork.cursor()
query = '''SELECT title, artist, score FROM reviews WHERE score=10'''
cursor.execute(query)
rows = cursor.fetchall()

rows[:5]
```

Out[19]:
```
[('metal box', 'public image ltd', 10.0),
 ('blood on the tracks', 'bob dylan', 10.0),
 ('another green world', 'brian eno', 10.0),
 ('songs in the key of life', 'stevie wonder', 10.0),
 ('in concert', 'nina simone', 10.0)]
```

## Issuing query using pd.read_sql_query

In [20]:
```
pandas_query = pd.read_sql_query(query, pitchfork)
pandas_query.head(5)
```

Out[20]:

| | title | artist | score |
|---|---|---|---|
| 0 | metal box | public image ltd | 10.0 |
| 1 | blood on the tracks | bob dylan | 10.0 |
| 2 | another green world | brian eno | 10.0 |
| 3 | songs in the key of life | stevie wonder | 10.0 |
| 4 | in concert | nina simone | 10.0 |

In [21]:
```
pitchfork.commit()
pitchfork.close()
```

### Part b

Follow the instructions in the Jupyter notebook for this module to install MySQL and `mysql.connector` on your computer. Make sure the MySQL server is running. Then import `mysql.connector` and do all of the tasks listed for part a using a MySQL database (including commiting changes and closing the database connection). Take steps to hide your password - do not let it display in your notebook. [3 points]

In [22]:
```
MYSQL_ROOT_PASSWORD = os.getenv('MYSQL_ROOT_PASSWORD')
```

In [23]:
```
dbserver = mysql.connector.connect(
    user='root',
    password=MYSQL_ROOT_PASSWORD,
    host='localhost',
    port='3306',
    database= 'hpz_db'
    #auth_plugin=MYSQL_ROOT_PASSWORD
)

if dbserver.is_connected():
    print("Connected to the MySQL database")
else:
    print("Failed to connect to the MySQL database")
```

```
Connected to the MySQL database
```

In [24]:
```
hpz_db = 'hpz_db'
engine = create_engine(f'mysql+mysqlconnector://root:{MYSQL_ROOT_PASSWORD}@localhost:3306/{hpz_db}')
```

In [25]:
```
reviews.to_sql('reviews', con=engine, chunksize=1000, if_exists='replace', index=False)
artists.to_sql('artists', con=engine, chunksize=1000, if_exists='replace', index=False)
content.to_sql('content', con=engine, chunksize=1000, if_exists='replace', index=False)
genres.to_sql('genres', con=engine, chunksize=1000, if_exists='replace', index=False)
labels.to_sql('labels', con=engine, chunksize=1000, if_exists='replace', index=False)
years.to_sql('years', con=engine, chunksize=1000, if_exists='replace', index=False)
```

Out[25]:
```
19108
```

## Issuing the query using the .cursor() method

In [26]:
```
cursor2 = dbserver.cursor()
query = '''SELECT title, artist, score FROM reviews WHERE score=10'''
cursor2.execute(query)
results = cursor2.fetchall()

results[:10]
```

Out[26]:
```
[('metal box', 'public image ltd', 10.0),
 ('blood on the tracks', 'bob dylan', 10.0),
 ('another green world', 'brian eno', 10.0),
 ('songs in the key of life', 'stevie wonder', 10.0),
 ('in concert', 'nina simone', 10.0),
 ("tonight's the night", 'neil young', 10.0),
 ('hounds of love', 'kate bush', 10.0),
 ('sign "o" the times', 'prince', 10.0),
 ('1999', 'prince', 10.0),
 ('purple rain', 'prince, the revolution', 10.0)]
```

## Issuing the query using pd.read_sql_query

In [27]:
```
pandas_query = pd.read_sql_query(query, con=engine)
pandas_query.head(10)
```

Out[27]:

| | title | artist | score |
|---|---|---|---|
| **0** | metal box | public image ltd | 10.0 |
| **1** | blood on the tracks | bob dylan | 10.0 |
| **2** | another green world | brian eno | 10.0 |
| **3** | songs in the key of life | stevie wonder | 10.0 |
| **4** | in concert | nina simone | 10.0 |
| **5** | tonight's the night | neil young | 10.0 |
| **6** | hounds of love | kate bush | 10.0 |
| **7** | sign "o" the times | prince | 10.0 |
| **8** | 1999 | prince | 10.0 |
| **9** | purple rain | prince, the revolution | 10.0 |

In [28]:
```
dbserver.commit()
dbserver.close()
```

### Part c

Follow the instructions in the Jupyter notebook for this module to install PostgreSQL and `psycopg2` on your computer. Then import `psycopg2` and do all of the tasks listed for part a using a PostgreSQL database (including commiting changes and closing the database connection). Take steps to hide your password - do not let it display in your notebook. [3 points]

In [29]:
```
import psycopg2
from psycopg2 import sql
POSTGRES_PASSWORD = os.getenv('POSTGRES_PASSWORD')
DATABASE_NAME = hpz_db
```

In [30]:
```
conn = psycopg2.connect(
    dbname='hpz_db',
    user='postgres',
    password=POSTGRES_PASSWORD,
    host='localhost'

)
conn.autocommit = True
```

In [31]:
```
engine2 = create_engine(f'postgresql+psycopg2://postgres:{POSTGRES_PASSWORD}@localhost:5432/{hpz_db}')
```

In [32]:
```
reviews.to_sql('reviews', con=engine2, if_exists='replace', index=False)
artists.to_sql('artists', con=engine2, if_exists='replace', index=False)
content.to_sql('content', con=engine2, if_exists='replace', index=False)
genres.to_sql('genres', con=engine2, if_exists='replace', index=False)
```

```
labels.to_sql('labels', con=engine2, if_exists='replace', index=False)
years.to_sql('years', con=engine2, if_exists='replace', index=False)
```

Out[32]: 108

## Issuing the query using the .cursor() method

In [33]:
```
cursor4 = conn.cursor()
cursor4.execute(query)
results = cursor4.fetchall()
results[:10]
```

Out[33]:
```
[('metal box', 'public image ltd', 10.0),
 ('blood on the tracks', 'bob dylan', 10.0),
 ('another green world', 'brian eno', 10.0),
 ('songs in the key of life', 'stevie wonder', 10.0),
 ('in concert', 'nina simone', 10.0),
 ("tonight's the night", 'neil young', 10.0),
 ('hounds of love', 'kate bush', 10.0),
 ('sign "o" the times', 'prince', 10.0),
 ('1999', 'prince', 10.0),
 ('purple rain', 'prince, the revolution', 10.0)]
```

## Issuing the query using pandas

In [34]:
```
filtered = reviews[reviews['score'] == 10]
filtered = filtered.loc[:, ['title', 'artist', 'score']]
filtered.head(10)
```

Out[34]:

| | title | artist | score |
|---|---|---|---|
| **191** | metal box | public image ltd | 10.0 |
| **200** | blood on the tracks | bob dylan | 10.0 |
| **355** | another green world | brian eno | 10.0 |
| **451** | songs in the key of life | stevie wonder | 10.0 |
| **530** | in concert | nina simone | 10.0 |
| **654** | tonight's the night | neil young | 10.0 |
| **706** | hounds of love | kate bush | 10.0 |
| **857** | sign "o" the times | prince | 10.0 |
| **858** | 1999 | prince | 10.0 |
| **861** | purple rain | prince, the revolution | 10.0 |

In [35]:
```
conn.close()
```

### Problem 4

Colin Mitchell is a web-developer and artist who has a bunch of cool projects that play with what data can do on the internet. One of his projects is Today in History, which provides an API to access all the Wikipedia pages for historical events that happened on this day in JSON format. The records in this JSON are stored in the `['data']['events']` path. Here's the first listing for today:

In [36]:
```
import requests
```

In [37]:
```
history = requests.get("https://history.muffinlabs.com/date", verify=False)
```

```
/Users/kianadane/Documents/GitHub/ds6001databases/.conda/lib/python3.12/site-packages/urllib3/connectionpool.
py:1099: InsecureRequestWarning: Unverified HTTPS request is being made to host 'history.muffinlabs.com'. Add
ing certificate verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-usag
e.html#tls-warnings
  warnings.warn(
```

For this problem, you will use MongoDB and the `pymongo` library to create a local document store NoSQL database containing these historical events.

Follow the instructions in the Jupyter notebook for this module to install MongoDB and `pymongo` on your computer. Make sure the local MongoDB server is running. Then import `pymongo`, connect to the local MongoDB client, create a database named "history" and a collection within that database named "today". Insert all of the records in `events` into this collection. Then issue the following query to find all of the records whose text contain the word "Virginia":

```
query = {
    "text":{
        "$regex": 'Virginia'
    }
}
```

If there are no results that contain the word "Virginia", choose a different work like "England" or "China". Display the count of the number of documents that match this query, display the output of the query, and generate a JSON formatted variable containing the output. [3 points]

In [38]:
```
from pymongo import MongoClient
load_dotenv
```

Out[38]:
```
<function dotenv.main.load_dotenv(dotenv_path: Union[str, ForwardRef('os.PathLike[str]'), NoneType] = None,
stream: Optional[IO[str]] = None, verbose: bool = False, override: bool = False, interpolate: bool = True,
encoding: Optional[str] = 'utf-8') -> bool>
```

In [39]:
```
mongo_init_db = os.getenv('MONGO_INIT_DB')
mongo_user = os.getenv('MONGO_USER')
mongo_password = os.getenv('MONGO_PASSWORD')
```

In [40]:
```
client = MongoClient(f"mongodb://{MONGO_INITDB_ROOT_USERNAME}:{MONGO_INITDB_ROOT_PASSWORD}@localhost:27017/"
try:
    databases = client.list_database_names()
    print("Connected to MongoDB!")
    print("Databases:", databases)
except Exception as e:
    print("Could not connect to MongoDB:", e)
```

```
Connected to MongoDB!
Databases: ['admin', 'config', 'local', 'mongo_init_db']
```

In [41]:
```
history_db = client['History']
```

In [42]:
```
today = history_db['Events']
```

In [43]:
```
import requests
history = requests.get("https://history.muffinlabs.com/date")
history
```

Out[43]:
```
<Response [200]>
```

In [44]:
```
history = json.loads(history.text)
```

In [52]:
```
today = history['data']['Events']
today = pd.DataFrame(today)
```

In [57]:
```
query = 'China'

results = today[today['text'].str.contains(query, case=False, na=False)]
results
```

Out[57]:

| | year | text | html | no_year_html | links |
|---|---|---|---|---|---|
| **35** | 1941 | World War II: In response to the Japanese occu... | 1941 - World War II: In response to the Japane... | World War II: In response to the Japanese occu... | [{'title': 'French Indochina', 'link': 'https:... |

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

## Problem 5 [No points, no need to write anything here, but do it anyway!]

When you are done working, go to the same terminal window you used to launch the databases with `docker compose up`. Here, type CONTROL + C on your keyboard to shut down the container.

Next type

```
docker compose down
```

This step removes extra database software, networks, volumes, etc. running on your computer. If you don't need them, don't clog your computer.

Whenever you need to work with databases, return to the terminal, navigate to this folder and type

```
docker compose up
```

to bring all these resources back online.