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. To receive full credit, make sure you address every part of the problem, make sure your document is formatted in a clean and professional way, and make sure the notebook is converted to a PDF and submitted to Gradescope according to these instructions: https://docs.google.com/document/d/1B9ZkK7n_hP_hQ9lIGm31Web4S6hGnwMz9Ad7EWm3N50/edit?usp=sharing (https://docs.google.com/document/d/1B9ZkK7n_hP_hQ9lIGm31Web4S6hGnwMz9Ad7EWm3N50/edit?usp=sharing).

This assignment requires you to include tables and images.

To create a table in a markdown cell, I recommend using the markdown table generator here: https://www.tablesgenerator.com/markdown_tables (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

To put an image into a markdown cell in a Jupyter notebook, save the image as a .png or .jpg file in the same folder where you have saved your Jupyter notebook, and use markdown code that looks like this:

```
![](imagefile.png)
```

where you will need to replace `imagefile.png` with the name of your own image file. Alternatively, if you want to control the size of the image in your notebook, type the following code on its own line in the markdown cell:

```
<img src="imagefile.png" width="600">
```

Here the `width` option allows you to control the size of the image by making this number larger or smaller. When converting the notebook to

PDF format

, make sure that the images display correctly in the PDF prior to submitting to Gradescope.

Problem 0

Import the following libraries, load the `.env` file where you store your passwords (see the notebook for module 4 for details), and turn off the error tracebacks to make errors easier to read:

In [173]:

```
#!jupyter nbconvert --execute --to html labassignment6.ipynb
```

In [103]:

```
import numpy as np
import pandas as pd
import wget
import sqlite3
import sqlalchemy
import requests
import json
import os
import sys
import dotenv
os.chdir("/Users/dmitrymikhaylov/Documents/learn/uva/spring2022/DS6001/surfing_data_pipeline/M6") # change to the directory where your .env file is
dotenv.load_dotenv() # register the .env file where passwords are stored
#sys.tracebacklimit = 0 # turn off the error tracebacks
```

Out[103]:

```
True
```

Problem 1

Suppose that we have (fake) data on people who are currently being hospitalized. Here are five records in the data:

patient	conditions	dateofbirth	age	sex	attendingphysician	APmedschool	APyearsexperiece	hosp
Nkemdilim Arendonk	[Pneumonia, Diabetes]	2/21/1962	58	M	Earnest Caro	University of California (Irvine)	14	UP Presbyte Shadys
Raniero Coumans	[Appendicitis, Crohn's disease]	8/15/1990	29	M	Pamela English	University of Michigan	29	Northwest Memo Hosp
Mizuki Debenham	[Kidney Cancer]	3/12/1977	43	F	Lewis Conti	North Carolina State University	8	Hous Metho Hosp
Zoë De Witt	[Cardiomyopathy, Diabetes, Sciatica]	11/23/1947	72	F	Theresa Dahlmans	Lake Erie College of Medicine	17	Mount S Hosp
Bonnie Hooper	[Pancreatic Cancer, Sciatica]	7/4/1951	68	F	Steven Garbutt	Ohio State University	36	UC Med Cer

The columns in this dataset are:

- **patient**: Patient name
- **conditions**: A list of the conditions that are relevant to the patient's hospitalization
- **dateofbirth**: The patient's date of birth
- **age**: The patient's age
- **sex**: The patient's sex
- **attendingphysician**: The name of the attending physician for the patient
- **APmedschool**: The name of the school where the attending physician got a medical degree
- **APyearsexperiece**: The attending physician's number of years of experience post-residency
- **hospital**: The hospital where the attending physicial is employed
- **hospitallocation**: The location of the hospital

For this problem, assume that

1. Some people in the data share the same name, but no two people in the data share the same name and date of birth.
2. Every attending physician is employed at only one hospital.
3. Every hospital exists at only one location.
4. There's more than one doctor with the same name, but there are no doctors with the same name that work at the same hospital.

Part a

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

1NF:

- Data is stored in tables with rows uniquely identified by a primary key
- Data within each table is stored in individual columns in its most reduced form
- There are no repeating groups

patient_id	patient	condition1	condition2	condition3	dateofbirth	age	sex	attendingphysician	APmedsc
1	Nkemdilim Arendonk	Pneumonia	Diabetes	NA	2/21/1962	58	M	Earnest Caro	Univers Calif (I
2	Raniero Coumans	Appendicitis	Crohn's disease	NA	8/15/1990	29	M	Pamela English	Univers Micl
3	Mizuki Debenham	Kidney Cancer	NA	NA	3/12/1977	43	F	Lewis Conti	North Ca Univ
4	Zoë De Witt	Cardiomyopathy	Diabetes	Sciatica	11/23/1947	72	F	Theresa Dahlmans	Lake Colle Mec
5	Bonnie Hooper	Pancreatic Cancer	Sciatica	NA	7/4/1951	68	F	Steven Garbutt	Ohio Univ

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]

2NF:

- Everything from 1NF
- Only data that relates to a table’s primary key is stored in each table

Table: patients

patient_id	patient	condition1	condition2	condition3	dateofbirth	age	sex	physician_id
1	Nkemdilim Arendonk	1	2	0	2/21/1962	58	M	1
2	Raniero Coumans	3	4	0	8/15/1990	29	M	2
3	Mizuki Debenham	5	0	0	3/12/1977	43	F	3
4	Zoë De Witt	6	2	7	11/23/1947	72	F	4
5	Bonnie Hooper	8	7	0	7/4/1951	68	F	5

Table: conditions

condition_id	condition_name
0	NA
1	Pneumonia
2	Diabetes
3	Appendicitis
4	Crohn's disease
5	Kidney Cancer
6	Cardiomyopathy
7	Sciatica
8	Pancreatic Cancer

Table: physicians

physician_id	physician	medschool	experiece	hospital_id
1	Earnest Caro	University of California (Irvine)	14	1
2	Pamela English	University of Michigan	29	2
3	Lewis Conti	North Carolina State University	8	3
4	Theresa Dahlmans	Lake Erie College of Medicine	17	4
5	Steven Garbutt	Ohio State University	36	5

Table: hospitals

hospital_id	hospital	location
1	UPMC Presbyterian Shadyside	Pittsburg, PA
2	Northwestern Memorial Hospital	Chicago, IL

hospital_id	hospital	location
3	Houston Methodist Hospital	Houston, TX
4	Mount Sinai Hospital	New York, NY
5	UCSF Medical Center	San Francisco, CA

Part c

Rearrange the data on the five patients into a group of data tables that together meet the requirements of third normal form.

Note that the patient's age is a derived attribute from the patient's date of birth, but please don't make an extra data table just for age. In principle, if we are worried about data inconsistencies we can simply remove age from the database and calculate it when needed from date of birth. But for this exercise, leave age in the table and ignore its dependency with date of birth. [2 points]

3NF:

- Everything from 2NF
- There are no in-table dependencies between the columns in each table

Table: patients

patient_id	patient	condition1	condition2	condition3	dateofbirth	sex	physician_id
1	Nkemdilim Arendonk	1	2	0	2/21/1962	M	1
2	Raniero Coumans	3	4	0	8/15/1990	M	2
3	Mizuki Debenham	5	0	0	3/12/1977	F	3
4	Zoë De Witt	6	2	7	11/23/1947	F	4
5	Bonnie Hooper	8	7	0	7/4/1951	F	5

NOTE:

- Removing age as it depends on date of birth.

Table: conditions

condition_id	condition_name
0	NA
1	Pneumonia
2	Diabetes
3	Appendicitis
4	Crohn's disease
5	Kidney Cancer
6	Cardiomyopathy
7	Sciatica
8	Pancreatic Cancer

NOTE:

- Depending on the business case, medical conditions may also have dependencies, for example "Pneumonia" may depend on "Diabetes".

Table: physicians

physician_id	physician	medschool	experiece	hospital_id
1	Earnest Caro	University of California (Irvine)	14	1
2	Pamela English	University of Michigan	29	2
3	Lewis Conti	North Carolina State University	8	3
4	Theresa Dahlmans	Lake Erie College of Medicine	17	4

physician_id	physician	medschool	experiece	hospital_id
5	Steven Garbutt	Ohio State University	36	5

Table: hospitals

hospital_id	hospital	city	state
1	UPMC Presbyterian Shadyside	Pittsburg	PA
2	Northwestern Memorial Hospital	Chicago	IL
3	Houston Methodist Hospital	Houston	TX
4	Mount Sinai Hospital	New York	NY
5	UCSF Medical Center	San Francisco	CA

NOTE:

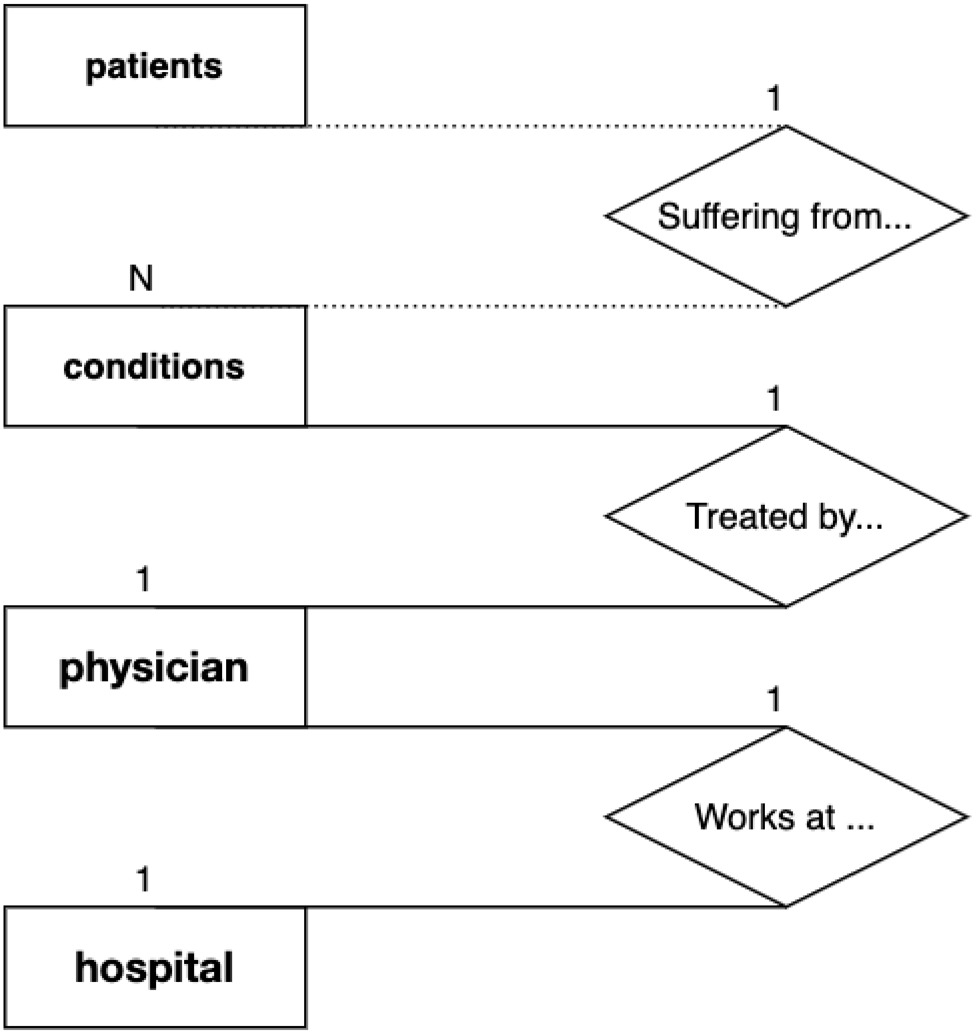
- Splitting location into 2 columns as state depends on city

Problem 2

For this problem, create ER diagrams of the database you created in problem 1, part c using draw.io: [https://app.diagrams.net/ (https://app.diagrams.net/)](https://app.diagrams.net/). The symbols used for both Chen's notation and IE notation are on the left-hand toolbar.

Part a

Create a conceptual ER diagram using Chen's notation. [2 points]

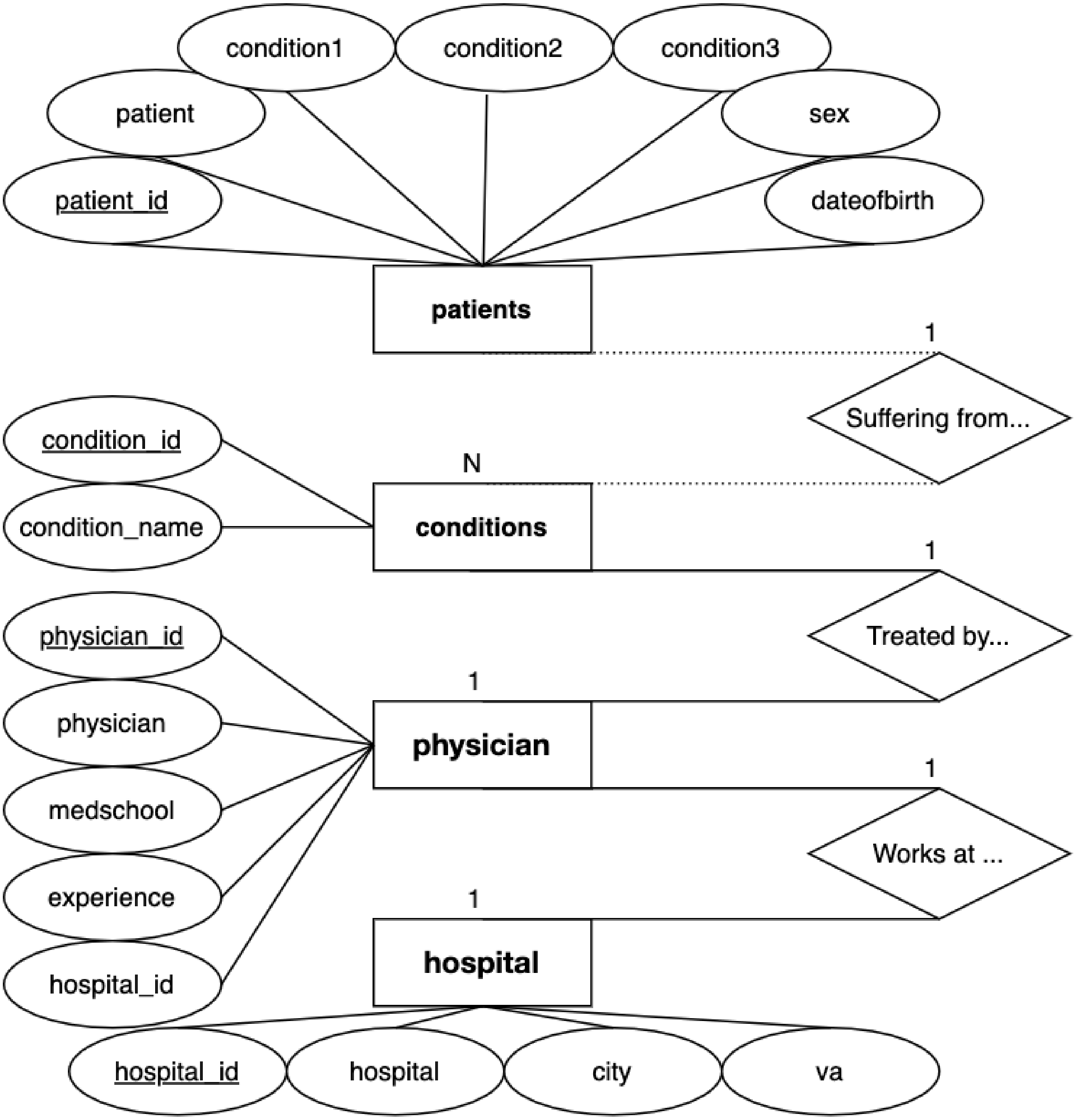

patients
1
Suffering from...
N
conditions
1
Treated by...
1
physician
1
Works at ...
1
hospital

Conceptual ER diagram:

- One **patient** suffering from 1 or many conditions (possibly 0)
- One or many **conditions** treated by one **physician**
- One **physician** works in one **hospital**

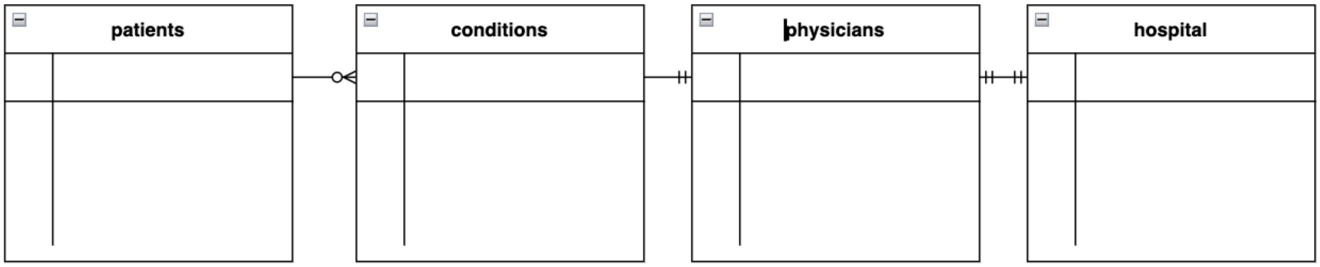
Part b

Create a logical ER diagram using Chen's notation. [2 points]


condition1
condition2
condition3
patient
sex
patient_id
dateofbirth
patients
1
Suffering from...
condition_id
N
condition_name
conditions
1
Treated by...
physician_id
1
physician
physician
medschool
1
Works at ...
experience
1
hospital_id
hospital
hospital_id
hospital
city
va

Part c

Create a conceptual ER diagram using IE notation. [2 points]



Problem 3

For this problem, you will download the individual CSV files that comprise a relational database on album reviews from Pitchfork Magazine (https://pitchfork.com/), collected via webscraping by Nolan B. Conaway (https://github.com/nolanbconaway/pitchfork-data), and use them to initialize local databases using SQlite, MySQL, and PostgreSQL.

To get the data, first set the working directory the folder on your computer to the folder where you want the CSV files to be. This should be the same folder where you saved our lab notebook and all associated files. Then change this line of code to the address for that folder:

In [104]:
```
os.chdir("/Users/dmitrymikhaylov/Documents/learn/uva/spring2022/DS6001/surfing_data_pipeline/M6")
```

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

In [3]:
```
# 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 is to practice creating databases from individual data frames. Next we load the CSVs to create the data frames in Python:

In [106]:
```
reviews = pd.read_csv("reviews.csv", index_col=0)
artists = pd.read_csv("artists.csv", index_col=0)
content = pd.read_csv("content.csv", index_col=0)
genres = pd.read_csv("genres.csv", index_col=0)
labels = pd.read_csv("labels.csv", index_col=0)
years = pd.read_csv("years.csv", index_col=0)
```

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 [107]: # Create and populate the database with 6 tables
          sqlite_database = sqlite3.connect("sqlite_database.db")
```

```
In [108]: # Populate respective tables in the "mysqlite_database"
          reviews.to_sql('reviews', sqlite_database, index=False, chunksize=1000, if_exists = 'replace')
          artists.to_sql('artists', sqlite_database, index=False, chunksize=1000, if_exists = 'replace')
          content.to_sql('content', sqlite_database, index=False, chunksize=1000, if_exists = 'replace')
          genres.to_sql('genres', sqlite_database, index=False, chunksize=1000, if_exists = 'replace')
          labels.to_sql('labels', sqlite_database, index=False, chunksize=1000, if_exists = 'replace')
          years.to_sql('years', sqlite_database, index=False, chunksize=1000, if_exists = 'replace')
```

```
Out[108]: 19108
```

```
In [109]: sqlite_database.commit()
```

```
In [110]: sqlite_database.close()
```

```
In [119]: # Query the database using .cursor
          sqlite_conn = sqlite3.connect('sqlite_database.db')
          cursor = sqlite_conn.cursor()
```

In [121]:

```
cursor.execute("SELECT title, artist, score FROM reviews WHERE score=10"
)
rows = cursor.fetchall()
for row in rows[:20]:
    print(row)
```

```
('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)
('dirty mind', 'prince', 10.0)
('off the wall', 'michael jackson', 10.0)
('"heroes"', 'david bowie', 10.0)
('low', 'david bowie', 10.0)
('a love supreme: the complete masters', 'john coltrane', 10.0)
("people's instinctive travels and the paths of rhythm", 'a tribe calle
d quest', 10.0)
('astral weeks', 'van morrison', 10.0)
('loaded: re-loaded 45th anniversary edition', 'the velvet undergroun
d', 10.0)
('sticky fingers', 'the rolling stones', 10.0)
('it takes a nation of millions to hold us back', 'public enemy', 10.0)
```

In [122]:

```
sqlite_conn.commit()
```

In [123]:

```
sqlite_conn.close()
```

In [124]:

```
another_conn = sqlite3.connect('sqlite_database.db')
pd.read_sql_query("SELECT title, artist, score FROM reviews WHERE score=
10", another_conn)
```

Out[124]:

	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
...	...	...	...
71	source tags and codes	...and you will know us by the trail of dead	10.0
72	the olatunji concert: the last live recording	john coltrane	10.0
73	kid a	radiohead	10.0
74	animals	pink floyd	10.0
75	i see a darkness	bonnie prince billy	10.0

76 rows × 3 columns

In [125]:

```
another_conn.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. [2 points]

In [126]:

```
# Importing the connector object
import mysql.connector
```

In [131]:

```
# Providing doyenv password
import dotenv
dotenv.load_dotenv()
mysqlpassword = os.getenv("mysqlpassword")
```

In [132]:

```
# Connecting to the local root
dbserver = mysql.connector.connect(
    user='root',
    passwd=mysqlpassword,
    host="localhost"
)
```

In [133]:

```
# Creating cursor object
cursor = dbserver.cursor()
```

In [134]:

```
# Try creating new database names "mysql_database"
try:
    cursor.execute("CREATE DATABASE mysql_database")
except:
    cursor.execute("DROP DATABASE mysql_database")
    cursor.execute("CREATE DATABASE mysql_database")
```

In [135]:

```
cursor.execute("SHOW DATABASES")
databases = cursor.fetchall()
databases
```

Out[135]:

```
[('information_schema',),
 ('mysql',),
 ('mysql_database',),
 ('performance_schema',),
 ('sys',)]
```

In [136]:

```
# Connecting to the specific database called "mysql_database"
mysql_database = mysql.connector.connect(
    user='root',
    passwd=mysqlpassword,
    host="localhost",
    database="mysql_database"
)
```

In [137]:

```
# Creating an “engine” to interface with the selected database
from sqlalchemy import create_engine
engine = create_engine("mysql+mysqlconnector://{user}:{pw}@localhost/{db}"
                       .format(user="root", pw=mysqlpassword, db="mysql_database"))
```

In [138]:

```
# Creating and populating tables in "mysql_database" using .to_sql method
reviews.to_sql('reviews', con = engine, index=False, chunksize=1000, if_exists = 'replace')
artists.to_sql('artists', con = engine, index=False, chunksize=1000, if_exists = 'replace')
content.to_sql('content', con = engine, index=False, chunksize=1000, if_exists = 'replace')
genres.to_sql('genres', con = engine, index=False, chunksize=1000, if_exists = 'replace')
labels.to_sql('labels', con = engine, index=False, chunksize=1000, if_exists = 'replace')
years.to_sql('years', con = engine, index=False, chunksize=1000, if_exists = 'replace')
```

Out[138]:

```
19108
```

In [139]:

```
# Using cursor object to get the results of query
cursor = mysql_database.cursor()
cursor.execute("SELECT title, artist, score FROM reviews WHERE score=10"
)
rows = cursor.fetchall()
for row in rows[:20]:
    print(row)
```

```
('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)
('dirty mind', 'prince', 10.0)
('off the wall', 'michael jackson', 10.0)
('"heroes"', 'david bowie', 10.0)
('low', 'david bowie', 10.0)
('a love supreme: the complete masters', 'john coltrane', 10.0)
("people's instinctive travels and the paths of rhythm", 'a tribe calle
d quest', 10.0)
('astral weeks', 'van morrison', 10.0)
('loaded: re-loaded 45th anniversary edition', 'the velvet undergroun
d', 10.0)
('sticky fingers', 'the rolling stones', 10.0)
('it takes a nation of millions to hold us back', 'public enemy', 10.0)
```

In [140]:
```
# Alternatively, use pandas read query method
pd.read_sql_query("SELECT title, artist, score FROM reviews WHERE score=
10", con=engine)
```

Out[140]:

	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
...	...	...	...
71	source tags and codes	...and you will know us by the trail of dead	10.0
72	the olatunji concert: the last live recording	john coltrane	10.0
73	kid a	radiohead	10.0
74	animals	pink floyd	10.0
75	i see a darkness	bonnie prince billy	10.0

76 rows × 3 columns

In [141]:
```
# Commit and close the database connection
dbserver.commit()
```

In [142]:
```
# ...by shutting down the entire server
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. [2 points]

In [143]:
```
import psycopg2
```

In [146]:
```
dotenv.load_dotenv()
pgpassword = os.getenv("pgpassword")
```

In [147]:
```
dbserver = psycopg2.connect(
    user='dmitrymikhaylov',
    password=pgpassword,
    host="localhost"
)
dbserver.autocommit = True
```

In [148]:

```
cursor = dbserver.cursor()
```

In [149]:

```
try:
    cursor.execute("CREATE DATABASE psql_database")
except:
    cursor.execute("DROP DATABASE psql_database")
    cursor.execute("CREATE DATABASE psql_database")
```

In [150]:

```
psql_database = psycopg2.connect(
    user='dmitrymikhaylov',
    password=pgpassword,
    host="localhost",
    database="psql_database"
)
```

In [151]:

```
engine = create_engine("postgresql+psycopg2://{user}:{pw}@localhost/{db}"
                       .format(user="dmitrymikhaylov", pw=pgpassword, db="psql_database"))
```

In [152]:

```
# Creating and populating tables in "psql_database" using .to_sql method
reviews.to_sql('reviews', con = engine, index=False, chunksize=1000, if_exists = 'replace')
artists.to_sql('artists', con = engine, index=False, chunksize=1000, if_exists = 'replace')
content.to_sql('content', con = engine, index=False, chunksize=1000, if_exists = 'replace')
genres.to_sql('genres', con = engine, index=False, chunksize=1000, if_exists = 'replace')
labels.to_sql('labels', con = engine, index=False, chunksize=1000, if_exists = 'replace')
years.to_sql('years', con = engine, index=False, chunksize=1000, if_exists = 'replace')
```

Out[152]:

```
19108
```

In [153]:

```
# Using cursor object to get the results of query
cursor = psql_database.cursor()
cursor.execute("SELECT title, artist, score FROM reviews WHERE score=10"
)
rows = cursor.fetchall()
for row in rows[:20]:
    print(row)
```

```
('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)
('dirty mind', 'prince', 10.0)
('off the wall', 'michael jackson', 10.0)
('"heroes"', 'david bowie', 10.0)
('low', 'david bowie', 10.0)
('a love supreme: the complete masters', 'john coltrane', 10.0)
("people's instinctive travels and the paths of rhythm", 'a tribe calle
d quest', 10.0)
('astral weeks', 'van morrison', 10.0)
('loaded: re-loaded 45th anniversary edition', 'the velvet undergroun
d', 10.0)
('sticky fingers', 'the rolling stones', 10.0)
('it takes a nation of millions to hold us back', 'public enemy', 10.0)
```

In [154]:

```
# Alternatively, use pandas read query method
pd.read_sql_query("SELECT title, artist, score FROM reviews WHERE score=
10", con=engine)
```

Out[154]:

	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
...	...	...	...
71	source tags and codes	...and you will know us by the trail of dead	10.0
72	the olatunji concert: the last live recording	john coltrane	10.0
73	kid a	radiohead	10.0
74	animals	pink floyd	10.0
75	i see a darkness	bonnie prince billy	10.0

76 rows × 3 columns

In [155]:

```
# Commit and close the database connection
dbserver.commit()
```

In [156]:

```
# Close it
dbserver.close()
```

Problem 4

Colin Mitchell (http://muffinlabs.com/) is a web-developer and artist who has a bunch of cool projects (http://muffinlabs.com/projects.html) that play with what data can do on the internet. One of his projects is Today in History (https://history.muffinlabs.com/), 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 [157]:

```
history = requests.get("https://history.muffinlabs.com/date")
history_json = json.loads(history.text)
events = history_json['data']['Events']
events[0]
```

Out[157]:

```
{'year': '0590',
 'text': 'Emperor Maurice proclaims his son Theodosius as co-emperor of
the Byzantine Empire.',
 'html': '0590 - <span style="visibility:hidden;color:transparent;">0</
span><a href="https://wikipedia.org/wiki/590" title="590">590</a> – Emp
eror <a href="https://wikipedia.org/wiki/Maurice_(emperor)" title="Maur
ice (emperor)">Maurice</a> proclaims his son <a href="https://wikipedi
a.org/wiki/Theodosius_(son_of_Maurice)" title="Theodosius (son of Mauri
ce)">Theodosius</a> as co-emperor of the <a href="https://wikipedia.or
g/wiki/Byzantine_Empire" title="Byzantine Empire">Byzantine Empire</a
>.',
 'no_year_html': '<span style="visibility:hidden;color:transparent;">0
</span><a href="https://wikipedia.org/wiki/590" title="590">590</a> – E
mperor <a href="https://wikipedia.org/wiki/Maurice_(emperor)" title="Ma
urice (emperor)">Maurice</a> proclaims his son <a href="https://wikiped
ia.org/wiki/Theodosius_(son_of_Maurice)" title="Theodosius (son of Maur
ice)">Theodosius</a> as co-emperor of the <a href="https://wikipedia.or
g/wiki/Byzantine_Empire" title="Byzantine Empire">Byzantine Empire</a
>.',
 'links': [{'title': '590', 'link': 'https://wikipedia.org/wiki/590'},
  {'title': 'Maurice (emperor)',
   'link': 'https://wikipedia.org/wiki/Maurice_(emperor)'},
  {'title': 'Theodosius (son of Maurice)',
   'link': 'https://wikipedia.org/wiki/Theodosius_(son_of_Maurice)'},
  {'title': 'Byzantine Empire',
   'link': 'https://wikipedia.org/wiki/Byzantine_Empire'}]}
```

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. [2 points]

In [158]:

```
import pymongo
```

In [159]:

```python
# Connect client to the local database server
myclient = pymongo.MongoClient("mongodb://localhost/")
```

In [160]:

```python
# Create a new document database "history"
history = myclient["history"]
```

In [161]:

```python
# Check if collection "today" is in the database; if yes, drop it
collist = history.list_collection_names()
if "today" in collist:
    history.today.drop()
```

In [162]:

```python
# Add collection "today"
today = history["today"]
```

In [163]:

```python
# Check the size of "events" JSON for testing
len(events)
```

Out[163]:

```
45
```

In [164]:

```python
# Insert all events in collection "today"
all_events = today.insert_many(events)
```

In [165]:

```python
# Check if the count matches expected JSON count
today.count_documents({})
```

Out[165]:

```
45
```

In [166]:

```python
# Define the query as per requirements above:
query = {
    "text":{
        "$regex": 'Virginia'
    }
}
```

In [167]:

```python
today.count_documents(query)
```

Out[167]:

```
0
```

In [168]:

```python
# Alternative query:
query = {
    "text":{
        "$regex": 'China'
    }
}
```

In [169]:

```python
# Returns one match for target "China"
today.count_documents(query)
```

Out[169]:

```
1
```

```
In [170]: # Use dumps() method to see what it is
          from bson.json_util import dumps, loads
          china_text = dumps(today.find(query))
```

```
In [171]: # Use loads() to convert string to readable text
          china_records = loads(china_text)
          china_records[0]
```

```
Out[171]: {'_id': ObjectId('623f4884e34de6fb6b34c03e'),
           'year': '2005',
           'text': 'Around 200,000 to 300,000 Taiwanese demonstrate in Taipei in
          opposition to the Anti-Secession Law of  China.',
           'html': '2005 – Around 200,000 to 300,000 <a href="https://wikipedia.o
          rg/wiki/Taiwan" title="Taiwan">Taiwanese</a> demonstrate in <a href="ht
          tps://wikipedia.org/wiki/Taipei" title="Taipei">Taipei</a> in oppositio
          n to the <a href="https://wikipedia.org/wiki/Anti-Secession_Law" title
          ="Anti-Secession Law">Anti-Secession Law</a> of <a href="https://wikipe
          dia.org/wiki/China" title="China">China</a>.',
           'no_year_html': 'Around 200,000 to 300,000 <a href="https://wikipedia.
          org/wiki/Taiwan" title="Taiwan">Taiwanese</a> demonstrate in <a href="h
          ttps://wikipedia.org/wiki/Taipei" title="Taipei">Taipei</a> in oppositi
          on to the <a href="https://wikipedia.org/wiki/Anti-Secession_Law" title
          ="Anti-Secession Law">Anti-Secession Law</a> of <a href="https://wikipe
          dia.org/wiki/China" title="China">China</a>.',
           'links': [{'title': 'Taiwan', 'link': 'https://wikipedia.org/wiki/Taiw
          an'},
            {'title': 'Taipei', 'link': 'https://wikipedia.org/wiki/Taipei'},
            {'title': 'Anti-Secession Law',
             'link': 'https://wikipedia.org/wiki/Anti-Secession_Law'},
            {'title': 'China', 'link': 'https://wikipedia.org/wiki/China'}]}
```

```
In [ ]:
```