Databases - Midterm

Due Sunday, Oct 24 by midnight

1) Please submit your notebook as a pdf, and make sure all the code fits on the screen!

2) Make sure to pay attention to places where you're asked to order the results!

3) In the census database (part 2), some of the data might come through as N or X where you would expect a number. Treat these as actual data, i.e. if I ask about "missing data", I'm **not** referring to these records.

In [1]:
```
# Import any libraries you'll need here
import sqlite3, pandas as pd
```

Part 1) Billboard database

These questions will make use of the bb.db database which contains the Billboard song data we have seen before.

This database has two tables: tSong, and tRating.

Recall that we have code from previous exercises you can use to list out the column names for each table in the database. You might also use the SQLite browser to help familiarize yourself with the data.

In [2]:
```
# Conenct to the bb.db database
conn = sqlite3.connect('data/bb.db')
curs = conn.cursor()
pd.read_sql("SELECT * FROM sqlite_master WHERE type='table';", conn)
```

Out[2]:

	type	name	tbl_name	rootpage	sql
0	table	tSong	tSong	2	CREATE TABLE tSong(\n year INTE...
1	table	tRating	tRating	3	CREATE TABLE tRating(\n id INTE...

In [3]:
```
x = pd.read_sql("SELECT name FROM sqlite_master WHERE type='table';", conn)
x.values
```

```
for table in x.values:
    print(table[0])
    sql = "PRAGMA table_info(" + table[0] + ");"
    print(pd.read_sql(sql, conn))
    print('\n')
```

```
tSong
   cid    name     type  notnull dflt_value  pk
0    0    year  INTEGER        1       None   0
1    1  artist     TEXT        1       None   0
2    2   track     TEXT        1       None   0
3    3    time     TEXT        1       None   0
4    4      id  INTEGER        0       None   1


tRating
   cid          name     type  notnull dflt_value  pk
0    0            id  INTEGER        1       None   1
1    1  date_entered     TEXT        1       None   0
2    2          week     TEXT        1       None   2
3    3        rating  NUMERIC        0       None   0
```

---

1) Which songs in the database have ever made it to the top of the chart, i.e., have ever had a rating = 1?

Have your query return 3 columns: track, artist, and time. Your results should not have any duplicate rows.

In [4]:

```
pd.read_sql("""SELECT track, artist, time
               FROM tSong
               JOIN tRating USING(id)
               WHERE rating LIKE '1'
               GROUP BY track;""", conn)
```

Out[4]:

	track	artist	time
0	Amazed	Lonestar	4:25
1	Be With You	Iglesias, Enrique	3:36
2	Bent	matchbox twenty	4:12
3	Come On Over Baby (A...	Aguilera, Christina	3:38

	track	artist	time
4	Doesn't Really Matte...	Janet	4:17
5	Everything You Want	Vertical Horizon	4:01
6	I Knew I Loved You	Savage Garden	4:07
7	Incomplete	Sisqo	3:52
8	Independent Women Pa...	Destiny's Child	3:38
9	It's Gonna Be Me	N'Sync	3:10
10	Maria, Maria	Santana	4:18
11	Music	Madonna	3:45
12	Say My Name	Destiny's Child	4:31
13	Thank God I Found Yo...	Carey, Mariah	4:14
14	Try Again	Aaliyah	4:03
15	What A Girl Wants	Aguilera, Christina	3:18
16	With Arms Wide Open	Creed	3:52

---

2) In this database, songs are retained for 76 weeks, even if they fell off the chart and did not have a rating for all 76 consecutive weeks. In those cases, the rating will be NULL.

Find all artists in the database who had a song that did not last for the 76 week duration, and return a count of the number of weeks they had null ratings.

Order the results by artist name, ascending.

In [5]:
```
pd.read_sql("""SELECT artist, COUNT(week) AS NumWeek
                FROM tSong
                JOIN tRating USING(id)
                WHERE rating IS NULL
                GROUP BY artist
                ORDER BY artist ASC;""", conn)
```

Out[5]:

	artist	NumWeek

	artist	NumWeek
0	2 Pac	69
1	2Ge+her	73
2	3 Doors Down	79
3	504 Boyz	58
4	98^0	56
...	...	...
223	Yankee Grey	68
224	Yearwood, Trisha	70
225	Ying Yang Twins	62
226	Zombie Nation	74
227	matchbox twenty	37

228 rows × 2 columns

3) It's often good to spot check your results. From question 2, take the first artist on the list and return:

artist, week, rating

for all entries where the rating is NULL. The number of rows should match the number you got for this artist in question 2.

In [6]:
```
pd.read_sql("""SELECT artist, week, rating
                FROM tSong
                JOIN tRating USING(id)
                WHERE rating IS NULL AND artist LIKE '2 Pac';""", conn)
```

Out[6]:

	artist	week	rating
0	2 Pac	wk8	None
1	2 Pac	wk9	None
2	2 Pac	wk10	None
3	2 Pac	wk11	None

	artist	week	rating
4	2 Pac	wk12	None
...	...	...	...
64	2 Pac	wk72	None
65	2 Pac	wk73	None
66	2 Pac	wk74	None
67	2 Pac	wk75	None
68	2 Pac	wk76	None

69 rows × 3 columns

---

4) What is the average rating for songs that are in week 10 of being on the Billboard chart?

In [7]:
```
pd.read_sql("""SELECT AVG(rating) AS AvgRating
                FROM tRating
                WHERE week LIKE 'wk10';""", conn)
```

Out[7]:

	AvgRating
0	45.786885

---

5) How many unique tracks in the database are there that are longer than 5 minutes?

Have your query return a single column with a single row: the number of songs.

Hint: To verify your result, you might also try listing them out. You might also check for the longest song in the database to make sure nothing else went wrong...

In [8]:
```
# longest song
pd.read_sql("""SELECT track, max(time)
                FROM tSong;""", conn)
```

Out[8]:

	track	max(time)
0	Auld Lang Syne (The ...	7:50

In [9]:

```
pd.read_sql("""SELECT COUNT(DISTINCT track) AS NumTrack
                FROM tSong
                WHERE time > '5:00';""", conn)
```

Out[9]:

	NumTrack
0	25

---

6) How many songs only had (non-null) ratings for a single week, and what are they?

Have your query return a list of these songs with: year, artist, track, time, date_entered, week, rating

In [10]:

```
pd.read_sql("""SELECT year, artist, track, time, date_entered, week, rating, count(*) AS NumNotNull
                FROM tSong
                JOIN tRating USING(id)
                WHERE rating IS NOT NULL
                GROUP BY track
                ORDER BY NumNotNull
                LIMIT 4;""", conn)
```

Out[10]:

	year	artist	track	time	date_entered	week	rating	NumNotNull
0	2000	Ghostface Killah	Cherchez LaGhost	3:04	2000-08-05	wk1	98	1
1	2000	Estefan, Gloria	No Me Dejes De Quere...	3:25	2000-06-10	wk1	77	1
2	2000	Master P	Souljas	3:33	2000-11-18	wk1	98	1
3	2000	Fragma	Toca's Miracle	3:22	2000-10-28	wk1	99	1

In [11]:

```
# Don't forget to close your connection to the database!
conn.close()
```

Part 2) Census database

These questions make use of the Census.db database. This is real data, albeit a bit out of date, from the US Census Bureau regarding things such as housing, income, employment, and population broken down by county, state, and year.

This database contains 4 tables. I have listed the columns below which we will be using. Other columns may be safely ignored.

- **tCounty**
  - county_id: a number which uniquely identifies each county
  - county: the name of the county
  - state
  - *Note: this is the ONLY table which is guaranteed to contain ALL counties in the data.*
- **tHousing**
  - county_id: same as county_id above.
  - year
  - units: An estimate of housing units (houses, apartments, etc. Check the census website for a more precise definition)
- **tEmployment**
  - county_id: same as in the previous tables
  - year
  - pop: An estimate of the adult population (i.e. the available workforce)
  - unemp_rate: The unemployment rate, expressed as a percentage, e.g. 5.0 = 5% = 0.05
- **tIncome**
  - county_id: same as in the previous tables
  - year
  - median_inc: median income
  - mean_inc: average (mean) income

In [12]:
```
# Connect to the Census.db database
conn = sqlite3.connect('data/Census.db')
curs = conn.cursor()
pd.read_sql("SELECT * FROM sqlite_master WHERE type='table';", conn)
```

Out[12]:

	type	name	tbl_name	rootpage	sql
0	table	sqlite_sequence	sqlite_sequence	6	CREATE TABLE sqlite_sequence(name,seq)
1	table	tCounty	tCounty	2	CREATE TABLE tCounty(\n county_id I...
2	table	tHousing	tHousing	3	CREATE TABLE tHousing(\n county_id...

	type	name	tbl_name	rootpage	sql
3	table	tEmployment	tEmployment	5	CREATE TABLE tEmployment(\n county_...
4	table	tIncome	tIncome	8	CREATE TABLE tIncome(\n county_id I...

In [13]:

```
x = pd.read_sql("SELECT name FROM sqlite_master WHERE type='table';", conn)
x.values

for table in x.values:
    print(table[0])
    sql = "PRAGMA table_info(" + table[0] + ");"
    print(pd.read_sql(sql, conn))
    print('\n')
```

```
sqlite_sequence
   cid  name type  notnull dflt_value  pk
0    0  name              0       None   0
1    1   seq              0       None   0


tCounty
   cid       name     type  notnull dflt_value  pk
0    0  county_id  INTEGER        1       None   1
1    1     county     TEXT        1       None   0
2    2      state     TEXT        1       None   0


tHousing
   cid       name     type  notnull dflt_value  pk
0    0  county_id  INTEGER        1       None   1
1    1       year  INTEGER        1       None   2
2    2      units  INTEGER        1       None   0


tEmployment
   cid            name     type  notnull dflt_value  pk
0    0       county_id  INTEGER        1       None   1
1    1            year  INTEGER        1       None   2
2    2             pop  INTEGER        1       None   0
3    3         pop_err  INTEGER        1       None   0
4    4        lab_part  NUMERIC        1       None   0
5    5    lab_part_err  NUMERIC        1       None   0
6    6       emp_ratio  NUMERIC        1       None   0
```

```
7     7   emp_ratio_err  NUMERIC        1       None    0
8     8      unemp_rate  NUMERIC        1       None    0
9     9  unemp_rate_err  NUMERIC        1       None    0


tIncome
   cid            name     type  notnull dflt_value  pk
0    0       county_id  INTEGER        1       None   1
1    1            year  INTEGER        1       None   2
2    2      median_inc  NUMERIC        1       None   0
3    3  median_inc_err  NUMERIC        1       None   0
4    4        mean_inc  NUMERIC        1       None   0
5    5    mean_inc_err  NUMERIC        1       None   0
```

---

7) In many places, the median income is less than the mean income, due to a relatively small number of individuals who make vastly more than the rest of the population.

Find all instances in this database where the opposite is true, that is, the median income is greater than the mean income.

Return four columns: county name, state, year, median income, mean income.

In [14]:
```
pd.read_sql("""SELECT county, state, year, median_inc, mean_inc
                FROM tCounty
                JOIN tIncome USING(county_id)
                WHERE median_inc > mean_inc AND median_inc != '(X)';""", conn)
```

Out[14]:

	county	state	year	median_inc	mean_inc
0	Daggett County	Utah	2016	75938	75200
1	Loving County	Texas	2017	80938	78119
2	Daggett County	Utah	2017	85000	76164

---

8) Assuming that population * unemployment rate = number of unemployed people, return a list of states with the highest number of unemployed people for the most recent year in the database

Have your query return five columns: state, year, population, unemployment rate, number of unemployed people. Limit the result to the top 10, sorted in descending order.

Note: Don't forget that the unemployment rates are expressed as percentages. A good sanity check here is that the number of unemployed people should be less than the population!

Also note: You can't simply average together unemployment rates for all counties in a state to get the overall unemployment rate for the state!

In [15]:
```
# The most recent year in the database
pd.read_sql("""SELECT MAX(year) AS MaxYear
               FROM tEmployment;""", conn)
```

Out[15]:

	MaxYear
0	2017

In [16]:
```
pd.read_sql("""WITH county_unemp AS
                (SELECT (unemp_rate/100)*pop AS num_unemp, state, year, pop, unemp_rate
                FROM tEmployment
                JOIN tCounty using(county_id)
                WHERE year LIKE '2017'
                GROUP BY county_id)
            SELECT state, year, SUM(pop) as pop, (SUM(num_unemp)/SUM(pop))*100 AS StateUnempRate,
            SUM(num_unemp) AS StateUnemp
            FROM county_unemp
            GROUP BY state
            ORDER BY StateUnemp DESC
            LIMIT 10;""", conn)
```

Out[16]:

	state	year	pop	StateUnempRate	StateUnemp
0	California	2017	31092029.0	4.152668	1291148.879
1	Florida	2017	16633043.0	5.326058	885885.539
2	Texas	2017	18888148.0	4.418055	834488.749
3	New York	2017	15348034.0	4.967867	762469.890
4	Illinois	2017	8786228.0	5.834497	512632.170

	state	year	pop	StateUnempRate	StateUnemp
5	Pennsylvania	2017	9666006.0	4.750235	459157.985
6	Ohio	2017	7883536.0	4.864421	383488.364
7	Michigan	2017	6849367.0	5.386399	368934.262
8	New Jersey	2017	7265350.0	4.994595	362874.832
9	Georgia	2017	6076879.0	5.388095	327427.994

---

9) Not all data exists for every county and every year in this database. Find all counties in Virginia that are missing population data.

Have your query return two columns: state, county name

In [17]:
```
pd.read_sql("""SELECT state, county
                FROM tCounty
                JOIN tEmployment USING(county_id)
                WHERE state = 'Virginia' AND pop = 'N';""", conn)
```

Out[17]:

	state	county
0	Virginia	James City County
1	Virginia	Frederick County

---

10) Find all counties where the number of housing units was less in 2017 than it was in 2015.

Have your query return 4 columns: state, county name, 2015 housing units, 2017 housing units.

In [19]:
```
curs.execute("""CREATE VIEW vNum2015 AS
                    SELECT county_id, county, units AS Num15
                    FROM tCounty
                    JOIN tHousing USING(county_id)
                    WHERE year = '2015';""")
```

Out[19]:
```
<sqlite3.Cursor at 0x7f3bf43cb730>
```

In [ ]:
```
curs.execute("""CREATE VIEW vNum2017 AS
                    SELECT county_id, county, units as Num17
                    FROM tHousing
                    JOIN tCounty USING(county_id)
                    WHERE year = '2017';""")
```

In [20]:
```
pd.read_sql("""SELECT state, tCounty.county, Num15, Num17
                FROM vNum2015
                JOIN vNum2017 USING(county_id)
                JOIN tCounty USING(county_id)
                WHERE Num17 < Num15;""", conn)
```

Out[20]:

	state	county	Num15	Num17
0	Alaska	Denali Borough	1766	1764
1	Alaska	Lake and Peninsula Borough	1514	1512
2	Alaska	Southeast Fairbanks Census Area	3909	3906
3	Arkansas	Arkansas County	9456	9453
4	Arkansas	Bradley County	5816	5797
...	...	...	...	...
395	West Virginia	Tucker County	5367	5366
396	West Virginia	Wayne County	19309	19290
397	West Virginia	Wetzel County	8151	8149
398	West Virginia	Wyoming County	10910	10894
399	Wisconsin	Ashland County	9662	9653

400 rows × 4 columns

---

11) Every town has a Main Street. There's a Miami in Florida and Ohio. There's a Roswell in New Mexico and Georgia.

Find all county names that exist more than once.

Have your query return two columns: county name, and a count of the number of times that county name exists. Order your results with the most frequently occurring county name at the top.

In [21]:
```
pd.read_sql("""SELECT county, COUNT(county) AS NumCounty
                FROM tCounty
                GROUP BY county
                HAVING NumCounty > 1
                ORDER BY NumCounty DESC;""", conn)
```

Out[21]:

	county	NumCounty
0	Washington County	30
1	Jefferson County	25
2	Franklin County	24
3	Lincoln County	23
4	Jackson County	23
...	...	...
418	Armstrong County	2
419	Alleghany County	2
420	Allegany County	2
421	Alexander County	2
422	Albany County	2

423 rows × 2 columns

In [22]:
```
# Don't forget to close the connection to the database!
conn.close()
```

Part 3) Conceptual Questions

---

12) What are the rules of tidy data?

1) Each variable forms a column

2) Each observations forms a row

3) Each type of observational unit forms a table

13) What normal form does Tidy Data most closely approximate?

3rd normal form (3NF)

---

14) In SQLite the RIGHT JOIN operation does not exist. Rewrite the following statement so that it would execute in SQLite:

```
SELECT column1,column2
FROM TableA
RIGHT JOIN TableB
ON TableB.id = TableA.id
```

SELECT column1, column2
FROM TableB
LEFT JOIN TableA
ON TableB.id = TableA.id

---

15) Suppose you have the following two tables:

TableA

x	y
1	cat
2	dog
3	bird
4	cow

TableB

x	z
2	blue
3	red
4	brown

and assume that we will be joining the tables on 'x'. Write a SQL statement that would produce the following output:

x	y	z
1	cat	NULL
2	dog	blue
3	bird	red
4	cow	brown

```
pd.read_sql("""SELECT x,y,z
FROM TableA LEFT JOIN TableB USING(x)
UNION
SELECT x,y,z
FROM TableB LEFT JOIN TableA USING(x);""", conn)
```

---

16) What is a Primary Key?

A primary key is a minimal set of columns needed to uniquely identify an observation.

---

17) Database normalization and Tidy Data have several benefits, but one of the main goals is to prevent certain things from occurring. What are those things called?

They're called anomalies. The three different types are:
1. Update anomaly
2. Insertion anomaly
3. Deletion anomaly