Databases - Fall 2020

Midterm - Due Sunday, October 4 by midnight

BY J.Mo Yang

If you would like to create views for any of these questions, please do so at the top of the section, in a cell immediately below where you connect to the database. This will help keep the rest of your submission clean and easy to read. Thanks!

In [6]:

```
import sqlite3
import pandas as pd
!rm -f Test.db
```

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 [7]:

```
conn=sqlite3.connect('./data/bb.db')
curs = conn.cursor()
```

In [8]:

```
x = pd.read_sql("""SELECT name
                FROM sqlite_master
                WHERE type = 'table'
                AND name LIKE 't%';""",conn)
for table in x.values:
    sql = "PRAGMA table_info(" + table[0] + ");"
    print(table)
    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 [9]:
```
pd.read_sql("""SELECT DISTINCT track, artist, time
                FROM tSong
                JOIN tRating USING (id)
                WHERE rating = 1;""", conn)
```

Out[9]:

	track	artist	time
0	Try Again	Aaliyah	4:03
1	Come On Over Baby (A...	Aguilera, Christina	3:38
2	What A Girl Wants	Aguilera, Christina	3:18
3	Thank God I Found Yo...	Carey, Mariah	4:14
4	With Arms Wide Open	Creed	3:52
5	Independent Women Pa...	Destiny's Child	3:38
6	Say My Name	Destiny's Child	4:31
7	Be With You	Iglesias, Enrique	3:36
8	Doesn't Really Matte...	Janet	4:17
9	Amazed	Lonestar	4:25
10	Music	Madonna	3:45
11	It's Gonna Be Me	N'Sync	3:10
12	Maria, Maria	Santana	4:18
13	I Knew I Loved You	Savage Garden	4:07
14	Incomplete	Sisqo	3:52
15	Everything You Want	Vertical Horizon	4:01
16	Bent	matchbox twenty	4:12

---

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.

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 [15]:
```
pd.read_sql("""SELECT artist, COUNT (track) as Week_NULL
                FROM tSong
                JOIN tRating USING (id)
                WHERE rating is NULL
                GROUP BY artist
                ORDER BY artist ASC""",conn)
```

Out[15]:

artist	Week_NULL

	artist	Week_NULL
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 [70]:

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

Out[70]:

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

Note: Make sure that NULL ratings are not included in your average! Do you need to add an additional condition in your query for this?

In [71]:

```
pd.read_sql("""SELECT week,  avg(rating) AS average_rating
                FROM tSong
                JOIN tRating USING (id)
                WHERE week LIKE '%wk10%'
                AND rating IS NOT NULL
                GROUP BY week;""", conn)
```

Out[71]:

	week	average_rating
0	wk10	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.

In [72]:

```
pd.read_sql("""SELECT COUNT(DISTINCT track) AS Num_track
                FROM tSong
                JOIN tRating USING (id)
                WHERE time > 5;""", conn)
```

Out[72]:

	Num_track
0	27

---

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 [103…

```
curs.execute("DROP VIEW IF EXISTS vtrack_num")
curs.execute("""CREATE VIEW vtrack_num AS
                SELECT DISTINCT year, artist, track, time, date_entered, week, ratin
                                COUNT(track) as track_num
                FROM tSong
                JOIN tRating USING (id)
                WHERE rating IS NOT NULL
                GROUP BY track;""")
```

Out[103…

```
<sqlite3.Cursor at 0x7f40bb2783b0>
```

In [104…

```
pd.read_sql("""SELECT * FROM vtrack_num;""",conn)
```

Out[104…

	year	artist	track	time	date_entered	week	rating	track_num
0	2000	Nelly	(Hot S**t) Country G...	4:17	2000-04-29	wk1	100	34
1	2000	Nu Flavor	3 Little Words	3:54	2000-06-03	wk1	97	9
2	2000	Jean, Wyclef	911	4:00	2000-10-07	wk1	77	19

	year	artist	track	time	date_entered	week	rating	track_num
3	2000	Brock, Chad	A Country Boy Can Su...	3:54	2000-01-01	wk1	93	3
4	2000	Clark, Terri	A Little Gasoline	3:07	2000-12-16	wk1	75	6
...	...	...	...	...	...	...	...	...
311	2000	Cyrus, Billy Ray	You Won't Be Lonely ...	3:45	2000-09-23	wk1	97	13
312	2000	Brooks & Dunn	You'll Always Be Lov...	2:58	2000-06-10	wk1	95	19
313	2000	Vertical Horizon	You're A God	3:45	2000-08-26	wk1	64	21
314	2000	Urban, Keith	Your Everything	4:10	2000-07-15	wk1	81	16
315	2000	Jackson, Alan	www.memory	2:36	2000-11-04	wk1	75	15

316 rows × 8 columns

In [105…
```
pd.read_sql("""SELECT year, artist, track, time, date_entered, week, rating
                FROM vtrack_num
                WHERE track_num = 1;""",conn)
```

Out[105…

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

In [106…
```
# 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 [107…

```
# Connect to the Census.db database
conn=sqlite3.connect('./data/Census.db')
curs = conn.cursor()
```

In [108…

```
x = pd.read_sql("""SELECT name
                FROM sqlite_master
                WHERE type = 'table'
                AND name LIKE 't%';""",conn)
for table in x.values:
    sql = "PRAGMA table_info(" + table[0] + ");"
    print(table)
    print(pd.read_sql(sql,conn))
    print('\n')
```

```
['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
```

In [109…

```
pd.read_sql("""SELECT state, unemp_rate from tEmployment JOIN tCounty USING (county_
```

Out[109…

	state	unemp_rate
0	California	5.3
1	California	10.5
2	California	6.4
3	California	8.7
4	California	9.9
...	...	...
115	California	7
116	California	10.6
117	California	5.2
118	California	6.9
119	California	3.3

120 rows × 2 columns

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 [110…

```
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[110…

	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!

In [111…

```
pd.read_sql("""SELECT state, year, sum(pop) as pop, avg(unemp_rate) as unemp_rate,
                        sum((pop*(unemp_rate/100))) as NumUnEmp
                FROM tCounty
                JOIN tEmployment USING (county_id)
                WHERE year = 2017
                GROUP BY state
                ORDER BY NumUnEmp DESC
                LIMIT 10;""", conn)
```

Out[111…

```
          state  year         pop  unemp_rate     NumUnEmp
0    California  2017  31092029.0    6.242500  1291148.879
1       Florida  2017  16633043.0    5.485366   885885.539
2         Texas  2017  18888148.0    4.857407   834488.749
3      New York  2017  15348034.0    5.341026   762469.890
4      Illinois  2017   8786228.0    6.034783   512632.170
5  Pennsylvania  2017   9666006.0    5.287500   459157.985
6          Ohio  2017   7883536.0    5.058974   383488.364
7      Michigan  2017   6849367.0    5.465517   368934.262
8    New Jersey  2017   7265350.0    5.619048   362874.832
9       Georgia  2017   6076879.0    5.402703   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 [112…

```
pd.read_sql("""SELECT DISTINCT state, county
                FROM tCounty
                LEFT JOIN tEmployment USING (county_id)
                WHERE [state] IN ('Virginia')
                AND pop IS NULL""", conn)
```

Out[112…

```
        state              county
0    Virginia      Accomack County
1    Virginia     Alleghany County
2    Virginia        Amelia County
3    Virginia       Amherst County
4    Virginia    Appomattox County
...       ...                  ...
98   Virginia          Salem city
99   Virginia        Staunton city
100  Virginia      Waynesboro city
101  Virginia    Williamsburg city
102  Virginia     Winchester city
```

103 rows × 2 columns

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 [113…

```
curs.execute("DROP VIEW IF EXISTS vHousing15;")
curs.execute("""CREATE VIEW vHousing15 AS
                SELECT state, county, year, units
                FROM tCounty
                JOIN tHousing USING(county_id)
                WHERE year LIKE '2015';""")
```

Out[113…

```
<sqlite3.Cursor at 0x7f40bb1d63b0>
```

In [114…

```
pd.read_sql("""SELECT * FROM vHousing15;""",conn)
```

Out[114…

	state	county	year	units
0	Alabama	Autauga County	2015	23104
1	Alabama	Baldwin County	2015	109412
2	Alabama	Barbour County	2015	11919
3	Alabama	Bibb County	2015	9114
4	Alabama	Blount County	2015	24107
...	...	...	...	...
3137	Wyoming	Sweetwater County	2015	19578
3138	Wyoming	Teton County	2015	13469
3139	Wyoming	Uinta County	2015	8937
3140	Wyoming	Washakie County	2015	3859
3141	Wyoming	Weston County	2015	3557

3142 rows × 4 columns

In [115…

```
curs.execute("DROP VIEW IF EXISTS vHousing17;")
curs.execute("""CREATE VIEW vHousing17 AS
               SELECT state, county, year, units
                FROM tCounty
                JOIN tHousing USING(county_id)
                WHERE year LIKE '2017';""")
```

Out[115…

```
<sqlite3.Cursor at 0x7f40bb1d63b0>
```

In [116…

```
pd.read_sql("""SELECT * FROM vHousing17;""",conn)
```

Out[116…

	state	county	year	units
0	Alabama	Autauga County	2017	23494
1	Alabama	Baldwin County	2017	114134
2	Alabama	Barbour County	2017	11970
3	Alabama	Bibb County	2017	9189

	state	county	year	units
4	Alabama	Blount County	2017	24313
...	...	...	...	...
3137	Wyoming	Sweetwater County	2017	19732
3138	Wyoming	Teton County	2017	13851
3139	Wyoming	Uinta County	2017	9018
3140	Wyoming	Washakie County	2017	3867
3141	Wyoming	Weston County	2017	3566

3142 rows × 4 columns

In [117…

```
pd.read_sql("""SELECT vHousing15.state, vHousing15.county, vHousing15.units as Units
                     vHousing17.units as Units_17
                FROM vHousing15
                JOIN vHousing17 USING (county)
                WHERE Units_17 < Units_15;""",conn)
```

Out[117…

	state	county	Units_15	Units_17
0	Alabama	Baldwin County	109412	20566
1	Alabama	Barbour County	11919	7923
2	Alabama	Butler County	10008	6797
3	Alabama	Butler County	10008	5952
4	Alabama	Butler County	10008	4068
...	...	...	...	...
6075	Wyoming	Sheridan County	14632	2166
6076	Wyoming	Sheridan County	14632	2913
6077	Wyoming	Sheridan County	14632	920
6078	Wyoming	Teton County	13469	5781
6079	Wyoming	Teton County	13469	2920

6080 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 in more than one state.

Have your query return two columns: county name, number of states it exists in. Order your results with the most frequently occurring county name at the top.

In [118…

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

Out[118…

	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 [119…

```
# 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 observation forms a row

3) Each type of observational unit forms a table

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

Third Normal Form

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 Table A

ON TableA.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

(SELECT *

FROM TableA

JOIN TableB

ON TableA.X=TableB.X)

16) What is a Primary Key?

(a minimal set of columns or attributes needed to uniquely identify an observation or row)

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?

(Benefits are limiting data anomalies and allows impossibility of data inconsistancies, this is called Data normalization and the goal is to prevent data anomalies)

In [ ]: