Databases_Midterm_Fall20

November 23, 2021

0.1 Databases - Fall 2020

0.1.1 Midterm - Due Sunday, October 4 by midnight

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!

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

0.1.2 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.

[2]:
```
# Conenct to the bb.db database
conn = sqlite3.connect('./bb.db')
curs = conn.cursor()
```

[3]:
```
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.

[4]:
```
pd.read_sql("""SELECT track, artist, time
                FROM tSong
                    JOIN tRating USING (id)
                WHERE rating = '1'
                GROUP BY track;""",conn)
```

[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
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.

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.

[5]:
```
curs.execute("DROP VIEW IF EXISTS vCountNull;")
curs.execute("""CREATE VIEW vCountNull AS
                SELECT *
                FROM tSong
                    JOIN tRating USING (id)
                WHERE rating ISNULL
                ORDER BY artist ASC;""")


pd.read_sql("""SELECT artist, count() as WeeksNull
                FROM vCountNull
                GROUP BY artist
                ;""", conn)
```

[5]:

	artist	WeeksNull
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 x 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.

[6]:
```
pd.read_sql("""SELECT artist, week, rating
                FROM tSong
                    JOIN tRating USING (id)
                WHERE artist LIKE "2 pac"
                AND rating ISNULL
```

```
;""", conn)
```

[6]:

	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 x 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?

[7]:
```
curs.execute("DROP VIEW IF EXISTS vWeek10;")
curs.execute("""CREATE VIEW vWeek10 AS
                SELECT *
                FROM tSong
                JOIN tRating USING (id)
                WHERE rating IS NOT NULL
                AND week LIKE 'wk10'
                ;""")




pd.read_sql("""SELECT AVG(rating) FROM vWeek10;""",conn)
```

[7]:

	AVG(rating)
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.

[8]:
```
curs.execute("DROP VIEW IF EXISTS vLongSongs;")
curs.execute("""CREATE VIEW vLongSongs AS
                SELECT *
                FROM tSong
                JOIN tRating USING (id)
                WHERE time > "5:00"
                GROUP BY track;""")


pd.read_sql("""SELECT count(*) as NumSongs
                FROM vLongSongs;""",conn)
```

[8]:

	NumSongs
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

[9]:
```
curs.execute("DROP VIEW IF EXISTS vWk2Ratings;")
curs.execute("""CREATE VIEW vWk2Ratings AS
                SELECT *
                FROM tSong
                JOIN tRating USING (id)
                WHERE week ='wk2'
                And rating ISNULL;""")

pd.read_sql("""SELECT year, artist, track, time, date_entered, rating FROM␣
 ↪vWk2Ratings;""",conn)
```

[9]:

	year	artist	track	time	date_entered	rating
0	2000	Anastacia	I'm Outta Love	4:01	2000-04-01	None
1	2000	Estefan, Gloria	No Me Dejes De Quere…	3:25	2000-06-10	None
2	2000	Fragma	Toca's Miracle	3:22	2000-10-28	None
3	2000	Ghostface Killah	Cherchez LaGhost	3:04	2000-08-05	None
4	2000	Master P	Souljas	3:33	2000-11-18	None

[10]:
```
# Don't forget to close your connection to the database!
conn.close()
```

0.1.3 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

[11]:
```
# Connect to the Census.db database
conn = sqlite3.connect('./Census.db')
curs = conn.cursor()
```

[12]:
```
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
```

---

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.

[13]:
```
pd.read_sql("""SELECT county, state, year, median_inc, mean_inc
               FROM tCounty
                   JOIN tIncome USING (county_id)
               WHERE mean_inc < median_inc;""", conn)
```

[13]:

	county	state	year	median_inc	mean_inc
0	Loving County	Texas	2015	(X)	54313
1	Daggett County	Utah	2016	75938	75200
2	Loving County	Texas	2017	80938	78119

```
3  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!

[14]:
```
pd.read_sql("""SELECT state, year, pop, unemp_rate,
                                0.01*pop*unemp_rate as NumUnemp
               FROM tCounty AS A
                   JOIN tEmployment AS B ON A.county_id = B.county_id
               WHERE year = '2017'
               GROUP BY state
               ORDER BY NumUnemp DESC
               LIMIT 10;""", conn)
```

[14]:

	state	year	pop	unemp_rate	NumUnemp
0	Nevada	2017	1746061	6.1	106509.721
1	California	2017	1355096	4.7	63689.512
2	Connecticut	2017	760442	6.6	50189.172
3	District of Columbia	2017	579654	6.6	38257.164
4	New Mexico	2017	544374	5.0	27218.700
5	New Jersey	2017	219723	8.5	18676.455
6	Nebraska	2017	433584	3.9	16909.776
7	Indiana	2017	287240	5.2	14936.480
8	Colorado	2017	382355	3.9	14911.845
9	Alaska	2017	228778	5.8	13269.124

---

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

[15]:
```
pd.read_sql("""SELECT state, county
               FROM tCounty
                   JOIN tEmployment USING (county_id)
               WHERE state LIKE 'virginia'
               AND pop = 'N';""", conn)
```

[15]:

	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.

[17]:
```
#PRobably need to set up two views to point to the 2015 housing units and the␣
 ↪2017 ones



curs.execute("DROP VIEW IF EXISTS v2017Units;")
curs.execute("""CREATE VIEW v2017Units AS
                    SELECT state, county_id, units
                    FROM tCounty
                        JOIN tHousing USING (county_id)
                    WHERE year = '2017';""")
curs.execute("DROP VIEW IF EXISTS v2015Units;")
curs.execute("""CREATE VIEW v2015Units AS
                    SELECT state, county_id, units
                    FROM tCounty
                        JOIN tHousing USING (county_id)
                    WHERE year = '2015';""")

#create table with state, conunty, 2015 units and 2017 units
#pd.read_sql("""SELECT * FROM v2015Units;""",conn)

pd.read_sql("""SELECT tCounty.state, tCounty.county, A.units as "2015_units", B.
 ↪units as "2017_units"
              FROM tCounty
              JOIN v2015Units as A USING (county_id)
              JOIN v2017Units as B USING (county_id)
              WHERE A.units > B.units
            ;""", conn)
```

[17]:

	state	county	2015_units	2017_units
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 x 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.

[18]:
```
pd.read_sql("""SELECT county, count(*) as CountyCount
                FROM tCounty

                GROUP BY county
                HAVING CountyCount > 1
                ORDER BY CountyCount DESC
                ;""",conn)
```

[18]:
```
                county  CountyCount
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 x 2 columns]
```

[19]:
```
# Don't forget to close the connection to the database!
conn.close()
```

0.1.4 Part 3) Conceptual Questions

---

12) What are the rules of tidy data?

Each variable forms a column Each observation forms a row Each observational unit forms a table

---

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

Codd's Third 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 JOIN Table A USING (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 JOIN TABLEB USING(x)

```
;""",conn)
```

---

16) What is a Primary Key?

A primary key is a field in a table which uniquely identifies each row/record in a database table.

Primary keys must contain unique values. A primary key column cannot have NULL values. A minimal set of columns (attributes) needed to uniquely identify and observation (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?

They aim to prevent redundencies, anomalies, and inconsistencies.

[ ]:

[ ]: