HW 5 - Building a normalized RDB

The goal of this homework is to take a semi-structured non-normalized CSV file and turn it into a set of normalized tables that you then push to your postgres database on AWS.

The original dataset contains 100k district court decisions, although I've downsampled it to only 1000 rows to make the uploads faster. Each row contains info about a judge, their demographics, party affiliation, etc. Rows also contain information about the case they were deciding on. Was it a criminal or civil case? What year was it? Was the direction of the decision liberal or conservative?

While the current denormalized format is fine for analysis, it's not fine for a database as it violates many normalization rules. Your goal is to normalize it by designing a simple schema, then wrangling it into the proper dataframes, then pushing it all to AWS.

For the first part of this assignment you should wind up with three tables. One with case information, one with judge information, and one that has casetype information. Each table should be reduced so that there are not then repeating rows, and primary keys should be assigned within each. These tables should be called 'cases', 'judges', and 'casetype'.

For the last part you should make a rollup table that calculates the percent of liberal decisions for each party level and each case category. This will allow for one to get a quick look at how the political party affiliation of judges impacts the direction of a decision for different case categories (e.g. criminal, civil, labor).

Submission 1) Run all cells. 2) Create a directory with your name. 3) Create a pdf copy of your notebook. 4) Download .py and .ipynb of the notebook. 5) Put all three files in it. 6) Zip and submit.

Bring in data, explore, make schema - 3 point

Start by bringing in your data to `cases`. Call a `.head()` on it to see what columns are there and what they contain.

```
import pandas as pd
cases = pd.read_csv('https://docs.google.com/spreadsheets/d/1AWLK06JOlSKImgoHNTbj7c
```

```
# head of cases
cases.head()
```

✓ 0s completed at 1:12 PM

	judge_name	party_name	gender_name	race_name	case_id	case_year	c
0	Thompson, Myron H.	Democrat	male	African-American/black	28321332	2011	
1	Shoob, Marvin A.	Democrat	male	white/caucasian	18110669	1993	
2	Bua, Nicholas J.	Democrat	male	white/caucasian	15660871	1983	

Make schema

OK, given that head, you need to make three related tables that will make up a normalized database. Those tables are 'cases', 'judges', and 'casetype'. If it's not clear what info should go into each, explore the data more.

Remember, you might not have keys, will need to reduce the rows, select certain columns, etc. There isn't a defined path here.

Make cases table. - 6 points

Start by making a table that contains just each case's info. I would call this table that you're going to upload `cases_df` so you don't overwrite your raw data.

This table should have six columns and 1000 rows.

Note, one of these columns should be a judge_id that links to the judges table. You'll need to make this foreign key.

Also, you can leave 'category_name' in this table as well as its id. Normally you'd split that off into it's own table as well, but you're already doing that for casetype which is enough for now.

```
# Make judge_id in cases
#
# cases_grouped = cases.groupby('judge_name')
cases['judge_id'] = pd.factorize(cases['judge_name'])[0]
cases.head()
```

	judge_name	party_name	gender_name	race_name	case_id	case_year	c
0	Thompson, Myron H.	Democrat	male	African-American/black	28321332	2011	
1	Shoob,	Democrat	male	white/caucasian	18110669	1993	

1	Marvin A.	Democrat	male	white/caucasian	18110669	1993
2	Bua, Nicholas J.	Democrat	male	white/caucasian	15660871	1983

```
# select necessary columns to make cases_df
cases_df = cases[['judge_id', 'category_id', 'category_name', 'case_id', 'case_year
cases_df.head()
```

	judge_id	category_id	category_name	case_id	case_year	casetype_id	caset
0	0	1	Criminal Justice Cases	28321332	2011	3	cri
1	1	2	Civil Liberties/Rights Cases	18110669	1993	14	fre
2	2	1	Criminal Justice Cases	15660871	1983	2	hab

```
# What's the shape of cases_df?
cases_df.shape
```

```
(1000, 7)
```

Make cases table in your database

Go bring over your connection function from the last workbook. **This must be in here for me to grade the homework** If I can't access your database you'll get a zero. It's a good idea to bring our other exploratory functions as well.

Once you do that you'll need to do the following

- Connect, make a table called 'cases' with the correct column names and data types. Be sure to execute and commit the table.
- Make tuples of your data
- Write a SQL string that allows you to insert each tuple of data into the correct columns
- Execute the string many times to fill out 'cases'
- Commit changes and check the table.

I'm not going to leave a full roadmap beyond this. Feel free to add cells as needed to do the above.

```
# Make our connection/cursor function
AWS_host_name = "test-hw-db.chb3guhvlmeq.us-west-2.rds.amazonaws.com"
```

```
AWS_dbname = "postgres"
AWS_user_name = "postgres"
AWS_password = "uW5uK$4xcBNvKL"

def get_conn_cur(): # define function name and arguments (there aren't any)
  # Make a connection
  conn = psycopg2.connect(
    host=AWS_host_name,
    database=AWS_dbname,
    user=AWS_user_name,
    password=AWS_password,
    port='5432')

  cur = conn.cursor()   # Make a cursor after

  return(conn, cur)   # Return both the connection and the cursor

# Same run_query function
def run_query(query_string):

  conn, cur = get_conn_cur() # get connection and cursor

  cur.execute(query_string) # executing string as before

  my_data = cur.fetchall() # fetch query data as before

  # here we're extracting the 0th element for each item in cur.description
  colnames = [desc[0] for desc in cur.description]

  cur.close() # close
  conn.close() # close

  return(colnames, my_data) # return column names AND data

# Column name function for checking out what's in a table
def get_column_names(table_name): # arguement of table_name
  conn, cur = get_conn_cur() # get connection and cursor

  # Now select column names while inserting the table name into the WERE
  column_name_query =  """SELECT column_name FROM information_schema.columns
       WHERE table_name = '%s' """ %table_name

  cur.execute(column_name_query) # exectue
  my_data = cur.fetchall() # store

  cur.close() # close
  conn.close() # close

  return(my_data) # return

# Check table names
```

```
def get_table_names():
  conn, cur = get_conn_cur() # get connection and cursor

  # query to get table names
  table_name_query = """SELECT table_name FROM information_schema.tables
       WHERE table_schema = 'public' """

  cur.execute(table_name_query) # execute
  my_data = cur.fetchall() # fetch results

  cur.close() #close cursor
  conn.close() # close connection

  return(my_data) # return your fetched results
```

```
from pandas.io.sql import execute
sql = """
CREATE TABLE cases (
  judge_id BIGINT,
  category_id BIGINT,
  category_name VARCHAR(255),
  case_id BIGINT PRIMARY KEY,
  case_year INTEGER,
  casetype_id BIGINT,
  casetype_name VARCHAR(255)
)
"""

con, cur = get_conn_cur()
cur.execute(sql)
con.commit()
```

```
cases_df.head()
cases_tupes = [tuple(x) for x in cases_df.to_numpy()]

sql = "INSERT INTO cases (judge_id, category_id, category_name, case_id, case_year,
cur.executemany(sql, cases_tupes)
con.commit()
```

```
# Use sql_head to check cases
sql_head(table_name='cases')
```

	judge_id	category_id	category_name	case_id	case_year	casetype_id	caset
0	0	1	Criminal Justice Cases	28321332	2011	3	cri
			Civil				

1	1	2	Liberties/Rights Cases	18110669	1993	14	fre
2	2	1	Criminal Justice Cases	15660871	1983	2	hab

Make judges - 6 points

Now make your judges table from the original `cases` dataframe (not the SQL table you just made).

Judges should have five columns, including the `judge_id` column you made. There should be 553 rows after you drop duplicates (remember that judges may have had more than one case).

After you make the dataset go and push to a SQL table called 'judges'.

```
judges_df = cases[['judge_id', 'judge_name', 'party_name', 'gender_name', 'race_nam
judges_df.head()
```

	judge_id	judge_name	party_name	gender_name	race_name
0	0	Thompson, Myron H.	Democrat	male	African-American/black
1	1	Shoob, Marvin A.	Democrat	male	white/caucasian
2	2	Bua, Nicholas J.	Democrat	male	white/caucasian
3	3	Kovachevich, Elizabeth	Republican	female	white/caucasian
4	4	Gilliam, Earl B.	Independent/Other /Unknown	male	white/caucasian

```
judges_df.shape
```

```
    (553, 5)
```

```
sql = """
CREATE TABLE judges (
  judge_id BIGINT,
  judge_name VARCHAR(255),
  party_name VARCHAR(255),
  gender_name VARCHAR(255),
  race_name VARCHAR(255)
)
"""
```

```
con, cur = get_conn_cur()
```

```
cur.execute(sql)
con.commit()


judges_tupes = [tuple(x) for x in judges_df.to_numpy()]

sql = "INSERT INTO judges (judge_id, judge_name, party_name, gender_name, race_name
cur.executemany(sql, judges_tupes)
con.commit()
sql_head(table_name='judges')
```

	judge_id	judge_name	party_name	gender_name	race_name
0	0	Thompson, Myron H.	Democrat	male	African-American/black
1	1	Shoob, Marvin A.	Democrat	male	white/caucasian
2	2	Bua, Nicholas J.	Democrat	male	white/caucasian
3	3	Kovachevich, Elizabeth	Republican	female	white/caucasian
4	4	Gilliam, Earl B.	Independent/Other/Unknown	male	white/caucasian

Make casetype - 6 points

Go make the casetype table. This should have only two columns that allow you to link the casetype name back to the ID in the 'cases' table. There should be 27 rows as well.

```
casetype_df = cases[['casetype_id', 'casetype_name']].drop_duplicates()
casetype_df.head()
```

	casetype_id	casetype_name
0	3	criminal court motions
1	14	free of religion
2	2	habeas corpus-state
4	6	alien petitions
5	19	environmental protection

```
casetype_df.dtypes
```

```
    casetype_id       int64
    casetype_name    object
    dtype: object
```

```
casetype_df.shape
```

```
(27, 2)
```

```
sql = """
CREATE TABLE casetype (
  casetype_id BIGINT,
  casetype_name VARCHAR(255)
)
"""

con, cur = get_conn_cur()
cur.execute(sql)
con.commit()
```

```
casetype_tupes = [tuple(x) for x in casetype_df.to_numpy()]

sql = "INSERT INTO casetype (casetype_id, casetype_name) VALUES (%s, %s)"
cur.executemany(sql, casetype_tupes)
con.commit()
sql_head(table_name='casetype')
```

	casetype_id	casetype_name
0	3	criminal court motions
1	14	free of religion
2	2	habeas corpus-state
3	6	alien petitions
4	19	environmental protection

A quick test of your tables - 3 point

Below is a query to get the number of unique judges that have ruled on criminal court motion cases. You should get a value of 119 as your return if your database is set up correctly!

```
run_query("""SELECT COUNT(DISTINCT(judges.judge_id)) FROM cases
    JOIN judges ON cases.judge_id = judges.judge_id
        WHERE casetype_id = (SELECT casetype_id FROM casetype
                  WHERE casetype_name = 'criminal court motions'); """)
```

```
(['count'], [(119,)])
```

Make rollup table - 6 points

Now let's make that rollup table! The goal here is to make a summary table easily accessed. We're going to roll the whole thing up by the judges party and the category, but you could imagine doing this for each judge to track how they make decisions over time which would then be useful for an analytics database. The one we're making could also be used as a dimension table where we needed overall party averages.

We want to get a percentage of liberal decisions by each grouping level (party_name, category_name). To do this we need first, the number of cases seen at each level, and second, the number of liberal decisions made at each level. `cases` contains the columns `libcon_id` which is a 0 if the decision was conservative in its ruling, and a 1 if it was liberal in its ruling. Thus, you can get a percentage of liberal decisions if you divide the sum of that column by the total observations. Your `agg()` can both get the sum and count.

After you groupby you'll need to reset the index, rename the columns, then make the percentage.

Once you do that you can push to a SQL table called 'rollup'

Let's get started

```
# Make a groupby called cases_rollup. This should group by party_name and categrory
cases_rollup = cases.groupby(['party_name', 'category_name']).agg({'libcon_id': ['c
cases_rollup
```

		libcon_id	
		count	sum
party_name	category_name		
Democrat	Civil Liberties/Rights Cases	218	112
	Criminal Justice Cases	107	36
	Labor & Economic Cases	126	69
Independent/Other/Unknown	Civil Liberties/Rights Cases	12	5
	Criminal Justice Cases	8	5
	Labor & Economic Cases	13	6
Republican	Civil Liberties/Rights Cases	237	73
	Criminal Justice Cases	125	34
	Labor & Economic Cases	154	67

```
# reset your index
cases_rollup = cases_rollup.reset_index()
cases_rollup
```

	party_name	category_name	libcon_id	
			count	sum
0	Democrat	Civil Liberties/Rights Cases	218	112
1	Democrat	Criminal Justice Cases	107	36
2	Democrat	Labor & Economic Cases	126	69
3	Independent/Other/Unknown	Civil Liberties/Rights Cases	12	5
4	Independent/Other/Unknown	Criminal Justice Cases	8	5
5	Independent/Other/Unknown	Labor & Economic Cases	13	6
6	Republican	Civil Liberties/Rights Cases	237	73
7	Republican	Criminal Justice Cases	125	34
8	Republican	Labor & Economic Cases	154	67

```
# rename your columns now. Keep the first to the same but call the last two 'total_
cases_rollup.columns = ['party_name', 'category_name', 'total_cases', 'num_lib_deci
cases_rollup
```

	party_name	category_name	total_cases	num_lib_decisions
0	Democrat	Civil Liberties/Rights Cases	218	112
1	Democrat	Criminal Justice Cases	107	36
2	Democrat	Labor & Economic Cases	126	69
3	Independent/Other/Unknown	Civil Liberties/Rights Cases	12	5
4	Independent/Other/Unknown	Criminal Justice Cases	8	5
5	Independent/Other/Unknown	Labor & Economic Cases	13	6
6	Republican	Civil Liberties/Rights Cases	237	73

Now make a new column called 'percent_liberal'

This should calucalte the percentage of decisions that were liberal in nature. Multiple it by 100 so that it's a full percent. Also use the `round()` function on the whole thing to keep it in whole percentages.

```
# make your metric called 'percent_liberal'
cases_rollup['percent_liberal'] = round((cases_rollup['num_lib_decisions'] / cases_

cases_rollup
```

	party_name	category_name	total_cases	num_lib_decisions	percent_liber
0	Democrat	Civil Liberties/Rights Cases	218	112	5
1	Democrat	Criminal Justice Cases	107	36	3
2	Democrat	Labor & Economic Cases	126	69	5
3	Independent/Other /Unknown	Civil Liberties/Rights Cases	12	5	4
4	Independent/Other /Unknown	Criminal Justice Cases	8	5	6
	Independent/Other	Labor &			

Now go and push the whole thing to a table called 'rollup'

There should be five columns and nine rows.

```
sql = """
CREATE TABLE rollup (
  party_name VARCHAR(255),
  category_name VARCHAR(255),
  total_cases INTEGER,
  num_lib_decisions INTEGER,
  percent_liberal INTEGER
)
"""


con, cur = get_conn_cur()
cur.execute(sql)
con.commit()


cases_rollup_tupes = [tuple(x) for x in cases_rollup.to_numpy()]

sql = "INSERT INTO rollup (party_name, category_name, total_cases, num_lib_decisior
sql = "INSERT INTO rollup (party_name, category_name, total_cases, num_lib_decisio
cur.executemany(sql, cases_rollup_tupes)
con.commit()


# check
sql_head('rollup')
```

	party_name	category_name	total_cases	num_lib_decisions	percent_liber
0	Democrat	Civil Liberties/Rights Cases	218	112	
1	Democrat	Criminal Justice Cases	107	36	
2	Democrat	Labor & Economic Cases	126	69	

Colab paid products - Cancel contracts here