STA220 Assignment 1

Preliminaries

Due **February 6, 11:59 PM**.

Submit your work by uploading it to Gradescope. Submission requires two files: the original Jupyter Notebook and its PDF export. Please rename this file as "H1_Lastname_Firstname_srnr", where srnr are the last four digits of your student's ID number and do the same for the PDF export file.

Objective

The objective of this homework assignment is to solidify your understanding of and proficiency with SQL and multithreading.

Instructions

1. Provide your solutions in new cells between the `Solution START` cell and the `Solution END` cell. Create as many new cells as necessary within these two blocks. Use code cells for your Python scripts and Markdown cells for explanatory text or answers to non-coding questions.

2. You must **execute the code following every `Validation` block to get credits** for the corresponding task. Failure to do so may result in a loss of points. (This, obviously, only applies to tasks with a `Validation` block.)

3. Prioritize code readability. Just as in writing a book, the clarity of each line matters. Adopt the **one-statement-per-line** rule. If you have a lengthy code statement, consider breaking it into multiple lines for clarity. Note you can use `'''` to start and end strings in Python that are written over multiple lines.

4. To help understand and maintain code, you should add comments to explain your code. Use the hash symbol (#) to start writing a comment.

5. Submit your work by uploading it to **Gradescope**. Submission requires two files: the original Jupyter Notebook (.ipynb) and its PDF export. To convert your Jupyter notebook file into a PDF, navigate to "File", select "Download as", and then choose either "PDF via LaTeX" or "HTML". If "PDF via LaTeX" does not work for you, export to "HTML", and then use Chrome to print the .html file into PDF.

6. This assignment will be graded on your proficiency in programming. Be sure to demonstrate your abilities and submit your own, correct and readable solutions.

Code of conduct

The usage of AI for this homework is strictly forbidden. Any use will result in 0 points for the whole homework assignment.

Setting

In this assignment, you'll use `sqlite3` to explore data in Lahman's Baseball Database, which contains "complete batting and pitching statistics from 1871 to 2022, plus fielding statistics, standings, team stats, managerial records, post-season data, and more." We use the 2022 version for this homework. You can find the database in SQLite format on Canvas. Documentation for the database, including a description of all tables, is in the `readme.txt` file included on Canvas.

The data are taken from github, and the corresponding `readme.txt` from here. Note that the `readme.txt` file contains a lot of information about the databases and the meaning of its columns. It is highly advised to use it for the upcoming tasks.

In [1]:

```
import os
import sqlite3 as sql
import pandas as pd
import numpy as np
import time
import requests
import threading
import concurrent.futures
```

Make sure that you can load the database. For this, make sure to replace the value for the `file_path` by the correct path on your computer where the database is stored.

In [2]:

```
# if the database is located in the same folder as this file, you can use th
# file_path = "./lahman.sqlite"

file_path = "/Users/heatherchilders/Documents/UCDavis/STA220/STA220/data/lah
if os.path.exists(file_path):
    print("File was found.")
else:
    print("File could not be found. Please change the file_path accordingly.
```

```
File was found.
```

If the file was not found, you must change the `file_path` before proceeding. If, otherwise, the file was found, you should be able to connect to the database.

In [3]:

```
db = sql.connect(file_path)
cur = db.execute("SELECT * FROM sqlite_master")
```

You can find a comprehensive overview of the tables in the database in the `readme.txt` file. Alternatively, you may use the following code to explore the database:

In [4]:

```
tables = pd.read_sql('SELECT * FROM sqlite_master WHERE type == "table"', db
tables
```

Out[4]:

	type	name	tbl_name	rootpage	sql
0	table	AllstarFull	AllstarFull	2	CREATE TABLE AllstarFull (\nplayerID TEXT,\nye...
1	table	Appearances	Appearances	3	CREATE TABLE Appearances (\nyearID INTEGER,\nt...
2	table	AwardsManagers	AwardsManagers	4	CREATE TABLE AwardsManagers (\nplayerID TEXT,\...
3	table	AwardsPlayers	AwardsPlayers	5	CREATE TABLE AwardsPlayers (\nplayerID TEXT,\n...
4	table	AwardsShareManagers	AwardsShareManagers	6	CREATE TABLE AwardsShareManagers (\nawardID TE...
5	table	AwardsSharePlayers	AwardsSharePlayers	7	CREATE TABLE AwardsSharePlayers (\nawardID TEX...
6	table	Batting	Batting	8	CREATE TABLE Batting (\nplayerID TEXT,\nyearID...
7	table	BattingPost	BattingPost	9	CREATE TABLE BattingPost (\nyearID INTEGER,\nr...
8	table	CollegePlaying	CollegePlaying	10	CREATE TABLE CollegePlaying (\nplayerID TEXT,\...
9	table	Fielding	Fielding	11	CREATE TABLE Fielding (\nplayerID TEXT,\nyearI...
10	table	FieldingOF	FieldingOF	12	CREATE TABLE FieldingOF (\nplayerID TEXT,\nyea...
11	table	FieldingOFsplit	FieldingOFsplit	13	CREATE TABLE FieldingOFsplit (\nplayerID TEXT,...
12	table	FieldingPost	FieldingPost	14	CREATE TABLE FieldingPost (\nplayerID TEXT,\ny...
13	table	HallOfFame	HallOfFame	15	CREATE TABLE HallOfFame (\nplayerID TEXT,\nyea...
14	table	HomeGames	HomeGames	16	CREATE TABLE HomeGames

	type	name	tbl_name	rootpage	sql
					(\nyearkey INTEGER,\nle...
15	table	Managers	Managers	17	CREATE TABLE Managers (\nplayerID TEXT,\nyearI...
16	table	ManagersHalf	ManagersHalf	18	CREATE TABLE ManagersHalf (\nplayerID TEXT,\ny...
17	table	Parks	Parks	19	CREATE TABLE Parks (\nID INTEGER,\nparkalias T...
18	table	People	People	21	CREATE TABLE People (\nID INTEGER,\nplayerID T...
19	table	Pitching	Pitching	23	CREATE TABLE Pitching (\nplayerID TEXT,\nyearI...
20	table	PitchingPost	PitchingPost	24	CREATE TABLE PitchingPost (\nplayerID TEXT,\ny...
21	table	Salaries	Salaries	25	CREATE TABLE Salaries (\nyearID INTEGER,\nteam...
22	table	Schools	Schools	26	CREATE TABLE Schools (\nschoolID TEXT,\nname_f...
23	table	SeriesPost	SeriesPost	27	CREATE TABLE SeriesPost (\nyearID INTEGER,\nro...
24	table	Teams	Teams	28	CREATE TABLE Teams (\nyearID INTEGER,\nlgID TE...
25	table	TeamsFranchises	TeamsFranchises	29	CREATE TABLE TeamsFranchises (\nfranchID TEXT,...
26	table	TeamsHalf	TeamsHalf	30	CREATE TABLE TeamsHalf (\nyearID INTEGER,\nlgI...

Exercise 1 [5.5 points]

Task 1a) [2 points]

Task description

Write a function `get_cols` that takes a table name as input and returns a dictorionary consisting of only one entry:

- the key is the table name
- the value is a list consisting of all column names the corresponding table has

Use this function to create another dictionary `table_info`, consisting of all table names as keys. Each entry (value of the dictionary) should be a list containing all column names the corresponding table has.

You may use the df `tables` to loop over all table names, but you must use some kind of sqlite request to get the column names.

From the dictionary `table_info`, define two integers `column_count_tot` and `table_count`:

- `column_count_tot` is the number of all columns of all tables
- `table_count` is the number of tables the database contains

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [5]:

```python
def get_cols(table_name):
    #SQL Query that gets the table info from each table
    cur.execute(f"PRAGMA table_info({table_name})")
    #Execute query
    column_info = cur.fetchall()
    #get the column names from the SQL output
    column_names = {col[0]: col[1] for col in column_info}
    #Make the output a dictionary
    dictionary = {
        table_name: list(column_names.values())
    }
    return dictionary

#Create a new dictionary called table_info
table_info = {}
#define a new parameter `column_count_tot` which is the number of all columr
column_count_tot = 0
# define a new parameter `table_count` which is the number of tables the dat
table_count = 0

#The for loop loops over all iterations in `tables`
for i in tables["tbl_name"]:
    #run get_cols
    columns = get_cols(i)
    #Add the output of get_cols to dictionary table_info
    table_info.update(columns)
    #for each iteration, add the number of columns to the column counter
    total_cols = list(columns.values())[0]
    column_count_tot += len(total_cols)
    table_count += 1

# print the `column_count_tot` for checking results
#print(f"The number of columns across all tables is {column_count_tot}")

# print to check results
#print(f"The number of tables is {table_count}")
```

Solution END

Validation

Please run the following code lines. Wrong results or errors in the following code may still get partial credits - as long as the following code is executed.

In [6]:

```
table_info
```

```
Out[6]:  {'AllstarFull': ['playerID',
           'yearID',
           'gameNum',
           'gameID',
           'teamID',
           'lgID',
           'GP',
           'startingPos'],
          'Appearances': ['yearID',
           'teamID',
           'lgID',
           'playerID',
           'G_all',
           'GS',
           'G_batting',
           'G_defense',
           'G_p',
           'G_c',
           'G_1b',
           'G_2b',
           'G_3b',
           'G_ss',
           'G_lf',
           'G_cf',
           'G_rf',
           'G_of',
           'G_dh',
           'G_ph',
           'G_pr'],
          'AwardsManagers': ['playerID', 'awardID', 'yearID', 'lgID', 'tie', 'note
         s'],
          'AwardsPlayers': ['playerID', 'awardID', 'yearID', 'lgID', 'tie', 'note
         s'],
          'AwardsShareManagers': ['awardID',
           'yearID',
           'lgID',
           'playerID',
           'pointsWon',
           'pointsMax',
           'votesFirst'],
          'AwardsSharePlayers': ['awardID',
           'yearID',
           'lgID',
           'playerID',
           'pointsWon',
           'pointsMax',
           'votesFirst'],
          'Batting': ['playerID',
           'yearID',
           'stint',
           'teamID',
           'lgID',
           'G',
           'G_batting',
           'AB',
           'R',
  'H',
  '2B',
  '3B',
  'HR',
  'RBI',
  'SB',
  'CS',
  'BB',
  'SO',
  'IBB',
  'HBP',
  'SH',
  'SF',
  'GIDP',
  'G_old'],
 'BattingPost': ['yearID',
  'round',
  'playerID',
  'teamID',
  'lgID',
  'G',
  'AB',
  'R',
  'H',
  '2B',
  '3B',
  'HR',
  'RBI',
  'SB',
  'CS',
  'BB',
  'SO',
  'IBB',
  'HBP',
  'SH',
  'SF',
  'GIDP'],
 'CollegePlaying': ['playerID', 'schoolID', 'yearID'],
 'Fielding': ['playerID',
  'yearID',
  'stint',
  'teamID',
  'lgID',
  'POS',
  'G',
  'GS',
  'InnOuts',
  'PO',
  'A',
  'E',
  'DP',
  'PB',
  'WP',
  'SB',
  'CS',
  'ZR'],
 'FieldingOF': ['playerID', 'yearID', 'stint', 'Glf', 'Gcf', 'Grf'],
 'FieldingOFsplit': ['playerID',
  'yearID',
  'stint',
  'teamID',
  'lgID',
  'POS',
  'G',
  'GS',
  'InnOuts',
  'PO',
  'A',
  'E',
  'DP',
  'PB',
  'WP',
  'SB',
  'CS',
  'ZR'],
 'FieldingPost': ['playerID',
  'yearID',
  'teamID',
  'lgID',
  'round',
  'POS',
  'G',
  'GS',
  'InnOuts',
  'PO',
  'A',
  'E',
  'DP',
  'TP',
  'PB',
  'SB',
  'CS'],
 'HallOfFame': ['playerID',
  'yearid',
  'votedBy',
  'ballots',
  'needed',
  'votes',
  'inducted',
  'category',
  'needed_note'],
 'HomeGames': ['yearkey',
  'leaguekey',
  'teamkey',
  'parkkey',
  'spanfirst',
  'spanlast',
  'games',
  'openings',
  'attendance'],
 'Managers': ['playerID',
  'yearID',
  'teamID',
  'lgID',
  'inseason',
  'G',
  'W',
  'L',
  'rank',
  'plyrMgr'],
 'ManagersHalf': ['playerID',
  'yearID',
  'teamID',
  'lgID',
  'inseason',
  'half',
  'G',
  'W',
  'L',
  'rank'],
 'Parks': ['ID',
  'parkalias',
  'parkkey',
  'parkname',
  'city',
  'state',
  'country'],
 'People': ['ID',
  'playerID',
  'birthYear',
  'birthMonth',
  'birthDay',
  'birthCity',
  'birthCountry',
  'birthState',
  'deathYear',
  'deathMonth',
  'deathDay',
  'deathCountry',
  'deathState',
  'deathCity',
  'nameFirst',
  'nameLast',
  'nameGiven',
  'weight',
  'height',
  'bats',
  'throws',
  'debut',
  'bbrefID',
  'finalGame',
  'retroID'],
 'Pitching': ['playerID',
  'yearID',
  'stint',
  'teamID',
  'lgID',
  'W',
  'L',
  'G',
  'GS',
  'CG',
  'SHO',
  'SV',
  'IPouts',
  'H',
  'ER',
  'HR',
  'BB',
  'SO',
  'BAOpp',
  'ERA',
  'IBB',
  'WP',
  'HBP',
  'BK',
  'BFP',
  'GF',
  'R',
  'SH',
  'SF',
  'GIDP'],
 'PitchingPost': ['playerID',
  'yearID',
  'round',
  'teamID',
  'lgID',
  'W',
  'L',
  'G',
  'GS',
  'CG',
  'SHO',
  'SV',
  'IPouts',
  'H',
  'ER',
  'HR',
  'BB',
  'SO',
  'BAOpp',
  'ERA',
  'IBB',
  'WP',
  'HBP',
  'BK',
  'BFP',
  'GF',
  'R',
  'SH',
  'SF',
  'GIDP'],
 'Salaries': ['yearID', 'teamID', 'lgID', 'playerID', 'salary'],
 'Schools': ['schoolID', 'name_full', 'city', 'state', 'country'],
'SeriesPost': ['yearID',
 'round',
 'teamIDwinner',
 'lgIDwinner',
 'teamIDloser',
 'lgIDloser',
 'wins',
 'losses',
 'ties'],
'Teams': ['yearID',
 'lgID',
 'teamID',
 'franchID',
 'divID',
 'Rank',
 'G',
 'Ghome',
 'W',
 'L',
 'DivWin',
 'WCWin',
 'LgWin',
 'WSWin',
 'R',
 'AB',
 'H',
 '2B',
 '3B',
 'HR',
 'BB',
 'SO',
 'SB',
 'CS',
 'HBP',
 'SF',
 'RA',
 'ER',
 'ERA',
 'CG',
 'SHO',
 'SV',
 'IPouts',
 'HA',
 'HRA',
 'BBA',
 'SOA',
 'E',
 'DP',
 'FP',
 'name',
 'park',
 'attendance',
 'BPF',
 'PPF',
 'teamIDBR',
 'teamIDlahman45',
  'teamIDretro'],
 'TeamsFranchises': ['franchID', 'franchName', 'active', 'NAassoc'],
 'TeamsHalf': ['yearID',
  'lgID',
  'teamID',
  'Half',
  'divID',
  'DivWin',
  'Rank',
  'G',
  'W',
  'L']}
```

```
In [7]: type(table_info)
```

```
Out[7]: dict
```

```
In [8]: table_count
```

```
Out[8]: 27
```

```
In [9]: column_count_tot
```

```
Out[9]: 374
```

```
In [10]: get_cols('people')
```

```
Out[10]: {'people': ['ID',
           'playerID',
           'birthYear',
           'birthMonth',
           'birthDay',
           'birthCity',
           'birthCountry',
           'birthState',
           'deathYear',
           'deathMonth',
           'deathDay',
           'deathCountry',
           'deathState',
           'deathCity',
           'nameFirst',
           'nameLast',
           'nameGiven',
           'weight',
           'height',
           'bats',
           'throws',
           'debut',
           'bbrefID',
           'finalGame',
           'retroID']}
```

```
In [11]: get_cols('Appearances')
```

```
Out[11]: {'Appearances': ['yearID',
           'teamID',
           'lgID',
           'playerID',
           'G_all',
           'GS',
           'G_batting',
           'G_defense',
           'G_p',
           'G_c',
           'G_1b',
           'G_2b',
           'G_3b',
           'G_ss',
           'G_lf',
           'G_cf',
           'G_rf',
           'G_of',
           'G_dh',
           'G_ph',
           'G_pr']}
```

Task 1b) [1.5 points]

Task description

Create a Pandas DataFrame named `tbl` by querying the Lahman database to find the total career games played by players during specific seasons where they appeared frequently but did not record any starts.

Logic Requirements

For each player, the SQL query should filter for "Special Seasons" that meet these two criteria:

- Games Started (GS) is NULL: The player did not start any games that year.
- Games Played (G_all) is more than 10: The player appeared in more than 10 games during that season.

DataFrame Specifications

The resulting DataFrame tbl must follow these formatting rules:

- Columns:
  1. playerID: The unique identifier for the player.
  2. total_apps: The sum of all G_all values across all the "special seasons" identified above for that player.
- Sorting:
  - Primary: Sort by total_apps in ascending order.

- Secondary: If total_apps is the same, sort by playerID alphabetically.

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [12]:

```
tbl = pd.read_sql('''SELECT playerID, SUM(G_all) AS total_apps
                  FROM Appearances
                  WHERE GS IS NULL
                  GROUP BY playerID
                  HAVING SUM(G_all) > 10
                  ORDER BY total_apps DESC,
                  playerID ASC''', db)
```

Solution END

Validation

Please run the following code lines. Wrong results or errors in the following code may still get partial credits - as long as the following code is executed.

In [13]:

```
tbl.head()
```

Out[13]:

	playerID	total_apps
0	ansonca01	2398
1	mcphebi01	2138
2	ryanji01	2014
3	connoro01	1998
4	becklja01	1997

In [14]:

```
tbl.shape
```

Out[14]:

```
(1579, 2)
```

Task 1c) [1 point]

Task description

Task: SQL View Creation – college_stats

Define a new database view named college_stats that aggregates career game statistics for players associated with Colleges.

Requirements:

- Source Data: Use the `CollegePlaying` table to identify the relationship between players (playerID) and schools (schoolID).
- Join Logic: Link these players to the `Appearances` table to retrieve their professional game history.
- Aggregation: For every unique player-school pairing, calculate the sum of all games played (G_all) across the player's entire professional career. This number shall be called `total_games`.

Note on multi-College Players: If a player is associated with multiple schools in the CollegePlaying table, their total career games should be reported for each school they attended.

Examples

The following examples are provided to help you for the task.

In [15]:
```
# pd.read_sql('''
            #SELECT * FROM college_stats LIMIT 5
            #''', db)
```

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [16]:
```
# Select the player d, school id and sum of all games
#from only distinct player/school pairings and name that selection "college
# then join the appearances table to that via player id
#then group by the player/school id
db.execute('''
CREATE VIEW college_stats AS
SELECT
    CollegePlaying.playerID,
    CollegePlaying.schoolID,
    SUM(Appearances.G_all) AS total_games
FROM (
    SELECT DISTINCT playerID, schoolID
    FROM CollegePlaying
) AS CollegePlaying
JOIN Appearances
    ON Appearances.playerID = CollegePlaying.playerID
GROUP BY
    CollegePlaying.playerID,
    CollegePlaying.schoolID''')
```

Out[16]:
```
<sqlite3.Cursor at 0x1795d52c0>
```

Solution END

Validation

Please run the following code lines. Wrong results or errors in the following code may still get partial credits - as long as the following code is executed.

In [17]:
```
pd.read_sql('''
            SELECT * FROM college_stats
            ''', db)
```

Out[17]:

	playerID	schoolID	total_games
0	aardsda01	pennst	331
1	aardsda01	rice	331
2	abadan01	gamiddl	15
3	abbeybe01	vermont	79
4	abbotje01	kentucky	233
...	...	...	...
7515	zoskyed01	fresnost	44
7516	zuberjo01	california	68
7517	zuninmi01	florida	850
7518	zupcibo01	oralrob	319
7519	zuvelpa01	stanford	209

7520 rows × 3 columns

In [18]:
```
db.execute('''DROP VIEW IF EXISTS college_stats''') # this removes the VIEW
#db.commit()
```

Out[18]:
```
<sqlite3.Cursor at 0x1795d5bc0>
```

Task 1d) [1 point]

Task description

Consider the CollegePlaying table: calculate how many years each player has played for each College and create a DataFrame `top10_college` consisting of the top10 players (in terms of: how long they have played for a specific college) in descending order, that is, starting with the Player that has played the most years for one college. The DataFrame should be created by the command `top10_college = pd.read_sql('''SOME SQL QUERY''', db)` and consist of three columns:

- First column, `total_years`, stating the number of years a player has played for this college.
- Second column: `playerID`

- Third column: `schoolID`

Examples

The following examples are provided to help you for the task.

In [19]:

```
top10_college.head(1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[19], line 1
----> 1 top10_college.head(1)

NameError: name 'top10_college' is not defined
```

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [ ]:

```
top10_college = pd.read_sql('''
    SELECT COUNT(DISTINCT yearID) AS total_years, playerID, schoolID
    FROM CollegePlaying
    GROUP BY playerID, schoolID
    ORDER BY total_years DESC, playerID ASC, schoolID ASC
    LIMIT 10''', db)
```

Solution END

Validation

Please run the following code lines. Wrong results or errors in the following code may still get partial credits - as long as the following code is executed.

In [ ]:

```
top10_college
```

Out[ ]:

	total_years	playerID	schoolID
0	9	burrial01	washcollmd
1	6	burkejo02	stbonny
2	6	currasa01	tuftsma
3	6	fenneho01	kalamazoo
4	6	gibsono01	notredame
5	6	harledi01	georgetown
6	6	hulveha01	shenandoah
7	6	tyngji01	harvard
8	5	batchri01	uscaiken
9	5	bellast01	fordham

Exercise 2 [2.5 points]

The purpose of this assignment is to practice accessing and analyzing data in a database. For full credit, query the correct table with `pandas.read_sql` and a **single SQL query**. Do not subset, group, sort, aggregate, etc. via `pandas` in this Exercise, but use SQL commands to return the desired table.

In other words, all solutions in this exercise must be of the following form:
pd.read_sql('''

> some sql statement
> over one
> or multiple lines

''', db)

Task 2a) [0.5 points]

Task description

List the World Series Winner of each year in a DataFrame showing the year and the name of the team, in chronological order (starting with the oldest entry). Please note that the database was last updated after the 2022 season.

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

```
In [ ]: pd.read_sql('''
                    SELECT yearID, TeamID
                    FROM Teams WHERE WSWin = 'Y'
                    ORDER BY yearID DESC''', db)
```

```
Out[ ]:
        yearID  teamID
    0     2022     HOU
    1     2021     ATL
    2     2020     LAN
    3     2019     WAS
    4     2018     BOS
  ...      ...     ...
  118     1889     NY1
  119     1888     NY1
  120     1887     DTN
  121     1886     SL4
  122     1884     PRO

123 rows × 2 columns
```

Solution END

Task 2b) [1 point]

Task description

Calculate the average `Rank` (taken from the `teams` table) of every team that played more than one season and return a table with the team's name and the average rank.

Sort the table, starting with the best team, that is, the lowest average rank).

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [ ]:

```
pd.read_sql("""
    SELECT TeamID, AVG(Rank)
    FROM Teams
    GROUP BY TeamID
    HAVING COUNT(*) > 1
    ORDER BY AVG(Rank) ASC;
""", db)
```

Out[ ]:

	teamID	AVG(Rank)
0	BS1	1.400000
1	CHF	2.000000
2	SL4	2.200000
3	PRO	2.250000
4	LAN	2.415385
...	...	...
87	KC2	7.500000
88	WS2	7.727273
89	WS8	7.750000
90	KC1	8.153846
91	LS3	10.625000

92 rows × 2 columns

Solution END

Task 2c) [1 points]

Task description

List all teams that have won the World Series at least three times. Make a table that shows the team's name and how often they won the title. Sort the table by the number of titles, starting with the most successfull team.

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

```
In [ ]: pd.read_sql('''
                SELECT teamID, COUNT(*) AS total_wins
                FROM Teams
                WHERE WSWin = 'Y'
                GROUP BY teamID
                HAVING COUNT(*) > 3
                ORDER BY total_wins DESC''', db)
```

Out[ ]:

	teamID	total_wins
0	NYA	27
1	SLN	11
2	BOS	9
3	NY1	7
4	LAN	6
5	CIN	5
6	PHA	5
7	PIT	5
8	DET	4
9	OAK	4

Solution END

Exercise 3 [3.5 points]

The purpose of this assignment is to practice accessing and analyzing data in a database. For full credit, query the correct table with `pandas.read_sql` and a **single SQL query**. Do not subset, group, sort, aggregate, etc. via `pandas` in this Exercise, but use SQL commands to return the desired table.

In other words, all solutions in this exercise must be of the following form:
pd.read_sql('''

> some sql statement
> over one
> or multiple lines

''', db)

Task 3a) [1 point]

Task description

Make a list of all managers that were also baseball players, consisting of the first name in the first, and the last name in the second column. Sort it alphabetically (start with the last name). A baseball player is defined as every player in the people database whose `debut` is not NULL. A manager is every person listed in the `managers` table.

The major challenge of this task is to combine the two tables.

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [ ]:

```
#select distinct pairs of first/last names, and join the managers table to t
# then only get the peopl who had not null debut
#order by last name the first name
pd.read_sql('''
    SELECT DISTINCT People.nameFirst, People.nameLast
    FROM People
    JOIN Managers
      ON People.playerID = Managers.playerID
    WHERE People.debut IS NOT NULL
    ORDER BY People.nameLast, People.nameFirst ASC
''',db)
```

Out[ ]:

	nameFirst	nameLast
0	Joe	Adcock
1	Bob	Addy
2	Bob	Allen
3	Doug	Allison
4	Sandy	Alomar
...	...	...
603	Eddie	Yost
604	Ned	Yost
605	Cy	Young
606	Charles	Zimmer
607	Don	Zimmer

608 rows × 2 columns

Solution END

Task 3b) 1.5 points]

Task description

Within all managers that coached at least 20 games in total, find the TOP 10 in terms of their win/loss ratio. The table should contain the first name, the last name and the win/loss ratio and the total number of games the manager coached. Sort the table of the these managers starting with the most successfull one (in terms of the win/loss ration).

The win/loss ratio is defined as the number of all wins a manager had during his entire career divided by the sum of all lost games during his entire career. The wins/losses are listed in the `managers` table (for each year spearately). The first and the last name is listed in the `people` table.

One of the challenges here is to combine these two tables in the right manner while simultaneously applying the restrictions/sorting/etc.

Make sure that the win/loss ratio is stored as a double and not as an integer. One possible solution for this is to define the win/loss ratio as total_wins/total_losses*1.0. (The key part is to multiply the ratio with 1.0.)

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [ ]:

```
#Select the individuals from people, the win/loss ratio and the total games
#add the personal info on playerid
#order by the best win/loss ratio first and show the top 10
pd.read_sql("""
    SELECT People.nameFirst,People.nameLast,
        (1.0 * SUM(Managers.W)) / SUM(Managers.L) AS winloss_ratio,
        SUM(Managers.G) AS total_games
    FROM Managers
    JOIN People
        ON People.playerID = Managers.playerID
    GROUP BY Managers.playerID
    HAVING SUM(Managers.G) >= 20
    ORDER BY winloss_ratio DESC
    LIMIT 10;
""", db)
```

Out[ ]:

	nameFirst	nameLast	winloss_ratio	total_games
0	Dick	Higham	2.636364	40
1	Joe	Start	2.571429	25
2	George	Wright	2.360000	85
3	Mase	Graffen	2.294118	56
4	Count	Campau	1.928571	42
5	Dick	McBride	1.894118	252
6	Tim	Bogar	1.750000	22
7	Dave	Roberts	1.713911	1034
8	Mike	Walsh	1.700000	110
9	Al	Spalding	1.659574	126

Solution END

Task 3c) [1 point]

Task description

Make a list of the TOP 10 colleges in the following sense: For each college, sum the total wins of each of its players over their whole career. List the top 10 most successfull colleges by Name (first column), the state where it is located (second column) together with the total sum of all wins that all their players achieved (third column). Get the players of each College from `CollegePlaying`, the wins of each player from the table `pitching` and the name/state of each College from the table `schools`.

Ignore the fact that some players have played for several colleges: We say a player is linked to a college, if he has played at least one season for this college.

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [ ]:

```
#Select the school and total wins from a subset of distinct school/player pa
# then join an additional subset of data from the pitching table on playerID
# the get one row per school, sort with most wins first and only show the to
pd.read_sql("""
    SELECT Schools.name_full, Schools.state,
        SUM(PlayerWins.sum_wins) AS total_wins
    FROM (
        SELECT DISTINCT schoolID, playerID
        FROM CollegePlaying
    ) AS CollegePlayers
    JOIN (
        SELECT playerID, SUM(W) AS sum_wins
        FROM Pitching
        GROUP BY playerID
    ) AS PlayerWins
      ON PlayerWins.playerID = CollegePlayers.playerID
    JOIN Schools
      ON Schools.schoolID = CollegePlayers.schoolID
    GROUP BY Schools.schoolID
    ORDER BY total_wins DESC
    LIMIT 10
""", db)
```

Out[ ]:

	name_full	state	total_wins
0	University of Southern California	CA	1605
1	University of Texas at Austin	TX	1369
2	University of Oklahoma	OK	984
3	Stanford University	CA	983
4	California State University Fresno	CA	950
5	University of Notre Dame	IN	829
6	University of Michigan	MI	812
7	University of Tennessee	TN	779
8	Arizona State University	AZ	754
9	Fresno City College	CA	746

Solution END

Exercise 4: Concurrency [3.5 points]

In this exercise, you will use multithreading to speed up your code. In this exercise, you are required to write code in Python and use SQlite commands.

Task 4a) [1 point]

Write a function `total_earnings` that takes a playerID as input variable and returns the sum of all salaries of the corresponding player over their entire career as a `numpy.float`. The function should use `pd.read_sql` once and then return one specific value of the database.

Make sure the function `total_earnings` is safe against **SQL injections**.

Task description

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [ ]:

```
#Function that gets the salary and sums all the salaries for a player
def total_earnings(playerID):
    query = """
        SELECT SUM(salary) AS total_salary
        FROM Salaries
        WHERE playerID = ?
    """
#Make a secure dataframe that is safe against sql injections
    df_secure = pd.read_sql(query , db, params=[playerID])
#Make the output a numpy.float
    salary = np.float64(df_secure.loc[0, "total_salary"])
 #Output that value
    return salary
```

Solution END

Examples

The following examples are provided to help you for the task.

In [ ]:

```
total_earnings("rodrial01")
```

Out[ ]:

```
np.float64(398416252.0)
```

Validation

Please run the following code lines. Wrong results or errors in the following code may still get partial credits - as long as the following code is executed.

In [ ]:

```
total_earnings("aaronha01")
```

Out[ ]:

```
np.float64(nan)
```

In [ ]:

```
type(total_earnings("aaronha01"))
```

Out[ ]:

```
numpy.float64
```

Task 4b) [0.5 points]

Task description

Query the `batting` table to obtain a `pandas.Series` object named `players` containing unique copies of all `playerID` s in that table.

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [ ]:

```
players = pd.read_sql("""
    SELECT DISTINCT playerID
    FROM Batting
""", db)["playerID"]
```

In [ ]:

```
type(players)
```

Out[ ]:

```
pandas.Series
```

Solution END

Examples

The following examples are provided to help you for the task.

In [ ]:

```
players.head()
```

Out[ ]:

```
0     aardsda01
1     aaronha01
2     aaronto01
3      aasedo01
4      abadan01
Name: playerID, dtype: str
```

Validation

Please run the following code lines. Wrong results or errors in the following code may still get partial credits - as long as the following code is executed.

In [ ]:

```
players.shape
```

Out[ ]:

```
(20469,)
```

Task 4c) [2 points]

Task description

Rewrite the function `total_earnings` for the following task. It should now also return the `playerID`. Furthermore, add other code if needed for multithreading.

Use multithreading to call `total_earnings` with each entry of `players`, that is, with each `playerID` of the `batting` table. Create a sorted DataFrame, consisting of the playerID in the first column and the total earnings per player in the second column. Note that this first DataFrame MUST contain all players.

Sort and slice this DataFrame to get a second DataFrame with the TOP 10 players with the highest accumulated salary in descending order.

Use 20 threads and make sure that every thread has its own connection to the Database. However, every thread should connect to the Database only once. You may write/use another function `get_db_access` to achieve this.

Wrap the task into a function called `task` that returns the second DataFrame consisting and prints the total time elapsed during the execution.

Solution START

All code for this task must be written between this `Solution START` and the following `Solution END` block.

In [ ]:

```python
# instantiates thread to create local data
thread_local = threading.local()  # thread-local storage for per-thread DB c

#connect tot the database
def get_db_access():
    if not hasattr(thread_local, "conn"):
        thread_local.conn = sql.connect(file_path)
    return thread_local.conn

#Copied from total earnings
def total_earnings(playerID):
    conn = get_db_access()
    query = """
        SELECT SUM(salary) AS total_salary
        FROM Salaries
        WHERE playerID = ?
    """
    df = pd.read_sql(query, conn, params=[playerID])
    salary = np.float64(df.loc[0, "total_salary"])

    return playerID, salary

# run total-earnings for multithreading using the 20 workers
def compute_all_players(players):
    with concurrent.futures.ThreadPoolExecutor(max_workers=20) as executor:
        return list(executor.map(total_earnings, players))


def task():
    start_time = time.time()

    #get the player details
    players = pd.read_sql( "SELECT DISTINCT playerID FROM Batting",
        sql.connect(file_path))["playerID"]

    # now get all unique playerids
    playerids = players.dropna().astype(str).unique().tolist()
    #get all their earnings
    results = compute_all_players(playerids)

    # dataframe with all players
    all_players = pd.DataFrame(results, columns=["playerID", "total_earnings

    # dataframe with just the top 10 players
    top10_players = (
        all_players
        .sort_values(["total_earnings", "playerID"], ascending=[False, True]
        .head(10)
    )
    #get the time the task takes
    print(time.time() - start_time)

    return top10_players
```

Solution END

Examples

The following examples are provided to help you for the task.

In [ ]:

```
total_earnings
```

Out[ ]:

```
<function __main__.total_earnings(playerID)>
```

Validation

Please run the following code lines. Wrong results or errors in the following code may still get partial credits - as long as the following code is executed.

In [ ]:

```
thrs = task()
```

```
5.168535947799683
```

In [ ]:

```
thrs
```

Out[ ]:

	playerID	total_earnings
15683	rodrial01	398416252.0
9050	jeterde01	264618093.0
16102	sabatcc01	218642856.0
18243	teixema01	214275000.0
15044	ramirma02	206827769.0
1237	beltrca01	205782782.0
14881	pujolal01	204040436.0
2588	cabremi01	188410623.0
1693	bondsba01	188245322.0
1236	beltrad01	183140000.0