COMP 651 Data Analytics Techniques Assignment 5

8 possible points

This assignment has you working with a SQLight database and unittest code. Write a Python module, **assignment5**, in which three functions are defined to create or access a database file **mydatabase.db**. Function **create_database()** makes a connection to **mydatabase.db**, creates a cursor, and makes a table **mytable** with columns **name** (type **TEXT**) and **age** (type **INTEGER**). It populates the table with data from file **students.txt** (available from the Blackboard page for this assignment), whose first few lines are

```
Ahmad,24
Jordan,26
Tyler,30
Jonathan,26
Nicholas,25
Richard,24
```

Each line has the **name** value, a comma, then the **age** value. You can use the following to remove the new line at the end of a line and to split the resulting string into the **name** value, **nm**, and the string version of the **age** value, **ag**.

```
nm, ag = line.replace('\n', '').split(',')
```

Note that **ag** must be converted to an integer before it is inserted into the table. To insert these values, use a prepared statement that is executed for each **name-age** pair:

```
INSERT INTO mytable(name, age) VALUES (?, ?);
```

Function **students_of_age(*years_old*)**, where ***years_old*** is an integer (age in years), returns a list of the names of the students whose age is ***years_old***. Write a prepared **SELECT** statement accessing table **mytable** and restricting to where the value of the **age** column is **?**, where '**?**' is the placeholder in the prepared statement. When this is executed, it expects one value, which is provided by a singleton tuple of the form **(*a*,)**. Use **fetchall()** to get a list of 1-tuples containing the names of those whose age is ***years_old***. Return the list that consists of just the names (not the 1-tuples with the names).

Function **average_age()** returns the average of the values in the **age** column of **mytable**. Have a **SELECT** command that lets you fetch the list of all **age** values as singleton tuples. To calculate the average, you must extract the values from the tuples, sum them, and divide the result by the length of the list. You may want to round the result to a value with just two digits after the decimal point; use built-in function **round()**.

Include the following test code at the end of you file.

```
if __name__ == '__main__':
    create_database()
    print(students_of_age(26))
    print(f'{average_age():.2f}')
```

The answers should be

```
['Jordan', 'Jonathan']
25.83
```

Manually remove **mydatabase.db** after each run; include docstrings and type hints.

Write a test file, **test_my_db.py**, for your database functions. Call the testcase **TestMyDataBase**. The **setUp()** method can simply call **assignment5.create_database()**. The **tearDown()** method can use the **remove()** function from the **os** module to delete **mydatabase.db**. And it should write an entry in the file **log.txt**. This file (also available from the Blackboard page) starts out with a single line

```
Log of executions of tests
```

The entry produced by **tearDown()** starts with the id of the method (use **self.id()**), followed by a comma, followed by the date and time in the format YYYY-MM-DD HH:MM:SS. The following is an example of the content of this file after the test file has been executed once.

```
Log of executions of tests
__main__.TestMyDataBase.test_average_age, 2023-02-19 19:58:17
__main__.TestMyDataBase.test_students_of_age, 2023-02-19 19:58:17
```

(We describe how to get current values for the data and time components below.) Note that file **log.txt** should be opened for appending, mode **a**. Write test methods for functions **students_of_age()** and **average_age()**. For the former, pass the integer 26 to **students_of_age()** (the integer used in the test code attached to **assignment5.py**), and compare the results to the list **['Jordan', 'Jonathan']**. For testing **average_age()**, compare the value returned to 25.83. Since these are floating-point values, use **assertAlmostEqual()** with **places=2**.

Include docstrings and (where they make sense) type hints (there will be only a few).

Class `datetime` in Module `datetime`

To have unqualified access to the class datetime in module datetime, use

```
from datetime import datetime
```

A class method of this class is **now()**, which returns a **datetime** object with the components of the current date and time as attributes. These attributes have self-explanatory names: **year**, **month**, **day**, **hour** (0-23), **minute**, **second**, **microsecond**. The following shows how to get a date-time in the desired format.

```
>>> print(f'{datetime.now():%Y-%m-%d %H:%M}')
2023-02-24 23:55
```