ASSIGNMENT 5

Contents

[Contents 1](#_Toc469074905)

[Purpose 2](#_Toc469074906)

[Assignment Background 2](#_Toc469074907)

[Assignment Statement 2](#_Toc469074908)

[Requirements: 2](#_Toc469074909)

[Code/Comment Format 5](#_Toc469074910)

[What to Deliver 6](#_Toc469074911)

[Notes 6](#_Toc469074912)

[Grading 7](#_Toc469074913)

# Purpose

This assignment is the second of a mini project where you are demonstrating your understanding of the modules of this class.

The mini project is about building a small ETL program in python. Each pat of the project focuses on different portions / stages of the ETL process.

# Assignment Background

Summarizing your acquired knowledge from module 5, we are going to focus on the L part of the ETL process. For reference, ETL stands for Extract, Transform, Load. We are going to focus here on Load.

# Assignment Statement

* Save data in a database
* Read data from a database

The file*\_\_init\_\_.py* is blank and should not be modified. You are required to supply *Last Name\_First Name\_Project\_Part2.py.*

# Requirements:

Let's create a place to store each dataset separately.

1. Create a “create\_dbs.py” file.
2. In this file created in step 1, write python code using sqlite3 to:
   1. Create a “baseball.db” database
      1. Create 1 table named “baseball\_stats” with the following columns:
      2. player\_name
      3. game\_played
      4. average
      5. salary

The "baseball.db" SQLite3 database should look like this:

Baseball\_stats

========================

name text

games\_played real

average real

salary real

1. In this file created in step 1, write python code using sqlite3 to:
   1. Create a “stocks.db” database:
   2. Create 1 table named “stock\_stats”
      1. company\_name
      2. ticker
      3. exchange\_country
      4. price
      5. exchange\_rate
      6. share\_outstanding
      7. net\_income
      8. market\_value\_usd
      9. pe\_ratio

The " stocks.db" SQLite3 database should look like this:

stock\_stats

========================

company\_name text

ticker text

country text

price real

exchange\_rate real

shares\_outstanding real

net\_income real

market\_value real

pe\_ratio real

1. Create AbstractDAO class. It should have the methods:
   1. it should have 1 (instance) member: db\_name
   2. insert\_records(records) – Should raise the NotImplementedError
   3. select\_all() - Should raise the NotImplementedError
   4. connect():
      1. connect to the database identified by db\_name
      2. returns the created connection
2. Create BaseballStatsDAO class
   1. Class should inherit AbstractDAO
   2. Class should implement the methods listed:
      1. insert\_records:
         1. takes a list of records as parameter ( BaseballStatsDAO takes BaseballStatRecord)
         2. call the method connect()
         3. using the returned connection, create a cursor.
         4. For each record in the list, write and execute an INSERT INTO statement to save the record's information to the correct table.

Example for baseball: cursor.execute(“INSERT INTO baseball\_stats VALUES ( ?, ? , ? , ?)”, (name, number\_game\_played, avg, salary))

* + - 1. Commit the connection
      2. Close the connection
    1. select\_all
       1. call the method connect()
       2. using the returned connection, create a cursor.
       3. create an empty deque to hold the records in memory
       4. write and execute a SELECT statement to get all the records of the table for the DAO

Example for baseball: cursor.execute(“SELECT player\_name, games\_played, average, salary FROM baseball\_stats;”)

* + - 1. For each row fetched, iterate with a for loop over the result of your select command
         1. Create a new record (BaseballStatRecord)
         2. Add the record to the deque
      2. Close the connection
      3. Return the deque

1. Create StockStatsDAO class
   1. Class should inherit AbstractDAO
   2. Class should implement the methods listed:
      1. insert\_records:
         1. takes a list of records as parameter (StockStatsDAO takes StockStatRecord)
         2. invokes the method connect()
         3. using the returned connection, create a cursor.
         4. For each record in the list, write and execute an INSERT INTO statement to save the record's information to the correct table.

Example for stocks: see insert statement for baseball stats

* + - 1. Commit the connection
      2. Close the connection
    1. select\_all
       1. invokes the method connect()
       2. using the returned connection, create a cursor.
       3. create an empty deque to hold the records in memory
       4. write and execute a SELECT statement to get all the records of the table for the DAO

Example for stocks: refer to select statement for Baseball

* + - 1. For each row fetched, iterate with a for loop over the result of your execute:
         1. Create a new record (StockStatRecord)
         2. Add the record to the deque
      2. Close the connection
      3. Return the deque

Use the code written for Project Part 1. If needed, a partial solution can be provided by your facilitator. This will only be done on a case by case basis and only in extreme circumstances. A 10-point deduction will be made a solution is needed.

This section loads the records into the correct database using the classes from Project Part1

1. load MLB2008.csv using the BaseballCSVReader
2. load StockValuations.csv using the StocksCSVReader
3. Instantiate a new DAO instance for BaseballStats
4. Instantiate a new DAO instance for StocksStats
5. Insert the loaded records into baseball database using Baseball DAO's insert\_records.
6. Insert the loaded records into stocks database using Stocks DAO's insert\_records.

Awesome! Now we have data in the database. Let's use it!

Stocks stats:

1. Using the instance of StockStatsDAO select\_all the records
   1. Calculate and print the number of tickers by exchange\_country using a dictionary.

Baseball stats:

1. Using the instance of BaseballStatsDAO select\_all the records
   1. Compute and print the average salary by “avg” using a dictionary

use round(record.avg, 3)) to get a more interesting average.

# Code/Comment Format

Good code includes well named variables that are consistent from the beginning to the end of the program. Naming of objects should be self-explanatory. For instance, iterator\_for\_noun\_list is much better than i.

Every program consists of a sequence of paragraphs, each of which has objectives, and which builds on the previous paragraphs. We are mostly interested in objectives that are valid at the end of the program so we can verify the program's design. The following is a preferred form for such paragraph headings. The # sign is adequate when the comment is a single line.

#This is an in-line comment – used to document the code for you, or anyone else, that intends

#To extend the code

In-line comments are helpful when one has to go back to the code 6 months later to make changes.

For doc strings, python allows the use of triple quotes. The triple quotes can be either single or double quotes. A doc sting is generally used as user documentation. It does not need to include details of the implementation of the program, but instead it provides documentation as how to use the API for the program (input, output etc.)

For example:

“””

This is an example of a doc string

It allows multiple lines within the string.

“””

‘’’

This is an example of a doc string

It allows multiple lines within the string.

‘’’

This becomes significant when using functions, classes etc. as the triple quotes help to self-document the parameters and return values of the function.

# What to Deliver

Supply

1. A .doc file with any remarks/comments you want to make. We can insert feedback into the file. Include screen shots of key output. Please label this Word file <Last Name>\_<First Name>\_Assignment \_2. For example, Smith\_Joe\_Assignment\_1.
2. The *Last Name\_First Name\_Assignment2.py* file (the *\_\_init\_\_.py* file is attached and should be left as is).

# Notes

* Assignments can be submitted once. If extenuating circumstances exist, contact your facilitator.
* Note the statement in the syllabus on timeliness of submissions (the gist being that all assignments must observe the deadlines).
* Start by identifying and ordering the objectives.
* There are no testing requirements for this assignment. However, it would be prudent to make sure your program does not crash and all input validation is performed correctly.

# Grading

Step 2 -10 pts

Step 3 – 10 pts

Step 4a – 2 pts

Step 4b – 2 pts

Step 4c – 2 pts

Step 4d – 4 pts

Step 5a – 1 pt

Step 5b.1 – 5 pts

Step 5b.2 – 5 pts

Step 6a – 1 pt

Step 6b.1 – 10 pts

Step 6b.2 – 10 pts

Step 7 – 2 pts

Step 8 – 2pts

Step 9 – 2 pts

Step 10 – 2 pts

Step 11 – 5 pts

Step 12 – 5 pts

Step 13 – 15 pts

Step 14 – 15 pts