Assignment No. 1 - Logging and Rollback

Description

You are going to simulate a **logging and rollback system** for a simple database that encompasses individuals, their salary, department to which they are affiliated to in the corporation, and their civil status.

As we are focusing on the conceptual side and will be running a simulation, we will not be using any specific RDBMS. After all, every RDBMS includes among their capabilities this logging and rollback functionality.

To simplify the problem, we will focus on the attributes ‘Salary’, ‘Department’ and Civil_status' of three individuals: the transaction list in the *starting code* will specify the update transactions that will be processed.

Below you’ll find an image of the DB (a simple .csv file):

```
['Unique_ID', 'First_name', 'Last_name', 'Salary', 'Department', 'Civil_status']
['1', 'John', 'Lennon', '230000', 'Projects', 'Married']
['2', 'Joan', 'Doe', '100000', 'Human Resources', 'Single']
['3', 'Mary', 'Carpenter', '250000', 'Projects', 'Separated']
['4', 'John', 'Ingham', '125000', 'Projects', 'Separated']
['5', 'Rachel', 'Sturgeon', '197000', 'Engineering', 'Married']
['6', 'Hanifa', 'Salima', '50000', 'Engineering', 'Married']
['7', 'Femi', 'Okeke', '425000', 'Industries', 'Married']
['8', 'Moe', 'Khalifa', '325000', 'Industries', 'Married']
['9', 'Katy', 'Jones', '475000', 'Management', 'Single']
['10', 'Lin', 'Wang', '435000', 'Engineering', 'Married']
['11', 'Art', 'Blanket', '137000', 'Projects', 'Single']
['12', 'Vivek', 'Singh', '231000', 'Industries', 'Married']
['13', 'Amal', 'Khan', '230000', 'Projects', 'Single']
['14', 'Richard', 'Carpenter', '123000', 'Human Resources', 'Single']
['15', 'Ryuichi', 'Sakamoto', '321000', 'Processing Facilities', 'Single']
```

Transaction to be executed

The following code shows the transactions that will be processed:

```
''' METADATA
transactions = [['id1',' attribute2', 'value1'], ['id2',' attribute2', 'val
                ['id3', 'attribute3', 'value3']]
'''
transactions = [['1', 'Department', 'Music'], ['5', 'Civil_status', 'Divorc
                ['15', 'Salary', '200000']]
```

There are three transactions in the queue:

- For ID='1' change the value in the attribute **Department** to 'Music'.
- For ID='5' change the value of the attribute **Civil-status** to 'Divorced'.
- for ID='15' change the value of the attribute **Salary** to '200000'.

Note

- Transactions are satisfactorily completed ONLY when a **commit** has occurred.
- If one is processing transaction No. 2 and an error occurs, the rest of the transactions in the list -those not processed yet- will be ignored (we are doing this to make the problem simpler).
- As the *simulated failure* occurs randomly, you may have to run your code several times until a *failure happens*.

Simulation Requirements

- Understanding whether your logging & rollback system is successful, will depend on 2 aspects:

    1. The contents of your logging & rollback data structures (your design)
    2. The status of the values for the attributes expected to be modified (in your DB).

- COMMIT: as we know, a COMMIT is the signal we send to the RDBMS, telling it that a logical-transaction has been executed successfully. Hence, it is time to make the changes we've performed to the Database, persistent (save them to secondary storage).

- In our simulation, Main_Memory corresponds to the data structure you are manipulating in your code (Main-Memory) to make a given transaction happen.

- In our simulation, SECONDARY_MEMORY -the place where the data will be persisted- will be a file in your computer (you can assume that the structure of the file is a .csv file). The .csv files for your assignment will be provided to you, and they correspond to the relations' images shown above.

- A successful transaction will show two things:

    1. Your .csv files (your table) should show the expected values **after** a successful transaction. You may want to save your updated files (.csv) with a different name, to prevent overwriting.
    2. Your Logging & Rollback Data Structure should show all transactions processed, indicating their status. Possible values are:
        - committed,
        - rolled-back, or
        - not-executed.

- An un-successful transaction will show two things:

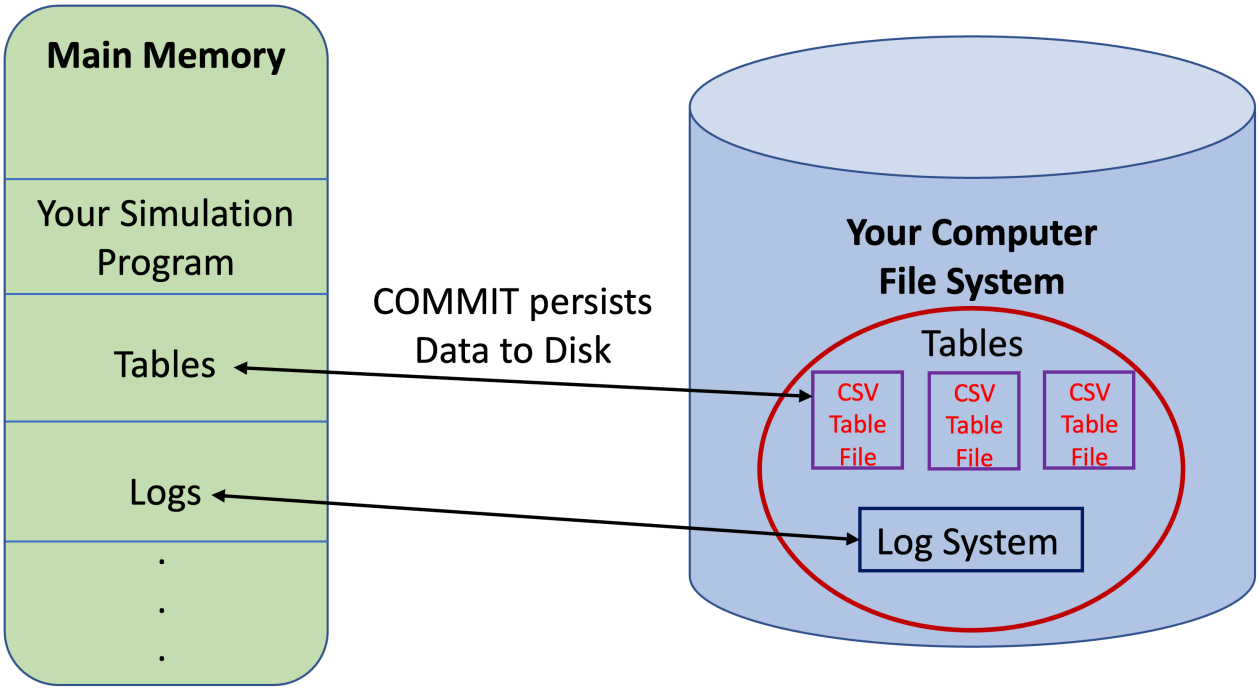
    1. Your .csv files (your Tables) should show the rolled-back values: no changes to the specific attribute in the database, if it was rolled-back.
    2. Your Logging & Rollback **Data Structure** should show what transaction(s) had been reversed (rolled-back).

You should print out the contents of your Logging & Rollback System as soon as an EXCEPTION/FAILURE is detected. If you prefer, you could make your data structure persistent by writing it to a file (this file MUST be different than the files that contain the original Database Tables/Relations). The recommendation would be to use one file (.csv) per relation/table.

You may want to record a video when you run your code. Please submit it in D2L and/or GitHub.

Proposed Architecture

The architecture for this "simulation" is as follows:



As you examine the diagram above, please consider the following:

- The .csv files representing the Database Tables should be read to main memory and manipulated there using a data structure appropriate for the job at hand (array, list, etc.)
- Once a COMMIT is issued -in the main program-, the data structure -representing the Tables- in Main Memory, is persisted to disk, in .csv format
- Your Log Sub-system, for which you must define its attributes and structure, will be in Main Memory and could be persisted to Disk, whenever you'd like, and using the format of your choice. You may want to think about this design aspect.
- The starting program provided will give you more insight into how tasks are supposed to be executed

Note: you **must** design a proper data structure for your **Log System**. Please document in your report your design decisions.

Starting Code (in Python)

The starting code has ben (purposely) written utilising very basic Python. This way, students that are not very proficient with the language, will still have the chance to understand and reuse the code. A brief explanation of the components of the starting code are given below:

The **main program** performs several actions:

```
- Reads the .csv file
- Navigates through the list of transactions
- A simulation of a random failure is executed:
    -  it could return failure or success
- If a failure did occur, it calls the program to perform roll-back
```

```
def main():
    number_of_transactions = len(transactions)
    must_recover = False
    data_base = read_file('Employees_DB_ADV.csv')
    failure = is_there_a_failure()
    failing_transaction_index = None
    while not failure:
        # Process transaction
        for index in range(number_of_transactions):
            print(f"\nProcessing transaction No. {index+1}.")
            #<--- Your CODE goes here (to process transaction at hand)
            print("UPDATES have not been committed yet...\n")
            failure = is_there_a_failure()
            if failure:
                must_recover = True
                failing_transaction_index = index + 1
                print(f'There was a failure whilst processing transaction \
                No. {failing_transaction_index}.')
                break
            else:
                print(f'Transaction No. {index+1} has been commited!
                Changes are permanent.')

    if must_recover:
        #Call your recovery script
        recovery_script(DB_Log)
        ### Call the recovery function to restore DB to sound state
    else:
        # All transactiones ended up well
        print("All transaction ended up well.")
        print("Updates to the database were committed!\n")

    print('The data entries AFTER updates -and RECOVERY, if necessary-
    are presented below:')
    for item in data_base:
        print(item)
```

Function for random stimulation of a **failure**. See header below:

```
def is_there_a_failure()->bool:
    '''
    Simulates randomly a failure, returning True or False, accordingly
    '''
    value = random.randint(0,1)
    if value == 1:
        result = True
    else:
        result = False
    return result
```

transaction_processing and **recovery_script**. See headers below (**these are the two functions you must write**. You are free to write any additional *help functions* you may need):

```
def recovery_script(log:list):  #<--- Your CODE
    '''
    Restore the database to stable and sound condition, by processing
    the DB log.
    '''
    print("Calling your recovery script with DB_Log as an argument.")
    print("Recovery in process ...\n")
    pass

def transaction_processing(): #<-- Your CODE
    '''
    1. Process transaction in the transaction queue.
    2. Updates DB_Log accordingly
    3. This function does NOT commit the updates, just execute them
    '''
    pass
```

Designing the Logging & Rollback Sub-system

- The design of the data structure -and its contents- for the Logging & Rollback sub-system is in your hands.
- You can choose any data structure that works for you. Having said that, the material covered in class -as well as the Class' Notes- should provide all the advice you need to design your solution.

Output and Documentation

```
- Your output should show the status of the DB Tables before and after
the List of Transactions is processed (see starting code).

- In addition, the contents of your Log System should be printed, too,
showing the status for each transaction.

- Documentation: in-line comments are welcomed.
- Design decisions and data structures and processes descriptions, must
be included in your report.
- Samples of your outputs, as described in the previous bullet points, shou
be included in your report.
```

Programming Language of Choice

- **Python** (preferred option)
- If for some reason you truly believe that you must use another programming language, please discuss it with your instructor.

Submitting your work / Due Date

Due Date: **Thursday 15/Feb/2024 at 5:00PM**

You could post your work three different ways: - Using this GitHub Repo (kindly post your code to the folder "YourCode"), or - Using the D2L option available to all students, or - Using both, the GitHub Repo and D2L

Note: only **one** member of the team needs to post the work. Make sure all the names of the members of each team are clearly referenced in your submission. You can work alone or in groups (a **maximum** of **4** *students* per group).