CSE 202: Fundamentals of Database Systems
Winter 2018
Home Assignment 2 [12 marks]
Due Date: 14-02-2018 (Three Weeks time: No extension will be allowed)

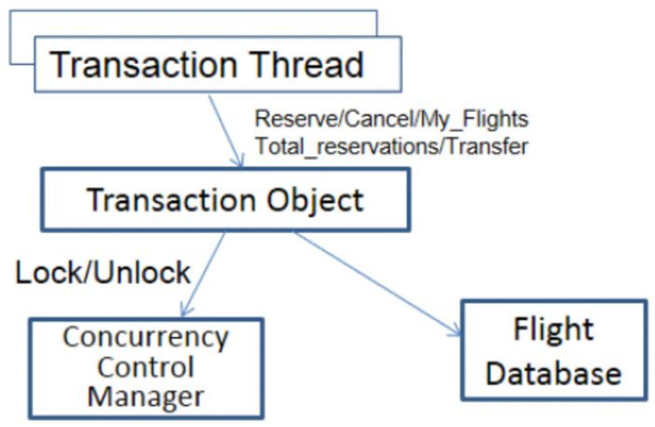
Instructions:

- Write the programs in **Java** only.
- Only one submission per group.
- The naming convention for the files: filename_rollnumber.extension like program1_2016001_2016002.java.
- **The assignment can be done individually or in groups of maximum 2 students.**
- **Compress all the input/output files along with the programs as tar.gz. Your submission (.tar.gz) must contain a text file with your group details (name and roll number of student). Your submission must contains files only with no folder hierarchy created.**

In this assignment, you have to implement a concurrency control algorithm for an airline database. The system consists of a transaction threads, concurrency control manager and an airline database. The database has a set of transactions and data structures containing flight information (passenger lists for all flights). Each transaction thread (TT) should have the following structure:

```
Repeat
    Select a transaction type randomly
    Select object(flight and passenger id) for transaction randomly
    Invoke transaction
```

The possible transactions are:
• **Reserve(F, i)**: reserve a seat for passenger with id i on flight F, where i > 0.
• **Cancel(F, i)**: cancel reservation for passenger with id i from flight F.
• **My_Flights(id)**: returns the set of flights on which passenger i has a reservation.
• **Total_Reservations()**: returns the sum total of all reservations on all flights.
• **Transfer(F1,F2,i)**: transfer passenger i from flight F1 to F2. This transaction should have no impact if the passenger is not found in F1 or there is no room in F2.


Transaction Thread
Reserve/Cancel/My_Flights
Total_reservations/Transfer
Transaction Object
Lock/Unlock
Concurrency Control Manager
Flight Database

You must choose appropriate data structures to implement the flight database. However, these data structures must contain no synchronization code. The Concurrency Control Manager (CCM) must implement the mechanism to control access to the data structures. At the minimum, it should include locking and unlocking operations. The Transaction object must implement the operations Reserve, Cancel, My_Flights, Total_reservations and Transfer operations (described above). Each of these operation must invoke the necessary lock and unlock operations and operations on the data structures.

- **CCM implementation with lock and unlock operations ⟶ 1 mark (0.5 marks if the lock compatibility is not checked before locking an item i.e. a transaction can't acquire a shared lock on item X if another transaction has locked item X using exclusive lock. - You need to check the code for it.)**

The goal is to study **performance improvements** based on the granularity of locking. **The first version of the program must implement only serial schedules (lock the entire database). You must then compare the performance to your second version which does two-phase locking at a more fine-grained level. You must exploit semantics of the operations to allow as much concurrency as possible.**

- **Correct implementation of both versions ⟶ 2 marks (0.5+1.5) (0.5 marks for the serial version i.e. locking the database every time and 1.5 marks for implementing 2 PL correctly.) (Deduct 0.5 marks if students have not implemented the global ordering to avoid deadlocks i.e. every transaction must acquire locks in a fixed global order. As an example, I may decide to lock seats first, followed by passenger and then flights.)**
  [Here, check for ordering of items through serial schedules, and order of locks acquired by each transaction to avoid any deadlocks; as throughput for 2-phase locking shall be greater than the serial version]

The performance must be measured in terms of transaction throughput (number of transactions completed over a specific interval of time). **You must have a sleep statement inside the function** calls of the data structures to simulate time required to access the database. You must plot the following data for each version: **(a) Impact of number of TT threads on throughput, (b) Impact of the transaction mix on throughput**.

- **Use of Sleep statement ⟶ 1 mark**
- **Plot for each version ⟶ 2 marks [zero marks if no plots and zero marks also if 2PL implemented incorrectly.]**
  [Plots for both the versions shall be relevant, where 2-phase locking shall show greater slope than the serial version]
- **For correct creation of threads with specified nos. created ⟶ 1 mark**

You should have information for around 5 flights. Each flight should have a different bound of the number of seats available. The number of TT threads should be increased to a sufficient number to see a trend the performance graphs.

- **HW2 demo ⟶ 3 marks**
  - ➢ ***Test case - 1*** **[for checking concurrency control (*1 mark*)]**
    reserve(F2,1)
    reserve(F1,2)
    cancel(F2,1)
    transfer(F1, F2, 2)
    my_flights(2)
    total_reservations
  - ➢ ***Test case - 2*** **[for displaying no change if F2 is already full (*1 mark*)]**
    F2 full
    transfer(F1,F2, 1)
    reserve(F1, 2)
  - ➢ ***Test case - 3*** **[to check what it does when there is no such passenger, e.g. Passenger 2 not in F1 (*1 mark*)]**
    reserve(F1,1)
    transfer(F1,F2,2)
- **Viva ⟶ 2 marks [binary]**
  [zero marks if anything is not answered related to the code by both group partners or student(s) who've submitted individually]