CSE 4701, Fall 2019

Project 2 – Bank Accounts and Transaction Processing

Part II (Due November 20/2019 (Wed), midnight at HuskyCT) (100 pts)

1. Part II Assignment

Extend your program **menu** from part 1 to support the following features:

i. **Deposit** – Ask the bank teller for account number. Show the account details (number, name, balance, and account open date) and **lock** the account for update. Then ask the teller to enter deposit amount. Update balance on the account and release the lock. Show new balance.

ii. **Withdraw** – Ask for the account number. Show the account details and **lock** the account for update. Ask for the withdrawal amount. Update balance on the account and release the lock. Show new balance. Display error if balance is insufficient for withdrawal.

iii. **Transfer** – Ask for the source account number. Show account details and **lock** the source account. Ask for target account number. Show details and **lock** the target account. Ask for the transfer amount. Withdraw amount from source account. Display error message in case of insufficient balance. Make your program go to sleep for 10 seconds to simulate network congestion. Deposit the amount into the target account. Go to sleep for 10 seconds again. Commit the transaction and show confirmation of transfer.

2. Deliverables

You are required to submit your source code (zip file if there are multiple source files) and a report (PDF file) with following contents (use same number system in your report):

i. **Screenshots:** Attach screenshots from your program and corresponding code snippet to demonstrate each subsection of assignment in Section 4 above. Also demonstrate that your program satisfies the following **ACID** properties.

ii. **Atomicity:** Atomicity must be maintained during Transfer transaction. Both withdrawal and deposit operations should be committed at once. Or none of them should be committed. Imagine you have a power failure during the transfer (between withdrawal and deposit). You can simulate this by closing your program while your program is sleeping after withdrawal. Now run your program again and check account balances. Demonstrate that your program maintains atomicity property.

iii. **Consistency:** In case of unexpected power-failures or other network disruptions, database should always be in valid state. No transaction or operation should put database into invalid state.

iv. **Isolation**: Now assume that multiple bank tellers are using your program simultaneously. Open multiple windows of your program and demonstrate the **concurrency control** feature of your program. Demonstrate that same bank account cannot be simultaneously accessed by multiple bank tellers to withdraw, deposit or transfer. When one teller is working on an account, this account should be locked for all the other bank tellers. However, other tellers should be able to work on other

accounts (with different account number). Experiment using various lock methods (table lock or row lock) and discuss which one is preferred in this case.

v. **Durability:** This is something you have already demonstrated in 3(ii).