**CSE 5331/4331 Database 2 Summer 2021**

**Professor Ramez Elmasri**

**Project 1**

In this project, you will implement a program that **simulates** the behavior of two variations of the **two-phase locking (2PL) protocol** for concurrency control.

Part 1: The first protocol to be implemented will be **rigorous 2PL, with the *cautious waiting* method for dealing with deadlock.**

Part 2: The second protocol to be implemented will be **rigorous 2PL, with the *wait die* method for dealing with deadlock.**

The input to your program will be a file of transaction operations in a particular sequence. Each line has a single transaction operation. The possible operations are: **b (begin transaction), r (read item), w (write item), and e (end transaction)**. Each operation will be followed by a **transaction id that is an integer between 1 and 9**.

For r and w operations, an item name follows between parentheses (**item names are single letters from A to Z**). Two example are given below.

Examples of two input files:

b1;                                                                          b1;

r1 (Y);                                                                     r1(Y);

w1 (Y);                                                                    w1(Y);

r1 (Z);                                                                     r1(Z);

b3;                                                                          b2;

r3 (X);                                                                     r2(Y);

w3 (X);                                                                   b3;

w1 (Z);                                                                    r3(Z);

e1;                                                                          w1(Z);

r3 (Y);                                                                     e1;

b2;                                                                          w3(Z);

r2 (Z);                                                                     e3;

w2 (Z);

w3 (Y);

e3;

r2 (X);

w2 (X);

e2;

Important Note: The order of operations in the input sequence will be affected during the simulation – for example, some commands may cause a transaction to be blocked (wait). In such cases, the simulation should keep track of all subsequent operations of this transaction, and if the transaction is resumed, these operations should be simulated.

Your simulation should assign transaction timestamps using an incremental counter, based on the transactions start order, and the timestamps are integer numbers: 1, 2, 3, …. For example, in the first example input file (on left above), TS(T1) = 1, TS(T3) = 2, TS(T2) = 3, because T1 starts first, then T3, then T2.

You should do the following steps for Project 1:

1. Design and implement appropriate data structures to keep track of transactions (**transaction table**) and locks (**lock table**), as well as any other needed information (it is your responsibility as part of the project to determine any additional needed information).
2. In the **transaction table**, you should keep relevant information about each transaction. This includes transaction id, transaction timestamp, transaction state (active, blocked (waiting), aborted (cancelled), committed, etc.), list of items currently locked by the transaction, plus *any other relevant information*. (It is part of your work to determine and specify other relevant information needed.) For blocked transaction, you should also keep an ordered list of the operations of that transaction (from the input file) that are waiting to be executed if the transaction can be resumed.
3. In the **lock table**, you should keep relevant information about each locked data item. This includes item name, lock state (read (share) locked, or write (exclusive) locked), transaction id for the transaction holding the lock (for write locked) or list of transaction ids for the transactions holding the lock (for read locked), list of transaction ids for transactions waiting for the item to be unlocked (if any), plus *any other relevant information*. (It is part of your work to determine and specify other relevant information.)
4. Write a program that reads the operations from an input file representing a schedule, and simulates the appropriate actions for each operation by referring to and updating the entries in the transaction table and lock table. Your program should **print a short summary of the simulation action** that the program takes to simulate each command, including information on any updates to the system tables (transaction table and lock table), and if the simulation will commit or abort or block a transaction, or just allow the operation to execute.

Some basic information about the actions that your program should take (this list is *not* an exhaustive list):

1. A transaction record should be created in the transaction table when a **begin transaction operation (b)** is processed. The state of this transaction should be set to *active* when it begins. A transaction timestamp is created for the transaction and stored with the transaction record.
2. Before processing a **read operation (r(X)),** the appropriate **read lock(X)** request should be simulated by your program simulation, and the lock table should be updated appropriately. If the item is not locked, the operation is allowed and the lock table is updated to indicate that the requested item is now read-locked by the transaction.

* If the item X is already locked by a conflicting write lock by another transaction, the requesting transaction is either: (i) blocked (*wait*) and its transaction state is changed to blocked (in the transaction table), or (ii) the simulation will abort the requesting transaction requesting. The decision to block (wait) or abort is based on the particular protocol rules (cautious waiting for Part 1, wait-die for Part 2).
* If the item is already locked by a non-conflicting read lock, the transaction is added to the list of transactions that hold the read lock on the requested item (in the lock table) and is not blocked.

1. Before processing a **write operation (w(X)),** the appropriate **write lock(X)** request should be processed by your program simulation (lock upgrading is permitted if the upgrade conditions are met – that is, if the item is read locked by only the transaction that is requesting the write lock). The lock table should be updated appropriately.

* If the item is not locked, the operation is allowed and the lock table is updated to indicate that the requested item is now write-locked by the transaction.
* If the item is already locked by a conflicting read or write lock, the transaction is either: (i) blocked (*wait*) and its state is changed to blocked (in the transaction table), or (ii) the simulation will abort the requesting transaction. The decision to block (wait) or abort is based on the particular protocol rules (cautious waiting for Part 1, wait-die for Part 2).

1. Your simulation program should include code **for each operation needed**. In addition to writing the code for each operation that can appear in the input schedule **b(i), r(i, X), w(i, X), e(i),** you should also write code for the **following additional operations** that will be needed by your simulation and can be called from other procedures in your code: **readlock(i, X), writelock(i, X), unlock(i, X), commit(i), and abort(i).**
2. Before processing an operation in the input list, you should check if the transaction **is in a blocked state or aborted state.** (i) If it is **blocked**, add the operation to the ordered list of the operations of that transaction that are waiting to be executed (in the transaction table); these operations will be executed if the transaction is resumed, subject to the rules of the locking protocol. (ii) If the transaction has already been **aborted**, its subsequent operations in the input list are ignored (you can print “transaction already aborted” in your program output).
3. Before changing a transaction state to **blocked,** your program should check the *deadlock prevention* protocol rules (*cautious waiting* for Part 1; *wait-die* for Part 2) to determine if the transaction should wait (be blocked) or abort.
4. The process of **aborting a transaction** should release **(unlock)** any items that are currently locked by the transaction, one at a time, and changing the transaction state to “aborted” in the transaction table. Any subsequent operations of the aborted transaction that are read from the input file should be ignored by the simulation (you can print “transaction already aborted” in your program output).
5. If a transaction reaches its end (e) operation successfully, it should be **committed**. The process of committing a transaction should release **(unlock)** any items that are currently locked by the transaction, one at a time, and changing the transaction state to committed in the transaction table.
6. The process of **unlocking an item** should check if any transaction(s) are blocked because of waiting for the item. If any transactions are waiting, it should remove the first waiting transaction and grant it access to the item (note that this will relock the item) and thus resume the transaction. All waiting operations of the transaction are processed (as discussed above, subject to the locking protocol) before any further operations from the input file are processed.

Due Dates:

1. You should turn in an *intermediate report* by  **by 11.59pm on Tuesday July 20** that includes your preliminary design of the program **psuedo-code** (high-level code description – this is written in English with programming language constructs as needed). Include the descriptions of the data structures that will be used for **transaction table, lock table,** etc. Include one **procedure/function psuedo-code** with documentation corresponding to the actions your simulation will do for each input operation, as well as for commit, abort, block (wait), etc.
2. The *completed project* is due by **Tuesday July 27 by 11.59pm.** Include: (i) Sufficient **documentation**. (ii) An **output file for each input** should be turned in that prints the actions taken by your program when processing each operation, including the description of any changes to the transaction table and lock table. (iii) The **source code** for your program with sufficient **internal documentation (comments)** for the GTA to understand the code.
3. Clear **instructions on how to execute your program** so that the GTA can run your program code.
4. If the project is done in a group of 2, then for both items 1 and 2 above, describe **which part was done by each group member**.

 Note 1: This program can be written in any programming language.

 Note 2: This project will be done groups of 2 students or individually.

Note 3: A number of input files will be provided on Canvas. You should run each of the provided input files and turn in your program output for each provided input file for both Part 1 (cautious waiting) and Part 2 (wait-die).