CS 5331 Project 1 Design document

Team members:

1. Tanveer Ahmed Shaik – 1001704423
2. Satish Rella – 1001677995

Objective:

Implement a program that simulates the behavior of two-phase locking (rigorous 2PL) protocol for concurrency control using Wound -Wait deadlock prevention method.

Programming language: Java 1.8.

Classes and Data structures used:

- Transaction table –class (TransactionDetails.java)

```
private int tId;
private Timestamp timestamp;
private String state;
private List<String> dataItemsHeld;
private Queue<String> waitingOperations;
```

  TansactionList will hold the TransationDetails objects.

- Lock table –class (Lock.java)

```
private String dataItem;
private String lockType;
private List<Integer> accessingTIds;
private Queue<Integer> waitingTids;
```

  LockList will hold the Lock objects.

Class level pseudocode:

```
class Rigorous2PLWithWoundWait {

    static List<TransactionDetails> transactionList = new
    ArrayList<transactiondetails>();

    static List<Lock> lockList = new ArrayList<Lock>();;
}
```

Methods used:

1. **Main method:**
   Starting point of the program where the input files is read line by line and various methods are called appropriately.

   Pseudocode:
   Main() {

   1. Read input file line by line till entire file is parsed.
   2. If line starts with b
      then call beginTransaction(tId);
   3. If line starts with e
      then call endOrAbrtTransaction(tId, "COMMIT")
   4. If line starts with r
      then readTransaction(tId, dataItem)
   5. If line starts with w then
      then callWriteTransaction(tId, dataItem)
   6. Call display() to print.

   }

2. **Begin Transaction method:**
   This method is called whenever 'b' operation is read from the input file. Tid (transaction id) is the input to this method.

   Pseudocode:
   beginTransaction(Integer tId) {

   1. Create ArrayLists dataItemsHeld.
      List<String> dataItemsHeld = new ArrayList<String>();
   2. Create a LinkedList waitingOperations and reference it as a Queue.
      Queue<String> waitingOperations = new LinkedList<String>();
   3. Create new TransactionDetails object.
      TransactionDetails tObj = new TransactionDetails(tId, new Timesatmp()
      , "Active", dataItemsHeld, waitingOperations);
   4. Now, created transactionDetails object i.e. tObj is added to the transactionList

   }

3. **Read transaction method:**
   This method is called whenever 'r' operation is read from the input file. Tid (transaction id) and dataItem are the input to this method.

Readtransaction (Integer tId, String dataItem) {

1. Get transaction details of the corresponding transaction id from the transactionList.
2. If state is ABORTED
   Then Ignore the operation.
   Else if state is BLOCKED
   Add the operation to the waitingOperations list of the corresponding transaction.
   Add the tId to the waitingTids queue of lock object of corresponding data item.
   ELSE If state is ACTIVE
   If Lock on dataItem is present in the lock list.
   Then If Read_Lock is present on the dataItem
   Add the tId to the accessingTids list of lock object.
   Also get the transaction details from the transactionsList and add the data item to the dataItemsHeld list.
   Else If Write_Lock is present on the dataItem
   Then call the WoundWait method with transaction id Ti, dataItem and transactionId Tj (i.e. transaction that is holding dataItem in Write_Lock).
   Else
   Then change the lockType to “Read_Lock” and add the tId to the accessingTids list of lock object.
   Also get the transaction details from the transactionsList and add the data item to the dataItemsHeld list.
   End If;
   Else
   Then add data item to dataItemsHeld list of the transaction.
   Create a Lock object with following dataItem, lockType, accessTids, waitingTIds.
   List<Integer> accessingTids = new ArrayList<Integer>();
   accessingTids.add(Ti);
   Queue<Integer> waitingTids = new LinkedList<Integer>();
   Lock obj = new Lock(dataItem, “Read_Lock”, accessingTids, waitingTids)
   And add the lock object to the locklist.
   End If;
   End If;

}

4. Write transaction method:

This method is called when a "w" operation is read from the input file. Transaction id and data item are the inputs to this method.

Pseudocode:

writeTransaction(Integer tId, String dataItem) {

1. Get the transaction details from transaction list.
2. If state is ABORTED
   Ignore the operation.
3. Else If state is BLOCKED
   Add the operation into waiting operations list of the respective transaction details object.
   Add the tId to the waitingTids queue of lock object of corresponding data item.
4. Else If state is ACTIVE
   If Lock on dataItem is present in the lock table
   Then If lockType is Read_Lock
   Get the accessingTids list from the lock object.
   If accessingTids.size() > 1
   Then call the WoundWait method with transaction id Ti, dataItem and on each transactionId Tj i.e. transaction that is holding dataItem) except Ti.
   Else if accessingTids.size() == 1
   If accessingTid == Tid(current Tid)
   Then upgrade Read_Lock on data item of lock object to Write_Lock.
   Else
   Then call the WoundWait method with transaction id Ti, dataItem and transactionId Tj (i.e. transaction that is holding dataItem in Read_Lock).
   End If;
   End If;
   Else If lockType is Write_Lock
   Then call the WoundWait method with transaction id Ti, dataItem and transactionId Tj (i.e. transaction that is holding dataItem in Write_Lock).
   Else
   Then change the lockType to "Write_Lock" and add the tId to the accessingTids list of lock object.
   Also get the transaction details from the transactionsList and add the data item to the dataItemsHeld list.
   End If;

Else

Then add data item to dataItemsHeld list of the transaction.
Create a Lock object with following dataItem, lockType, accessTids, waitingTIds.
List<Integer> accessingTids = new ArrayList<Integer>();
accessingTids.add(Ti);
List<Integer> waitingTids = new ArrayList<Integer>();
Lock obj = new Lock(dataItem, "Write_Lock", accessingTids, waitingTids);

End If;

}

5. **End or Abort transaction method:**
   This method is called whenever 'e' operation is read from the input file. Tid (transactionid) is the input to this method.

   Pseudocode:
   EndorAbortTransaction (Integer tId, String State) {
   1. Get the transaction details of tId.
   2. Get the data items held by the aborting or committed transaction from transaction details object.
   3. If state is Blocked or Active
      Then change the state to End or Committed i.e. (input state)
      Empty the dataItemsHeld list and waitingOperations queue from the transaction details object.
      Remove the corresponding tId from accessingTids and waitingTids from each lock object present in the lockList. If accessingTids list is empty change state to Unlock and remove a tId from waitingTids list in FIFO order and process it.
      Else if state is Aborted
      Then ignore the operation.
      End If;

   }

6. **WoundWait method:**
   This method is used for deadlock prevention using wound-wait protocol.

   Pseudocode:
   woundWait(Integer tId1,Integer tId2 String dataItem, String operation) {

   1. Get timestamp of tId1 by getting transaction details of tId2.
   2. Get timestamp of Tj
   3. If tId1 < tId2

Then call endOrAbortTransaction method with tId2 as input
Proceed with the operation for tId1.
Else
Set state of tId1 to BLOCKED in transaction details object of tId1.
Add tId1 to the waitingOperations queue of Tid.
Add tId1 to waitingTids queue of lock object of data item.
End if;
}

7. **Display method:**
This method prints the transaction table and lock table after every operation onto the console.