**Project 1:**

**CSE 5331:** Database Models and Implementation.

**Intermediate Report:** Documentation related to the Data Structures, sample Pseudo Code and operations of the program on the given input.

**By,**

**Pramod Kemisetty**

**1001570333.**

**Files To be Included in the Project:**

**Files:**

1. TransactionTable.java.
2. LockTable.java.
3. WaitDieAlgorithm.java.

* **WaitDieAlgorithm.java:** This file is the main entry point into the program and it contains all the logic to handle all the operations according to the Timestamp value on each of the transactions. It also reads the input (i.e. all the transaction from the input file) and wakes all the waiting transactions when the item is released or unlocked.
* **TransactionTable.java:** This file contains all the attributes of the transaction like the Transaction ID, Transaction Timestamp, Transaction State (i.e. active, Blocked, waiting, committed etc.), List of items locked by the transaction, operations waiting to be executed.
* **LockTable.java:** This file contains all the information about each locked data item like item name, lock state (read lock, or write lock), transaction id for the transaction holding the lock (for write locked) , list of transaction ids for the transactions holding the lock (for read locked), list of transaction ids for transactions waiting to be executed.

**Instructions to run the java files:**

Navigate to the directory which has all the required source files.

**Compile** all the java files using: **javac \*.java**

**Execute** all the java files using **java WaitDieAlgorithm <<Input file path>>**

**Sample Input Execution:**

**Input 1:**

b1; 🡪 Begin transaction: 1.

r1 (Y); 🡪 Item Y is “Read – locked” by Transaction: 1.

w1 (Y); 🡪 Item Y is upgraded to “Write Locked” by Transaction: 1.

r1 (Z); 🡪 Item Z is “Read Locked” by Transaction: 1.

b3; 🡪 Begin Transaction 3.

r3 (X); 🡪 Item X is “Read- Locked” by Transaction 3.

w3 (X); 🡪 Item X is upgraded to “Write Locked” by Transaction 3.

w1 (Z); 🡪 Item Z is upgraded to “Write Locked” by Transaction 1.

e1; 🡪 End of Transaction 1. Transaction 1 is committed.

r3 (Y); 🡪 Unsuccessful Operation. Transaction 3 aborted.

b2; 🡪 Begin Transaction 2.

r2 (Z); 🡪 Unsuccessful Operation. Read – Write Conflict. Transaction 2 Aborted.

w2 (Z); 🡪 Operation Ignored. Since Transaction 2 is already aborted.

w3 (Y); 🡪 Operation Ignored. Since Transaction 3 is already aborted.

e3;🡪 Operation Ignored. Since Transaction 3 is already aborted.

r2 (X); 🡪 Operation Ignored. Since Transaction 2 is already aborted.

w2 (X); 🡪 Operation Ignored. Since Transaction 2 is already aborted.

e2; 🡪 Operation Ignored. Since Transaction 2 is already aborted.

**Input 2:**

b1; 🡪 Begin Transaction 1.

r1(Y); 🡪 Item Y is “Read-Locked” by Transaction 1.

w1(Y); 🡪 Item Y is upgraded to “Write-Locked” by Transaction 1.

r1(Z); 🡪 Item Z is “Read-Locked” by Transaction 1.

b2; 🡪 Begin Transaction 2.

r2(Y); 🡪 Unsuccessful Operation. Read Write Conflict. Transaction 2 is Aborted.

b3; 🡪 Begin Transaction 3.

r3(Z); 🡪Item Z is “Read Locked Shared” by Transaction 3 (Shared with Transaction 1).

w1(Z); 🡪 Unsuccessful Operation. Transaction is Blocked (Waiting), since it is the older transaction.

e1; 🡪 End of Transaction 1. Commits.

w3(Z); 🡪 Item Z is upgraded to “Write Locked” by Transaction 3.

e3; 🡪 End of Transaction 3. Commits.

**Data Structure:**

**Transaction Table:**

* **T\_ID <INTEGER>** 🡪 Gives the Transaction Identification like 1,2,3…
* **T\_TimeStamp <INTEGER>** 🡪 Gives the Timestamp of the Transaction (Usually the Starting time of the Transaction).
* **TState <Enum>** 🡪 State of the Transaction i.e. Aborted, committed etc., Enum is the Datatype to accommodate all these fixed state names.
* **Items Locked <List> 🡪** List of all the items that have been locked.
* **Operations List <List> 🡪** List of all operations that can be performed.

**Lock Table:**

* **Item Name <String> 🡪** Name of the Item.
* **Lock State <Enum> 🡪** State of the Transaction i.e. Aborted, committed etc., Enum is the Datatype to accommodate all these fixed state names.
* **Locked By <List> 🡪** List of all Transactions that have locked the item. The shared access of the Read operation can have many transactions accessing the item.
* **Waiting List <List>** 🡪 List of all Transactions that are waiting for the access of the item.

**Pseudo Code:**

1. Create 3 Lists namely, TransList, LockList and ActiveTrans
2. TransList is a HashMap that contains the details of the transaction Timestamps.
3. LockList is also a HashMap that contains the details of the locks on the items.
4. ActiveTrans is an Array List of Transactions that contains all the list of transactions that are waiting on the queue.
5. //To check if the Transaction began or not.
6. If (readLine.charAt(0) == ‘b’)
7. Create the Transaction ID.
8. Create the Transaction state as “Active”.
9. System.out.println(“Operation Successful and Transaction (TID has started)”);
10. Set Transaction Timestamp.
11. Else if (readLine.charAt(0) == ‘r’)
12. If (transactionState.equalsIgnoreCase(“BLOCKED”)
13. Add operation to the List of operations that are waiting to be executed.
14. Else If (transactionState.equalsIgnoreCase(“ABORTED”)
15. //Print that the transaction is ignored.
16. System.out.println(“Operation Ignored”);
17. If (lockObj.LockMode. equalsIgnoreCase(“NOTLOCKED”)
18. Set the status as “Locked”.
19. Append it to the LockList.
20. Else if (lockObj.LockMode. equalsIgnoreCase(“READLOCKED”)
21. Append it to the LockList.
22. Else it means that it is “Write-Locked”
23. Call function check Deadlock (data Item, TID).
24. Else if (readLine.charAt(0) == ‘w’)
25. If(transactionState.equalsIgnoreCase(“BLOCKED”)
26. Add operation to the List of operations that are waiting to be executed.
27. Else If(transactionState.equalsIgnoreCase(“ABORTED”)
28. //Print that the transaction is ignored.
29. System.out.println(“Operation Ignored”);
30. //else perform Lock Upgrade.
31. If (dataItem.contains(“LOCK”) && dataItem.contains(“READLOCK”) && NoOfLocks == 1))
32. Set Lock Status “Write Locked”.
33. Else if (dataItem == “NOTLOCKED”)
34. Set LockStatus “WriteLocked”.
35. Append this to the existing list of LockList.
36. Else Conflicting Read-Write Locks.
37. Call to the Function (dataItem, TID)
38. Else If (readLine.charAt(0) == ‘e’)
39. Set Transaction Status as “Committed”.
40. //Need to release the locks on all the items held by the transaction.
41. Call function Free Transaction (TID)
42. Else
43. System.out.Println(“Invalid Operation”);
44. //Defining the Free\_Transaction (TID)
45. For (All dataItems locked by TID)
46. dataItem.unlock(TID);
47. If (Transactions in waitlist)
48. Dequeue the transaction from the list.
49. Lock the dataItem.
50. //Defining the function check\_DeadLock.
51. Retrieve transactions that have lock status == “locked”
52. //Checking wait-die deadlock
53. If(Timestamp (this Transaction) < Timestamp(T))
54. //Older transactions get to wait until the younger ones complete.
55. Append the transaction to the list of waiting transactions.
56. Set status of Transaction == “BLOCKED”
57. ELSE
58. //Younger Transaction dies.
59. Set status of Transaction == “ABORTED”
60. Call to the function “Free\_Transaction (TID).

**Pseudo – Code Description:**

* Firstly, we have 3 lists namely TransList, LockList and ActiveTrans.
* TransList is a HashMap that is used to save the transaction timestamps.
* LockList is also a HashMap that is used to save the locked items list.
* ActiveTrans is an Array List that contains the transactions that are in the waiting list.
* Also, we have few transaction statuses which are self-explanatory like “Blocked”, “Active”, “NotBlocked”, “Aborted”, Committed”, “ReadLocked”, “WriteLocked” which are used to explain the status of the data Item.
* We also have 2 fucntions namely, Check\_DeadLock and Free\_Transaction.
* Check Deadlock takes a Transaction ID which have the lock status as “Locked” and then perform the Wait Die algorithm on them to kill the younger transactions by setting their status as “Aborted” and allowing the older transaction to be in the waiting queue by setting their status as “Blocked” and adding them to the LockList.
* Next, we have the FreeTransaction function which takes the Transaction ID alone as the argument and then unlocks them one by one by running a for loop of all the wait list data Items.