**DB 2 Project Report**

By

Dhara Patel (ID : 1001053878)

Archana Mohan (ID: 1001091734)

The following is a brief overview of the implementation of our project.

For the TRANSACTION table, we make use of Hash Maps. Each entry in the table will contain:

|  |  |  |
| --- | --- | --- |
| Trans\_id | Trans\_timestamp | Trans\_state |
|  |  |  |
|  |  |  |
|  |  |  |

For the LOCK table, again we make use of Hash Maps.

|  |  |  |
| --- | --- | --- |
| Item\_id | Lock\_mode | Trans\_id |
| X | Read | T2 |
| Y | Write | T3 |
|  |  |  |

For the blocked transactions, we make use of the LINKED HASH MAP data structure because of it is FIFO property. All the transactions and Lock\_mode which wait on the locked item will be serially added to the Queue and will be executed in the same order.

**Language of Implementation:**

The primary language for our implementation is Java. We believe that Java is the ideal language since threading and deadlock handling can be done by means of Java methods.

We plan on making four modules- one for each operation BeginTransaction, Read Operation, Write Operation and EndTransaction.

When the input file is read for a particular sequence of transaction operations, the

**BeginTransaction:**

Information regarding new transaction will be entered in Transaction table. In this method, the main purpose will be to determine the Trans\_id of the transaction. It maintains a STACK which will hold the different transaction ids. Also, the timestamp of each transaction is assigned here. The Timestamp[] array will hold all the values.

**Read Operation**:

When the ‘r’ operation is encountered in the file, the code searches for the data item in the parenthesis of the operation and accordingly, if the item is *UNLOCKED,* the item is *READ-LOCKED*. We keep a counter here to keep track of all the transactions that have a read-lock on this item. If the item that one transaction wants to access is *WRITE-LOCKED* by another item, the timestamp of the current transaction is compared with the timestamp of the locking transaction and appropriate steps are taken(wait-die protocol). The lock table is also updated indicating the latest locks on the item. The pseudo-code of this is as follow:

**read\_lock(X)**

{

if LOCK(X) == “unlocked”

then **begin** LOCK(X) ← “read-locked”;

no\_of\_reads(X) ← 1

**end;**

else if LOCK(X) == “read-locked”

then no\_of\_reads(X) ← no\_of\_reads + 1;

else **begin**

if TS(Tnew) <TS(Told) //Tnew = transaction requesting lock and Told= old transaction that has an item locked

**begin**

block the transaction;

update the Transaction Table;

system wakes the waiting transaction;

go to read\_lock(X);

**end;**

else **begin**

unlock(X);

change the state in the Transaction Table

**end;**

**end;**

}

**Write Operation:**

For this operation, the code checks if the item under consideration is *UNLOCKED*. If it is, then it acquires a *WRITE-LOCK* the item. Else if the item is already locked by a conflicting read or write lock, then a wait-die protocol is enforced. The transaction is either blocked and it’s state is changed to block in the transaction table or the transaction is aborted if the deadlock prevention protocol determines that the conflicting transaction should be aborted.

The pseudo-code for the write function is given below:

**write\_lock(X)**

{

if LOCK(X) == “unlocked”

then LOCK(X) ← “write-locked”;

update the lock table;

else **begin**

if TS(Tnew) < TS(Told) //Tnew = transaction requesting lock and Told= old transaction that has an item locked

**begin**

block the transaction;

update the Transaction Table;

system wakes the waiting transaction;

go to write\_lock(X);

**end;**

else **begin**

unlock(X);

change the state in the Transaction Table

**end;**

**end;**

}

**EndTransaction**: The EndTransaction signals a successful end of the transaction. At this juncture, all the locks that are held by the transaction are released and the transaction table is updated to indicate that the transaction has committed.

**unlock(X)**

{

if LOCK(X) == “write-locked”

then **begin** LOCK(X) ← “unlocked”;

wake up one of the transactions waiting in the queue

**end;**

else if LOCK(X) == “read-locked”

then **begin**

no\_of\_reads(X) ← no\_of\_reads - 1;

if no\_of\_reads(X) = 0

then **begin** LOCK(X) = “unlocked”;

wake the next transaction that’s waiting

**end;**

**end;**

}

**Note:** We have modified the pseudo-code at program level in detail. Above is general logic of the given description.