7/25/2017

Project I

CSE 5331 Database Models and Implementation

Name: Virti Bipin Sanghavi ID: 1001504428

Name: Dhinesh Kumar Sivakumaran ID: 1001393555

**Files:**

Transaction.java – Transaction Object which holds Transaction Id, Timestamp, Transaction State, Items locked by Transaction object and Operations pending to be executed.

*It maintains the list of transactions and all their corresponding attributes(data items)*

LockObj.java – Lock Object which hold Resource Name, Lock Mode, Transaction Id which has Locked Resource, Transaction Id which are waiting to be executed.

*It maintains locking table, list of resources and their attributes locked by the various transactions.*

ConcurrencyControl.java(Main file) – Read Input file and preform corresponding actions along with wait-die deadlock mechanism. It also handles task of waking up waiting transaction when resource become free.

*It created the corresponding transaction and the lock objects and takes care of the concurrency control in terms of transaction management*.

**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 concurrency/ConcurrencyControl <<Input file path>>**

**Execution:**

**Input1.txt**

b1;

r1 (Y);

w1 (Y);

r1 (Z);

b2;

r2 (X);

w2 (X);

w1 (Z);

e1;

r2 (Y);

b3;

r3 (Z);

w3 (Z);

w2 (Y);

e2;

r3 (X);

w3 (X);

e3;

**Output1.txt**

b1; - Operation is Successful => Transaction 1 has started

r1 (Y); - Operation is successful => Y is locked by 1

w1 (Y); - Operation is Successful => Lock on Y by 1 upgraded to Write-Lock

r1 (Z); - Operation is successful => Z is locked by 1

b2; - Operation is Successful => Transaction 2 has started

r2 (X); - Operation is successful => X is locked by 2

w2 (X); - Operation is Successful => Lock on X by 2 upgraded to Write-Lock

w1 (Z); - Operation is Successful => Lock on Z by 1 upgraded to Write-Lock

e1; - Operation is Successful => Transaction 1 has Committed

r2 (Y); - Operation is successful => Y is locked by 2

b3; - Operation is Successful => Transaction 3 has started

r3 (Z); - Operation is successful => Z is locked by 3

w3 (Z); - Operation is Successful => Lock on Z by 3 upgraded to Write-Lock

w2 (Y); - Operation is Successful => Lock on Y by 2 upgraded to Write-Lock

e2; - Operation is Successful => Transaction 2 has Committed

r3 (X); - Operation is successful => X is locked by 3

w3 (X); - Operation is Successful => Lock on X by 3 upgraded to Write-Lock

e3; - Operation is Successful => Transaction 3 has Committed

**Input2.txt**

b1;

r1 (Y);

w1 (Y);

r1 (Z);

b2;

r2 (X);

w2 (X);

w1 (Z);

r2 (Y);

e1;

b3;

r3 (Z);

w3 (Z);

w2 (Y);

e2;

r3 (X);

w3 (X);

e3;

**Output2.txt**

b1; - Operation is Successful => Transaction 1 has started

r1 (Y); - Operation is successful => Y is locked by 1

w1 (Y); - Operation is Successful => Lock on Y by 1 upgraded to Write-Lock

r1 (Z); - Operation is successful => Z is locked by 1

b2; - Operation is Successful => Transaction 2 has started

r2 (X); - Operation is successful => X is locked by 2

w2 (X); - Operation is Successful => Lock on X by 2 upgraded to Write-Lock

w1 (Z); - Operation is Successful => Lock on Z by 1 upgraded to Write-Lock

r2 (Y); - Operation is Unsuccessful => Transaction 2 is Aborted

e1; - Operation is Successful => Transaction 1 has Committed

b3; - Operation is Successful => Transaction 3 has started

r3 (Z); - Operation is successful => Z is locked by 3

w3 (Z); - Operation is Successful => Lock on Z by 3 upgraded to Write-Lock

w2 (Y); - Operation is Ignored

e2; - Operation is Ignored

r3 (X); - Operation is successful => X is locked by 3

w3 (X); - Operation is Successful => Lock on X by 3 upgraded to Write-Lock

e3; - Operation is Successful => Transaction 3 has Committed

**Input3.txt**

b1;

r1 (Y);

w1 (Y);

r1 (Z);

b2;

r2 (X);

w2 (X);

w1 (Z);

r2 (Y);

b3;

r3 (Z);

e1;

w3 (Z);

w2 (Y);

e2;

r3 (X);

w3 (X);

e3;

**Output3.txt**

b1; - Operation is Successful => Transaction 1 has started

r1 (Y); - Operation is successful => Y is locked by 1

w1 (Y); - Operation is Successful => Lock on Y by 1 upgraded to Write-Lock

r1 (Z); - Operation is successful => Z is locked by 1

b2; - Operation is Successful => Transaction 2 has started

r2 (X); - Operation is successful => X is locked by 2

w2 (X); - Operation is Successful => Lock on X by 2 upgraded to Write-Lock

w1 (Z); - Operation is Successful => Lock on Z by 1 upgraded to Write-Lock

r2 (Y); - Operation is Unsuccessful => Transaction 2 is Aborted

b3; - Operation is Successful => Transaction 3 has started

r3 (Z); - Operation is Unsuccessful => Transaction 3 is Aborted

e1; - Operation is Successful => Transaction 1 has Committed

w3 (Z); - Operation is Ignored

w2 (Y); - Operation is Ignored

e2; - Operation is Ignored

r3 (X); - Operation is Ignored

w3 (X); - Operation is Ignored

e3; - Operation is Ignored

**Input4.txt**

b1;

r1 (Y);

w1 (Y);

r1 (Z);

b2;

r2 (Y);

b3;

r3 (Z);

w1 (Z);

w2 (Y);

r2 (X);

e1;

e3;

w2 (X);

e2;

**Output4.txt**

b1; - Operation is Successful => Transaction 1 has started

r1 (Y); - Operation is successful => Y is locked by 1

w1 (Y); - Operation is Successful => Lock on Y by 1 upgraded to Write-Lock

r1 (Z); - Operation is successful => Z is locked by 1

b2; - Operation is Successful => Transaction 2 has started

r2 (Y); - Operation is Unsuccessful => Transaction 2 is Aborted

b3; - Operation is Successful => Transaction 3 has started

r3 (Z); - Operation is successful => Z is locked by 3

w1 (Z); - Operation is Unsuccessful => Transaction 1 is Blocked

w2 (Y); - Operation is Ignored

r2 (X); - Operation is Ignored

e1; - Operation is Successful => Operation added to Operation list

e3; - Operation is Successful => Transaction 3 has Committed

Inside ActiveTransaction List

w1 (Z); - Operation is Successful => Lock on Z by 1 upgraded to Write-Lock

e1; - Operation is Successful => Transaction 1 has Committed

w2 (X); - Operation is Ignored

e2; - Operation is Ignored

**Input5.txt**

b1;

r1 (Y);

w1 (Y);

r1 (Z);

b2;

r2 (Y);

b3;

r3 (Z);

w1 (Z);

w2 (Y);

r2 (X);

e1;

w3 (Z);

e3;

w2 (X);

e2;

**Output5.txt**

b1; - Operation is Successful => Transaction 1 has started

r1 (Y); - Operation is successful => Y is locked by 1

w1 (Y); - Operation is Successful => Lock on Y by 1 upgraded to Write-Lock

r1 (Z); - Operation is successful => Z is locked by 1

b2; - Operation is Successful => Transaction 2 has started

r2 (Y); - Operation is Unsuccessful => Transaction 2 is Aborted

b3; - Operation is Successful => Transaction 3 has started

r3 (Z); - Operation is successful => Z is locked by 3

w1 (Z); - Operation is Unsuccessful => Transaction 1 is Blocked

w2 (Y); - Operation is Ignored

r2 (X); - Operation is Ignored

e1; - Operation is Successful => Operation added to Operation list

w3 (Z); - Operation is Unsuccessful => Transaction 3 is Aborted

Inside ActiveTransaction List

w1 (Z); - Operation is Successful => Lock on Z by 1 upgraded to Write-Lock

e1; - Operation is Successful => Transaction 1 has Committed

e3; - Operation is Ignored

w2 (X); - Operation is Ignored

e2; - Operation is Ignored

**Phase I**

**Data Structure:**

**Transaction Table:**

* Transaction Id <Integer>
* Transaction Timestamp <Integer>
* Transaction State <Enum>
* Items Locked <List>
* Operations List <List>

**Lock Table:**

* Item Name <String>
* Lock State <Enum>
* Locked By <List>
* Waiting List <List>

**Pseudo Code:**

line = Read Line of input file

if(line starts with "b")

create transaction record

set transaction state "active"

set transaction timestamp

else if(line starts with "r")

if(this transaction is blocked)

add operation to list of operation to be done by this transaction

else if(this transaction is aborted)

#Ignore, do nothing

else

if data item not in lock table, insert data item

if(data item not locked)

set lock status as "read-locked"

add this transaction to list of transaction holding the locks

else if(data item read-locked)

add this transaction to list of transaction holding the locks

elses

#write-locked

check\_deadlock(data item, this transaction Id)

else if(line starts with "w")

if(this transaction is blocked)

add operation to list of operation to be done by this transaction

else if(this transaction is aborted)

#Ignore, do nothing

else

#lock upgrade

if(data item is read-lock by this transaction AND Only this transaction has locked data item)

update lock status as "write-locked"

else if(data item not locked)

set lock status as "write-locked"

add this transaction to list of transaction holding the locks

else

#conflicting write-locks and read locks

check\_deadlock(data item, this transaction Id)

else if(line starts with "e")

update transaction state to be "committed"

#release locks on items been lock by this transaction (progressively)

unlock\_by\_transaction(this transaction id)

else

print "invalid operation"

Function unlock\_by\_transaction(Tid)

{

For(all data items locked by Tid)

{

data\_item.unlock(Tid)

If(another transaction in wait list)

Remove it from waitlist and aquire lock on data\_item

Start performing the operations from the operation list for this transaction

}

}

Function check\_deadlock(data\_item, Tid)

{

T = Get transaction that holds lock on data\_item

// For checking wait-die deadlock detection

if(timestamp(this transaction) < timestamp(T))

Append Tid into waitlist of data\_item

// older transaction waits for younger one in wait-die

Update Tid state as “blocked”

else

// since this transaction is younger then it is aborted

Update Tid state as “aborted”

unlock\_by\_transaction(Tid)

}

**Functions Description:**

//For Deadlock Detection using Wait-Die//

*function* wait\_die(this transaction):

T = Get transaction that holds lock on data\_item

Comparing the timestamps of the T and this transaction

// For checking wait-die deadlock detection

if(timestamp(this transaction) < timestamp(T))

Append Tid into waitlist of data\_item

// older transaction waits for younger one in wait-die

Update Tid state as “blocked”

else

// since this transaction is younger then it is aborted

Update Tid state as “aborted”

unlock\_by\_transaction(Tid)

*endfunction*

*function* abort(this transaction):

set the transaction status of this transaction to 'aborted' in the transaction table

unlock\_by\_transaction (this transaction)

*endfunction*

*function* commit(this transaction):

set the transaction status of this transaction to 'committed' in the transaction table

unlock\_by\_transaction (this transaction)

*endfunction*

*function* blocked(this operation, this transaction):

add this operation to the list of waiting operations in the transaction table for this transaction

update this transaction state “blocked”

*endfunction*

**Various Transaction Actions mentioned in the pseudo-code:**

1. **Begin (b):**

When you find a begin operation (For eg. b1;) in the input file, then the begin() is called which creates a new record with the respective transaction Id where the transaction status is set as ‘active’.

1. **Read (r) :**

When a read operation (For eg. r2(Y);) is found in the input file, then the read() is called. The readlock() is processed for the requested item and when the lock is set, the item can be accessed by the transaction requesting it.

readlock():

Parameters: data item, this transaction

Here, it checks the lock table for checking the presence of the data item, if the item does not exist then a new record will be inserted with that data item in the Lock table and the lock status for this item will be set as RL (Read Lock). If the item is already present in the lock table, then appropriate actions are taken either to abort this transaction or to block this transaction. We have used the wait-die deadlock prevention protocol by using the time stamps of the respective transaction.

3. **Write (w):**

A write operation (For eg. w2(Y);) is found in the input file, then the write() is called. Then we process writelock() for the requested item and when the lock is set successfully, the item is accessed by the transaction.

writelock():

parameters to function: data item, this transaction

Here, it checks the lock table for the checking the data item, if the item does not exist then a new record will be inserted with that data item in the Lock table and the lock status for this item will be set as WL (Write Lock). If the item is already in the lock table, then appropriate actions are taken either to abort this transaction or to block this transaction. If this transaction already hold Read Lock and Only transaction holding that lock, then upgrade lock from RL to Write Lock. We have used the wait-die deadlock prevention protocol by using the time stamps of the respective transaction.

4. **End (e):**

When you come across an end operation (For eg. e2)in the input file, then the commit() will be executed. The status of the transaction in the transaction table is set to 'committed'. By executing this function, we release all the locks corresponding to that transaction and transactions that have been waiting to access the items locked by this transaction are resumed from where it left.

5. **Abort:**

Here, if an abort() is called in any of the functions, then the transaction has to be aborted, that is the status of the transaction in the transaction table will be set to 'aborted' and releasing all the locks corresponding to that transaction. Also, all the operations encountered after the transaction is aborted will be ignored.

6. **Blocked:**

When you call the blocked() in any of the functions, then the recent operation which is read from the input file is added to the list of the waiting operations in the transaction table. These operations will be executed once the transaction becomes active.