**Database Models and Implementation**

**Project 1**

**Complete Project Report**

**Submitted by,**

1. **Kaushik Pulivendula Balasubramanyam ( 1001529713)**
2. **Subrahmanya Basavapatna Nagaraja Rao (1001540208)**
3. **PseudoCodes of different methods**

**Main Method :-**

Initialize transaction count to 0

// in main method

//This is the entry point of the program. It reads input file line by line and call respective method for further processing

Printstream out == new Printstream(new FileoutputStream(split the output into a output file))

//Logs all system.out into a separate output file. The output gets displayed on seperate file rather than command prompt

Assign Filereader( ) function to Filereader object

Declare Buffer reader( ) method with Filereader argument

Declare a String object line

while (input is not null)

Ask for input and print line to console line by line

Call OperationKey() method with line as argument.

switch(operation)

case b: createTransaction(line)

**Create a transaction:-**

createTransaction(String Inputline)

{

try {

create tokens

create a new transaction object

add the transaction object to the transaction hashmap table using the tokens created

**}**

catch (exception){

Rise an exception using packetStackTrace

}

**Write Operation**

writeOperation(line)

try{

create tokens

Use Fetch current transaction from table

if Transaction is blocked

Assign current blocked transaction value to a blocked operations queue

else if transaction is already aborted

{

Output no action taken

}

else

{

if(data item is not present in lock table)

Declare a new Arraylist() method and assign it to a list type

Add to the list

Declare new lock() method with create tokens, lock state, transactionholdinglock as arguments

Declare lockHashmap.put() function with current token, lockCurrent arguments

Assign locked transactions list to itemLockedlist of lock type

Update itemLockedList

}

// if item is already present in lock table, this condition is used

else

{

Fetch item from lock(hash) table, assign it to lock type object

if(item is present in lock(hash) table)

{

call Lockcurrent.setLockState() function

Fetch Transid holding lock

if(transaction already exist)

Then do not add

else

Update lock list

}

else if(data item is unlocked)

{

//Writelock is given to transaction and updated accordingly

Call lockCurrent.getTransholdingLock() function, assign it to list object

if( check whether transaction already exists do not exist)

else

add to locktable

}

elseif( condition contains “die”)

{

call setTransactionState with ABORTED argument

release() function is called with Current transaction as argument, assign it to TransactionCurrent

call transactionCurrent.getBlockedOperation() function and assign it to priority queue

while( priority queue is not empty)

{

fetch next Transaction

if(read)

{

call readOperation(eachline)

executeBlockedTransactions(eachline)

}

else if(end)

{

call commitOperation(eachline)

}

show Locktable by calling showLocktable(Current transaction)

}

// in case of exception

catch (Exception ex)

{

Rise an exception using printStackTrace()

}

**Read Operation**

try{

create tokens

Get current transactions from lock table

if(transaction is blocked)

add to blocked priority queue

}

else if(Transaction is already aborted)

{

take no action

}

else {

if(data item is not present in lock table)

{

// item along with its initial value is entered into locktable below

create new ArrayList(), assign it to Locktable

Fetch Locktable token from Data item

Create lockCurrent object of lock type, call new lock with tokens,lock state and transaction id as arguments

Call LockHashmap.put() function with tokens,current lock as arguments

if(locked list does not have current transaction token)

{

Add it to locked list

}

show locked list by using showLockTable () with items Locked List as argument

else if(item is already present in lock table when condition is entered)

{

Call LockHashmap.get(tokens), assign it to lock current of lock type

if(data item has read lock)

Current transaction also gets read lock

}

else if( Transaction has write lock)

{

//Wait-die condition is checked

checkWaitDie() function is called with timestamp, transaction holding lock as arguments and assigned to a string

if( Condition has wait)

{

set transaction state as blocked

Update BlockTransaction object, assign current transaction to it by calling hashmap table

Update priority Queue by removing Current transaction

Update Locktable

}

else if(condition contains die)

{  
Abort current transaction

release() current transaction

Fetch blocked transaction into priority queue

while(temporary variable is not empty)

{

create tokens

Take current active transaction from transaction Hashmap

if( eachline contain read)

call readOperation(eachline)

execute blocked transactions(eachline)

}

else if( eachline contains write)

{

call writeOperation(eachline)

execute blocked transactions(eachline)

}

else if( eachline contains e)

{

call commitOperation(eachline)

}

else if(current item is unlocked)

{

Update both tables

}

in case of exception

catch (Exception ex)

{

call printStackTrace()

}

**Commit Operation :-**

commitOperation(String InputLine)

{

try

{

create tokens

get current transaction object from transaction table using tokens created

if(transactionstate of current transaction is blocked)

{

get the blocked operation using getBlockedOperation method implemented by PriorityQueue

set the blocked operation on temporary variable

}

if(transaction is aborted)

{

print that transaction is already aborted

}

else

{

set transaction state to committed as the transaction is neither blocked or aborted

Release all locks of the current transactions using release method

By creating a temporary variable,get all the operations that were blocked, to execute

while(temporary variable is not empty)

{

remove each line in the temporary variable string

create tokens

Create a transaction object from the transaction hash map table using the created tokens

set the created transaction to ACTIVE

get the transaction ID

if(line in transaction contains “r”)

{

perform read operation

execute that blocked operation

}

else

if(line in transaction contains “w”)

{

perform write operation

execute that blocked operation

}

else if ( line in transaction contains “e”)

perform commit operation

print that transaction has been committed

show the current transaction table

}

catch (Exception)

{

Rise an exception using printStackTrace

}

**Returning Lock state based on input** :-

LockState getLockStateMain(String LockType)

{

try{

if(LockType is equal to “r”)

return RL i.e. ReadLock as lockState

else if (LockType is equal to “w”)

return WL i.e. WriteLock as lockState

else

return UL i.e. User defined Lock as lockState

}

catch(exception)

{

Rise an exception using printStackTrace

}

**Method to check for wait and die condition of transaction :-**

String checkWaitDie(int timestampOfcurrentTransaction, list holdingTransactionID)

{initialize a variable called toReturn to null

try

{

if(size of holdingtransactionID is greater than 1)

{

create an iterator object on holdingtransactionID

while(iterator object is not empty)

{

get next transaction

if(timestamp of current transaction is greater)

abort transaction i.e. die

else if(timestamp of current transaction is lesser)  
 transaction will be waiting

}

}

else

{

Create a new transaction object

if(timestamp of current transaction is greater)

abort transaction i.e. die

else if(timestamp of current transaction is lesser)  
 transaction will be waiting

}

}

catch(Exception)  
 Rise an exception using printStackTrace

return the toReturn variable

}

**Releasing or Unlocking all items locked by a transaction**

Transaction release( Transaction current Transaction)

{

Create a list of items locked

Create an Iterator Object on list of items locked

while(itemlist is not empty)

{

Create a new lock object

if( lock of current item is a write lock)

unlock that lock on item using loops

else if( lock of current item is a read lock and if item list is >1)

Remove that lock on item using loops

else if(current item is unlocked)

remove index of the iterator object

set transaction holding lock on current item

put it back into the table

}

Set items locked

return the current transaction

}

**Returning the type of Operation based on the input :-**

String getOperationKey(String InputLine)

{

try

{

if (line contains “b” )

return b i.e. begins transaction

else if (line contains “r”)

return r i.e. read operation

else if (line contains “w”)

return w i.e. write operation

else return e i.e. end of operation

}

catch (Exception) {

Rise an exception using printStacktrace

}

**Printing the Transaction table :-**

showTransactionTable(Transaction currentTransaction)

{

Print transaction table using System.out.print and getTransactionId method

Print the timestamp of transaction using getTransactionTimeStamp method

for(initialize i; i < size of getItemsLocked method of current transaction; i++)

Print items locked using get method in getItemsLocked in current transaction

**}**

**Printing the Lock table :-**

showLockTable(Lock currentLock)

{

Prints the dataitem using getItemname of currentlock object

Prints LockState of the current item using getLockState method

for(initialize i; i < size of getTransIdHoldingLock method of current lock; i++)

Prints the Transaction ID that is holding the current lock using get method

}

**Transaction State Class**

public enum TransactionState {

//Enum data structure to store possible types of states that a transaction can be in.

Declare 4 states Active, Blocked, aborted , committed

}

**Lock State Class**

public enum LockState

{

//Enum data structure to store possible types of locks that can be imposed on a data item

Declare Readlocked, Write locked, Unlocked

}

**Lock Class**

public class Lock

{

public class lock( Pass itemName, lockState, transIdHoldingLock as arguments)

initialize arguments

}

use selectors and mutators i.e get and set methods for arguments

create a new Array list and assign it to transIdHoldingLock

main method

transIdHoldingLock.indexof(1) and assign it to index of integer type

**Transaction Class**

public class Transaction

{

Declare transactionId, transaction state, items locked, Transaction timestamp

Create new Priority Queue and assign it to blocked list

public Transaction(Pass transaction State, tranaction id and Transaction timestamp as arguments)

initialize the arguments

}

Use selectors and mutators, get and set methods for arguments

Use get and set method for items locked and blocked Operations

**B. Data structures** used :-

1. **Lock Table** :- The data structure used to implement the Lock Table is **enum** data type. It stores **RL(Read Lock)**, **Write Lock(WL)** and **Unlock(UL)** as elements.
2. **Transaction Table** :- Same as in the Lock Table, **enum** data type is used to implement the Transaction table. The enum data type stores **ACTIVE, BLOCKED, ABORTED** and **COMMITTED** as **state elements.**
3. **Data Item name :-** A **String** data type is used to store **names** of **data items** , on which operations like **read**, **write** are performed.
4. **Transaction ID** :- **Integer** datatype is used to **assign ID numbers** to transactions.
5. **Transaction Timestamp** :- Like Transaction ID’s, **Integer** datatype is used to store the **timestamps** of Transactions. Timestamp values are assigned in the order in which the transactions are **submitted to the system**. Hence, Timestamping can be thought of as a **transaction start time**.
6. **Transaction holding Lock :-** Both a **List** and **ArrayList** data structures are used to store the **ID’s** of t**ransactions** which are currently **holding locks**.
7. **Transaction that is waiting :-** Here, a **Priority Queue** is used to store the **ID** of the transaction that is waiting to **lock** a certain data item.
8. **Items Locked :- List** datatype is used to store a set of **data items** that are **locked** by specific transactions.
9. **Blocked operation :-** A **Priority Queue** is used to store a number of **operations** which are **blocked** by the system.