**SAAHIL PRALHAD ANANDE**

**1001855308**

Project language: python

**Transaction table columns**:

using dataframe data structure to store this table

|  |  |  |  |
| --- | --- | --- | --- |
| **transid** | **tstamp** | **status** | **itemslocked** |
|  |  |  |  |

**transid** --stores unique transcation id given for sperate transcation

**tstamp** -- saves sperate time for each operation performed by the transacation

**status** -- keeps taps if the transaction is unblocked, blocked or commited that it could wait for procceed

**itemslocked** -- saves the data item which is accessed and type of lock (read or write) the transcation is locking

**Lock table columns**:

using dataframe data structure to store this table

|  |  |  |
| --- | --- | --- |
| **item** | **state** | **lockingtransaction** |
|  |  |  |

**item** -- saves the data item which is locked

**state** -- save what type of lock transcation is locking

**lockingtransaction** -- saves the transcation id of seperate transcation

**Blocking table**:

using dataframe data structure to store this table

|  |  |  |  |
| --- | --- | --- | --- |
| **transdid** | **item** | **operation** | **lockingtransaction** |
|  |  |  |  |

**transid** --stores unique transcation id given for sperate transcation

**item** -- saves the data item which is blocked

**operation –** operation performed on that data item

**lockingtransaction** -- saves the transcation id of seperate transcation

Code Structure:

It uses 3 dataframes for transaction table , lock table, blocktable

1. Main function extracts each operation from the txt file, including the transaction id and data item if applicable, then calls respective begin(..), read(..), write(..), commit(..) and abort functions.
2. read and write functions checks transaction state and time stamp to grant read/write lock on given data item
3. There are two functions which are used in one of the program code for deadlock prevention wait(..) for wait and die and cautious(..) for cautious wait
4. Times stamp of requesting transaction is compared with each and every transaction's time stamp separately which are holding the lock; if one of them is older than requesting transaction which is requesting lock then, requesting transaction is aborted.
5. read(..) and write(..) functions internally calls wait(..) and abort(..) functions.

5. There are three types of results which are returned from functions: blocked or aborted or commited

6. In case one of transaction is aborted/committed then it checks for unlocking any other blocked transactions and executes its previous commands which were ignored because it was blocked.

**pseudo code**:

**# read the inputfile and put in list**

inputfile = file.open(inputfile)

for lines in inputfile

infile.append(lines)

file.close

**#iterate each items in file list and search for operations**

main(infile)

for items in infile

if items == 'b'

**#begin the transaction**

transid = findall(r'\d+', items)

tstamp = tstamp + 1

begintransaction(transid, tstamp)

infile.pop(0)

if items == 'r'

**#begin read transaction**

transid = findall(r'\d+', items)

tstamp = tstamp +1

dataitem = get data item from items

if status == 'active':

read(transid,dataitem)

if status == 'aborted':

#the trans is aborted

Print (transcation is aborted)

if status == 'blocked':

queue(transid,dataitem,operation)

if items == 'w'

**#begin write transaction**

transid = findall(r'\d+', items)

tstamp = tstamp +1

dataitem = get data item from items

status = getstatus(transid)

if status == 'active':

write(transid,dataitem)

if status == 'aborted':

#the trans is aborted

Print (transcation is aborted)

if status == 'blocked':

queue(transid,dataitem,operation)

if items == 'e'

**#end transcation**

transid = findall(r'\d+', items)

if status == 'blocked':  
 executeque(transid,operation)  
 if status == 'aborted':  
 #the trans is aborted

Print (transcation is aborted)

if status == 'commited':

#the trans is commited

Print (transcation is commited)

else:  
 commit(transid)

**#begin method**

begintransaction(id,time):

**#add a row in transcation table**

transtable = transtable.append(pd.Series([id, time, 'active', 'none'],

**#read method**

read(id,time,data):

if id in transtable: **#checks if there is any transcation in transtable with this id**

if status == 'blocked'

return 0

else:

if data in locktable: **# when data item is locked in locktable**

if lockstatus == 'rl'

**#add new row to lock table since the lock is shared**

locktable = locktable.append(pd.Series([data, 'rl', id]

if lockstatus == 'wl'

**#since the lock is shared we perform cautious wait or waitdie**

wait(id,data,'r')

Else: **#no data item detected in locktable**

**#add a new row in locktable**

locktable = locktable.append(pd.Series([data, 'rl', id])

**#wait method**

wait(id,data):

**#get time stamps**

**#compare the time stamps if the older transcation requesting lock the current transcation abort itself**

if currentts > holdingts

abort(id)

**#if the current transcation is requesting lock it is allow to wait**

else:

#update blocktable and transtable

transtable.loc[transtable['transid'] == id, ['status']] = 'blocked'

blocktable = blocktable.append(pd.Series([id, data, operation, x]

**#commit method**

commit(id):

**#update transtable state value to committed using transid**

row = get by id in transtable

transtable.row.update(state = 'commited')

**#release lock hold by transcation with using id**

for rows in locktable:

**# checks if there exsist any row which are holding locks**

if rows.lockingtransaction == id:

**#delete that row, release the locks**

locktable.pop(rows)

**#abort method**

abort(id):

**#update transtable state value to aborted using transid**

row = get by id in transtable

transtable.row.update(state = 'abort')

**#release lock hold by transcation with using id**

for rows in locktable:

**# checks if there exsist any row which are holding locks**

if rows.lockingtransaction == id:

**#delete row, release the locks**

locktable.pop(rows)

**#write method**

write(id,data):

if id in transtable:

if status == 'blocked'

**#if the data item is wl locked by othertrans perform wait die or cautious wait**

wait(id,data,'w')

if lockstatus == 'rl'

**#upgrade from read log to write log**

locktable.loc[locktable['item'] == data, ['state']] = 'wl'

Else: **#no data item detected in locktable**

**#add a new row in locktable**

locktable = locktable.append(pd.Series([data, 'wl', id]

**References**:

https://www.geeksforgeeks.org/introduction-to-tstamp-and-deadlock-prevention-schemes-in-dbms/

https://www.exploredatabase.com/2014/04/deadlock-prevention-algorithms-in.html