DB2 PROJECT 1 DESIGN DOCUMENT

Team Members for Group 1:

Name	**ID**
Elsy Fernandes	**1001602253**
Maria Lancy Devadoss	**1001639262**

Project:

In this project, we implement a program that simulates the behavior of the two-phase locking (2PL) protocol for concurrency control. The protocol to be implemented will be rigorous 2PL, with the wound-wait method for dealing with deadlock.

Programming Language: Python

Data Structures Used: Array List, Dictionary

We are using two tables, Transaction table and Lock table, to store the transaction details and the lock information of data item in any transactions.

Table Structure

Transaction table:

The Transaction table contains the following six fields:

tranId – This field contains transaction Id of the transactions (T1, T2…).

Timestamp - This field contains timestamps of each transaction (1,2,3…). The time when transaction begin executing.

state- This field contains information about state of the transaction (active, waiting, aborted, committed)

Itemslockedbytransaction- This list contains information about the list of data Items locked by each transaction (X, Y,Z)

transactionBlocked- This list contains information about the operations which are waiting to be executed once the transaction state changes from 'waiting' to 'active'

changeState :- This method to change the current state of transaction from Active to 'Waiting' or 'Abort' based on wound-wait method.

‘lockby’:- This attribute tells if the transaction is in waiting state , then which attribute the transaction is waiting for.

‘lockedop’:-This list tells if the transaction is in waiting state , then list of all the waiting operations will be displayed.

<u>Lock table:</u>

The Lock table contains following four fields:

dataItem– This field contains information about item name each transaction request (X, Y, Z).

Lock - This field contains lock state of the data item (ReadLock, writeLock)

tranId - This field contains list of transaction ids for the transactions which hold the lock for the data item (T1, T2...).

TransactionHoldingReadLock- This list contains Transaction Ids of all the data items which are Read Locked

TransactionHoldingWriteLock- This list contains Transaction Ids of all the data items which are write Locked (basically it should have only one Transaction still we maintain a list)

waitingTransactions- This field contains list of transaction ids for transactions which are waiting for the data item to be unlocked (Operations waiting to be executed)

Description

1.Project is implemented using Python

2.The program consists the following function:

- Main
- BeginDatabaseTransaction
- Read
- Write
- checkreadState
- checkwriteState
- Woundwait
- checkDuplicateTransaction
- Abort
- startblockTrans
- printable
- writetofile
- End

Main Function:

When Python program starts it does the following operations:

- Place input files in Python root directory
- Enter the file name which you want to process
- Read file line by line (Here line means Operation to be executed)
- We check the state of the transaction Id in transaction table using checkstate functions
- We check two checkstates functions 1) checkreadState for read operations 2)checkwriteState for write operations
- If transaction is ‘Abort’ Ignore the operation if its ‘waiting’ we add the operation to the waiting list, if its ‘Active’ we perform the read/write operations.

```
If (line.find( ‘b’) !=-1):
          Call beginDatabaseTransaction(line,writefile)
if (line.find( ‘r’) !=-1):
          Call checkreadState(line,writefile)
if (line.find( ‘w’) !=-1):
          Call checkwritestate(line,writefile)
if (line.find( ‘e’) !=-1):
          Call endtransaction (Tid)
```

BeginDatabaseTransaction function:

Insert TransactionId,Timestamp,Transaction State(‘Active’) in transaction .Since the transaction is in active state lockby and lockedop should be initialized to None. Timestamp gets incremented as new transaction begins

```
def beginDatabaseTransaction(line,writefile):
          for digit in line:
             if digit.isdigit()
                digit=k.split()
          counter=len(counter)+1
          timestamp=counter
```

```
transactionTable.append(digit,Timestamp,'Active')
counter.append(1)
transactiontable=prettytable([tranId,Timestamp,State])
for i in transactionTable
        transactiontable.add_row([i.tranId,i.Timestamp,i.State])
 writefile.wrilelines(str(transactiontable))
```

Note: Output will be printed to the corresponding ouput.txt file

Read Function

1.Here we get the Tid, dataItem from the passed input line.

2.Once the transaction state is active, we check if lock table has any entries for that data item. If it doesn't, we go ahead and create entry for data item along with lock type and Tid to lock table.

3.If lock table is not empty, we check if requested data item is existing in lock table. If it does, we check whether lock state is read lock or write lock. If read lock, we go ahead and add transaction to data item along with other transaction in read list. If write locked, we call wound-wait function

4. If lock table is not empty and requested data item is not in lock table then we go ahead and create entry for data item along with lock type and Tid to lock table.

```
Def readTransaction (line,writefile):

        Get the Tid, dataItem from the passed input line

            If (len(lock_table)==0):

                    1.Read lock the data Item and add entry to lock table with
                    dataItem,Lock_state,Tid

                    2. Add the dataItem to the transaction Table Locked Items List

            Elif (len(lock_table)!=0):

                    Iterate through the lock table,checks if data Item in lock table is equal to
                    requesting data Item.

                    If(lockObject[i].dataItem==dataItem):

                            If(lockObject[i].lock=='Read Lock'):

                                    1.Add the Tid to the lock table transactionHoldingReadLock
                                    Function which contains all the Tid which holds read lock
                    2.It adds the dataItem to the transaction Table Locked Items List

                Elif(lockObject[i].lock==’Write Lock’):

                    1.Call woundWait(lock_holding_Tid,lock_requesting_Tid)

            Elif(lockObject[i].dataItem!=dataItem):

                1.Read lock the data Item and add entry to lock table with dataItem,Lock_state,Tid

                2. It adds the dataItem to the transaction Table Locked Items List
```

Write Function

1.Here we get the Tid, dataItem from the passed input line.

2.Once the transaction state is active, we check if lock table has any entries for that data item. If it doesn’t, we go ahead and create entry for data item along with lock type and Tid to lock table.

3.If lock table is not empty, we check if requested data item is existing in lock table. If it does we check whether lock state is read lock or write lock. If read lock we go ahead and check if transaction holding read lock is same as one requesting write lock. If so we update lock state of data item. If write locked we call wound wait function

4. If lock table is not empty and requested data item is not in lock table then we go ahead and create entry for data item along with lock type and Tid to lock table.

```
Def writeTransaction (line,writefile):

    Get the Tid, dataItem from the passed input line

        If (len(lock_table)==0):

            1.Write lock the data Item and add entry to lock table with dataItem,Lock_state,Tid

            2.It adds the dataItem to the transaction Table Locked Items List

        Elif (len(lock_table)!=0):

          Iterate through the lock table,checks if data Item in lock table is equal to requesting data Item.

            If(lockObject[i].dataItem==dataItem):
```

```
            If(lockObject[i].lock==’Read Lock’):
                If(len(lockObject[i].transactionIdHoldingReadock)==1and
        lockObject[i].transactionIdHoldingReadock[0]==digit):
                lockObject[i].lock=’Write Lock’
            Elif(lockObject[i].lock==’Write Lock’):
                1.Call woundWait(lock_holding_Tid,lock_requesting_Tid)
        Elif(lockObject[i].dataItem!=dataItem):
            1.Write lock the data Item and add entry to lock table with
        dataItem,Lock_state,Tid
            2. It adds the dataItem to the transaction Table Locked Items List
```

checkreadState Function: -

1.Here we get the Tid from the passed input line.

2.Here we check the transaction is active/aborted/waiting. If transaction is Active, we proceed with read operation. If transaction is waiting, we add the transaction to the waiting list. If the transaction is aborted, then we ignore the operation.

```
Def checkreadstate(line,writefile)
    1. Get the digit from the input line
    If t.traId==digit:
      t.State=’Abort’:
         1. Ignore the operation
      t.State=’Waiting’
         1. Add transaction to waiting queue
      t.State=’Active’
         1. Call read(j,writefile) function
```

checkwriteState Function: -

1.Here we get the Tid from the passed input line.

2.Here we check the transaction is active/aborted/waiting. If transaction is Active, we proceed with read operation. If transaction is waiting, we add the transaction to the waiting list. If the transaction is aborted, then we ignore the operation.

```
Def checkwritestate(line,writefile)

    1.  Get the digit from the input line

    If t.traId==digit:

       t.State='Abort':

          1. Ignore the operation

       t.State='Waiting'

          1. Add transaction to waiting queue

       t.State='Active'

          1. Call write(j,writefile) function
```

WoundWait Function:

From the passed inputs we get the transaction information for the particular Tids.

```
Def woundwait(lockHoldingTid,requestingTid,line):

        Get transaction information from transaction table for Tid

        If(requestingTid_timestamp)< lockHoldingTid_timestamp):

                Update state ='abort' for lockHoldingTid

                Release all locks held by lockHoldingTid

                Ignore all upcoming operations of the lockHoldingTid

                requestingTid gets access to the data item

                Update the lock table and transaction table
        Else:
                Update state ='waiting' for requestingTid
                transactionBlocked.append(line) (note:Add operations of requestingTid to the waiting)
                Update Lock table and transaction table
```

checkDuplicateTransaction Function:-

def checkDuplicateTransaction(transaction):

1. Checks if the transaction is duplicated in transactionwaiting

Abort:-

def Abort(holding):

1. This function unlocks all the data items held by holding transaction
2. It removes the entry in the lock table for the holding transaction

startblockTrans Function:-

```
def startblockTrans(holding):
    if t.State='Abort'
        1. Remove the transaction from transaction waiting
    Else:
      If i.lockby==holding and i.state='waiting':
          S=0;
          While S<len(i.lockedop) and i.lockby==holding:
              1. Performs the read or write operations based on the operation in waiting queue
              2. Changes i.state==Active
              3. i.lockby==None
```

Printtable Function:

def printatble(transactionobjects):

1. Function is for printing the transaction table

writeToFile Function: -

def writeToFile(file,message):-

1. This function is to print the output to output files

End Function :

This Function is called when the end transaction from the input files executes.

```
Def end(i,writefile):
        Get the transaction id from input file

        If t.state !='Abort' and t.state!='waiting'

                Transaction commits

        Elif t.state==Abort:

                Transaction will be ignored since its already aborted

        Elif t.state=Waiting:

                Transaction is in waiting cannot commit
```

Sample Input: -

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;

Expected Output: -

```
b1;         - Begin Transaction T1
r1 (Y);     - DataItem Y is Read Locked by Transaction T1
w1 (Y);     - Upgraded Read Lock to Write Lock on DataItem Y by Transaction T1
r1 (Z);     - DataItem Z is Read Locked by Transaction T1
b2;         - Begin Transaction T2
r2 (X);     - DataItem X is Read Locked by Transaction T2
w2 (X);     - Upgraded Read Lock to Write Lock on DataItem X by Transaction T2
w1 (Z);     - Upgraded Read Lock to Write Lock on DataItem Z by Transaction T1
e1;         - End of Transaction T1. Unlock all the locks held by T1. (on Y and Z)
```

r2 (Y);	- DataItem Y is Read Locked by Transaction T2
b3;	- Begin Transaction T3
r3 (Z);	- DataItem Z is Read Locked by Transaction T3
w3 (Z);	-Upgraded Read Lock to Write Lock on DataItem Z by Transaction T3
w2 (Y);	- Upgraded Read Lock to Write Lock on DataItem Y by Transaction T2
e2;	- End of Transaction T2. Unlock all the Locks held by T2. (on Y, X)
r3 (X);	- DataItem X is Read Locked by Transaction T3
w3 (X);	- Upgraded Read Lock to Write Lock on DataItem X by Transaction T3
e3;	- End of Transaction T3. Unlock all the locks held by T3. (on X and Z)