Ishita Gangal (ig1043) and Divya Juneja (dj1322)
Advanced Database design document
Final Project

In this project several classes and enums have been used. Code starts with running main file StartRepCRec.py. It contains following components:

1. **StartRepCRec:**
   Main Function that initializes siteManager, Locktable, Transaction Manager. Calls parser in ParseCommands(...) to start reading the test file. Output is written out to file if specified with --outFile else prints to stdout

2. **Command (Class):**
   Reads the commands given in the test files
   Stores:
   - type of transaction: begin, end, read, beingRO, write, dump, fail, recover
   - transaction parameters: everything within (...)

   Methods:
   - get_transactionType(): Getter for transaction type which returns the type of transaction if it is begin, beingRO, read, write, end, dump, fail, recover.
   - get_transactionParameters(): Getter for transaction parameters

3. **ParseCommands (Class):**
   Responsible for reading each line in the file of operations. Processes each line and decides whether to call Site Manager or Transaction Manager. Site Manager is called for fail, recover, dump transaction types. Transaction Manager is called for begin, beginRO, read, write, end.
   Methods:
   - getLine(): Gets next line in file
   - getNextCommand(): Returns next command in the line
   - processCommand(): Removes comments and takes line as input. Returns commands list.
   - parseFileCommands(): Calls Site Manager functions or Transaction Manager functions based on type of transaction in command.

4. **DataManager (Class):**
   Every Site has its own data manager. The DM manages variables stored on the site and its locktable
   Methods:
   - getVariable(): Returns variable from its dictionary of stored variables, takes name of variable as input and returns variable if present else None
   - getLockTable(): Getter for Site's lock table and returns lockTable
   - getAllVariables(): Get all variables on this site and returns variables dictionary
   - addVariable(): Adds variable to dictionary. Takes name of variable and variable object as input
   - siteHasVariable(): Checks if site has variable. Takes name of variable being searched and returns True if the variable is present else False
   - acquireLock(): Transaction tries to acquire a lock on a variable. If the transaction already had a lock and it was only of one type then a lock is granted. Otherwise:
     - if locktype is read and the variable isn't already write-locked, a lock is granted.
     - if locktype is write and the variable isn't already locked, a lock is granted.

     Takes transaction that wants a lock, type of lock requested, variable on which lock is requested as inputs. Returns True if lock was acquired else False.

- writeVariableForTransaction(): Writes out variables that transaction wrote at commit time. Takes transaction that is being committed, variable to be written, new value to be written as input. Returns True if transaction had the appropriate lock else False.
- releaseLock(): Release lock on variable. Takes lock to remove and variable that is locked and needs to be removed lock on as input

5. **Lock (Class):**
   Represents a lock and holds information of type of lock and Transaction that holds it. Takes type of the lock, R or W and transaction that holds this lock as inputs.
   Methods:
   - getTransaction(): Gets transaction holding this lock
   - setTransaction(): Sets transaction for this lock. Takes transaction to be set as input and returns error if invalid Transaction.
   - getType(): Returns type of lock
   - setType(): Sets the type of Lock

6. **LockTable (Class):**
   Lock Table maintained by Data Manager on each site. Holds the lock dictionary.
   Methods:
   - get_lockDict(): Gets lock dictionary
   - getNumOfLock(): Returns the number of locks on a variable in the lock dictionary and takes the variable we want to look up as input and the length of lock dict for this variable is returned
   - setLock(): Sets the lock for transaction on variable. Takes transaction requesting lock, lock type that is being requested, variable on which lock is requested as inputs. Returns True if the lock is set, else False.
   - checkReadLocked(): Checks if variable has a read lock on it. Takes variable to look up as input and returns True if it read locked, else False.
   - checkWriteLocked(): Checks if variable has a write lock on it. Takes variable to look up as input and returns True if it write locked, else False.
   - checkLocked(): Checks if variable has a lock on it. Takes variable to look up as input and returns True if it has lock(read or write), else False.
   - checkTransactionHasLock(): Checks if transaction already has a lock on the variable. Takes transaction to check if it has a lock on variable, variable to look up, type of lock if specified else None as inputs and returns True if curTransaction has lock else False.
   - releaseAllLocks(self, variable):Releases all locks on variable. Takes variable to be released as input.

7. **Graph (Class):**
   Initializes this Graph. A graph is represented as a dictionary mapping nodes to lists of neighbors.
   Methods:
   - addEdge(): Inserts the given nodes into the graph as a directional mapping of start -> end. If a lock is acquired by a transaction on a variable then the arrow goes from Variable -> Transaction. If a transaction wants a lock but wasn't able to acquire it then arrow goes from Transaction -> Variable. Takes start i.e., the initial vertex/node and end, the terminal vertex/node as inputs.
   - addVertices(): Adds variable and transactions as nodes to the graph

   - removeEdge(): Removes the given node from the graph, as well as any references to it among the neighbor lists of other nodes. Takes the node to remove from the graph as input and returns True if nodeToRemove was successfully removed, and False if it wasn't found.

- removeSpecificEdge(): Removes edge from transaction to variable on state change. Input transaction and variable to be removed are taken as inputs.
- getNeighbors(): Returns the list of neighbors for node or None if node does not exist. Takes node to find neighbors of as input.
- keys(): Returns a list of this Graph's keys
- isCyclic(): If there is a cycle, it adds all the transactions in the cycle to list: cycleTransactions. This list is then used to find youngest transaction that should be aborted. Returns False if there is no cycle else True
- isPartOfCycle(): Returns True if the given node is part of a cycle and False otherwise. Takes the node at which to start the search as input
- visitedTwice(): Returns True if the given node has been visited twice and False otherwise.

8. Variable (Class):

Variable represents each variable(data) stored on the sites.

Args:

- name: Variable name
- index: Variable index
- value: Initial value
- siteId: Index of site which holds this variable

Methods:

- isLocked(): Checks if this variable has a lock. Returns the type of lock if present else None.
- setLockType(): Sets the type of lock to variable. Takes type of lock Read or Write as input
- getListOfSites(): Returns list of sites that have this variable. Takes index of variable as input and returns the list of sites that have the variable.
- getCurrentSite(): Returns current siteID

9. LockType (Class), TransactionState (Class), SiteStatus(Class): Enums

10. Transaction (Class):

Transaction objects holds information of the transaction.

Stores:

- id - Transaction ID
- name - Transaction Name
- readyOnly - Boolean indicating whether transaction is ready only or not
- state - State of trasaction : running, waiting, blocked, committed, aborted
- variablestoBeCommitted - list of variables written by Transaction to be changed on sites.
- startTime - time wen transaction started, used as an indicator of age when choosing youngest transaction to abort

Methods:

- getTransactionState(): Get transaction state if running, waiting, blocked, committed, aborted.
- getTransactionStartTime(): Returns start time of transaction, used to indicate age.
- checkIfTransactionReadOnly(): Returns true if transaction is read only
- getTransactionName(): Returns transaction name
- getReadVariables(): Get all variables read by this transaction
- getVariablesToBeCommitted(): Returns dictionary of variables written by this transaction
- setTransactionState(): Set state of transaction. Takes state to be set to as input and returns value error if invalid state.
- clearAllUncommittedVariables(): Clears dict of variables to be committed when transaction is aborted/committed

11. **Site (Class):** Site instance.
    - status: If the site is up, down or is in recovery state
    - id: Index of the site
    - variables: list of variables it stores
    - SiteDataManager: data managaer instance for site
    - foundVariables : list of variables this site has written/can access after failure

    Methods:
    - getSiteID(): Returns Site ID
    - getStatus(): Returns Site status
    - setStatus(): Sets site status. Takes status to be set to as input.
    - setFail(self): Sets site status to fail
    - setRecover(): Set site status to recover
    - getDataManager(): Getter for data manager instance
    - acquireLockOnSite(): Tries to acquire a lock on the site. Takes transaction requesting lock, type of lock - read or write, variable on which lock is requested as inputs and returns True if acquired else False.
    - releaseLockOnSite(): Releases lock for variable. Takes lock object to be released and variable on which this lock is present as inputs.
    - writeVariable(): Writes variable onto site. Takes transaction doing the writing, variable to be written, new value of variable as inputs.
    - fail(): Called when site fails. Sets transaction state to aborted if it used this site
    - recover(): Called when site recovers. Sets status to recovering.
    - dump(): Called when dump() is called in site manager. Prints all values of all variables on this site
    - getVariablesOnThisSite(): Returns list of variables present on this site

12. **SiteManager (Class):** Site instance. Responsible for processing site management requests. Takes care of sites if they are in recovery, UP and DOWN state. Makes calls to sites and maintains variable values on each one of them.

    Methods:
    - mainSiteManager(): Main method of site manager, calls fail(), recover() for particular siteId. Takes command that is being executed/has been read as input
    - getLock(): Getter for a read or write lock on variable by a transaction. Returns if lock is acquired, no site is available,
    - getVariablesValues(): Gets value of a variable if index given, else return values of all variables
    - getAllLocksSet(): Gets all the locks that are stored in locktable. Returns locktable containing information of which variable has been acquired by which transaction and type of its lock.
    - releaseLocks(): Releases lock for variable. Takes lock object to be released and variable on which this lock is present as inputs.
    - fail(): Calls the site fail method to mark site as failed
    - recover(): Calls the site recover method to mark site as recovering
    - getUPsites(): Get a list of all the sites that are UP
    - writeCommitToSites(): Write the values to be committed to the sites that are UP. Takes transaction that needs the value to be written, variable on which value needs to be written and value that needs to be written on a variable as inputs

13. **TransactionManager (Class):** Responsible for processing transactions. Detects deadlock and resolves it. Makes calls to site manager to release locks when transaction aborts or is committed.

    Stores:
    - transactionDict : dictionary of transactions
    - locktable : copy of locktable

- siteManager: global site manager instance
- graph: directed graph that is used for cycle detection
- waitingTransactions: dictionary of dictionaries the operation it needs to do and is waiting to perform
- blockedQueue: dictionary of dictionaries that stores transaction blocked and the transaction its waiting for along with the operation it needs to do
- abortedTransaction : dictionary of all aborted transactions

Methods:

- mainTransactionManager(): Main method of transaction manager, calls readVariable(), writeVariable(), endTransaction(). Initializes read only transaction or a normal transaction when they begin, creates a new Transaction instance. Calls function to check if there is a deadlock in blockedQueue or the graph. Aborts appropriate transaction if there is a deadlock. Takes: transaction that is being executed/has been read and time used for startTime of transaction as inputs.
- writeVariable(): Responsible for processing write request. Obtains write lock for variable for this transaction, if lock couldn't be acquired its state is changed to blocked or waiting. Adds transaction to blockedQueue or waitingTransactions appropriately. Takes parameters of the transaction parsed from input command as input.
- readVariable(): Responsible for handling read requests. Obtains read locks on variables. If can't acquire lock then transaction's status is changed to blocked or waiting. Adds transactions blockedQueue or waitingTransactions appropriately. Takes as inputs, parameters parsed from command and a flag which is if true, then this was a waiting transaction and we try to give it a lock.
- readVariableRO(): Responsible for read operations for read only transactions. Takes as input transaction that is requesting read operation, variable to be read, name of current transaction and a flag which if true we are trying to allow a waiting transaction read.
- endTransaction(): Responsible for ending transaction and calls release lock function to release its locks. Transaction to be ended is taken as input.
- checkAbortedTransaction(): Check if any transaction was marked aborted then abort it.
- abortTransaction(): Aborts transaction, removed from queues and releases its locks. Transaction to be aborted is taken as input.
- changeStateFromBlockedToWaiting(): Check if any transaction's state can be changed from blocked to waiting. If so changes state and try to give it a lock.
- releaseLockForThisTransaction(): Release lock for transaction, clears the locks it holds on all variables at all sites. Takes transaction releasing lock as input.
- checkYoungestTransaction(): Finds youngest transaction in deadlock in the graph to kill. Takes graph instance and transaction to be killed (youngest) as inputs.
- tryToExecuteWaitingTransactions(): Tries to execute transactions in the waiting queue
- getFlattenedBlockedQueue(): Change time ordered dict and flatten to obtain list of dictionaries in blockedQueue
- getFlattenedWaitingQueue(): Change time ordered dict and flatten to obtain list of dictionaries in waiting transaction.
- checkLoopInBlockedQueue(): Checks if there is a loop in blocked queue. Returns true if there is a deadlock else false
- findLoop(): Finds the loop in blockedQueue and kills appropriate transaction. Takes transaction, visited list for dfs, current list, list of blocked transactions as inputs and returns true if loop found else false