Advanced Database Systems - CSCI-GA.2434-001 - Fall 2020

Final Project Design Doc - RepCRec

Professor: Dennis Shasha

Students: Lingwei Luo (ll4123), Yuyang Fan (yf1357)

There are mainly 2 objects in this DB system: *site* (1-10) and *transaction*, correspondingly, we will have 2 major classes to encapsulate their methods, i.e. *DataManager* and *TransactionManager*.

# A *DataManager* represents a site, where all variables and locks would be stored here.
Class DataManager:

# initialize variables in each site, create data table and lock table
def __init__():
Input: site Id
Output: finish initialization of site

# output all useful info about this site
def dump():
Input: None
Output: print site status, list of commit value for all variables

# a transaction T want to read a variable i from this site
def read():
Input: transaction Id, variable Id
Output: First judge the current lock type on this variable, then try to get read lock of this variable, return True or False, which indicate whether this read is success or fail

# a read-only transaction T want to read a snapshot of variable i from this site
def read_snapshot():
Input: transaction Id, variable Id
Output: find the latest commit value of this variable before begin time of T in commit queue.

# Judge whether write lock of variable_id can be obtained from this site
def can_get_write_lock():
Input: transaction Id, variable Id
Output: True / False means can/can't get write lock

# a transaction T want to write a variable i to value V from this site

```
    def write():
        Input: transaction Id, variable Id, value
        Output: As write operation would be first judged by can_get_write_lock(), so when we do write, all write lock can must be obtained, we can safely write value to var and set new write lock.

    # a transaction is aborted, do corresponding operations in current site
    def abort():
        Input: transaction Id
        Output: release current locks and queued locks of this transaction, update lock table.

    # a transaction is committed, do corresponding operations in current site
    def commit():
        Input: transaction Id
        Output: Add all temp value of this transaction in this site to commit list, release all current and queued locks, update lock table.

    # a site fail, do corresponding operations in current site
    def fail():
        Input: site fail timestamp
        Output: Set site status to down and clear lock table in this site

    # a site recover, do corresponding operations in current site
    def recover():
        Input: site recover timestamp
        Output: Set site status to up

    # update lock table to move queued lock
    def update_lock_table():
        Input: None
        Output: After a transaction commit/abort, move possible queued lock to variable.

    # return this site's wait for graph for cycle detection
    def get_blocking_graph():
        Input: None
        Output: a waits-for all variables in current site


# TransactionManager is used to process all instructions and conduct corresponding operations for various transactions
Class TransactionManager:
```

```
# initialize Transaction Manager
def __init__():
        Input: None
        Output: call DM to finish initialization of all sites

# output all useful info about all sites
def dump():
        Input: None
        Output: call DM to print info about each site

# process a line from input file
def process_line():
        Input: a pared line of input
        Output: True or False, which indicate whether this line is processed correctly

# do corresponding operation according to the command ("begin", "beginRO", "read",
"write", "dump", "end", "fail", "recover")
def process_command():
        Input: parsed commands
        Output: do operations

# execute all operations in current operation queue
def execute_operations():
        Input: None
        Output: loop through operation queue, call read/write to execute, if execution
        succeed, remove it from queue, otherwise let it remain there

# add read operation to operation queue, in case read fail, it will remain there to be
executed later
def add_read_opration():
        Input: transaction Id, variable Id
        Output: add read operation to operation queue

# a transaction T want to read a variable i
def read():
        Input: transaction Id, variable Id
        Output: call DM to read from any sites which have this variable, return True or
        False, which indicate whether this read is success or fail

# a read-only transaction T want to read a snapshot of variable i
def read_snapshot():
        Input: transaction Id, variable Id
        Output: call DM to read from any sites which have this variable, return True or
        False, which indicate whether this read is success or fail
```

```
# add write operation to operation queue, in case read fail, it will remain there to be
executed later
def add_write_opration():
    Input: transaction Id, variable Id, value
    Output: add write operation to operation queue

# a transaction T want to write a variable i to value X
def write():
    Input: transaction Id, variable Id, value
    Output: first check whether can obtain all write locks from relevant up sites, if at
    least 1 relevant up site cannot be written, give up. Otherwise, call DM to write to
    all relevant up sites.

# a transaction is about to begin, do corresponding operations
def beigin():
    Input: transaction Id, a flag indicating whether we begin a read-only transaction
    Output: Initialize this transaction with current timestamp and add it into
    transaction table

# a transaction is about to end, do corresponding operations
def end():
    Input: transaction Id
    Output: if the abort flag of this transaction is true, abort it. Otherwise commit it.

# a transaction is about to abort, do corresponding operations
def abort():
    Input: transaction Id
    Output: call DM to abort this transaction, remove transaction id from transaction
    table in TM.

# a transaction is about to commit, do corresponding operations
def commit():
    Input: transaction Id
    Output: call DM to commit this transaction, remove transaction id from
    transaction table in TM.

# a site fail, do corresponding operations for related transactions
def fail():
    Input: site Id
    Output: call DM to do failure operations in site, for all transactions which have
    ever accessed this failed site, set their abort flag to true

# a site recover, do corresponding operations for related transactions
```

```
def recover():
    Input: site Id
    Output: call DM to do recovery operations in site

# return this site’s wait for graph for cycle detection
def solve_deadlock():
    Input: None
    Output: collect waits-for graphs from all sites, then abort youngest transaction if there is a cycle.
```