CSE-5331-001-DBMS MODELS AND IMPLEMENTATION

Project 1 Phase 1

Rigorous 2PL with Wound-Wait protocol

Pavithra Rathinasabapathy(1001698736)

Tharoon T Thiagarajan(1001704601)

Data Structure:

The data structure used for rigorous 2PL with wound-wait protocol is as follows:

- Hash Map for Transaction Table and Lock Table
- Queue for List of blocked transactions operations

Additional Information in the lock table:

A separate column to keep track on the number of read operation in the lock table is added.

Information stored in transaction Table:

- transaction_id
- transaction_timestamp
- transaction_state
- list_of_items_locked

Information stored in Lock Table:

- item_name
- lock_state
- transaction_holding_lock
- transaction_id_waiting_for_lock
- lock_status
- no_of reads

Pseudo Code for rigorous 2PL with wound-wait protocol:

This program simulates the behavior of rigorous 2 phase locking

```
Create a hash map for Transaction table
Create a hash map for Lock table
declare timestamp_counter to 0
declare transaction_id, data_item
read the input file
get the operation, transaction id and data item from the file


if operation is not equal to "b"
        check if transaction_id is already blocked or aborted in the transaction table
        if transaction_state is equal to "blocked":
                add the operation to the list of blocked operations
        else if transaction_state is equal to "aborted":
                Ignore the operation and fetch the next operation and proceed

Iterate until all the operations are read from the file
if operation is equal to "b"
        increment timestamp_counter
        set transaction_id
        goto: begin_transaction(transaction_id, timestamp_counter)

else if operation is equal "r"
        set transaction_id and data_item;
        goto: read_lock(transaction_id, data_item)


else if operation is equal to "w"
        set transaction_id and data_item;
        goto: write_lock(transaction_id, data_item)

else if operation is equal to "e"
        set transaction_id;
        goto: end_transaction(transaction_id)

Fetch next operation and proceed

// Pseudo code for every new transaction begin
begin transaction(transaction_id, timestamp_counter):
```

```
        set transaction_id in the transaction table
        set the counter value to the transaction_timestamp in the transaction table
        set transaction_state to active in the transaction table
        set null in the list_of_items_locked
        print "A record is created for the transaction_id in the transaction table"

// Pseudo code for read lock(data_item)
read_lock(transaction_id, data_item):
        traverse the lock table for the requested data_item
        check if the data_item is already locked by any other transaction in lock table
        if the data_item is not in the table
                set data_item to item_name in the lock table
                set read_lock to lock_state in the lock table
                set the transaction_id to the transaction_holding_lock in the lock table
                set lock_status to locked
                set data_item to the list_of_items_locked in the transaction table
                increment no_of_reads by 1 in the lock table for that data_item
                print "transaction_id acquires read_lock on data_item"

        else if lock_state is "write_locked" and transaction_holding_lock is equal to
                transaction_id
                downgrade the lock_state to read_locked in lock table
                print "transaction_id downgrades from write_lock to read_lock on data_item"

        else if lock_state is "read_locked"
                add the transaction_id to the transaction_holding_lock
                add the data_item to the list_of_items_locked for the current transaction
                increment no_of_reads by 1 in the lock table for that data_item
                print "transaction_id acquires read_lock on data_item"

        else if lock_state is "write_locked and transaction_id is not equal to
                transactionID_holding_lock
                // prevent deadlock using wound-wait protocol
                if timestamp of requesting transaction is less than transaction holding the lock
                        abort the transaction holding the lock and release all the locks held by
                        transaction holding the lock
                        set lock_status to unlock to all the items held by transaction_holding_lock
                        change the transaction_state of the transaction_holding_lock to abort in
                        transaction table
                        print "transaction holding the lock is aborted"

                else
```

```
                    add transaction_id of transaction_requesting_write_lock to
                    transactionID_waiting_for_lock in lock table
                    change the transaction_state of the transaction requesting write lock to
                    waiting in transaction table
                    print "requesting transaction is blocked"


// Pseudo code for write_lock(data_item)
write_lock(transaction_id,data_item)
        traverse the lock table for the requested data_item
        check if data_item is already write_locked by any other transaction in lock table
        if the data_item is not in the table
                set data_item to item_name
                set write_locked to lock_state
                set the transaction_id to the transaction_holding_write_lock
                set lock_status to lock
                print "transaction_id acquires write_lock on data_item"

        else if transactionID_holding_lock is equal to transactionID_requesting_write_lock and
                lock_state is equal to "read_locked" and no_of_reads < 2
                upgrade the lock_state to write_lock in lock table
                print "transaction_id upgraded from read_lock to write_lock on data_item"

        else if lock_state is  "write_locked" or "read_locked" and transaction_id is not equal to
                transactionID_holding_lock
                // prevent deadlock using wound-wait protocol
                if timestamp of requesting transaction is less than transaction holding the lock
                        abort the transaction holding the lock and release all the locks held by
                        transaction holding the lock
                        update the no_of_reads in the lock table
                        set lock_status to unlock to all the items held by transaction
                        change the transaction_state of the transaction_holding_write_lock to
                        abort in transaction table

                        print "transaction holding the lock is aborted"
                else
                        add transaction_id of transaction_requesting_write_lock to
                        transaction_id_waiting_for_lock in lock table
                        change the transaction_state of the transaction_requesting_write_lock to
                        waiting in transaction table
                        print "requesting transaction is blocked"
```

```
// Pseudo code for end transaction
end_transaction(transaction_id):
        check the transaction state of the current transaction to determine whether to commit or
        abort the transaction
        if transaction_state is equal to "active"
                goto: commit
        else if transaction_state is equal to "blocked"
                goto:abort_transaction(transaction_id)
        else if transaction_state is equal to "aborted"
                ignore the end operation

// Pseudo code for abort transaction
abort_transaction(transaction_id):
        if transaction_id is equal to transactionID_holding_lock
                goto: unlock_transaction(transaction_id)
                change the status of the transaction_state to aborted in the transaction table
                print "transaction_id aborted"

// Pseudo code for commit
commit(transaction_id):
        change the status of the transaction_state to committed in the transaction table
        goto:unlock_transaction(transaction_id)
        print "transaction_id commited"

// Pseudo code for unlock transaction
unlock_transaction(transaction_id):
        release all locks held by the transaction_id
        change the lock_status to unlock in the lock table
        update the no_of_reads in the lock table if it is read lock
        traverse the list of blocked operation
        if there are any waiting operation in the list of blocked operations, process list one by one
                iterate each item in the list of blocked operations
                if operation is equal to "r"
                        goto:read_lock(transaction_id, data_item)
                else if operation is equal to "w"
                        goto:write_lock(transaction_id, data_item)
        else

                return
```