CSE5331 - DBMS Model and Implementation

REPORT - PROJECT 2 (CONCURRENCY CONTROL)

Project Team:

Mahesh Hingane - 1001017122

- OVERALL STATUS OF THE PROJECT -

  The project is ***completed*** and can be executed successfully. Here is the brief description of the implementation of some major components:

  1. *begintx()* method in zgt_tx.C - This method creates a new transaction object after acquiring semaphore. The new transaction is also inserted into the linked list of active transactions.
  2. *ReadTx()* method in zgt_tm.C - This method creates a new thread for the read operation. On successful creation of the thread, *readtx()* method from zgt_tx.C is called.
  3. *readtx()* method in zgt_tx.C - This method performs the read operation, i.e., decrements the value of the objarray[obno] variable by 1. This is done only if the object to be read is not locked by any other transaction, or it is locked by the calling transaction itself. If the object is locked by some other transaction, the calling transaction waits until this transaction releases the lock on the object.
  4. *WriteTx()* method in zgt_tm.C - This method creates a new thread for the write operation. On successful creation of the thread, *writetx()* method from zgt_tx.C is called.
  5. *writetx()* method in zgt_tx.C - This method performs the write operation, i.e., increments the value of the objarray[obno] variable by 1. This is done only if the object to be written is not locked by any other transaction, or it is locked by the calling transaction itself. If the object is locked by some other transaction, the calling transaction waits until this transaction releases the lock on the object.
  6. *CommitTx()* method in zgt_tm.C - This method creates a new thread for the commit operation. On successful creation of the thread, *committx()* method from zgt_tx.C is called.
  7. *committx()* method in zgt_tx.C - This method performs the commit operation, i.e., releases all locks held by current transaction and changes status of the transaction to *END*.
  8. *AbortTx()* method in zgt_tm.C - This method creates a new thread for the abort operation. On successful creation of the thread, *aborttx()* method from zgt_tx.C is called.
  9. *aborttx()* method in zgt_tx.C - This method performs the abort operation, i.e., releases all locks held by current transaction and changes status of the transaction to *ABORT*.

- CHALLENGING PART -

  The thread logic and flow of the project were easy to understand, but the semaphore logic was a little complicated for me. The links given by Dr. Sharama helped to understand the coding of this semaphore logic.

- LOGICAL ERRORS -

  1. Semaphore and mutex logic - Understanding semaphore and mutex logic and coding for that was a little challenging. I had to make sure that the mutex were used appropriately so that the ordering between the operations of a transaction and serializability of multiple transactions is maintained.

2. Which variable to modify in read and write operations? - In read and write operations, the decrement and increment is done on the objarray[obno] variable. Initially, I found it hard to identify which variable to use, as I was confusing it with the count variable in param.
3. Freeing transaction waiting on a transaction - To achieve S2PL logic, we need to release all the transactions waiting on a particular transaction on its completion. For this, we need to use the zgt_v() method for as many times as the number of transaction that are waiting on this transaction. Using the zgt_v() method multiple times was the tricky part.