**CSE DBMS 5331 Project – 1 Report**

**Team No. 2**

**Hemakumar Gokulakannan(1001415255)**

**Overall Status**

The Project has been completed and all the methods that were to be implemented have been completed. Methods **begintx, readtx, writetx, aborttx & committx** have been coded as per the requirements give in **cse5331\_fall-16\_project2\_v1.pdf**. Following are briefs about how we implemented the methods in this project.

1. **begintx():**

Since the code for this method was already provided to us there were no changes that we did in this.

1. **readtx():**

In this function we first check the type of transaction that we are dealing with, depending on that we assign the type of lock mode which we are going to assign i.e S or X. We later check if the transaction is active, if yes we check if it has acquired a lock successfully, if not we assign it a lock by using set lock method and then proceed. Once we have successfully assigned a lock we perform the read operation by invoking perform\_readWrite(…) method.

If the transaction that we are dealing has the status as abort, in that case we enter the details of the transaction to the log file.

1. **writetx():**

In this function we check if the transaction that we are dealing with is active, if it is active we assign it an exclusive lock (since the transaction we are dealing with is a Write). Once the lock has been successfully acquired using set lock method, we invoke the perform\_readWrite. If we know the transaction status has changed to abort we write the transaction details to log file.

**Set\_lock():** In this method we are actually checking whether a lock can be granted to a transaction on a particular object or it has to wait. If there is no record of the object in the hash table used for locking, or the object is locked by the same transaction which is again requesting the lock or the lock type was ‘S’ requested by a Read type transaction before, then we grant the lock to the new transaction without any wait, else the new transaction is kept on hold till the one who held the lock commits and releases the lock. This fails in the deadlock scenario as deadlock handling has not been done

1. **perform\_readWrite(…):**

In this we check what kind of lock was requested from the calling function. If it’s a read operation then we decrement the Object No. value by 1. In a case where we have write operation, we increment the Object No. value by 1. In either case we append transaction and log details to the log file

1. **aborttx():**

In order to perform this operation, we first start the transaction then assign a semaphore to it, do the actual abort using do\_commit\_abort(…) method, and once it is done release the semaphore and finish the transaction.

1. **committx():**

Inthis method we start by changing the status of the transaction to TR\_END, once this is done we perform the actual commit using do\_commit\_abort(…) method. We also calculate the number of time a V operation needs to be carried out in order to commit the transaction, till that time we make the semaphore wait and once the transaction is committed we release it.

1. **do\_commit\_abort(…):**

In this method we check if the calling function has requested for a commit or an abort, we do this based on the value of the status i.e TR\_END and TR\_ABORT respectively. Once we recognize the status, we append an appropriate message to log file. At the end of these operations we free all the locks we acquired during the course of the transaction.

**File Description**

No new files were created for this project.

**Logical Errors and How they were handled**

According to us following were the logical errors we face while coding –

1. Doing a P operation in a P operation hangs the program.
2. Missing V operations to P operations also hangs the program.
3. If the log file is updated after releasing the thread, then sometimes the order of operations in the log file is not correct (Mostly was happening during multiple runs), as some other thread might get processed and does some activity till the time log is updated.
4. Nwait gives an output of -1 if it is not processed correctly.
5. Received Segmentation Fault(Core Dumped) error initially on few test files execution.