**CSE 5331: DBMS MODELS AND IMPLEMENTATION**

**FALL 2018**

**Project 2: Implementation of Transaction Manager**

Under the guidance of –

Prof. Sharma Chakravathy

Abhishek Santra (TA)

Submission by Team no.17

Kanthi Sannapa Komar (1001529656)

Shreyas Mohan (1001669806)

**Index**

|  |  |  |
| --- | --- | --- |
| **Sl.No.** | **Topic** | **Page Number** |
| 1 | Acknowledgement | 3 |
| 2 | Overall Status | 4 |
| 3 | File Description | 5 |
| 4 | Division of Labor | 6 |
| 5 | Logical Errors | 7 |
| 6 | References | 8 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Acknowledgement**

We would like to express special thanks of gratitude to our professor, Prof. Sharma Chakravathy who gave us the golden opportunity to do this project on Transaction Manager. Through this project, our foundation on Transaction Manger logic became stronger and the understanding of the concept became more clearer through practical knowledge. We also like to thank our teaching assistant, Mr. Abhishek Santra who gave his valuable time in explaining how to begin with the project.

Kanthi Komar

Shreyas Mohan

**Overall Status**

In our project we have implemented all 5 functions such as **BeginTx, ReadTx, WriteTx, AbortTx and CommitTx.**

The first operation any transaction encounters is **BeginTx**. Transaction Manager object will be initialised with appropriate values in this function. This is done after locking the transaction manager. This brings the transaction to active state. Each operation in the BeginTx function explained with easy understandable comments in the code.

Second function is ReadTx, which is called when the transaction has to access the object for reading the data. This in turn call the **set\_lock** function which grants two type of locks. Exclusive lock Read/Write transaction. Shared lock for Read only transaction. If the requesting transaction already holds a lock on the object, it directly moves on to **perform\_readWrite** function. If the requesting transaction needs a lock on a object which is currently being used by another transaction, it is put in the waiting queue. To grant the object a lock, it is put on to Hash table and then it can move on to perform\_readWrite. Comment section in the code explains each and every scenario of set\_lock function.

Third function **WriteTx** is also similar to readTx, this function is called when the transaction has to access an object to write on to it. WriteTx also calls set\_lock and exclusive lock is granted, and in perform\_readWrite, write is simulated.

After Read and Write Transaction has to commit if it executed smoothly or has to be Aborted if there occurs any error. **AbortTx** aborts any transaction. AbortTx call the **do\_commit\_abort** function. Here all the holding locks are set to free. All the waiting transactions are informed that locks are set free by doing the ‘v’ operation on the sem\_no. Similarly, **CommitTx** calls do\_commit\_abort and sets all the locks free of the transactions being commited.

**File Description**

No additional files have been created in this project.

**Division of Labor**

In this project we (Shreyas Mohan, Kanthi Komar) worked together on major 5 functions. Understanding the concepts and planning on what functions to use for what operations was worked out in the team. Major part of Set\_locks was worked out by Shreyas Mohan. Major part of do\_commit\_abort, Perform\_readWrite was done by Kanthi Komar. Testing was done by Shreyas and Report was done by Kanthi.

**Encountering Logical Errors**

1. Even after a transaction commits, it was moving on to performing ReadTx or writeTx. This was because we were not releasing all the locks in do\_commit\_abort function. After doing the ‘V’ operation on the number of transaction waiting in the queue, this problem was solved. We had to release all the semaphores on the waiting transaction.
2. We missed out of a few conditions to be applied when making a transaction wait. If Lock requested for is a Shared lock, the current lock on the object is an Shared lock and there is more than 1 transactions waiting for it, the transaction has to be added to the waiting queue. This condition was not applied and we encountered major problems and were unable to get the output. After adding this condition in the set\_lock function we could move further.
3. We had not called start\_operation in some functions which gave us errors. We understood that start\_operation manages each transaction operations to execute one after the other.
4. We had missed on releasing the transaction manager in commit and abort operations, which was leading to malfunctioning of the program. Later,we were able to resolve it.
5. While testing the program, we were encountering ‘permission denied’. By giving executable permissions to tmtest, the error got resolved.

**References**

1. <https://wweb.uta.edu/faculty/sharmac/courses/cse5331And4331/current-offering/module-CC/CC1.pdf>
2. <https://wweb.uta.edu/faculty/sharmac/courses/cse5331And4331/current-offering/module-CC/CC2.pdf>
3. <https://wweb.uta.edu/faculty/sharmac/courses/cse5331And4331/current-offering/module-CC/indxLock.pdf>
4. Chapter 16,17 of Database Systems by Ramakrishnan and Gehrke 3rd Edition
5. Set Lock method’s waiting condition was understood from-

https://github.com/vasanth-mahendran/transaction-manager