CSE 5331- DBMS Models and Implementation

Project 2: Implementation of a Transaction Manager

Team no.: 30

Group members:

1. Pratik Antoni Patekar (1001937948)
2. Anurag Reddy Pingili (1001863623)

Table of contents

1. Overall status:
Following is the status on implementation tasks assigned in this project:

1. zgt_tx.C - In this class, we have completed the implementation of the four functions (excluding begintx function) assigned.
    a. readtx
    b. writetx
    c. aborttx
    d. committx
2. zgt_tm.C - In this class, we have completed the implementation of the following functions.
    a. TxRead
    b. TxWrite
    c. CommitTx
    d. AbortTx
3. We have successfully combined and ran using the test data files given.

Below is more information on how we have implemented the above mentioned classes and functions.

1.1. Transaction manager class (zgt_tm.C): In this class, we first have set the team_no to our team no. i.e. 30. This class basically creates threads for each operation of a transaction as per mentioned in the input text file. There are 6 operations that are basically performed.
1. openlog(string lfile): This function creates a log file and writes the initial headers in the log file.In all the other functions, we have implemented a similar procedure to be followed.

2. BeginTx(long tid, int thrNum, char type): This function basically creates a thread corresponding to the begin transaction operation mentioned in the input text file. For doing so it creates a node with the required information such as tid (transaction id), Txtype (transaction type), object number (obno = -1) and count (= 0). This node is then passed as arguments to the begintx function in zgt_tx.C (discussed later).

3. TxRead(long tid, long obno, int thrNum): This function basically performs the same thing as BeginTx but here it sets the object number to the required obno and count is the sequence number of the tid decremented by 1. And then this node is then passed as arguments to the readtx function in zgt_tx.C (discussed later).

4. TxWrite(long tid, long obno, int thrNum): Similar to the TxRead function. The difference is just that the node is passed as arguments to the writetx function in zgt_tx.C (discussed later).

5. CommitTx(long tid, int thrNum): The arguments passed are tid and count only to the committx function in zgt_tx.C (discussed later).

6. AbortTx(long tid, int thrNum): Same as in the CommitTx function.

7. endTm(int thrNum): This function is called when end all is encountered in the input text file. This function was already implemented in the given code files.

1.2. Transactions class (zgt_tx.C): This class basically performs the transaction operations by setting the required locks for different threads.

1. begintx(void *arg): This function is called from the BeginTx function in the transaction manager class. It starts the operation by calling start_operation function. The start_operation function makes all the threads belonging to the same transaction mutually exclusive from each other depending upon the condition set. Then adds the transaction node to the transaction manager data structure at LASTR (while modifying the transaction manager values, we had to lock the transaction manager using semaphore 0). Once this is done, finish the operation by unlocking the threads.

2. readtx(void *arg): This function calls the start_operation() function and then gets the transaction from the transaction manager using the get_tx() function. Then we send the transaction values i.e, tid, obno and count in a shared lockmode('S') to the set_lock() function (discussed below in detail). Finally, it calls the finish_operation() function and exits the assigned thread using pthread_exit(NULL).

3. writetx(void *arg): This function performs the same operations as in the readtx function above. The only difference is that it sets an exclusive lock.

4. aborttx(void *arg): This function first starts the operation using the start_operation function and then calls the do_commit_abort() function to perform the abort and commit operations. While calling the function do_commit_abort() function, the operation (abort or commit) is indicated by the status argument. If the status argument is passed as 'A' then the abort operation is performed and if it is 'C' then a commit operation is performed. After that, the operation is completed by calling finish_operation function.

5. committx(void *arg): This function behaves the same way as the aborttx() function but sends a 'C' character to the do_commit_abort() function as the status to indicate a commit of the transaction.

6. do_commit_abort(long t, char status): This function performs the actual commit and abort of the transactions. It first gets the transaction from the transaction manager using the get_tx() function and then checks if the status is 'C' or 'A'. If the transaction status is 'A', it writes the abort of the transaction to the logfile. It frees all the locks that the transaction is holding using the free_locks() function and then removes the transaction from the Transaction manager using the remove_tx() function so that none of the operations done by the transaction are made permanent. If the transaction status is 'C', it does the same thing that it does for 'A' with one key difference: it calls the end_tx() function instead of the remove_tx() function so that the operations done by the transaction are not removed.

7. remove_tx () and end_tx(): These two functions are used by do_commit_abort() functions. The remove_tx() function is used to remove the transaction from the transaction list in the transaction manager without committing the changes to the disk. Whereas the end_tx() function is used to remove transactions as well as committing the changes to the disk. Thus the remove_tx() function is used to perform the abort operation and end_tx() function is used to perform the commit operation.

8. free_locks(): This function simply removes all the locks that a transaction holds. It does this by using the head pointer of the transaction to locate all the objects locked by the transaction i.e, all the objects that the transaction put into the hash table. It removes all these objects from the hash table, thus unlocking them.

9. set_lock(long tid1, long sgno1, long obno1, int count, char lockmode1): As mentioned in the above points, the readTx and writeTx functions use the set_lock function. The set_lock function basically handles the locks (shared and exclusive) for all objects. Following are the steps that we have followed for locking a particular object:

a. Check the hash table (lock table) in the transaction manager data structure to find if there is any existing lock on the object with obno1. Note: Before reading or making any changes in the lock table, we need to lock the semaphore 0 (semaphore for locking transaction manager).
b. If there is no lock (shared or exclusive) on the required object then grant the access. On granting the access, perform read/ write by calling the perform_read_write() function (discussed below).
c. If there is a lock on the required object then check if the same current transaction is having the access then perform read/ write by calling the perform_read_write() function as there is no need to grant access.
d. If there is a lock on the required object but the transaction is different then find the transaction that is holding the current lock. And then,
    i. If the current transaction requires shared lock and the transaction holding the current lock is a read-only transaction then grant the shared lock on the required object and perform read write.
    ii. If the transaction currently holding the object is a read/ write transaction or if current transaction requires an exclusive lock or if current transaction requires shared lock and transaction holding the lock is read-only transaction but there are other transactions that are waiting on the required object then do the following: Set a semaphore for the current transaction and make it wait till the current transaction performs its operations. This is done using the setTx_semno() function and then zgt_p() and zgt_v() functions. Setting the value of zgt_p(x) allows the transaction x to complete its critical section and it is unset by using zgt_v(x). While the transaction having the lock on the required object executes its critical section, the current transaction is made to wait by setting its values to default values as mentioned in the project description file (i.e. obno = -1, lockmode = ‘ ‘ and status = TR_ACTIVE).

10. perform_read_write(long tid, long obno, char lockmode): This function performs the actual read and write operations on the objects in a simulated manner. First, it checks the lockmode of the object. If it is a shared lock denoting a read operation, it accesses the object in the object array using the obno and decrements its value by 1 and writes this to the log file. After this, to simulate the amount of time required for an actual disk read operation, the control sleeps for the given amount of time using the sleep() function. The time for which the operation is made to sleep depends on the optime array which is generated randomly using a large prime number as seed.
If the lockmode is exclusive denoting a write operation, the function does the same thing as a read operation but instead of decrementing the value, it increments the value of the object in the object array by 1 and then sleeps for the required amount of time.

2. Input/output, log files and overall status of outputs:

2.1. Input and log files:
1. no_conflists_2Txs.txt

```
// serializable history
// 2 transactions (no conflicts)
// same object accessed
// multiple times
Log no_conflicts_2Txs.log
BeginTx 1 R
Read    1 1
Read    1 2
BeginTx 2 W
Read    2 8
Read    2 7
Write   2 6
Write   2 5
Commit  2
read    1 3
read    1 4
Commit  1
end all
```

```
[axp3623@omega src]$ cat no_conflicts_2Txs.log
---------------------------------------------------------------------------
TxId    Txtype  Operation       ObId:Obvalue:optime     LockType        Status          TxStatus
T1      R               BeginTx
T1                      readTx          1:-1:29614              ReadLock        Granted         P
T2      W               BeginTx
T2                      readTx          8:-1:3675               ReadLock        Granted         P
T2                      readTx          7:-1:3675               ReadLock        Granted         P
T2                      writeTx         6:1:3675                WriteLock       Granted         P
T2                      writeTx         5:1:3675                WriteLock       Granted         P
T2                      CommitTx
T1                      readTx          2:-1:29614              ReadLock        Granted         P
T1                      readTx          3:-1:29614              ReadLock        Granted         P
T1                      readTx          4:-1:29614              ReadLock        Granted         P
T1                      CommitTx
```

2. interleaved_RW.txt

```
log interleaved_RW.log
BeginTx 1 W
Read 1 1
Write 1 2
Read 1 3
Write 1 4
BeginTx 2 W
Write 2 8
Write 2 2
Read 2 1
BeginTx 3 W
Write 3 1
Write 3 2
Write 3 9
Read 3 8
Commit 1
commit 2
Commit 3
end all
```

```
[axp3623@omega src]$ cat interleaved_RW.log
------------------------------------------------------------------------
TxId    Txtype Operation       ObId:Obvalue:optime LockType       Status          TxStatus
T1      W      BeginTx
T1             readTx 1:-1:29614            ReadLock      Granted              P
T2      W      BeginTx
T3      W      BeginTx
T2             writeTx8:1:3675              WriteLock     Granted              P
T2             writeTx2:1:3675              WriteLock     Granted              P
```

3. Multi_ROTxs.txt

```
// Multiple RO Txs test case
// read only transactions
log Multi_ROTxs.log
// op     Tx#      type
// op     Tx#      Obj
BeginTx 1 R
Read 1 1
Read 1 2
Read 1 3
Read 1 8
BeginTx 2 R
Read 2 1
Read 2 8
Read 2 5
BeginTx 3 R
Read 3 1
Read 3 5
Read 3 3
read 3 7
Commit 2
commit 3
Commit 1
end all
```

```
[axp3623@omega src]$ cat Multi_ROTxs.log
--------------------------------------------------------------------------
TxId    Txtype  Operation      ObId:Obvalue:optime     LockType       Status        TxStatus
T1      R          BeginTx
T1                 readTx         1:-1:29614              ReadLock       Granted        P
T2      R          BeginTx
T2                 readTx         1:-2:3675               ReadLock       Granted        P
T2                 readTx         8:-1:3675               ReadLock       Granted        P
T2                 readTx         5:-1:3675               ReadLock       Granted        P
T3      R          BeginTx
T3                 readTx         1:-3:19271              ReadLock       Granted        P
T2                 CommitTx
T3                 readTx         5:-2:19271              ReadLock       Granted        P
T1                 readTx         2:-1:29614              ReadLock       Granted        P
T3                 readTx         3:-1:19271              ReadLock       Granted        P
T3                 readTx         7:-1:19271              ReadLock       Granted        P
T1                 readTx         3:-2:29614              ReadLock       Granted        P
T3                 CommitTx
```

```
T1          readTx      8:-2:29614          ReadLock    Granted     P
T1          CommitTx
```

4. disj_multi_accesses.txt

```
// serial history
// 2 transactions
// same disjoint objects accessed
// multiple times
Log disj_multi_accesses.log
BeginTx 1 W
Read    1 1
Read    1 2
Write   1 3
Write   1 4
read    1 1
write   1 2
write   1 4
write   1 4
commit 1
begintx 2 W
read    2 5
write   2 5
write   2 6
read    2 6
commit  2
end all

[axp3623@omega src]$ cat disj_multi_accesses.log
--------------------------------------------------------------------------
TxId    Txtype  Operation     ObId:Obvalue:optime    LockType     Status      TxStatus
T1      W         BeginTx
T1                readTx      1:-1:29614             ReadLock     Granted     P
T2      W         BeginTx
T2                readTx      5:-1:3675              ReadLock     Granted     P
T2                writeTx     5:0:3675               WriteLock    Granted     P
T2                writeTx     6:1:3675               WriteLock    Granted     P
T2                readTx      6:0:3675               ReadLock     Granted     P
T2                CommitTx
T1                readTx      2:-1:29614             ReadLock     Granted     P
T1                writeTx     3:1:29614              WriteLock    Granted     P
T1                writeTx     4:1:29614              WriteLock    Granted     P
T1                readTx      1:-2:29614             ReadLock     Granted     P
T1                writeTx     2:0:29614              WriteLock    Granted     P
```

```
T1          writeTx     4:2:29614          WriteLock    Granted      P
T1          writeTx     4:3:29614          WriteLock    Granted      P
T1          CommitTx
```

5. ddlk_3Txs.txt

```
// possible deadlock test case
// Two write transactions
log ddlk_3Tx.log
// op    Tx#     type
BeginTx 1 W
// op    Tx#     Obj
Read 1 1
Write 1 2
Read 1 6
BeginTx 2 W
Read 2 2
Write 2 1
Read 2 7
commit 2
Commit 1
begintx 3 R
read 3 2
write 3 1
read 3 2
end all
```

```
[axp3623@omega src]$ cat ddlk_3Tx.log
-------------------------------------------------------------------------------
TxId   Txtype  Operation      ObId:Obvalue:optime    LockType      Status       TxStatus
T2     W          BeginTx
T1     W          BeginTx
T1             readTx       1:-1:29614             ReadLock      Granted       P
T2             readTx       2:-1:3675              ReadLock      Granted       P
T3     R          BeginTx
T3             readTx       2:-2:19271             ReadLock      Granted       P
```

6. ddlk_2Txs.txt

```
// 2 transactions
// classic deadlock
// will hang w/o deadlock resolution
Log ddlk_2Tx.log
BeginTx 1 W
BeginTx 2 W
Read    1 1
Read    2 2
Write   1 2
Write   2 1
Commit  1
commit 2
end all
```

```
[axp3623@omega src]$ cat ddlk_2Tx.log
--------------------------------------------------------------------------------
TxId    Txtype  Operation       ObId:Obvalue:optime     LockType        Status          TxStatus
T1      W               BeginTx
T2      W               BeginTx
T1                      readTx          1:-1:29614              ReadLock        Granted         P
T2                      readTx          2:-1:3675               ReadLock        Granted         P
```

7. test_abort.txt

```
// simple deadlock test case
// Two write transactions
log test_abort.log
// op     Tx#      type
// op     Tx#      Obj
BeginTx 1 W
read 1 6
write 1 7
write 1 7
read 1 6
begintx 2 W
read 2 8
write 2 7
abort 2
begintx 3 R
read 3 4
write 3 5
read 3 9
commit 3
commit 1
end all
```

```
[axp3623@omega src]$ cat test_abort.log
---------------------------------------------------------------------------
TxId    Txtype  Operation      ObId:Obvalue:optime     LockType       Status        TxStatus
T1      W            BeginTx
T1                   readTx         6:-1:29614              ReadLock       Granted         P
T2      W            BeginTx
T2                   readTx         8:-1:3675               ReadLock       Granted         P
T2                   writeTx        7:1:3675                WriteLock      Granted         P
T2                   AbortTx
T3      R            BeginTx
T3                   readTx         4:-1:19271              ReadLock       Granted         P
T3                   writeTx        5:1:19271               WriteLock      Granted         P
T3                   readTx         9:-1:19271              ReadLock       Granted         P
T1                   writeTx        7:2:29614               WriteLock      Granted         P
T3                   CommitTx
T1                   writeTx        7:3:29614               WriteLock      Granted         P
T1                   readTx         6:-2:29614              ReadLock       Granted         P
T1                   CommitTx
```

8. RW_disjoint.txt

```
// Multiple RW Txs test case with no deadlock
log RW_disjoint.log
// op     Tx#      type
// op     Tx#      Obj
BeginTx 1 W
Read 1 1
Write 1 2
Read 1 3
BeginTx 2 W
Write 2 4
Write 2 5
BeginTx 3 W
Write 3 6
Write 3 7
read 3 8
Commit 3
commit 2
Commit 1
begintx 5 R
read 5 9
read 5 10
read 5 11
read 5 12
read 5 13
read 5 1
commit 5
end all
```

```
[axp3623@omega src]$ cat RW_disjoint.log
------------------------------------------------------------------------
TxId    Txtype  Operation       ObId:Obvalue:optime     LockType        Status          TxStatus
T1      W               BeginTx
T1                      readTx          1:-1:29614              ReadLock        Granted         P
T2      W               BeginTx
T2                      writeTx         4:1:3675                WriteLock       Granted         P
T2                      writeTx         5:1:3675                WriteLock       Granted         P
T3      W               BeginTx
T3                      writeTx         6:1:19271               WriteLock       Granted         P
T2                      CommitTx
T5      R               BeginTx
T5                      readTx          9:-1:16371              ReadLock        Granted         P
T3                      writeTx         7:1:19271               WriteLock       Granted         P
```

```
T5          readTx      10:-1:16371          ReadLock    Granted     P
T1          writeTx     2:1:29614            WriteLock   Granted     P
T3          readTx      8:-1:19271           ReadLock    Granted     P
T5          readTx      11:-1:16371          ReadLock    Granted     P
T3          CommitTx
T5          readTx      12:-1:16371          ReadLock    Granted     P
T1          readTx      3:-1:29614           ReadLock    Granted     P
T5          readTx      13:-1:16371          ReadLock    Granted     P
T5          readTx      1:-2:16371           ReadLock    Granted     P
T1          CommitTx
T5          CommitTx
```

9. RW_pot_ddlk.txt

```
// Multiple RW Txs test case with no deadlock
log RW_pot_ddlk.log
// op    Tx#     type
// op    Tx#     Obj
BeginTx 1 W
Read 1 1
Write 1 2
Read 1 3
Write 1 8
BeginTx 2 W
Write 2 4
Write 2 5
BeginTx 3 W
Write 3 6
Write 3 7
Read 3 9
Commit 3
commit 2
Commit 1
begintx 5 R
read 5 1
read 5 2
read 5 3
read 5 8
read 5 6
read 5 7
commit 5
end all
```

[axp3623@omega src]$ cat RW_pot_ddlk.log

TxId	Txtype	Operation	ObId:Obvalue:optime	LockType	Status	TxStatus
T1	W	BeginTx				
T1		readTx	1:-1:29614	ReadLock	Granted	P
T2	W	BeginTx				
T2		writeTx	4:1:3675	WriteLock	Granted	P
T2		writeTx	5:1:3675	WriteLock	Granted	P
T3	W	BeginTx				
T3		writeTx	6:1:19271	WriteLock	Granted	P
T2		CommitTx				
T5	R	BeginTx				
T5		readTx	1:-2:16371	ReadLock	Granted	P
T3		writeTx	7:1:19271	WriteLock	Granted	P
T5		readTx	2:-1:16371	ReadLock	Granted	P
T3		readTx	9:-1:19271	ReadLock	Granted	P
T5		readTx	3:-1:16371	ReadLock	Granted	P
T3		CommitTx				
T5		readTx	8:-1:16371	ReadLock	Granted	P
T5		readTx	6:0:16371	ReadLock	Granted	P
T5		readTx	7:0:16371	ReadLock	Granted	P
T5		CommitTx				
T1		writeTx	2:0:29614	WriteLock	Granted	P
T1		readTx	3:-2:29614	ReadLock	Granted	P
T1		writeTx	8:0:29614	WriteLock	Granted	P
T1		CommitTx				

10. unlikely_ddlk.txt

```
// serializable history
// ddlk unlikely
log unlikely_ddlk.log
BeginTx 1 W
Read    1 3
Read    1 2
BeginTx 2 W
Read    2 1
Write   2 3
Write   1 3
Write   1 2
Write   2 2
Commit  1
commit 2
end all
```

```
[axp3623@omega src]$ cat unlikely_ddlk.log
--------------------------------------------------------------------------------
TxId    Txtype  Operation       ObId:Obvalue:optime     LockType        Status          TxStatus
T1      W               BeginTx
T1                      readTx          3:-1:29614              ReadLock        Granted         P
T2      W               BeginTx
T2                      readTx          1:-1:3675               ReadLock        Granted         P
T1                      readTx          2:-1:29614              ReadLock        Granted         P
T1                      writeTx         3:0:29614               WriteLock       Granted         P
T1                      writeTx         2:0:29614               WriteLock       Granted         P
T1                      CommitTx
T2                      writeTx         3:1:3675                WriteLock       Granted         P
T2                      writeTx         2:1:3675                WriteLock       Granted         P
T2                      CommitTx
```

11. multiple_aborts.txt

```
// Multiple RW Txs test case with no deadlock
// 23 operations
log multiple_aborts.log
// op     Tx#      type
// op     Tx#      Obj
BeginTx 1 W
Read 1 1
Write 1 2
Read 1 3
Write 1 8
BeginTx 2 W
Write 2 4
Write 2 5
BeginTx 3 W
Write 3 6
Write 3 7
Read 3 9
abort 3
commit 2
abort 1
begintx 5 R
read 5 1
read 5 2
read 5 3
read 5 8
read 5 6
read 5 7
abort 5
end all
```

```
[axp3623@omega src]$ cat multiple_aborts.log
-------------------------------------------------------------------------
TxId    Txtype  Operation       ObId:Obvalue:optime     LockType        Status          TxStatus
T1      W               BeginTx
T1                      readTx          1:-1:29614              ReadLock        Granted         P
T2      W               BeginTx
T2                      writeTx         4:1:3675                WriteLock       Granted         P
T2                      writeTx         5:1:3675                WriteLock       Granted         P
T3      W               BeginTx
T3                      writeTx         6:1:19271               WriteLock       Granted         P
T2                      CommitTx
T5      R               BeginTx
```

```
T5          readTx      1:-2:16371          ReadLock    Granted     P
T5          readTx      2:-1:16371          ReadLock    Granted     P
T3          writeTx     7:1:19271           WriteLock   Granted     P
T3          readTx      9:-1:19271          ReadLock    Granted     P
T5          readTx      3:-1:16371          ReadLock    Granted     P
T3          AbortTx
T5          readTx      8:-1:16371          ReadLock    Granted     P
T5          readTx      6:0:16371           ReadLock    Granted     P
T5          readTx      7:0:16371           ReadLock    Granted     P
T5          AbortTx
T1          writeTx     2:0:29614           WriteLock   Granted     P
T1          readTx      3:-2:29614          ReadLock    Granted     P
T1          writeTx     8:0:29614           WriteLock   Granted     P
T1          AbortTx
```

2.2. Overall status of outputs:

	test file	**zgt_test**	**tmtest 100**	**tmtest 1000**	**max tmtest value < 50000 for which it works correctly**
1	no_conflict_2Txs.txt	ok	works	works	50000
2	interleaved_RW.txt	hangs (as it should be)	hangs	hangs	hangs
3	Multi_ROTxs.txt	ok	works	works	50000
4	disj_multi_accesses.txt	ok	works	works	50000
5	ddlk_3Txs.txt	hangs	hangs	hangs	hangs
6	ddlk_2Txs.txt	hangs	hangs	hangs	hangs
7	test_abort.txt	aborts T2 correctly	works	works	50000
8	RW_disjoint.txt	ok	works	works	50000
9	RW_pot_ddlk.txt	ok	works	works	50000
10	unlikely_ddlk.txt	ok	works	works	50000
11	multiple_aborts.txt	ok	works	works	50000

3. Difficulties and logical errors:

3.1. Where we encountered difficulty:
The difficulties we faced were mostly centered around the set_lock() function. It took a lot of time for us to correctly implement all the possible cases. We faced some problems with making the transaction wait using a semaphore when another transaction had the lock.
We also encountered some difficulties with the perform_read_write() function because we were not sure how the sleep() function corresponded to the execution. When the deadlock test case hung, we were unsure what was causing it: a deadlock or some error in the sleep() function.

3.2. File descriptions: There are no new files, data structures or additional test cases that we have added in the submission zip file.

3.3 Division of Labor:

Group member name	**Functions implemented**	**Hours spent on project on weekdays**	**Hours spent on project on weekends**
Pratik Antoni Patekar (1001937948)	Read and abort related functions in both the classes	Utmost 3 hours	4 to 5 hours
Anurag Reddy Pingili(1001863623)	Write and Commit functions in both the classes.	Utmost 3 hours	4 to 5 hours

3.4. Logical Errors:

- Some test cases were leading to a segmentation fault. We weren't sure what was causing this but it went away after we implemented our set_lock() function correctly.
- Race condition: Our test cases were hanging because we didn't check the number of threads waiting for a lock and granted the lock to a transaction immediately. This went away after we checked the wait_tx_no > 0.
- Locking and unlocking TM: We were not doing this before so we were not sure why our test cases kept hanging but we realized our error once we properly understood the beginTx function.