Name: Patel Tejas **UTA ID:** 1001729131
Name: Jariwala Prayag **UTA ID:** 1001719373

Report for Project-1

CSE – 5331

Data Structure used for tables:

Mongodb Collections: db.transactiontable

db.locktable

Following operations will be used in the program,

For updating tables : transactiontable.update_one()

locktable.update_one()

For insertion: transactiontable.insert()

locktable.insert()

Pseudo code:

```
def 2pl():
    timestamp = 0
    Read input file f:
    for a in f: //iterate through all elements in file
        If a[0] = 'b':
            Check transaction is already exists or not
            if exists skip this step otherwise Add  new transaction in transaction table
        If a[0] = 'r':
            If transaction_state == 'active':
                Check a[1] transaction id already exist and in active state
                    if a[2]=='('
                        dataitem = a[3]
                    else
                        dataitem = a[2]
                count = Check with lock table that dataitem already exist or not
```

```
if count ==0
    add data item into  lock table and update the T ID's list
    which contains this resources
if count > 0
    state = Check lock state of that data item from lock table
    if state == 'read'
        give current a[1] tid read_lock for given dataitem
    if state == 'write'
        // RW conflict occurs
        wantingtidTimeStamp = TimeStamp of resouce requesting transaction
        // its write lock only one transaction holding that resource
        holdingtidTimeStamp = TimeStamp of transaction who already holds the
                              resource
        if wantingtidtimetamp = holdingtidTimeStamp :
            update convert write lock into read lock beacause both are same
        if wantingtidtimetamp > holdingtidTimeStamp :
            // Resource requesting transaction is younger than resource holding
             transaction
            Put Resource requesting transaction in waiting queue and do changes
            for that transaction in transaction table
        if wantingtidtimetamp < holdingtidTimeStamp :
            // Resource requesting transaction is older than resource holding
            transaction
            Abort resource holding transaction and give resouce to resource
            requesting transaction
```

```
Itemholds = Items hold by aborted transaction
// Distribute Released dataitem over remaining transaction
for j in Itemholds:
    state = retrieve state of Itemholds[ j ] (dataitem) from locktable
    if state == write:
        LockwaitingTid = Retrive all lock waiting Tids
        if len(lockwaitingtid)>0:
             newstate=lockwaitingtid[1]
                 if newstate=='r':
                     newstate='read'
                if newstate=='w':
                     newstate='write'
                newlockholdingtransaction=lockwaitingtid[2]
                newlockwaitingtransactions=lockwaitingtid[3:]
                // give resouce to first waiting transaction
                update resouce holding list  with Lockwaiting[1]
    if state == read:
        // delete aborted transaction id from resouce holding list
        if len(lockholdingtid)>1:
            // lockingholdingtid = resouce holding list
            z= lockholdingtid
            for k in range(0,len(z)):
                 if z[k]==holdingtid:
                     str1=z[0:k-1]
                     str2=z[k+1:]
```

```
                        else:
                            LockwaitingTid = Retrive all lock waiting Tids
                            // give resouce to first waiting transaction update resource
                            holding list  with Lockwaiting[1] update waiting list with
                            Lockwaiting[2:]
        If transaction_state == ‘blocked’:
            If a[3] == ‘(‘:
                dataitem = a[4]
            else:
                dataitem = a[3]
                // give resouce to first waiting transaction
                update resouce holding list with Lockwaiting[1]


    If a[0] = ‘w’:
        If transaction_state == ‘active’:
            Check a[1] transaction id already exist and in active state
            if a[2]==‘(‘
                dataitem = a[3]
             else
                dataitem = a[2]
            count1 = Check with lock table that dataitem already exist or not
            if count ==0
                add data item into  lock table and update the T ID’s list which contains this
            resources
            if count > 0
                state = Check lock state of that data item from lock table
                // WR and WW conflict occurs
                wantingtidTimeStamp = TimeStamp of resouce requesting transaction
```

```
// its write lock only one transaction holding that resource
holdingtidTimeStamp = TimeStamp of transaction who already holds the
resource
if wantingtidtimetamp = holdingtidTimeStamp :
    update convert lock into write lock beacause both transaction  are same
if wantingtidtimetamp > holdingtidTimeStamp :
    // Resource requesting transaction is younger than resource holding
    transaction.
    Put Resource requesting transaction in waiting queue of that resouce in
    locktable and do changes for that.
if wantingtidtimetamp < holdingtidTimeStamp :
    // Resource requesting transaction is older than resource holding
    transaction.
    Abort resource holding transaction and give resouce to resource
    requesting transaction.
    Itemholds = Items hold by aborted transaction
    // Distribute Released dataitem over remaining transaction
    for j in Itemholds:
        state = retrieve state of Itemholds[ j ] (dataitem) from lock table
        if state == read:
            // delete aborted transaction id from resouce holding list
            if len(lockholdingtid)>1:  // lockingholdingtid = resouce holding list
                z= lockholdingtid
                for k in range(0,len(z)):
                    if z[k]==holdingtid:
                    str1=z[0:k-1]
                    str2=z[k+1:]
            else :
                LockwaitingTid = Retrive all lock waiting Tids
                // give resouce to first waiting transaction
                update resouce holding list  with Lockwaiting[1]
                update waiting list with Lockwaiting[2:]
        if state == write:
            LockwaitingTid = Retrive all lock waiting Tids
            if len(lockwaitingtid)>0:
                newstate=lockwaitingtid[1]
                if newstate=='r':
                    newstate='read'
                if newstate=='w':
                    newstate='write'
                newlockholdingtransaction=lockwaitingtid[2]
                newlockwaitingtransactions=lockwaitingtid[3:]
                // give resouce to first waiting transaction
                update resouce holding list  with Lockwaiting[1]

If transaction_state == ‘blocked’:
    If a[3] == ‘(‘:
dataitem = a[4]
    else:
dataitem = a[3]
// give resouce to first waiting transaction
update resouce holding list  with Lockwaiting[1]
```

```
If a[0] == 'e':
    // transaction state needs to be active for commit
    if transaction_state == 'active':
        update transaction state=='commited'
        release resources
        Itemholds = Items hold by committed transaction
        // Distribute Released dataitem over remaining transaction
        for j in Itemholds:
            state = retrieve state of Itemholds[ j ] (dataitem) from lock table
            if state == read:
                // delete aborted transaction id from resouce holding list
                if len(lockholdingtid)>1:     // lockingholdingtid = resouce holding list
                    z= lockholdingtid
                    for k in range(0,len(z)):
                    if z[k]==holdingtid:
                        str1=z[0:k-1]
                        str2=z[k+1:]
                else :
                    LockwaitingTid = Retrive all lock waiting Tids
                    // give resouce to first waiting transaction
                    update resouce holding list  with Lockwaiting[1]
                    update waiting list with Lockwaiting[2:]
            if state == write:
                LockwaitingTid = Retrive all lock waiting Tids
                if len(lockwaitingtid)>0:
                    newstate=lockwaitingtid[1]
                    if newstate=='r':newstate='read'
```

```
                        if newstate=='w':
                            newstate='write'
                        newlockholdingtransaction=lockwaitingtid[2]
                        newlockwaitingtransactions=lockwaitingtid[3:]
                    // give resouce to first waiting transaction
                    update resouce holding list  with Lockwaiting[1]

// after every iteration print transaction table and lock table
print(“transaction table”)
print(“lock table”)
// at the end
If transaction.state == ‘blocked’ || transaction.state == ‘aborted’:
    transaction.state = ‘active’
```