“锁管理器模拟程序”

设计报告

学号：PB15111662

姓名：李双利

计算机科学与技术学院

中国科学技术大学

2018 年 6 月

目 录

1 概述

1.1 设计目标

实现一个支持 S 锁、X 锁的简化的锁管理器模拟程序。该锁管理器从 stdin 获取事务的锁请求（实际中应来自事务管理子系统 以及并发控制子系统），然后将事务的锁信息输出到 stdout，同时为了满足查询需求，方面实时监测锁管理模拟器，增加打印锁表等用户查询请求命令。

输入命令的格式为：<request type> <transaction ID> <object>

其中<request type>是锁管理器接收的处理请求。输入命令包括 Start、End、XLock、SLock、Unlock 等五种命令和其他查询命令，分别表示开始事务、结束事务(abort or commit)、请求 X 锁、请求 S 锁、释放锁。<transaction ID>是事务标识，规定其为 0 到 255 之间的一个整数；<object>是请求加锁的数据库对象，以单个的大写字母来表示

1.2 需求说明

锁管理器模拟器的的主要功能需求如下:

1. 支持锁的读写命令
   a) SLock 对数据对象添加 S 锁，进行读操作；
   b) XLock 对数据对象添加 X 锁，进行写操作（当事件以对数据 A 加入 S 锁时，此时可升级为 X 锁）
   c) Unlock 对数据对象进行解锁（对数据的 X/S 锁解绑，并检查等待队列）
2. 支持事务的控制命令
   a) Start 开启相应的事件请求
   b) End 关闭相应的事件请求
3. 支持模拟器的打印查询、退出等命令
   a) Exit 退出模拟程序
   b) PrintAll 在终端打印执行情况以及输出的操作结果

c) PrintLock 在终端打印事务中的数据对象

d) PrintSLock 在终端打印特定数据对象上当前持有 SLock 锁的事务列表

1.3 本报告的主要贡献

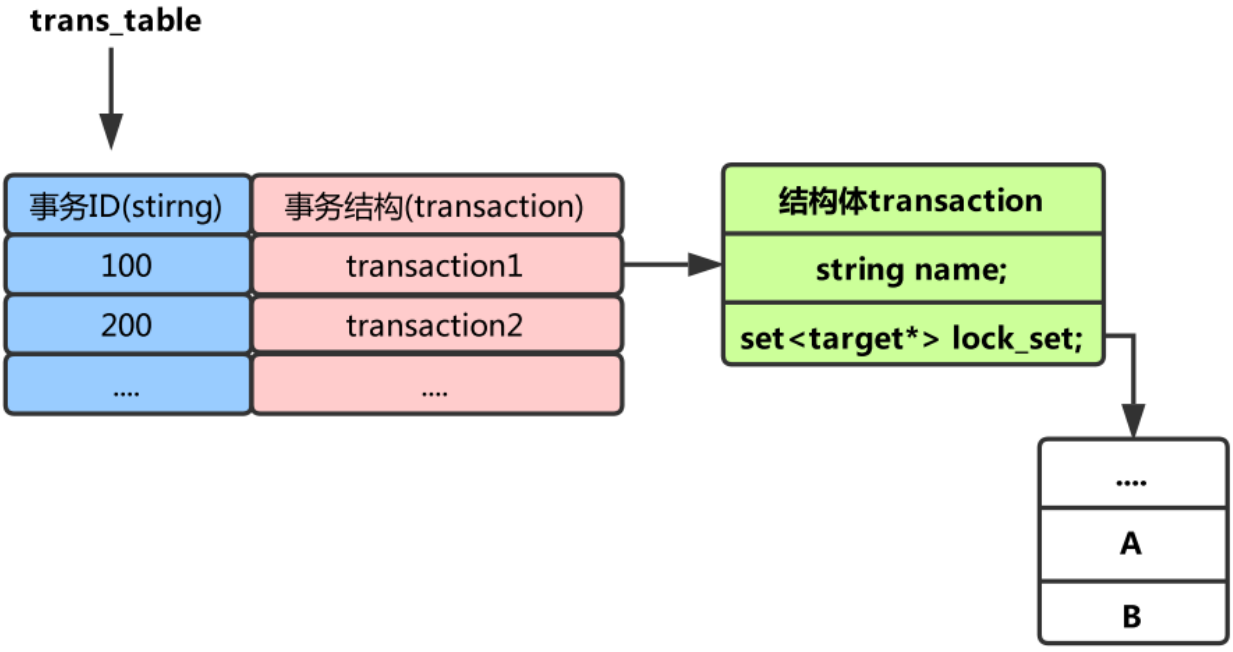
此设计报告主要是针对锁管理模拟程序提出的具体需求进行分析，从功能需求分析、命令设计说明到锁表类设计、具体代码实现等整个模拟的设计过程进行详细的阐述说明，并且针对完成的锁管理器进行功能测试以及容错性测试，从而对整个项目流程进行说明展示，并且进行项目系统总结，提出针对性的改进计划。

2 锁管理器总体设计

2.1 锁表设计

为方便锁管理模拟程序的维护和执行更新操作，为锁管理模拟建一个类 LockSim，在 LockSim 中有两个类成员变量，分别是事务表 trans_table 和数据对象表 target_table，锁管理模拟类在程序运行时维护这两个状态表，锁表的信息随着事务的执行动态更新，反映当前的锁状态。

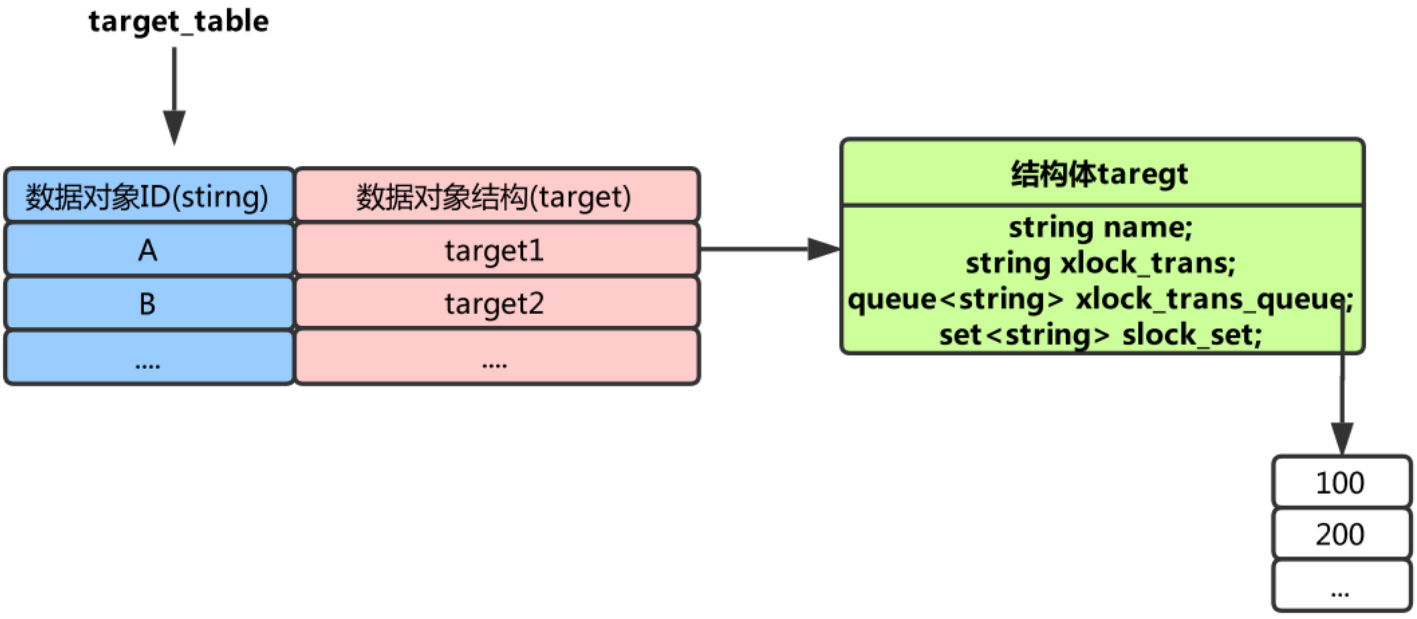
其数据结构如下图所示：



在上图中结构体 transaction 的成员分别为

name：事务名称（与 table 中事务 ID 一致）

lock_set：当前事务所操作的数据对象对象集合



在上图中结构体 target 的成员分别为

mObjectList：为 map 结构的对象树，可方便快速查找相应对象。

name：目标数据对象的名称（与 table 中数据对象 ID 一致）

xlock_trans：当前写操作的事务

xlock_trans_queue：写等待的事务队列

slock_set： 读操作的事务共享集，当 xlock_trans 不为空时，说明被写操作独占，变为共享等待队列

2.2 工作流程

此锁管理模拟程序的所有命令在功能需求部分已经提及，具体每一个命令的执行过程如下：

1. Start transID

   开始一个事务 transID。新建一个结构体 transaction 变量，加入到事务表中。

2. End transID

   结束事务 transID。此时需要释放该事务所持有的所有锁。如果有别的事务在等待 transID 上的锁，则需要根据给定的策略将锁授予等待的事务。同时从事务表中删除该事务。

3. SLock transID object

   事务 transID 请求 object 上的一个 S 锁。如果在数据对象表中没有找到该 object，那么新建一个 object 结构体加入到 target_table 中。该请求可能的输出有两种：如果锁请求被批准，则更新锁表信息同时输出“Lock granted”；如果锁请求不能被批准，则将该事务放入等待队列并输出“Waiting for lock (X-lock held by: <trans_ID>)”。

4. XLock transID object

   事务 transID 请求 object 上的一个 X 锁。如果在数据对象表中没有找到该 object，那么新建一个 object 结构体加入到 target_table 中。该请求同样有两种可能的执行结果，即批准或者等待。如果事务之前已经持有了 object 上的 S 锁，则该操作将 S 锁升级为 X 锁。

5. Unlock transID object

   事务 transID 释放对象 object 上的锁。释放锁之后，如果有别的事务在等待该锁，首先检测是否有等待的事务请求 Xlock 写锁，如果存在那么按照 FIFO 的策略授予等待的事务，如果没有 Xlock 等待，则将所有等待请求 Slock 的事务进行分配。

6. PrintAll

   在终端打印输出当前的所有命令执行结果。在模拟程序执行模拟每一条锁命令时会将返回结果保存在一个 vector 中，此时将所有的 input 和 output 格式化输出打印。

7. PrintLock transID

   打印事务 transID 的所有数据对象。在事务表中查找索引为“transID”的 transaction，找到之后遍历其 lock_set 输出所有数据对象。

8. PrintSLock object

   打印数据对象 object 的共享或者等待的事务集合。在数据对象表中查找索引为“object”的 target，找到之后遍历其 slock_set 输出所有事务。

   （说明：由于 XLock 的等待队列是使用 STL 中的 queue 实现，不支持遍历，所以模拟程序并未提供打印等待 XLock 事务的命令）。

9. Exit

   退出程序。

3 实现与测试

3.1 实现结果

1. Start object

```
1  if(inst[0] == "Start")
2  {
3      transaction* trans_p = new transaction();
4      trans_p->name = inst[1];
5      // insert new transaction into table
6      trans_table[inst[1]] = trans_p;
7      res += "Transaction " + inst[1] +" started\n";
8  }
```

```
>>> Start 100
Transaction 100 started
```

2. End transID

```
if(inst[0] == "End")
{
    res += "Transaction "+ inst[1] +" ended\n";
    // Release all locks
    set<target*>::iterator target_p;
    while(trans->second->lock_set.size() != 0)
    {
        target_p = trans->second->lock_set.begin();
        sim_unlock(trans->second, *target_p, res);
    }
    // remove transaction from table
    trans_table.erase(trans);
}
```

```
>>> End 100
Transaction 100 ended
```

3. SLock transID object

```
if(inst_op == "SLock")
{
    // No XLock, SLock is granted
    if(target_p->xlock_trans == "")
    {
        target_p->slock_set.insert(trans_p->name);
        trans_p->lock_set.insert(target_p);
        res += "S-Lock granted\n";
    }
    // XLock exists, SLock waiting
    else
    {
        target_p->slock_set.insert(trans_p->name);
        res += "Waiting for lock (X-lock held by: " + target_p->xlock_trans +")\n";
    }
}
```

```
>>> SLock 100 A
S-Lock granted
```

4. XLock transID object

```
else if(inst_op == "XLock")
```

```
2   {
3       // No XLock
4       if(target_p->xlock_trans == "")
5       {
6           int shareNum = target_p->slock_set.size();
7           if(shareNum > 1)
8           {
9               string sTemp = "";
10              for(set<string>::iterator it_index = target_p->slock_set.begin();
11                  it_index != target_p->slock_set.end(); it_index++)
12              {
13                  sTemp += " " + *it_index;
14              }
15              target_p->xlock_trans_queue.push(trans_p->name);
16              res += "Waiting for lock (S-lock held by:" + sTemp + "\n";
17          }
18
19          else if(shareNum == 1)
20          {
21              // update
22              if(*(target_p->slock_set.begin()) == trans_p->name)
23              {
24                  target_p->xlock_trans = trans_p->name;
25                  target_p->slock_set.clear();
26                  res += "Upgrade to XLock granted\n";
27              }
28              else
29              {
30                  target_p->xlock_trans_queue.push(trans_p->name);
31                  res += "Waiting for lock (S-lock held by:" + *(target_p->slock_set.b
    egin()) + ")\n";
32              }
33          }
34
35          else if(shareNum == 0)
36          {
37              target_p->xlock_trans = trans_p->name;
38              trans_p->lock_set.insert(target_p);
39              res += "XLock granted\n";
40          }
41      }
42      // XLock exists, XLock waiting
43      else
44      {
```

```
45          target_p->xlock_trans_queue.push(trans_p->name);
46          res += "Waiting for lock (X-lock held by: "+ target_p->xlock_trans +")\n";
47      }
48  }
```

```
>>> XLock 200 B
XLock granted
```

5. Unlock transID object

```
1   void sim_unlock(transaction* trans_p, target* target_p, string& res)
2       {
3           if(target_p->xlock_trans != "")
4           {
5               if (target_p->xlock_trans == trans_p->name)
6               {
7                   target_p->xlock_trans = "";
8                   trans_p->lock_set.erase(target_p);
9                   res += "Lock released\n";
10              }
11              else
12                  res += "I can not find the transaction.\n";
13          }
14          else
15          {
16              set<string>::iterator shareIndex = target_p->slock_set.find(trans_p->
    name            );
17              if(shareIndex != target_p->slock_set.end())
18              {
19                  target_p->slock_set.erase(shareIndex);
20                  trans_p->lock_set.erase(target_p);
21                  res += "Lock released\n";
22              }
23              else
24                  res += "I can not find the transaction.\n";
25          }
26          // FIFO and XLock takes precedence
27          int flag = 0;
28          if(target_p->xlock_trans_queue.size() != 0)
29          {
30              string s = "";
31              while(!target_p->xlock_trans_queue.empty())
32              {
33                  s = target_p->xlock_trans_queue.front();
34                  target_p->xlock_trans_queue.pop();
```

```
35                  if(trans_table.find(s) != trans_table.end())
36                      break;
37                  s = "";
38              }
39              if(s != "")
40              {
41                  target_p->xlock_trans = s;
42                  res += "X-Lock on "+ target_p->name +" granted to "+ target_p->xl
    ock_trans +"\n";
43                  flag = 1;
44              }
45          }
46          if(target_p->slock_set.size() != 0 && flag == 0)
47          {
48              string temp = "";
49              for(set<string>::iterator it_index = target_p->slock_set.begin();
50                  it_index != target_p->slock_set.end(); it_index++)
51              {
52                  temp += " " + *it_index;
53              }
54              res += "S-Lock on "+ target_p->name +" granted to "+ temp +"\n";
55          }
56      }
57  };
```

```
>>> Unlock 100 A
Lock released
```

6. PrintAll

```
1   void print_table()
2   {
3       for(int i = 0;i < output.size();i++)
4       {
5           cout << "[" << i << "]" << left << setw(12) << input[i];
6
7           string temp = "";
8           int k = 0;
9           for(int j = 0;j < output[i].size();j++)
10          {
11              if(output[i][j] == '\n')
12              {
13                  if(k == 0)
14                      cout << "   " << temp << endl;
15                  else
```

```
                    cout << "                    " << temp << endl;
                k = 1;
                temp = "";
                continue;
            }
            temp += output[i][j];
        }
    }
}
```

```
>>> PrintAll
[0]Start 100     Transaction 100 started
[1]SLock 100 A   S-Lock granted
[2]Start 200     Transaction 200 started
[3]XLock 200 B   XLock granted
[4]Unlock 100 A  Lock released
[5]End 100       Transaction 100 ended
```

7. PrintLock transID

```
if(inst[0] == "PrintLock")
{
    map<string, transaction*>::iterator trans = sim.trans_table.find(inst[1]);
       if(trans == sim.trans_table.end())
       {
           cout << "[ERROR]Transaction " + inst[1] + " doesn't exist\n";
        return;
       }

       set<target*>::iterator target_p = trans->second->lock_set.begin();
       cout << "[INFO]Transaction " << inst[1] << " all targets:";
    while(target_p != trans->second->lock_set.end())
    {
        cout << (*target_p)->name << "  ";
        ++target_p;
    }
    cout << endl;
}
```

```
>>> PrintLock 200
[INFO]Transaction 200 all targets:B
```

8. PrintSLock object

```
if(inst[0] == "PrintSLock")
{
    map<string, target*>::iterator tar = sim.target_table.find(inst[1]);
       if(tar == sim.target_table.end())
       {
           cout << "[ERROR]Target " + inst[1] + " doesn't exist\n";
        return;
       }
        set<string>::iterator slock_p = tar->second->slock_set.begin();
        cout << "[INFO]Target " << inst[1] << " all Transactions who share/wait this SLock:";
       while(slock_p != tar->second->slock_set.end())
       {
           cout << *slock_p << "  ";
           ++slock_p;
       }
       cout << endl;
}
```

```
>>> PrintSLock A
[INFO]Target A all Transactions who share/wait this SLock:100
```

3.2 测试结果

测试用例：

```
1. Start 100
2. Start 200
3. SLock 100 A
4. XLock 200 A
5. PrintSLock A
6. Unlock 100 A
7. XLock 100 B
8. XLock 200 B
9. PrintLock 100
10. XLock 100 A
11. End 100
12. Unlock 200 A
13. End 200
14. PrintAll
```

测试结果

```
shuangli@DESKTOP-JFO0682:/mnt/f/course/DataBase/lab4$ ./sim
[info]Please input instructions, shut down with "exit" instruction>>> Start 100
Transaction 100 started
>>> Start 200
Transaction 200 started
>>> SLock 100 A
S-Lock granted
>>> XLock 200 A
Waiting for lock (S-lock held by:100)
>>> PrintSLock A
[INFO]Target A all Transactions who share/wait this SLock:100
>>> Unlock 100 A
Lock released
X-Lock on A granted to 200
>>> XLock 100 B
XLock granted
>>> XLock 200 B
Waiting for lock (X-lock held by: 100)
>>> PrintLock 100
[INFO]Transaction 100 all targets:B
>>> XLock 100 A
Waiting for lock (X-lock held by: 200)
>>> End 100
Transaction 100 ended
Lock released
X-Lock on B granted to 200
>>> Unlock 200 A
Lock released
>>> End 200
Transaction 200 ended
>>> PrintAll
[0]Start 100     Transaction 100 started
[1]Start 200     Transaction 200 started
[2]SLock 100 A   S-Lock granted
[3]XLock 200 A   Waiting for lock (S-lock held by:100)
[4]Unlock 100 A  Lock released
                 X-Lock on A granted to 200
[5]XLock 100 B   XLock granted
[6]XLock 200 B   Waiting for lock (X-lock held by: 100)
[7]XLock 100 A   Waiting for lock (X-lock held by: 200)
[8]End 100       Transaction 100 ended
                 Lock released
                 X-Lock on B granted to 200
[9]Unlock 200 A  Lock released
[10]End 200       Transaction 200 ended
>>> Exit
```

4 总结与讨论

通过此次锁管理模拟程序的实现，我对 Slock 读操作锁和 XLock 写操作锁有了进一步的理解，尽管实现的只是一个模拟器，并不是数据库真正运行的场景，但是却充分表现了数据库在工作时为了保证事务的正常进行所采用的读写锁机制带来的作用，这样才能使得各个事务在交叉进行时不会出错。

在进行实验时，应当考虑到的是一个事务结束时必须要事务中的锁都释放，否则会造成后序事物的错误执行。该模拟器可以实现对模拟过程中的状态查询，但是仍有不足之处，比如未能完整的打印一个数据对象所有的对应事务列表，以及在执行中出现死锁的提示和处理等，这些是锁管理模拟程序需要改进的方向。