



南開大學
Nankai University

南 开 大 学

计 算 机 学 院

数据库实验报告

lab4

姓名：林帆

学号：2113839

2023 年 5 月 25 日

目录

一、 实验概述	1
二、 exercise1&2	1
(一) 锁类 PageLock	1
(二) 锁管理器类 LockManager	1
1. acquiredLock()	1
2. releaseLock()	2
3. completeTranslation	3
(三) BufferPool	3
(四) HeapFile	3
三、 exercise3	3
四、 exercise4	4
五、 exercise5	4
六、 代码提交记录	5
七、 实验总结	5

一、 实验概述

在本次实验中，需要完成事务和锁的相关类和函数，并对前三个 lab 中需要用到事务和锁的函数进行局部修改。过程中需要保证锁的原子性、正确性、隔离性和持久性。

二、 exercise1&2

在 exercise1 中首先完成了锁和锁管理器类的创建。

(一) 锁类 PageLock

锁类对象中主要有四个重要变量——当前锁对应事务的 Id——TransactionId，锁的类型——type，以及类型对应的两个 int 型静态变量的表示——SHARE 表示共享锁，EXCLUSIVE 表示排他锁。

除了构造函数以外，类中还有获取事务 Id/获取锁的类型/设置锁的类型的函数，分别为 getTid(), getType() 和 public void setType(int type)。

(二) 锁管理器类 LockManager

锁管理器类中存储了各事务和相应各锁的对应关系，可看作事务与锁的桥梁，通过类中函数实现事务的获取锁/解锁/升级锁等操作。

事务和锁的对应关系存储在类型为 ConcurrentHashMap<PageId, ConcurrentHashMap<TransactionId, PageLock>> 的哈希表 lockMap 中。从中也映射出了缓存中的页、事务和锁的对应关系——一页-> 多个事务；一个事务-> 一个锁。

1. acquiredLock()

在 LockManager 中，最重要的函数是获取锁的函数 **acquiredLock()**，传入函数的参数有查找页面的 PageId，事务的 tid 和获取锁的请求类型 requiredType（请求读锁/写锁，对应共享锁/排他锁）。返回类型为请求是否成功的 bool 类型值。实现思路如下所示：

先通过判断 lockMap 中是否含有参数 pageId 的键值，来判断该页是否有锁。

若**无锁**，则直接加锁即可：新建锁并与 tid 构成键值对，存入内层哈希中，然后更新内层哈希表，再更新外层哈希表。

加锁过程

```
1  if(lockMap.get(pageId)==null) {
2      //创建锁（自定义类）——tid和pageLock对应
3      PageLock pageLock=new PageLock(tid,requiredType);
4      ConcurrentHashMap<TransactionId,PageLock> tpMap=new ConcurrentHashMap
        <TransactionId,PageLock>();
5      //放入该页的锁集合
6      tpMap.put(tid,pageLock);
7      lockMap.put(pageId,tpMap);
8      return true;
9  }
```

若该页**有锁**，则需考虑下列两种情况——该页上是否有当前事务 tid 之前上的锁。需要先找出当前 pageId 下的哈希表 tpMap，亦即前页下的事务和锁的对应关系集合。寻找 tpMap 中是否有事务键值 tid。随后判断出属于哪种情况：

case1: 不存在该页之前上的锁:

判断当前页的锁的个数。

若**个数 >1**，说明该页面此时可允许多个锁，即都是共享锁，因而可直接上锁，上锁步骤同上。

若**个数 =1**，需要判断唯一的旧锁的类型，以及当前的 requiredType。4 种排列组合方式中，只有两个均为 SHARE 共享类型时，才能上锁并返回 TRUE。

旧锁	当前锁	返回
EXCLUSIVE	EXCLUSIVE	FALSE
EXCLUSIVE	SHARE	FALSE
SHARE	EXCLUSIVE	FALSE
SHARE	SHARE	TRUE

表 1: 唯一的旧锁和当前锁的四种组合情况

case2: 存在该页之前上的锁在内层哈希表 tpMap 中，通过 get() 函数找到 tid 之前上的锁，并判断旧锁类型。

若**旧锁为排他锁**，本来就达到写锁要求，直接返回 TRUE 即可。

若**旧锁为共享锁**，需要判断当前锁的类型。为共享锁时直接返回，否则判断锁的个数——只有唯一一个锁（即事务 tid 对应的旧锁）时，将旧锁升级为写锁；个数大于 1 证明其他锁的存在，而当前锁本身是排他锁，不行，返回 false。

2. releaseLock()

releaseLock() 函数完成了当前事务在当前页上的锁的释放，在释放前，首先需要通过查询哈希表，判断该页是否有锁，以及是否有当前事务的锁，然后在内层哈希表将此 tid 对应键值对 remove()。注意，若释放后该页没有锁了，需要在外层哈希表中将该 pageId 的键值对删除。

releaseLock() 函数

```

1 //先找到该页的锁集合
2 ConcurrentHashMap<TransactionId, PageLock> locks=lockMap.get(pageId);
3 //判断锁集合是否为空
4 if(locks==null)
5     return false; //为空肯定不行
6 //判断锁集合中是否有当前事务的
7 if(locks.get(tid)==null)
8     return false;
9 //存在，在哈希表中删除
10 locks.remove(tid);
11 //若释放锁后此时页没有锁了，需要将该页在哈希表中删除
12 if(locks.size()==0)
13     lockMap.remove(pageId);
14 return true;

```

3. completeTranslation

而类中的 completeTranslation() 函数则通过遍历内存中所有页，并执行 releaseLock(), 完成了事务在内存中所有的锁的释放。

(三) BufferPool

在 BufferPool 中，首先需要声明一个锁管理器 lockManager，并完成 releasePage() 和 hold-sLock() 函数——只需在函数内部调用 lockManager 的相应函数即可。

在 getPageSize() 函数中，在事务读取 pageStore 中的 page 前，需要添加对能否读取的状态的判断，并在进入读取状态时，为该页面加锁，保证事务的原子性。

getPageSize() 读取前判断

```

1 //先判断需要获取的锁的类型
2 int lockType=perm==Permissions.READ_ONLY?PageLock.SHARE:PageLock.
  EXCLUSIVE;
3 //循环判断是否可以加锁
4 boolean isAcquired=false;//初始化不能
5 while(!isAcquired)
6     isAcquired=lockManager.acquiredLock(pid,tid,lockType);
7 //跳出循环表明可以加锁了，否则一直忙等待

```

(四) HeapFile

在 insertTuple() 函数中，遍历所有 page 后发现没有空余 slot 时，由于后续需要花费较大时长新建页面和插入，为了提高系统整体的效率，在创建页面前，释放当前事务的锁，让其他事务继续进行。执行下列语句即可：

没有空余 slot 时

```

1 else
2     Database.getBufferPool().releasePage(tid,tempId);

```

三、 exercise3

在 BufferPool 中的 evictPage() 函数，原本使用最近最少使用的 LRU 原则，将链表中的第一页摘下，丢弃后作为新页利用。在本次实验中，需要保证未完成事务的完整性，即 commit 前，内存中的对应的脏页不能丢弃。因而将原则修改为，找到一个非脏页丢弃即可。

evictPage() 函数

```

1 //脏页代表事务未完成，不能直接丢弃
2 for(Page p:pageStore.values()){
3     if(p.isDirty()==null){
4         //不是脏页，丢弃
5         discardPage(p.getId());
6         return;
7     }
8     //是脏页则寻找下一个

```

```

9         else
10         continue;
11     }
12     //到这里没返回说明全都是脏页，没有可以丢弃的
13     throw new DbException("All Page Are Dirty Page");

```

四、 exercise4

transactionComplete() 函数的实现思路是,若需要直接 commit 事务,则调用 flushPages() 函数刷新 tid 事务修改的所有页面即可;若需要 roll back, 则遍历 tid 的所有脏页, 将磁盘中该页的原有数据重新覆盖内存。最终事务完成, 调用锁管理器中的 completeTranslation() 函数即可。

transactionComplete() 函数

```

1     //若提交，刷新页面→磁盘
2     if(commit) {
3         flushPages(tid);
4     }
5     else {
6         //若回滚，还原事务对内存的所有改变
7         //遍历所有页面
8         for (Page p : pageStore.values()) {
9             //找到该事务的脏页
10            if (tid.equals(p.isDirty())) {
11                //获取磁盘中的原有状态
12                int tableId = p.getId().getTableId();
13                DbFile table = Database.getCatalog().getDatabaseFile(tableId);
14                ;
15                Page pageInDisk = table.readPage(p.getId());
16                //写回内存
17                pageStore.put(p.getId().hashCode(), pageInDisk);
18            }
19        }
20        //事务完成
21        lockManager.completeTranslation(tid);

```

五、 exercise5

exercise5 需要解决死锁的问题, 而 getPageSize() 函数中有些事务的执行时间过长, 导致 while 循环始终无法跳出, 后一个锁始终不能加上, 而在前期测试中也发现部分测试函数因此执行时间过长。因而修改后加入时间检测, 时间过长则抛出异常。修改后的函数如下所示:

修改后的 getPageSize() 函数

```

1     //先判断需要获取的锁的类型

```

```

2      int lockType=perm==Permissions.READ_ONLY?PageLock.SHARE:PageLock.
        EXCLUSIVE;
3      // 计算超时时间 (设置为 500 ms)
4      long startTime = System.currentTimeMillis();
5      //循环判断是否可以加锁
6      boolean isAcquired=false;//初始化不能
7      while(!isAcquired){
8          isAcquired=lockManager.acquiredLock(pid,tid,lockType);
9          long now = System.currentTimeMillis();
10         // 如果超过 500 ms没有获取就抛出异常
11         if(now - startTime > 500){
12             // 放弃当前事务
13             throw new TransactionAbortedException();
14         }
15     }
16     //跳出循环表明可以加锁了, 否则一直忙等待

```

六、 代码提交记录

24 May, 2023 - 3 commits

sail-b, Lab4 exercise5
sail-brother authored 20 hours ago

sail-b, Lab4 exercise4
sail-brother authored 1 day ago

sail-b, Lab4 exercise4
sail-brother authored 1 day ago

23 May, 2023 - 2 commits

sail-b, Lab4 exercise3
sail-brother authored 1 day ago

sail-b, Lab4 exercise2
sail-brother authored 2 days ago

22 May, 2023 - 1 commit

sail-b, Lab4 exercise1
sail-brother authored 2 days ago

图 1: gitlab 提交记录

七、 实验总结

通过本次实验, 对数据库中的事务和锁, 及其相关机制和原则有了更深入的了解, 并对 java 语言有了进一步的掌握