

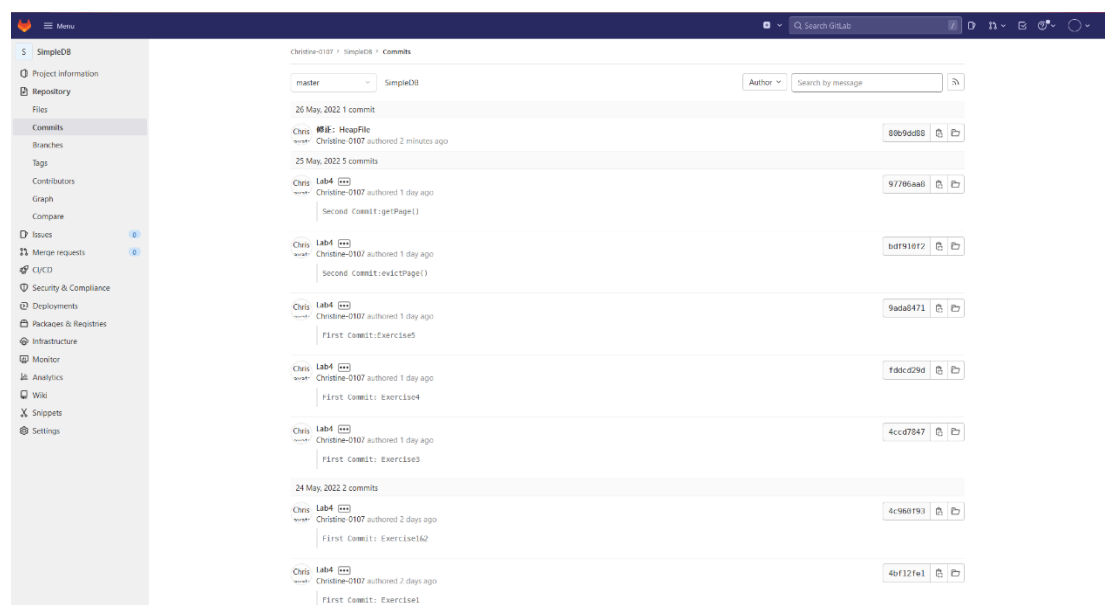
SimpleDB Lab4 实验报告

2010239 李思凡

一、Lab4 概述

通过 lab3, 将主要实现一个基于锁操作的事务系统, 需要在适当的位置加锁、解锁, 管理事务的锁的添加和释放。在本次实验中需要在 `BufferPool` 类中完善基于两段锁协议的管理锁的机制; 完善 `evictPage()` 方法, 使置换 `BufferPool` 中的页时满足 NO STEAL 策略; 实现事务的提交和回滚功能; 实现死锁的发现, 出现死锁时抛出异常。

二、Git Commit History



三、Granting Locks

实现 `BufferPool.java` 中的 `releasePage()` 和 `holdsLock()` 方法, 完善 `getPage()` 方法。从而实现 `page` 级别的锁管理器, 实现在 `getPage()` 获取页之前进行加锁, 判断某个事务是否持有某一页上的锁, 以及释放某个事务对某一页的锁。

需要满足的锁为 `shared` 和 `exclusive` 锁, 规则如下:

- (1) 在事务读取一个对象之前, 需要有一个共享锁 (读锁);
- (2) 在事务可以写一个对象前, 需要有一个排他锁 (写锁);
- (3) 多个事务可以在一个对象上共用一个共享锁;

(4) 只有一个事务能对一个对象使用排他锁；

(5) 如果一个对象上只持有对一个事务的共享锁，可以升级为对该对象的排他锁。

1.设计思路：

(1) 由于需要实现授予锁的方法，锁又有两种类型，因此添加一个有关锁的辅助类 `Lock`。在 `Lock` 里有两个静态属性标识锁的类型，`SHARE` 代替整数 0 表示共享锁，`EXCLUSIVE` 代替整数 1 表示排他锁。字段 `TransactionId` 对象 `tid` 表示持有该锁的事务 ID；字段整型对象 `type` 表示这个锁的类型，0 表示共享锁，1 表示排他锁。编写构造函数赋初值，`get` 方法获取该锁的 `tid` 和 `type`，以及 `set` 方法重置锁的类型。

(2) 需要实现对锁的管理，因此编写辅助类 `LockManager`，其中包括获取锁的方法 `acquireLock()`，判断某个事务是否持有某一页上的锁 `holdsLock()`，和释放某事务在某一页上锁的方法 `releasePage()` 方法。具体实现见重难点。

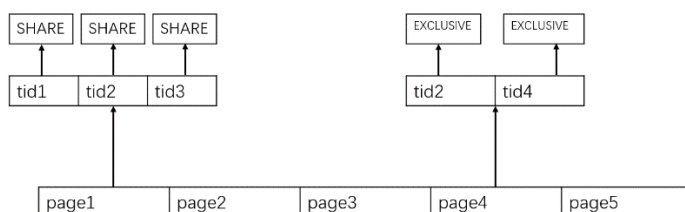
(3) 在 `BufferPool` 类中添加属性 `LockManager` 类的对象 `lockManager`，并在构造函数中初始化，通过这个对象来管理锁。

(4) 在 `getPage()` 方法中，通过传进来的 `Permissions` 参数判断应该为这个页授予的锁的类型。若为 `RAED_ONLY` 只读类型，锁的类型赋为 `Lock.SHARE`，否则赋为 `Lock.EXCLUSIVE`。

(5) 在 `releasePage()` 方法中，直接通过 `lockManager` 对象调用辅助类中 `releasePage()` 方法；在 `holdsLock()` 方法中，也直接通过 `lockManager` 调用辅助类中的 `holdsLock()` 方法并返回。

2.重难点：实现辅助类 `LockManager`

(1) 由于要实现基于页的锁管理机制，需要能够通过每一页对应到该页上的事务，进而对应到该事务上的锁。因此设置两级的 `ConcurrentHashMap`，由 `pageId` 查找 `transactionId`，再由 `transactionId` 查找 `lock`，如下图所示：



声明该 map 为 lockMap。

(2) 实现 acquireLock()方法。加锁时需要分情况判断能够加锁成功，假设当前页面为 page，事务为 t，ID 相应为 pid 和 tid。当事务 tid 向页 pid 申请锁时，能否获取成功分为以下多种情况：首先分为 pid 上没有锁和有锁。如果 pid 上没有锁，无论事务 tid 请求的是读锁还是写锁，都可以直接新建一个锁，将它放入 map 中，然后返回获取成功。如果 pid 上有锁，则需要分为有事务 tid 的锁和没有事务 tid 的锁两种情况讨论。

若有 tid 的锁，且为事务 tid 的读锁。当请求的锁的类型也为读锁时，可以直接获取这个已有的读锁，返回成功。当请求的锁的类型是写锁时，如果页 pid 只有一个事务 tid 的读锁，则根据规则可将它升级为写锁，也可直接获取；如果页 pid 上还有其他事务的读锁，为了避免发生死锁，直接抛出 TransactionAborted 异常，返回失败。若有 tid 的锁，且为事务 tid 的写锁，则无论请求的是读锁还是写锁，都可以直接获取并返回成功。

若有锁但没有事务 tid 的锁，需要考虑页面 pid 上锁的个数。若页面 pid 上锁的个数大于 1，当请求为读锁时，可以直接获取，当请求为写锁时，需要等待一段时间，等其他锁释放。若页面 pid 上锁的个数等于 1，若该锁为读锁，当请求的锁也为读锁时，可以直接获取，当请求的锁为写锁时，需要等待一段时间。若该锁为写锁，无法获取，必须等待。

(3) 实现 holdsLock()方法，判断事务 tid 是否持有 pid 上的锁。根据建立的 lockMap，若 lockMap 中 pid 对应的值为空，说明该页上没有锁，直接返回 false。否则就判断 pid 和 tid 确定的锁是否为空，若为空，代表 tid 不持有 pid 上的锁，返回 false，否则返回 true。

(4) 实现 releasePage()方法，释放事务 tid 在某一页上持有的所有锁。调用 holdsLock 方法判断给出的 tid,pid 是否有锁，若没有直接返回 false，若有则调用 map 的 remove 方法移除 tid 对应的锁，若由 tid 到 pid 的 map 大小变为 0，则移除 lockMap 中 pid 对应的值。移除后唤醒所有等待加锁的线程执行操作。

3.未改动 API 和测试代码。

四、Lock Lifetime

根据两段锁协议，在访问数据库元素之前，相应的事务必须获得相应类型的锁，在事务提交之后才可以解锁。在本练习中，需要确认并完善方法，使得操作符合两段锁协议。

1. 设计思路：

(1) 实现在访问数据库元素之前，为相应的事务添加相应类型的锁。在上一个练习中，已经实现了在 `getPage()` 方法里，获取页之前先获取锁。因此，`HeapFile` 中的插入删除操作获取页时，需要通过 `BufferPool` 中的 `getPage()` 方法，传入的访问权限为可读可写，以便先获得加锁。

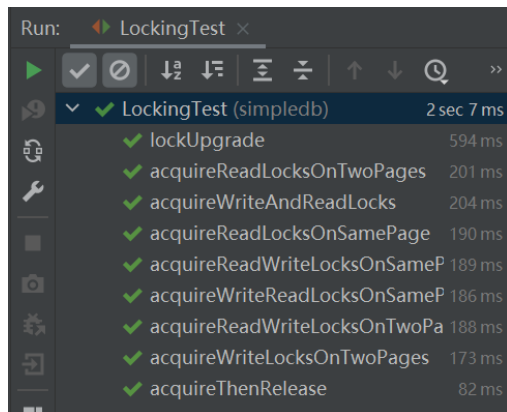
(2) 加锁后的页面需要及时释放锁，否则会导致操作无法正常进行。在 `HeapFile` 的插入方法中，需要找到有空 slot 的页进行插入，但若某一页没有空 slot，应该及时解除该页上的锁，调用 `BufferPool` 中的 `releasePage` 方法。

2. 重难点：

需要了解两段锁协议，并且要注意在某一页遍历了但未找到空 slot 的时候及时释放上面的锁，此时虽然该事务仍未提交，但之后不会用到这一页。

3. 未改动 API 和测试代码。

4. 测试结果：



The screenshot shows the Run console of a Java IDE. The test suite 'LockingTest (simplifiedb)' has passed with a total time of 2 seconds and 7 milliseconds. The following table lists the individual test methods and their execution times:

Test Method	Execution Time
lockUpgrade	594 ms
acquireReadLocksOnTwoPages	201 ms
acquireWriteAndReadLocks	204 ms
acquireReadLocksOnSamePage	190 ms
acquireReadWriteLocksOnSameP	189 ms
acquireWriteReadLocksOnSameP	186 ms
acquireReadWriteLocksOnTwoPa	188 ms
acquireWriteLocksOnTwoPages	173 ms
acquireThenRelease	82 ms

五、实现 NO STEAL 策略

实现 NO STEAL 策略，也就是一个事务中对于数据库元素的修改，必须在这个事务提交后才能写入磁盘。在本练习中，需要完善 `evictPage` 方法，使其能够满足 NO STEAL 策略。

1. 设计思路：

事务没有提交之前，不能将该事务对数据库元素的修改写入磁盘。因此在 `BufferPool` 类中的 `evictPage()`方法中置换缓冲池中的页时，不能将脏页换走。如果当前访问的是脏页，则不淘汰、继续找下一个；如果 `BufferPool` 中当前所有页都是脏页，则都不能置换，抛出异常。

2.未改动 API 和测试代码。

六、Transactions

实现对事务完成时的处理，成功要提交，失败要回滚。

1.设计思路：

(1) 实现 `transactionComplete` 算法，其中传入参数 `tid` 和标识是提交还是回滚的布尔值。当事务成功可以提交时，调用 `flushTidPage()`方法将事务 `tid` 对应的脏页写入磁盘；如果失败，调用 `restorePage()`方法，会从磁盘中获取脏页对应的原数据页。对脏页处理完毕后，调用 `releasePage` 方法释放事务 `tid` 在所有数据页加的锁。

(2) `BufferPool` 中还有一个与 `transactionComplete` 方法同名的方法，只有 1 个参数 `tid`，它表示事务一定完成。通过调用 `transactionComplete(tid, true)`方法实现。

2.重难点：

(1) `flushTidPage()`方法，将事务 `tid` 对应的脏页写入磁盘。遍历 `buffer` 中所有 `pageId`，找到它们对应的 `page`，调用 `isDirty` 方法判断是不是 `tid` 对应的脏页，如果是则调用 `flushPage` 方法。

(2) `restorePage()`方法，通过 `pid` 找到该页对应的 `tableId`，再通过 `tableId` 找到对应的数据库文件 `file`，调用 `file` 的 `readPage` 方法获取到原始的页，将它放到 `buffer` 中。

3.未改动 API 和测试代码。

4.测试结果：

TransactionTest unit test:

Run: TransactionTest (1) ×

▶

✓

⏸

⏮

⏭

⏪

⏩

⏴

⏵

🔍

»

📁

✓

TransactionTest (simplifiedb)

654 ms

🔄

✓

abortTransaction

464 ms

🔧

✓

commitTransaction

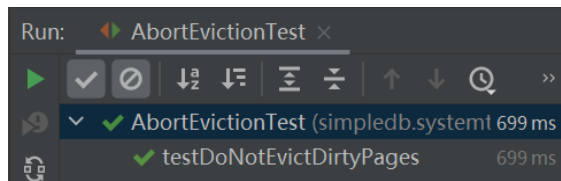
96 ms

✓

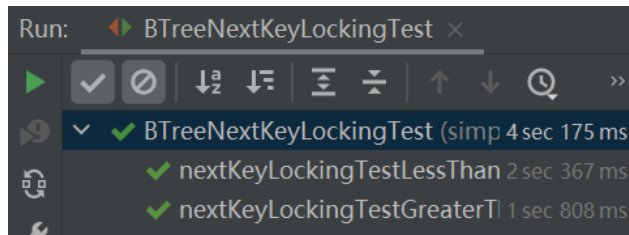
attemptTransactionTwice

94 ms

AbortEvictionTest system test:



BtreeNextKeyLockingTest:



七、DeadLock and Aborts

需要检测死锁的发生，当认为发生死锁时，需要将事务中断。

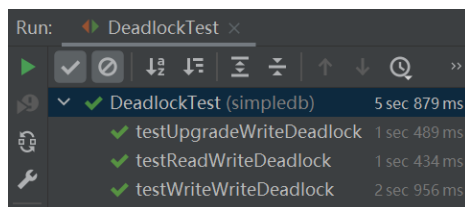
1.设计思路:

本实验中采取超时的方式检测死锁是否发生。在 `getPage()` 方法里设置 `while` 循环，调用获取锁的方法之前设置开始时间，然后调用获取锁的方法，如果成功获取锁，就退出循环，直接进行获取页的操作，再之后设置结束时间。如果结束时间-开始时间大于某一数，说明获取锁的操作执行时间过长，可能发生死锁，抛出 `TransactionAbortedException` 异常。

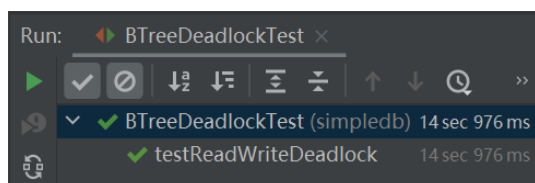
2.未改动 API 和测试代码。

3.测试结果:

DeadLockTest:



BtreeDeadLock:



TransactionTest system test:

Run: TransactionTest x

✓ [stop] [sort] [filter] [compare] [refresh] [search] [expand]

TransactionTest (simplifiedb.syst 36 sec 530 ms)

- ✓ testFiveThreads 6 sec 152 ms
- ✓ testSingleThread 25 ms
- ✓ testTenThreads 29 sec 782 ms
- ✓ testTwoThreads 433 ms
- ✓ testAllDirtyFails 138 ms