simpleDB实验四实验报告

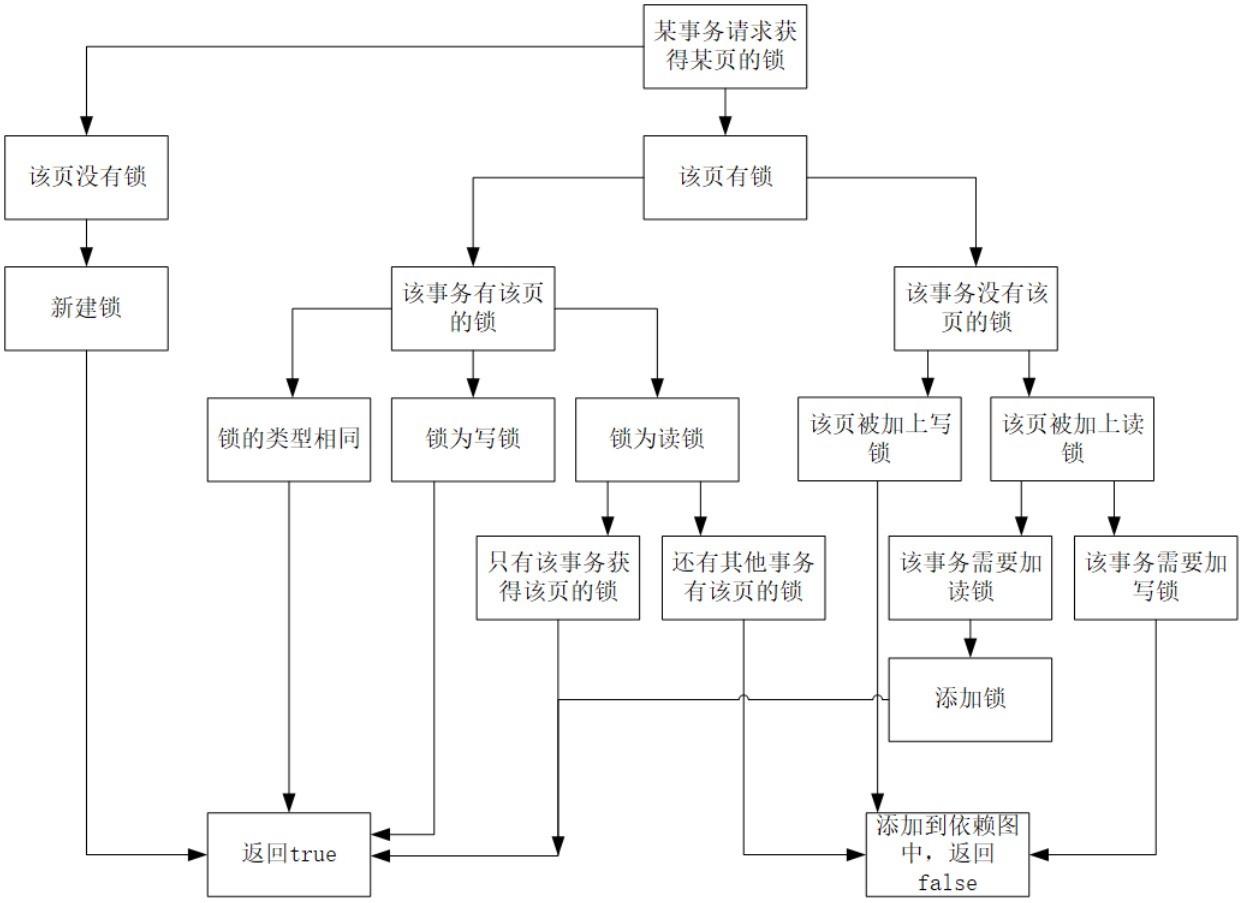
1. exercise1

1.1 设计思路

- 新增内部类`Lock`，存储该锁对应的的事务和类型
- 新增内部类`PageLockManager`，管理锁的获取、释放和查询

1.2 重难点

- `PageLockManager`类中存在成员变量`ConcurrentHashMap<PageId,Vector<Lock>> lockMap`，用于存储页对应的锁。
- `PageLockManager`类提供的获取、释放和查询方法均为同步方法。
- 获取锁要考虑如下方面：
    - 在一个transaction能够读资源前，必须获得资源的 shared lock。
    - 在一个transaction能够写资源前，必须获得资源的 exclusive locks。
    - 多个transaction可以同时获得资源的 shared lock。
    - 同一时间，只有一个transaction可以获得资源的 exclusive locks。
    - 如果transaction t 是唯一一个拥有资源 o shared lock 的transaction，t可以将它对资源o的锁更新为exclusive locks。
- `releasePage`方法直接调用`PageLockManager`类中的`releaseLock`方法即可。



2. exercise2

2.1 设计思路

- 在`BufferPool`类的`getPage`方法中，判断读写权限并尝试获取锁，只有获取到锁才返回页面
- 在`HeapFile`类的`insertTuple`方法中，将没有多余槽位的页面的锁给释放
- 在`BTreeFile`类中，将读取的内部节点的权限改为`READ_ONLY`

2.2 重难点

- 在`BufferPool`类的`getPage`方法中，维护一个变量`lockAcquired`，调用`lockManager.acquireLock`方法并将返回值赋给`lockAcquired`，如果一直返回`false`就循环调用，直到返回`true`，退出循环并返回页面。
- 在`HeapFile`类的`insertTuple`方法中，如果获取的页面槽位已满，则调用`Database.getBufferPool().releasePage`方法将刚刚加上的锁释放。

3. exercise3

3.1 设计思路

- 找到第一个不是脏页的页，移除出缓冲池，如果所有页都是脏页，抛出异常

3.2重难点

- 遍历`pageStore`中的页面，如果是脏页则继续找下一个，如果不是脏页就移除。如果遍历完之后也没有找到不是脏页的页面就抛出异常。

4. exercise4

4.1 设计思路

- `transactionComplete(TransactionId tid)`调用`transactionComplete(TransactionId tid, boolean commit)`
- 如果事务提交，那么将`BufferPool`中该事务修改的页面存入磁盘，如果事务没有提交，从磁盘中读取该文件并更新`BufferPool`

4.2 重难点

- `flushPages`方法：遍历BufferPool中的所有页面，如果该页面的`isDirty()`等于事务的`tid`，则表明该页最后被该事务修改，需要更新到磁盘中。
- `transactionComplete`方法：如果`commit`为`true`，则调用`flushPages`方法，否则遍历所有页面，如果该页面的`isDirty()`等于事务的`tid`，则从磁盘中读取该页面在修改之前的值，并存入BufferPool中。最后遍历页面，判断该页面是否被该事务加锁，如果加锁则释放该锁。

5. exercise5

5.1 设计思路

- 在`BufferPool`中，需要在`readPage`方法中，获取计算从读页面到获得锁的时间，如果时间大于给定值2000ms，则说明可能存在死锁。
- 在`BtreeTest`中，需要添加锁来保证并发插入删除时不冲突。
    - 当scan时，对查询路径上的每页加读锁。
    - 当插入时，对查询路径上每页加读锁，叶子节点加读写锁，如果需要分裂节点，对邻居和父节点加读写锁，并一直循环向上到根节点加读写锁。
    - 当删除时，对叶子节点加读写锁，如果需要合并节点或移动节点之间的entry，对邻居和父节点加读写锁，并一直循环向上到根节点加读写锁。

5.2 重难点

- 在`BTreeFile`中的`splitLeafPage`方法中，对左邻居加锁加锁，然后向上循环加锁。
- 在`BTreeFile`中的`deleteTuple`方法中，在获取到页面且还未插入之前，向上循环加锁。
- 在`BTreeFile`中的`mergeLeafPages`方法、`stealFromLeftInternalPage`方法、`stealFromRightInternalPage`方法中，在方法开始的时候向上循环加锁。没有对`mergeInternalPages`方法加锁，因为调用该方法只能通过`mergeLeafPages`或者`mergeInternalPages`调用，但是这时他的路径上的节点已经被加锁了，无需重复加锁。

6. exercise6

选择死锁的判断，通过两种方式实现：超时和依赖图。

6.1 超时

当循环获取锁的时间超过2000ms时，抛出异常，判断有死锁。

6.2 依赖图

使用数据结构`ConcurrentHashMap<TransactionId, Set<TransactionId>> dependencyMap;`来存储事务之间的依赖图。set中不会有重复的元素，保证依赖图中的两个事务之间不会有两条边。

当一个事务获取页面的锁失败时，会将这个页面的锁里的所有事务在`dependencyMap`中对应的`Set<TransactionId>`里添加获取该锁失败的事务，这样当一个事务完成时，只需要删除对应的键就可以删除所有他指向的边。

每一次添加完成后，需要通过图的DFS遍历判断是否存在环，如果存在，则说明存在死锁，抛出异常。

6.3性能测试

采取`BTreeDeadlockTest`测试死锁检测的性能：

- 当采取超时检测时，通过该测试需要3.587秒
- 当采取依赖图检测时，通过该测试需要1.181秒

依赖图检测的效果明显优于超时检测。

超时检测可以通过降低获取锁的时间的长短来提升性能，但是降低时间也有可能导致将正常的事务判断为死锁。

7. 提交记录