Lab 5

1. 代码逻辑

在讲解每个 exercise 的逻辑之前，我先介绍我所实现的 Lock 机制。

——Lock 类用于管理数据库系统中的页面锁，通过定义锁的类型、锁定的页面 ID 以及持有共享锁和独占锁的事务集合，并提供相应的方法获取和设置这些属性，同时重写了 equals 和 hashCode 方法来支持锁对象的比较和哈希操作。

——LockManager 实现了用于管理数据库系统中的锁定机制。它包括获取和释放页面锁的方法，并通过事务 ID 和页面 ID 来管理锁。acquireLock 方法会阻塞直至成功获取所需锁，releaseLock 和 releaseAllLocks 方法用于释放锁，getExclusiveLockedPids 和 getSharedLockedPids 方法分别获取某事务持有独占锁和共享锁的页面 ID 集合，holdsLock 方法检查某事务是否持有锁。它实现了一个线程安全的锁管理系统，确保并发事务的正确执行。

1.1 exercise 1

exercise1 要求我们在 BufferPool 中实现获取和释放锁的方法，修改 getPage() 方法以在返回页面之前获取所需的锁，实施 releasePage() 以释放页面的锁，以及实现 holdsLock() 以确定事务是否已经锁定了某个页面。

——getPage 函数实现了根据事务 ID、页面 ID 和权限获取页面，并在需要时获取相应的锁的功能。首先，根据权限类型（只读或读写）请求共享或独占锁。如果权限无效，抛出异常。然后，在同步块中，检查页面是否已缓存在 pageIndexHashMap 中。如果未缓存，则从磁盘读取页面，并在需要时驱逐旧页面以腾出空间，最后将页面加入缓存和 LRU 列表。如果页面已缓存，则更新 LRU 列表中的位置以反映最近使用情况，最后返回目标页面。

1.2 exercise 3

exercise 3 要求我们在 BufferPool 的 evictPage 函数中实现正确的页面驱逐逻辑，确保不驱逐脏页面。

——evictPage 函数实现了在缓冲池（BufferPool）中驱逐一个页面（Page），以便为新的页面腾出空间。首先，创建一个迭代器 iter 来遍历最近最少使用（LRU）列表 LRUList。其次，初始化变量 evictPid 为 null，用于存储待驱逐的页面 ID。然后，遍历 LRUList，找到第一个没有被修改（即未被标记为脏页）的页面。如果找到了，将其页面 ID 赋值给 evictPid，并跳出循环。如果遍历完所有页面都没有找到未被修改的页面，则抛出一个数据库异常，提示所有页面都是脏页，无法驱逐。接着，获取待驱逐页面在页面数组 pageArray 中的索引 pageArrayId。之后，调用 flushPage 函数将待驱逐的页面刷回磁盘，如果发生 IOException，打印错误信息。最后，将 readyBitSet 中对应位置的比特设置为 false，表示该位置的页面已被驱逐，并从页面索引哈希表 pageIndexHashMap 和 LRUList 中移除该页面的相关信息。

1.3 exercise 4

exercise4 要求我们实现 BufferPool 类中的 transactionComplete 函数。此函数有两个版本，一个接收布尔参数 commit，另一个不接收。无参版本应调用 transactionComplete(tid, true) 来提交事务。提交时，应将与事务相关的脏页刷入磁盘；回滚时，应将页面恢复到磁盘上的状态。此外，无论事务提交或回滚，都应释放 BufferPool 中与该事务相关的所有状态，包括事务持有的任何锁。

——transactionComplete 函数实现了根据给定事务的 ID 和一个布尔标志来提交或回滚事务，并释放与该事务相关的所有锁的功能。首先，如果提交标志 commit 为 true，则调用 flushPages(tid) 方法将该事务的脏页刷入磁盘；如果 commit 为 false，则检查该事务是否持有锁，并获取该事务独占锁定的页面集合，调用 discardPage(pid) 方法丢弃每个页面。最后，无论事务提交或回滚，都释放该事务持有的所有锁，如果事务不再持有任何锁，则调用 notifyAll() 方法通知所有等待线程。

2. 实验心得

该实验花费了我 4 天时间完成。其中，我花费了 2 天时间学习了锁机制，由于对于操作系统相关知识的相关记忆比较模糊，因此学起来花费了较多的时间和精力。