This assignment is a collaboration between:

- Megan Fantes (U56999246)
- Efstathios Karatsiolis (U65214507)

PA 5
Design Document

Describe any design decisions you made, including your deadlock detection policy, locking granularity, etc.

- We created a LockManager class to track the state of the locks throughout the system. The LockManager class implements all methods for acquiring lock, detecting deadlocks, and releasing locks. There are different methods to acquire and release a shared vs. exclusive lock.
- The LockManager has the following objects:
    - Sharers = a hashmap of page IDs and the set of transactions that are all currently accessing the corresponding page
    - Owners = a hashmap of page IDs and the single transaction that is currently accessing that page
    - sharedPages = a hashmap of transaction IDs and all of the pages that each transaction currently has a shared lock on
    - ownedPages = a hashmap of transaction IDs and all of the pages that each transaction currently has an exclusive lock on
    - waiters = a hashmap of page IDs and the set of transactions that are currently waiting to access each page
    - waitedPages = a hashmap of transaction IDs and the page each transaction is waiting for (each transaction can only be waiting on one page at a time)
    - waitingTx = a hashmap of transaction IDs and the set of transactions on which each transaction is waiting to complete so it can acquire the locks it needs.
- The LockManager has the following methods:
    - acquireLock() = checks the permissions and assigns the correct lock if the page is available, adds the transaction to waitingTx otherwise.
    - detectDeadlock() = scans through all transactions in waitingTx to see if it detects a loop (i.e. if transaction2 is in the set of transactions that transaction1 is waiting on, and transaction1 is in the set of transactions that transaction2 is waiting on).
    - checkConsistency() = asserts that all sharers also appear in sharedPages, and vice versa. Asserts that all owners also appear in ownedPages, and vice versa.
    - holdsLock() = checks that the given transaction has an exclusive lock on the given page.
    - addWaiter() = adds the given transaction to the set of transactions waiting for the given page. removeWaiter() removes the transaction.
    - addSharer() = adds the given transaction to the set of transactions with a shared lock on the given page. removeSharer() removes the transaction.
    - addOwner() = adds the given page and transaction as a key, value pair to the owners hashmap to indicate the transaction has an exclusive lock on the page. removeOwner() removes the hashmap entry.
    - acquireExclusiveLock() / acquireSharedLock() = checks the consistency of the pages, checks the states of the given transaction and page, and if all checks pass, then it assigns the lock to the transaction.

    - releaseLock() = checks if the given transaction has an exclusive or shared lock on the given page, then removes the given transaction as a sharer or owner of the page. releaseAllLocks() does so for all transactions.
- getPage()
    - getPage() iterates through the levels of the database hierarchy until we acquire the lock we need
    - Then it checks if the page we want is in the buffer
        - If it is, it retrieves the page
        - If it is not, it evicts a page from the buffer and then brings in the page from the disk
    - Then it checks the permissions, and if the permission of the transaction is to write to the page, it marks the page as dirty
    - Then it returns the page ID
- releasePage()
    - The LockManager releases the lock on the page
- holdsLock()
    - the LockManager checks that the given transaction has an exclusive lock on the given page
- evictPage() so that it does not evict dirty pages
    - To make sure that evictPage() does not evict dirty pages, we make an ArrayList of clean pages in the buffer pool. If that list is empty, we flush ALL pages in the buffer to the disk, so all pages are clean now, then we re-make the ArrayList of clean pages and try again.
    - To implement this logic, we implemented the flushAllPages() method
        - This method just loops through every page in the buffer pool and flushes it with flushPage()
- transactionComplete()
    - we loop through every page in the buffer, and check if the page is dirty
    - if the page is dirty, we check if the given transaction was the one to write to the page
        - if it is, we check if this transactionComplete() method call is the commit (commit = true) or abort (commit = false)
        - if we are committing, we flush the dirty page to the disk
        - if we are aborting, we return the page to the beforeImage that we stored when we wrote to it.
- Detecting and resolving deadlock
    - Every time transaction tries to acquire the lock, the LockManager runs the detectDeadlock() method. If it detects a deadlock, it throws a transactionAbortedException and aborts the transaction.

Describe how long you spent on the lab, and whether there was anything you found particularly difficult or confusing.

- This assignment took us about 20 hours of coding between 2 people.
- This assignment was relatively straightforward, it just involved a lot of design decisions.
- We passed the QueryTest system test in PA4, but for some reason the test was failing due to timeout while we were testing for PA5. When we asked George about it, he said we could increase the timeout limit for the QueryTest, and once we did, the test passed.