EX 1: Acquire and Release locks

getPage() Method: Retrieves the specified page with the associated permissions. Acquires a lock and may block if that lock is held by another transaction. Checks if the page is present in the buffer pool. If not, it adds the page to the buffer pool. Implements a simple Least Recently Used (LRU) eviction policy when there is insufficient space in the buffer pool.

insertTuple() Method: Adds a tuple to the specified table on behalf of a transaction. Acquires a write lock on the page the tuple is added to and any other pages that are updated. Marks any pages that were dirtied by the operation as dirty and adds versions of any pages that have been dirtied to the cache.

deleteTuple() Method: Removes the specified tuple from the buffer pool. Acquires a write lock on the page the tuple is removed from and any other pages that are updated. Marks any pages that were dirtied by the operation as dirty and adds versions of any pages that have been dirtied to the cache.

EX 2: Locking Throughout SimpleDB

flushPage() Method: Flushes a certain page to disk if it is dirty.

transactionComplete() Method: Commits or aborts a given transaction and releases all locks associated with the transaction. If committed, flushes all pages associated with the transaction to disk.

EX 3: Implementation of evictPage method in BufferPool

Initialisation of variables: oldestPageCounter is initialized with the maximum integer value to keep track of the least used page. oldestPageId is initialized to null to store the page ID of the least used clean page found so far. **Iterate through pages to find the least used clean page:** check each page to see if it is clean and if it is less used (oldest) than the current candidate for eviction. **Check for Clean Page:** If a clean page is found during the iteration, its ID is stored as the candidate for eviction. If the page is dirty, it is skipped, and the search continues for a clean page. **Handle No Clean Page Found:** If no clean page is found (all pages in the buffer pool are dirty), the method throws a DbException as required by the NO STEAL policy. **Flush the Clean Page:** If a clean page is found, it is flushed to disk to ensure that any modifications are written before eviction. ensuring that the page's state is persisted before removing it from the buffer pool. **Remove the Page:** Once the clean page is successfully flushed, it is removed from the buffer pool and the LRU tracking data structures. This completes the eviction process, ensuring that only clean pages are evicted from the buffer pool.

EX 4: transactionComplete method in BufferPool

Handle Commit: If the commit flag is true, indicating that the transaction should be committed: 1) Iterate through all pages associated with the transaction and flush any dirty pages to disk using flushPage method. This ensures that any changes made by the transaction are persisted. 2) Release all locks held by the transaction using the deleteLock method of LockManager. 3) Remove any state related to the transaction from the BufferPool. **Handle Abort:** If the commit flag is false, indicating that the transaction should be aborted: 1) Iterate over all pages associated with the transaction and discard any changes made by the transaction by removing its associated pages from the buffer pool using discardPage method. 2) Release all locks held by the transaction using the deleteLock method of LockManager. 3) Remove any state related to the transaction from the BufferPool.

EX5: Deadlock detection/prevention in BufferPool

Lock Manager: The LockManager instance manages locks acquired by transactions. The lock manager is used to acquire read or write locks on pages (getPage method), release locks (unsafeReleasePage method), check if a transaction holds a lock (holdsLock method), and delete locks associated with a transaction (transactionComplete method). **Acquiring and Releasing Locks:** In the getPage method, locks are acquired on pages before accessing them. Depending on the requested permissions (read-only or read-write), the lock manager is used to acquire appropriate locks. In the **transactionComplete** method, locks associated with a transaction are released. **Deadlock Detection**: When a transaction requests a lock (getPage method), the lock manager checks the current lock state and transaction dependencies. If a potential deadlock is detected, the transaction is aborted by throwing a TransactionAbortedException. This ensures that deadlock situations are identified and avoided. **Transaction Abortion:** When a deadlock is detected, a TransactionAbortedException is thrown to signal the need for transaction abortion. The transactionComplete method then cleans up after the aborted transaction.

Appendix:

LockingTest (unit) for EX 1 and 2

```
    [junit] Testsuite: simpledb.LockingTest
    [junit] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.85 sec
    [junit] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.85 sec
    [junit]
    [junit] Testcase: acquireWriteReadLocksOnSamePage took 0.3 sec
    [junit] Testcase: acquireReadLocksOnTwoPages took 0.187 sec
    [junit] Testcase: acquireReadLocksOnSamePage took 0.201 sec
    [junit] Testcase: acquireReadWriteLocksOnTwoPages took 0.202 sec
    [junit] Testcase: lockUpgrade took 0.279 sec
    [junit] Testcase: acquireReadWriteLocksOnSamePage took 0.169 sec
    [junit] Testcase: acquireThenRelease took 0.057 sec
    [junit] Testcase: acquireWriteAndReadLocks took 0.161 sec
    [junit] Testcase: acquireWriteLocksOnTwoPages took 0.27 sec

BUILD SUCCESSFUL
Total time: 5 seconds
```

LockingTest (unit) for EX 3

```
PS C:\Users\Sree Devi Rajavelu\Desktop\DATABASE SYSTEMS\DB_LAB\project_24-main> ant runtest -Dtest=LockingTest
Buildfile: C:\Users\Sree Devi Rajavelu\Desktop\DATABASE SYSTEMS\DB_LAB\project_24-main\build.xml

compile:

testcompile:

runtest:
    [junit] Running simpledb.LockingTest
    [junit] Testsuite: simpledb.LockingTest
    [junit] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.988 sec
    [junit] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.988 sec
    [junit]
    [junit] Testcase: acquireWriteReadLocksOnSamePage took 0.354 sec
    [junit] Testcase: acquireReadLocksOnTwoPages took 0.23 sec
    [junit] Testcase: acquireReadLocksOnSamePage took 0.195 sec
    [junit] Testcase: acquireReadWriteLocksOnTwoPages took 0.212 sec
    [junit] Testcase: lockUpgrade took 0.336 sec
    [junit] Testcase: acquireReadWriteLocksOnSamePage took 0.202 sec
    [junit] Testcase: acquireThenRelease took 0.084 sec
    [junit] Testcase: acquireWriteAndReadLocks took 0.198 sec
    [junit] Testcase: acquireWriteLocksOnTwoPages took 0.156 sec

BUILD SUCCESSFUL
Total time: 3 seconds
```

DeadlockTest (unit) for EX 5

```
PS C:\Users\Sree Devi Rajavelu\Desktop\DATABASE SYSTEMS\DB_LAB\project_24-main> ant runtest -Dtest=DeadlockTest
Buildfile: C:\Users\Sree Devi Rajavelu\Desktop\DATABASE SYSTEMS\DB_LAB\project_24-main\build.xml

compile:

testcompile:

runtest:
    [junit] Running simpledb.DeadlockTest
    [junit] Testsuite: simpledb.DeadlockTest
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.163 sec
    [junit] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.163 sec
    [junit] ------------- Standard Output ---------------
    [junit] testReadWriteDeadlock constructing deadlock:
    [junit] testReadWriteDeadlock resolved deadlock
    [junit] testUpgradeWriteDeadlock constructing deadlock:
    [junit] testUpgradeWriteDeadlock resolved deadlock
    [junit] testWriteWriteDeadlock constructing deadlock:
    [junit] testWriteWriteDeadlock resolved deadlock
    [junit] ------------- ---------------- ---------------
    [junit] ------------- Standard Error -----------------
    [junit] simpledb.transaction.TransactionAbortedException
    [junit]     at simpledb.transaction.LockManager.acquireWriteLock(LockManager.java:203)
    [junit]     at simpledb.storage.BufferPool.getPage(BufferPool.java:96)
    [junit]     at simpledb.TestUtil$LockGrabber.run(TestUtil.java:338)
    [junit] simpledb.transaction.TransactionAbortedException
    [junit]     at simpledb.transaction.LockManager.acquireWriteLock(LockManager.java:203)
```

```
    [junit] simpledb.transaction.TransactionAbortedException
    [junit]     at simpledb.transaction.LockManager.acquireWriteLock(LockManager.java:203)
    [junit]     at simpledb.storage.BufferPool.getPage(BufferPool.java:96)
    [junit]     at simpledb.TestUtil$LockGrabber.run(TestUtil.java:338)
    [junit] simpledb.transaction.TransactionAbortedException
    [junit]     at simpledb.transaction.LockManager.acquireWriteLock(LockManager.java:203)
    [junit]     at simpledb.storage.BufferPool.getPage(BufferPool.java:96)
    [junit]     at simpledb.TestUtil$LockGrabber.run(TestUtil.java:338)
    [junit] simpledb.transaction.TransactionAbortedException
    [junit]     at simpledb.transaction.LockManager.acquireWriteLock(LockManager.java:203)
    [junit]     at simpledb.storage.BufferPool.getPage(BufferPool.java:96)
    [junit]     at simpledb.TestUtil$LockGrabber.run(TestUtil.java:338)
    [junit] ------------- ---------------- ---------------
    [junit]
    [junit] Testcase: testReadWriteDeadlock took 0.42 sec
    [junit] Testcase: testUpgradeWriteDeadlock took 0.279 sec
    [junit] Testcase: testWriteWriteDeadlock took 0.451 sec

BUILD SUCCESSFUL
Total time: 2 seconds
PS C:\Users\Sree Devi Rajavelu\Desktop\DATABASE SYSTEMS\DB_LAB\project_24-main> ant runsystest -Dtest=TransactionTest
```

TransactionTest (system) for EX 5

```
Total time: 2 seconds
PS C:\Users\Sree Devi Rajavelu\Desktop\DATABASE SYSTEMS\DB_LAB\project_24-main> ant runsystest -Dtest=TransactionTest
Buildfile: C:\Users\Sree Devi Rajavelu\Desktop\DATABASE SYSTEMS\DB_LAB\project_24-main\build.xml

compile:

testcompile:

runsystest:
    [junit] Running simpledb.systemtest.TransactionTest
    [junit] Testsuite: simpledb.systemtest.TransactionTest
    [junit] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.536 sec
    [junit] Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.536 sec
    [junit]
    [junit] Testcase: testTwoThreads took 0.135 sec
    [junit] Testcase: testAllDirtyFails took 0.071 sec
    [junit] Testcase: testSingleThread took 0.018 sec
    [junit] Testcase: testTenThreads took 0.239 sec
    [junit] Testcase: testFiveThreads took 0.064 sec

BUILD SUCCESSFUL
Total time: 1 second
PS C:\Users\Sree Devi Rajavelu\Desktop\DATABASE SYSTEMS\DB_LAB\project_24-main>
```