CSCI 3357: Database System Implementation

Homework Assignment 5
Due Tuesday, October 17

SimpleDB currently uses timeout to detect deadlock. Change it so that it uses the wait-die deadlock detection strategy, as described in Figure 5.22 of the text. Your code should modify the class `LockTable` as follows:

- The methods `sLock`, `xLock`, and `unLock` will need to take the transaction's id as an argument.
- The variable `locks` must be changed so that a block maps to a list of the transaction ids that hold a lock on the block (instead of just an integer). Use a negative transaction id to denote an exclusive lock.
- Each time through the while loop in `sLock` and `xLock`, check to see if the thread needs to be aborted (that is, if there is a transaction on the list that is older than the current transaction). If so, then the code should throw a `LockAbortException`.
- You will also need to make trivial modifications to the classes `Transaction` and `ConcurrencyMgr` so that the transaction id gets passed to the lock manager methods. I'm sure you can figure out what those changes must be.

My solution required replacing a lot of code, but the resulting amount of code did not change much.

A word of warning about Java lists. The Java `List` classes have two methods to remove an element from a list `L`:

`L.remove(e)` removes the object `e` from `L`, if it exists.
`L.remove(i)` removes the object at position I from `L`.

If `L` is a list of numbers, then a method call such as `L.remove(6)` is ambiguous. Does it remove the value 6 from the list, or the sixth element? The answer is that it removes the sixth element. If you want to remove the value 6, then you must convert the value to an `Integer` object, either by type casting, as in `L.remove((Integer)6)`, or by explicit conversion, as in `L.remove(Integer.valueOf(6))`.

I have included the test program `HW5Test` for you to download. This program is similar to the class `ConcurrencyTest` of Figure 5.19 in the text, and creates three transactions, each in their own thread. In it, transactions A and C both need a lock held by transaction B. Under the wait-die algorithm, transaction A should wait, whereas transaction C should throw an exception. When I run the program on my solution, I get the output shown on the next page:

```
new transaction: 1
Transaction A starts
Tx A: request slock block 1
Tx A: receive slock block 1
new transaction: 2
Transaction B starts
Tx B: request xlock block 2
Tx B: receive xlock block 2
new transaction: 3
Transaction C starts
Tx C: request xlock block 1
Transaction C aborts
transaction 3 rolled back
Tx A: request slock block 2
Tx B: request slock block 1
Tx B: receive slock block 1
transaction 2 committed
Transaction B commits
Tx A: receive slock block 2
transaction 1 committed
Transaction A commits
```

The code for `HW5Test` contains a statement (i.e. line 97) which, when uncommented, results in an output in which transaction C does not abort. You should try that also.

When you are done, create a zip file containing `LockMgr.java`, `Transaction.java`, and `ConcurrencyMgr.java`, and submit the zip file to Canvas.