HW4
Junfu Chen
4684872
Please count Homework 4 twice to replace the Homework 2, Thanks!

16.2
1. After T1 writes X and before T1 reads Y, T2 writes X and writes Y. In this case, T2 overwrites uncommitted data and T1's update of X is lost. Moreover, the result is not equivalent to any serial execution of the transactions. This is a WW conflict and T2 interferes with T1.

2. When strict 2PL is used, T1 will obtain an exclusive lock on X before writing X and will release it after the transaction completes. In this way, T2 will not be able to get an exclusive lock on X and write X until T1 completes. Thus, T2 will access X and Y after T1 completes. And we can see that the interference between the two transactions is apparently prevented.

3.
<1> The Strict 2PL algorithm allows only serializable schedules. That is, conflicts are completely prevented from happening. Thus, we can save time since we don't have to undo transactions because of conflicts.

<2> The idea of the Strict 2PL algorithm is quite simple. We only have to grab an S lock for reading and an X lock for writing, then release the lock after the transaction is done. This can be easily done in the code, which saves tons of human labor.

17.4
1. Strict 2PL with timestamps used for deadlock prevention: (We use wait-die policy for demonstrating)

a) Sequence 81: <1> T1: R(X). In this step, T1 obtains a shared lock on X. <2> T2: W(X). Since T2 comes later than T1 (thus has lower priority) and wants a lock that T1 holds, T2 aborts. <3> T2: W(Y). This step is not executed since T2 aborts. <4> T3: W(Y). In this step, T3 obtains an exclusive lock on Y. <5> Tl: W(Y). Since T1 comes earlier than T3 (thus has lower priority) and wants a lock that T3 holds, T1 waits for T3. After T3 completes and release its locks, T1 obtains an exclusive lock on Y and writes Y. Then T1 completes and releases its locks. When T1 releases its locks and T2 restarts, T2 obtains an exclusive lock on X and writes X, obtains an exclusive lock on Y and writes Y, then releases its locks.

b) Sequence 82: <1> T1: R(X). In this step, T1 obtains a shared lock on X and reads X. <2> T2: W(Y). In this step, T2 obtains a shared lock on Y and reads Y. <3> T2: W(X). Since T2 comes later than T1 (thus has lower priority) and wants a lock that T1 holds, T2 aborts. <4> T3: W(Y). Since T2 aborts and releases its lock on Y, T3 obtains an exclusive lock on Y and writes Y. <5> Tl: W(Y). Since T1 comes earlier than T3 (thus has lower priority) and wants a lock that T3 holds, T1 waits for T3. After T3 completes and release its locks, T1 obtains an exclusive lock on Y and writes Y. Then T1 completes and releases its locks. When T1 releases its locks and T2 restarts, T2 obtains an exclusive lock on Y and writes Y, obtains an exclusive lock on X and writes X, then releases its locks.

2. Strict 2PL with deadlock detection.

a) Sequence 81: <1> T1: R(X). In this step, T1 obtains a shared lock on X. <2> T2: W(X). T2 waits for T1 to release the lock on X. <3> T2: W(Y). T2 is still waiting. <4> T3: W(Y). T3 obtains the lock on Y.

<5> Tl: W(Y). T1 waits for T3 to release the lock on Y. Since T3 is not waiting, it completes and releases the locks. Then T1 obtains the lock on Y and writes Y. T1 completes and releases its locks. Then T2 obtains the lock on X and writes on X, obtains the lock on Y and writes Y, and commit. We see there is no cycle of transactions waiting for locks to be released by each other. Namely, there is no deadlock.

Wait-for-graph:

b) Sequence 82: <1> Tl: R(X). T1 obtains a shared lock on X. <2> T2: W(Y). T2 obtains an exclusive lock on Y. <3> T2: W(X). T2 waits for T1 to release the lock on X. <4> T3: W(Y). T3 waits for T2 to release the lock on Y. <5> Tl: W(Y). T1 is queued to wait for the lock on Y. And we see that T2 is waiting for T1 while T1 is also waiting for T2. So a deadlock occurs.

Wait-for-graph:

17.8
1. An example query is SELECT EMP.salary WHERE EMP.name=”Santa”. Since this query run at the same time with the update command, it reads data (EMP.salary) that are not committed yet by the update transaction and thus yield erroneous results. This is called an Write-Read Conflict.

The way that locking tuples solve this problem is, by locking EMP.salary field of “Santa” in the update transaction, the query will not be able to obtain the lock and access the data until the update transaction completes writing and releases the locks. In this way, query access the data after update is done so no deadlock will happen.

18.2
1. Each log record has a unique LSN, thus it should be able to access a log record with one disk access if we know the LSN. Also, LSNs are always increasing.

2.
prevLSN: previous LSN of the transaction.
transID: id of the transaction of the log record.
type: the type of the log record.
pageID: id of the modified page.
offset: offset of the change.
length: length of the change.
before-image: value before the change.
after-image: value after the change.

3.
Redoable log records may be update log records and compensation log records. All redoable actions must be redone unless one of the following conditions holds: 1. Affected page is not in the Dirty Page

Table. This means that this page has alreay made it to disk. 2. Affected page is in the Dirty Page Table, but has recLSN > LSN. This means that this change is not the one that is responsible for making this page dirty, i.e., this change has already made it to disk. 3. Affected page is in the Dirty Page Table, but has pageLSN that is greater than or equal LSN. This means that either this update or a later update to the page was written to disk.

4.
Update log records are for recording the update transactions, while CLR are for recording the actions to undo the actions recorded in the log record. Also, we cannot undo the actions in update log records while we can undo the actions in CLR.

18.4
1.

LSN	LOG	prevLSN	undoNextLSN
00	Update: T1 writes T2	NA	NA
10	Update: T1 writes P1	00	00
20	Update: T2 writes P5	NA	NA
30	Update: T3 writes P3	NA	NA
40	T3 commit	30	NA
50	Update: T2 writes P5	20	20
60	Update: T2 writes P3	50	50
70	T2 abort	60	NA

2. We will have to undo every action of T2 in an reverse order. Thus, we will first restore P3 to the before-image in LSN 60. Second, we restore P5 to the before-image in LSN 50. And last, we restore P5 to the before-image in LSN 20.

3.

LSN	LOG	prevLSN	undoNextLSN
80	T2 CLR: UNDO LSN 60	70	50
90	T2 CLR: UNDO LSN 50	80	20
100	T2 CLR: UNDO LSN 20	90	NA
110	T2 END	100	NA