CS 186 Spring 2016 Midterm 2 Solutions

I. Locking and Serializability

1. For the following statements, list the true statement(s) in alphabetical order.
   a. Schedules that are conflict serializable are also view serializable.
      True.
   b. Under two-phase locking, once a transaction releases a lock, it can no longer acquire any new locks.
      True.
   c. Schedules that are conflict serializable have to be produced by two-phase locking.
      False. Note that the converse is true - 2PL guarantees conflict serializability.
   d. Schedules produced by two-phase locking are guaranteed to prevent cascading aborts.
      False. Strict 2PL is needed to guarantee this.
   e. Strict two-phase locking is both necessary and sufficient to guarantee conflict serializability.
      False. Sufficient but not necessary.
   f. Wound-wait and wait-die algorithms are pessimistic deadlock avoidance algorithms and can cause more transaction aborts than needed.
      True.
   g. Under multi-granularity locking, locks should be acquired and released from the top level to the bottom level.
      False. Locks are acquired top down, but released bottom up.
   h. Under multi-granularity locking, when a transaction T1 is holding an 'IX' lock on page A, it is possible for transaction T2 to hold a 'S' lock on record B of page A.
      True.

   *Instructor note: Some of these might seem like arbitrary rules, but they all exist for a reason. For each, do you understand why?

2. Draw the dependency graph for this schedule.

$$\begin{tikzcd}
T1 & T2 \arrow[l] \arrow[dl] \arrow[d] \\
T3 \arrow[r] & T4
\end{tikzcd}$$

3. Of the following, list all conflict equivalent serial orderings of this schedule in alphabetical order:
   *Can you think of *all* conflict equivalent serial orderings?
   a. T2 T1 T4 T3
   b. T2 T3 T4 T1
   c. T2 T1 T3 T4
   d. T2 T4 T1 T3

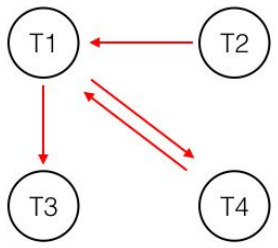
4. List one read that could occur between timestep 9 and 10 that would make this schedule not conflict serializable.
   There were multiple answers: T2:R(A) or T2:R(B) or T2:R(C) or T3:R(B)

5. For the following statements, list the true statement(s) in alphabetical order.
   a. It is possible for this schedule to be produced by two-phase locking.
      True.
   b. It is possible for this schedule to be produced by strict two-phase locking.
      False. Recall that strict 2PL releases locks at commit.
   c. This schedule is guaranteed not to have cascading aborts.
      True.

6. What locks have been acquired on table Y at timestep 9?
   T1:IX, T2: IX, T3: IX, T4: IX
   *All transaction tried to write to the table, so they all have IX locks. Note that transactions that started with IS locks will upgraded those locks to IX. Also note that no transaction has to wait for a lock at the table level, since there are no S or X locks at that level. All waiting happens at lower levels.

7. What locks have been acquired on page B at timestep 9?
   T1: IX, T2: IS
   *T1 has an IX lock, as it wrote to B_1. T2 has an IS lock, as it read from B_2 - it does <u>not</u> have an X lock, as it is waiting for T1.
   Tying this back to the previous question, before T2 writes to B, it will successfully acquire the IX on table Y, but will have to wait for T1 before acquiring the X on B.

8. There is a deadlock in this schedule (i.e. it is waiting indefinitely for locks).
   a. Draw the waits-for graph for this schedule.



   b. Which transactions are involved in the deadlock?
      These are the transactions in the cycle above: T2 T4
   c. Circle all true statements (on the answer sheet):
      i. We can resolve this deadlock by aborting any transaction involved in the deadlock.
         True.
      ii. No transaction can proceed until this deadlock is resolved.
         False. In this schedule, T3 can clearly proceed.

9. Suppose T1 wishes to scan page B between timestep 8 and 9. Which locks are upgraded at this time?
   IX(B) -> SIX(B)
   *At this time, T1 has IX(Y), IX(B), and X(B_1). So, T1 only needs to *upgrade* its lock on B. Recall that we are working with Strict 2PL, so it cannot drop the IX on B.

II. Query Optimization

1.
    a. When evaluating potential query plans, the set of left deep join plans are always guaranteed to contain the best plan.
    False - this is a heuristic that System R uses to shrink the search space.
    b. As a heuristic, the System R optimizer avoids cross-products if possible.
    True.
    c. Considering all join orders and join methods, there are n! ways to join n tables.
    False. During the exam we amended this question to say O(n!) -- it is still false, but ends up being a bit harder to prove, so we gave everyone a point for this question.
    d. A plan can result in an interesting order if it involves a sort-merge join.
    True.
    e. The System R algorithm is greedy because for each pass, it only keeps the lowest cost plan for each combination of tables.
    False - it is not greedy because it keeps track of interesting orders.
    f. If the statistics needed to compute the result size of a table are missing, the System R optimizer aborts.
    False - it uses 1/10 as reduction factor if it cannot be computed.
2.
    a.
        i. 1/50
        ii. 1/50000
        iii. 2501 / 5000
        iv. (3*10^6) / (7*10^6 + 1)
    b.
        i. B
        ii. Ø. The correct answer is 100020*R3
        iii. A
        iv. D
        v. C
    c. [NPages(Index III) + NPages(C)]*½ + ½ ½ NPages(C) * NPages(F)
    15 + 5*50 = 265 I/Os

    We will access City using Index (III) which gives an outer cost of [NPages(Index III) + NPages(C)]*½. We can only apply one reduction factor because Index III is only on population.

    How many pages worth of tuples will be candidates to be joined with F? ½ ½ NPages(C) because of the two predicates.

    To access F, we have to use a page scan: NPages(F).

    Common mistakes were to forget the index access cost for the outer loop, applying the wrong reduction factors, or having F as the outside releation. "A JOIN B" is typically interpreted to have A on the outside, and furthermore we always put the smaller relation on the outside to minimize I/O cost.

    d. After Pass 2, which of the following plans could be in the DP table?
    A. City [Index(III)] JOIN Airline [File scan] - False this is a cross product.

B. City [Index (III)] JOIN Flight [Index (I)] - True, this could possibly be in the table.
C. Flight [Index (II)] JOIN City [Index (III)] - False, Index (II) would not have been kept as a Single Table Access Method.

3. CDE

A. Maintain a more detailed histogram of Cities.population
We gave everyone points for A since it was vague. Maintaining the histograms helps, but only if we also update our optimizer to use these more advanced statistics for its cost estimation (which the answer choice did not explain).

B. Change Index (III) to be unclustered
An unclustered index would not minimize I/O cost, since it's more random I/O, and we may load a page more than once.

C. Reduce the size of Airline.name field using string compression
Fitting more records per page means fewer I/Os.

D. Add a hash index on Flights.aid
Hash index allows us to do a different kind of index nested loops join, which may reduce I/O cost.

E. Store City as a sorted file on population
Sorted file may provide more efficient range lookups.

III. Recovery

1. [6 points] List all the true statements in alphabetical order:
    a. When a transaction commits, any modified buffer pages must be written to durable storage.
    False. ARIES uses a NO FORCE policy.
    b. When aborting a transaction, it may be necessary to modify pages on disk.
    True. ARIES uses a STEAL policy.
    c. During recovery, the ARIES protocol may redo aborted transactions.
    True. This is a key feature of ARIES and essential for the correctness of the protocol.
    d. The pageLSN contains the LSN of the last operation to modify the page.
    True. This is the definition of the pageLSN.
    e. The tail of the log is always flushed after every update operation.
    False. We only require that the tail of the log be flushed on commit. Remember that flushedLSN keeps track of the log tail.
    f. A system that uses a FORCE, STEAL policy does not need to undo any operations after a crash.
    False. A system with a STEAL policy needs UNDO logging to ensure atomicity.

Questions 2 through 7 use the log below. Our database system is using ARIES for recovery. Some of the information for update and CLR log records is omitted for brevity. Adjacent log records are spaced exactly 10 LSN's apart. The system crashes after the last log record is written.

LSN	Record	prevLSN
0	update: T1 writes P1	null
10	update: T2 writes P2	null
20	update: T3 writes P1	null
30	update: T2 writes P3	10
40	begin_checkpoint	-
50	update: T1 writes P4	0
60	update: T2 writes P5	30
70	end_checkpoint	-
80	commit: T2	60
90	update: T3 writes P5	20
100	T2 end	80
110	T1 abort	50
120	CLR: T1 LSN 50	110

Transaction Table and Dirty Page Table recorded at end_checkpoint (LSN 70).

Transaction Table			Dirty Page Table	
Transaction	**lastLSN**	**Status**	**PageID**	**recLSN**
T1	0	Running	P1	20
T2	30	Running	P3	30
T3	20	Running		

2. [1 point] The page updated by LSN 0 has been written to disk when the system starts recovering from the crash.
    a. True
    b. False
    c. Not enough information

a. At the time of the begin_checkpoint record, P1 appears in the dirty page table with recLSN 20, so the update at LSN 0 has definitely been materialized on disk.

3. [1 point] The page updated by LSN 60 has been written to disk when the system starts recovering from the crash.
   a. True
   b. False
   c. Not enough information

c. When T2 committed, all log records for T2 were forced to the stable log. However, because we are using a NO FORCE policy, the data pages themselves may not have been forced to disk.

4. [1 point] What is the undoNextLSN of the CLR at LSN 120?

0. The undoNextLSN of a CLR indicates the LSN of the next CLR that will be written for that transaction, or null if all CLR's for that transaction have been written.

5. [6 points] Fill in the transaction table and dirty page table after Analysis. Do not add any rows to the transaction table (it should only contain T1 and T3). You may not need to use all rows provided in the dirty page table.

Transaction Table			**Dirty Page Table**	
Transaction	**lastLSN**	**Status**	**PageID**	**recLSN**
T1	120	Aborting	P1	20
T3	90	Running	P3	30
			P4	50
			P5	60

+½ for each correct lastLSN in the transaction table, +½ for each correct Status in the transaction table, +¾ for each correct (PageID, recLSN) in the dirty page table, +1 if P2 was not included in the dirty page table.

6. [3 points] List the LSN's of the actions that are redone during REDO. Assume that nothing in the buffer pool was flushed to disk between begin checkpoint and the time of crash.

20, 30, 50, 60, 90, 120

+½ for each correct LSN, -½ for each incorrect LSN

7. [6 points] What log records are written during UNDO? Fill in the remaining columns for the records below. You may not need to use all rows in the table.
   Hint: T1 and T3 are the only "loser" transactions.

LSN	**Record**	**prevLSN**
200	CLR: T3 LSN 90	90
210	CLR: T3 LSN 20	200
220	T3 end	210
230	CLR: T1 LSN 0	120
240	T1 end	230
250		
260		

-1 for each missing record above, -1 for most additional records included in the answer, -½ for each correct record with an incorrect prevLSN.

A common mistake was to include a record "CLR: T1 LSN 50". This log record is not written during UNDO because it was already written before the crash. Another common mistake was to include records "T3 abort" or "T1 abort". The only records written during UNDO are CLR and END records.