CS 245
Final Exam – Winter 2014

This exam is open book and notes. You can use a calculator. You can use your laptop only to access CS245 materials. You have 140 minutes (2 hours, 20 minutes) to complete the exam.

Print your name:_____

The Honor Code is an undertaking of the students, individually and collectively:

1. that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;
2. that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.

The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.

While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.

I acknowledge and accept the Honor Code.

Signed:_____

Problem	Points	Maximum
1		10
2		10
3		10
4		10
5		10
6		10
7		10
Total		70

Problem 1 (10 points)

Consider doing a natural join operation on two relations $R(A, B)$ and $S(B, C)$. Assume that the R tuples are stored contiguously on 200 disk blocks (i.e., $B(R) = 200$), while the S tuples are stored contiguously on 1000 blocks (i.e., $B(S) = 1000$). Each block holds 20 tuples (same for R as for S). There are 51 memory blocks available.

Calculate the I/O cost for each of the following join algorithms. Use an analysis similar to the one used in class. Ignore the I/O cost of writing the final join output to disk. Unless stated otherwise, the tuples in the relations are not sorted.

(a) **Iteration Join Algorithm, R First**. In this case, the algorithm reads 50 R blocks into memory, reads all of S (1 block at a time) into memory, joining with R. The process is repeated until all outputs are generated.

Number of IOs: 4200

200 IOs to read R into memory (50*4), then for each set of 50 R blocks we need 1000 IOs to read in R (1000*4). Therefore, the total is 50*4 + 1000*4 = 4200.

(b) **Merge Join, Sorted Relations**. Assume that both R and S are sorted by B.

Number of IOs: 1200

$B(R) + B(S) = 200 + 1000 = 1200$

(c) **Merge Join, Unsorted Relations**. Assume R and S are not sorted. First sort R by B using merge-sort, writing out all tuples in R into a sorted file. Then sort S by B in a similar fashion. Finally, do the merge-join on the sorted relations.

Number of IOs: 6000

Each tuple is read, written, read written. The first read, written is to sort each chunk, the second read, written is to merge sort all of the chunks.

Sort cost R: 4*200 = 800.

Sort cost S: 4*1000 = 4000.

Total cost: sort cost R + sort cost S + join cost = 800 + 4000 + 1200 = 6000

(d) **Index Join**. Assume that there is an index on the B column of S and that the index fits in memory. Furthermore, assume that on average each R tuple matches 4 S tuples. With this strategy, we read a block of R and for each tuple r of R in this block we use the index to find all matching tuples $\{s_1, ..., s_m\}$ of S. Each of these S tuples is read into memory and joined with r. We repeat the process for all R blocks.

Number of IOs: 16200

(4000 tuples in R * 4 matching tuples in S per R tuple) + 200 IOs to read in R = 16200

(e) **Hash Join**. We first hash tuples from R (using the B attribute) into 50 buckets. (No R buckets are kept in memory.) We then hash S into 50 buckets in a similar fashion. Then we join each pair of R_i and S_i buckets.

Number of IOs: 3600

Read and write R to hash, read and write S to hash, then read all buckets to join. That is a read, write, and read for each block in R and S.

Total cost = 3*($B(R)$ + $B(S)$) = 3*(200 + 1000) = 3600

(f) **Hybrid Hash Join**. We first hash tuples from R (using the B attribute) into 20 buckets (i.e., buckets $R_1, R_2, ..., R_{20}$). and keep 3 R buckets in memory (i.e., buckets R_1, R_2, R_3). We then hash S into 20 buckets in a similar fashion (i.e., buckets $S_1, S_2, ..., S_{20}$) but join buckets S_1, S_2, S_3 immediately with buckets R_1, R_2, R_3. Then we join each pair of R_i and S_i buckets for the remaining 17 buckets.

Number of IOs: 3240

Bucketize R = read R + (write 17 buckets * 10 blocks per bucket) = 200 + 17*10 = 370

Bucketize S, only write 17 buckets = read S + (write 17 buckets * 50 blocks per bucket) = 1000 + 17*50 = 1850

Compare join (3 buckets already done) = read 17*10 + 17*50 = 1020

Total cost = 370 + 1850 + 1020 = 3240

Problem 2 (10 points)

Consider the following two transactions:

- $T_1 : R_1(X)W_1(X)R_1(Y)W_1(Y)$
- $T_2 : R_2(Y)W_2(Y)$

Which of the following schedules would be valid under a legal, two-phase-locking scheduler (with well formed transactions)? (Assume L lock actions in schedules are exclusive.) Write YES for allowed, NO for not allowed. If your answer is NO, please circle the first action in the schedule that is problematic.

(a) $L_1(X)R_1(X)W_1(X)L_2(Y)R_2(Y)W_2(Y)U_2(Y)L_1(Y)R_1(Y)W_1(Y)U_1(X)U_1(Y)$

YES (allowed) or NO?: YES

(b) $L_1(Y)L_1(X)R_1(Y)W_1(Y)R_1(X)W_1(X)U_1(X)U_1(Y)L_2(Y)R_2(Y)W_2(Y)U_2(Y)$

YES (allowed) or NO?: NO. $R_1(Y)$ is the first problematic action because it reorders the sequence of the transaction T_1.

(c) $R_1(X)L_1(X)W_1(X)R_1(Y)L_1(Y)W_1(Y)U_1(X)U_1(Y)R_2(Y)L_2(Y)W_2(Y)U_2(Y)$

YES (allowed) or NO?: NO. $R_1(X)$ is the first problematic action.

(d) $L_1(X)R_1(X)W_1(X)U_1(X)L_1(Y)R_1(Y)W_1(Y)U_1(Y)L_2(Y)R_1(Y)W_1(Y)U_2(Y)$

YES (allowed) or NO?: NO. $L_1(Y)$ is the first problematic action.

(e) $L_1(X)L_1(Y)L_2(Y)R_1(X)W_1(X)R_1(Y)W_1(Y)R_2(Y)W_2(Y)U_2(Y)U_1(Y)U_1(X)$

YES (allowed) or NO?: NO. $L_2(Y)$ is the first problematic action.

(f) $L_2(X)R_2(Y)W_2(Y)U_2(X)L_1(X)L_1(Y)R_1(X)W_1(X)R_1(Y)W_1(Y)U_1(Y)U_1(X)$

YES (allowed) or NO?: NO. $R_2(Y)$ is the first problematic action.

For the next two questions, consider the given schedule for three transactions. Determine whether the schedule can be produced by a two-phase-locking scheduler. Write YES if it can, NO if it cannot.

(g) $R_3(C)R_1(A)W_1(B)R_2(B)W_3(A)$

YES (produced by 2PL) or NO?: YES

(h) $R_1(A)R_3(C)W_3(A)R_2(B)W_1(B)$

YES (produced by 2PL) or NO?: NO

Problem 3 (10 points)

Consider the schedule below, S stands for start, V for validation attempt and F for finish. As usual, each subscript indicates the transaction id.

- $H = S_1, S_2, V_2, V_1, F_1, S_3, F_2, S_4, V_3, V_4, F_3, F_4$

The objects in the database that can be read or written are A, B, C, D, E, F. Read and write sets are given by:

- $T_1 : RS(T_1) = \{A, B\}$, $WS(T_1) = \{C, D\}$
- $T_2 : RS(T_2) = \{A, C\}$, $WS(T_2) = \{D, F\}$
- $T_3 : RS(T_3) = \{C, E\}$, $WS(T_3) = \{B, F\}$
- $T_4 : RS(T_4) = \{A, D\}$, $WS(T_4) = \{B, E\}$

In the questions below, for deciding if a validation is successful assume that all previous validations succeeded. That is, assume that VAL(T_i) includes all transactions such that $V_j <_H V_i$ in the schedule H above. Note that this is different from the way we did a similar problem in class!

(a) Does T_1 validate?

YES or NO?: NO (Write-Write conflict with T_2 on D)

(b) Does T_2 validate?

YES or NO?: YES

(c) Does T_3 validate?

YES or NO?: YES

(d) Does T_4 validate?

YES or NO?: NO (Write-Write conflict with T_3 on B)

(e) Suppose we add another transaction T_5 to the schedule H_1 obtaining:

- $G = S_1, S_2, V_2, V_1, F_1, \mathbf{S_5}, S_3, F_2, S_4, V_3, \mathbf{V_5}, V_4, F_3, F_4, \mathbf{F_5}$

Suppose that validation V_5 succeeds. What could have been T_5's biggest read set? Continue to assume all validation attempts by previous transactions succeeded. Assume there are only six objects A, B, C, D, E, F.

Largest possible read set: A, C, E

(f) What could have been T_5's biggest write set? Continue to assume previous validations were successful and 6 objects.

Largest possible write set: A, C, D, E

(g) Suppose validation *failed* due to a write set-write set conflict, not a write set-read set conflict. What could have been T_5's *smallest* write set? (If there are multiple possible answers, give any one.) Continue to assume previous validations were successful and 6 objects.

Smallest possible write set: B (or F)

Problem 4 (10 points)

For the first part of this problem, consider two transactions that simultaneously attempt to acquire locks on object Z. The types of locks requested are given in parts (a) through (d) below. (For example, in part (d) one transaction requests an S lock while the other requests an IS lock.) Assume that each transaction has requested the appropriate lock on Z's parent in the hierarchy. For each case below, write 'yes' if the transactions can obtain the lock at the same time, and 'no' if one of the transactions will have to wait for the other to release its lock.

(a) Requested lock types are: SIX, SIX.

Can locks be held concurrently (YES) or not (NO)?: NO

(b) Requested lock types are: IX, IX.

Can locks be held concurrently (YES) or not (NO)?: YES

(c) Requested lock types are: IX, X.

Can locks be held concurrently (YES) or not (NO)?: NO

(d) Requested lock types are: S, IS.

Can locks be held concurrently (YES) or not (NO)?: YES

For questions in the second part, we give a sequence of three locks acquired by a single transaction, where the first lock is obtained for the root of the lock hierarchy, the second one for a child C_1 of the root, and the third lock is acquired for a child C_2 of C_1. No other locks are requested by the transaction. However, note that the transaction can access additional nodes. For example, if the transaction has locked C_1 in S mode, it can read any child of C_1 with no additional locks.

For each question, write 'yes' if the series of locks is valid and not-redundant; write 'no' otherwise. By valid we mean that when a node is locked the appropriate lock has been obtained on the parent. By redundant we mean that the transaction could have avoided requesting one of its locks and still been able to read/write exactly the same data.

(e) Lock sequence: IX, SIX, S

Is sequence valid and non-redundant?: NO (Since the final lock was an 'S', the intention-X lock wasn't needed)

(f) Lock sequence: IX, X, X

Is sequence valid and non-redundant?: NO (X lock on child is not needed if you have an X lock on the parent)

(g) Lock sequence: SIX, IX, X

Is sequence valid and non-redundant?: YES

(h) Lock sequence: IS, SIX, X

Is sequence valid and non-redundant?: NO (Root lock should have been IX, since an X lock was taken for the grandchild)

Problem 5 (10 points)

Consider a system that uses undo/redo value logging (not logical action logging). At recovery time the log contains, in this order:

- A begin checkpoint record, S_1
- An end checkpoint record, E_1
- A begin checkpoint record, S_2

There are no other checkpoint related records in this log. Using these 3 records we divide the log into the following four regions:

- Region A: From the oldest log record through the log record just before S_1
- Region B: From the first record after S_1 through the record just before E_1
- Region C: From the first record after E_1 through the record just before S_2
- Region D: From the first record after S_2 through the last record written before the crash.

A particular transaction T may have undo/redo action records in all 4 regions. For each scenario below, indicate what action should be taken for the T records in each region. The actions can be either (i) *undo* the actions in that region, (ii) *redo* the actions in the region, (iii) *ignore* the actions, or (iv) this case is *not applicable* because T has no actions in this region.

(a) Transaction T has a start record in region A and a commit record in region C.

For actions in A, undo/redo/ignore/not applicable: ignore

For actions in B, undo/redo/ignore/not applicable: redo

For actions in C, undo/redo/ignore/not applicable: redo

For actions in D, undo/redo/ignore/not applicable: not applicable

(b) Transaction T has a start record in region A and a commit record in region D.

For actions in A, undo/redo/ignore/not applicable: ignore

For actions in B, undo/redo/ignore/not applicable: redo

For actions in C, undo/redo/ignore/not applicable: redo

For actions in D, undo/redo/ignore/not applicable: redo

(c) Transaction T has a start record in region A but no commit record in any region.

For actions in A, undo/redo/ignore/not applicable: undo

For actions in B, undo/redo/ignore/not applicable: undo

For actions in C, undo/redo/ignore/not applicable: undo

For actions in D, undo/redo/ignore/not applicable: undo

(d) Transaction T has a start record in region B and a commit record in region C.

For actions in A, undo/redo/ignore/not applicable: not applicable

For actions in B, undo/redo/ignore/not applicable: redo

For actions in C, undo/redo/ignore/not applicable: redo

For actions in D, undo/redo/ignore/not applicable: not applicable

(e) Transaction T has a start record in region B but no commit record in any region.

For actions in A, undo/redo/ignore/not applicable: not applicable

For actions in B, undo/redo/ignore/not applicable: undo

For actions in C, undo/redo/ignore/not applicable: undo

For actions in D, undo/redo/ignore/not applicable: undo

Problem 6 (10 points)

Great news! Your boss just negotiated some great deals with your supplier. She got the following pricing (a gigabyte is 2^{30} bytes):

- 4 gigabyte RAM card, \$128/each card
- 128 gigabyte SSD drive, 50,000 IO/s, \$500/each drive
- 1000 gigabyte hard drive, 15,000 RPM, 500 IO/s, \$500/each drive

It's time to evaluate the five minute rule, to decide what data goes to memory and what data goes to secondary storage! You can assume:

- a page can be fetched in one IO;
- a page is 64kB (i.e., 64×2^{10} bytes).

(a) How much does one gigabyte of RAM cost?

DOLLAR COST OF 1 RAM GIGABYTE: $\frac{128}{4} = 32$

(b) How many pages fit in one gigabyte of RAM?

NUMBER OF PAGES: $\frac{2^{30}}{64 \times 2^{10}} = 2^{14} = 16384$

(c) How much does one IO cost with the SSD?

DOLLAR COST OF 1 SSD IO: $\frac{500}{50000} = 0.01$

(d) How much does one IO cost with the hard drive?

DOLLAR COST OF 1 HARD DRIVE IO: $\frac{500}{500} = 1$

(e) What is the break-even point for a page on SSD? In other words, how small does the interval between accesses (in seconds) have to be before it is more economical to have the page in RAM rather than on SSD?

ANSWER: $\frac{2^{14}}{32} \times 0.01 = 5.12$

(f) What is the break-even point for a page on a hard drive?

ANSWER: $\frac{2^{14}}{32} \times 1 = 512$

Problem 7 (10 points)

Consider a database system that logs logical actions and uses log sequence numbers (LSNs). For this first part of this problem, consider 4 transactions that have performed actions on an object that resides on disk block Z. The following list describes the actions and the current state of the transactions:

- Transaction T_1: inserts record r_1 into Z; LSN of this action is 10; T_1 has committed;
- Transaction T_2: inserts record r_2 into Z; LSN of this action is 20; T_2 has committed;
- Transaction T_3: inserts record r_3 into Z; LSN of this action is 30; T_3 is still active;
- Transaction T_4: inserts record r_4 into Z; LSN of this action is 40; T_4 is still active;

Note that there could be other transactions running in the system. These other transactions do not refer to records r_1, r_2, r_3 or r_4, but could access other records in Z or elsewhere.

(a) At this point in time, say disk block Z contains r_1 and r_2, but not r_3 nor r_4. What range of LSNs can block Z have?

Range of LSNs: 20-29

Explanation: Since the T_2 action is reflected on disk, the lsn of Z must be at least 20. Since the T_3 action is not reflected, the lsn must be less than 30.

(b) At this point in time, say disk block Z contains r_1, r_2, r_3 and r_4. What range of LSNs can block Z have?

Range of LSNs: ≥ 40

Explanation: Since all four actions are reflected on Z, lsn must be at least 40.

(c) At this point in time, say disk block Z has LSN = 35. Which of the four records should Z contain?

Records in Z: r_1, r_2, r_3

With page Z having lsn 35, all actions with an lsn smaller or equal to 35 must be reflected. But no action with lsn 36 or higher has been reflected on Z.

(d) Assume the system crashes at this time, without T_3 and T_4 having committed. At the end of step [1] of the recovery algorithm (Notes 10, slide 72), what will be the *minimum value* of the LSN of block Z?

Minimum LSN of Z: 40

At the end of phase 1, all actions in the log have been redone, so the lsn of Z must be at least 40.

(e) Assume that the recovery process of part (d) continues. At the end of step [2] of the recovery algorithm (Notes 10, slide 73), what will be the *minimum value* of the LSN of block Z?

Minimum LSN of Z: 42

Since T_3 and T_4 are rolled back, at least two log compensation records (one for the T_3 action and one for the T_4 action) must have been added to the log, hence the lsn of Z must advance to at least 42. Note that an abort transaction entry should be written after the last undo action is logged. Hence the lsn of the compensation actions could be 41 and 42 (and the abort records would have LSNs 43, 44), so the MINIMUM LSN of Z is 42.

The second part of this problem does not refer to transactions T_1, T_2, T_3 and T_4. This part is about LSN wrap-around. Since a finite number of bits is used to represent an LSN, eventually LSNs will wrap around to 0. For example, say we use 4 bits to represent LSNs, so the maximum representable LSN is `M` = 15. (This is a small number just to keep the example simple.) As we write log records, LSNs will go from 0, 1, 2,... to 14, 15, 0, 1, 2...

If we are not careful, the system can get confused between an old LNS and a new one. For example, say a block Z has LSN = 2 from the first cycle of LSNs, and a log record has LSN = 1 from the second round. If we compare the LSNs it appears that the log action has been reflected on Z, but this is not the case. To avoid such confusion, the system keeps a limit LSN, call it `L`, and uses the following operations:

```
To increment current LSN C:
    INCR(C):
    If C=L wait until L has been increased (see below)
    else return C := (C + 1) modulo M;
To check if LSN X has been reflected on block with LSN Y:
    REFLECTED(X, Y):
      If [(Y <= L) and (X <=L)] or [(Y > L) and (X > L)] then
          if  X <= Y then return TRUE else return FALSE;
      Else
          if X > Y then return TRUE else return FALSE.
```

To illustrate, say `L` = 4, `M` = 15 and `C` = 14. In this case, if we increment `C` once, we get 15. If we increment again, we get `C` = 0, and then 1, 2, 3, and 4. However, we cannot increment beyond `C` = 4 because we have reached the limit `L`, indicating that LSNs greater than 4 are still in use so we cannot recycle them yet.

At the point where `L` = 4, `M` = 15 and `C` = 3, we get that REFLECTED(1,2) is TRUE, REFLECTED(12, 12) is TRUE, REFLECTED(2, 14) is FALSE and REFLECTED(14, 2) is TRUE.

Our task is to figure out when we can advance `L` to some new value `L'`. The advance will be safe if the numbers between `L` and `L'` (inclusive) are no longer "in play" in the system, i.e., these numbers will not be needed in future REFLECTED comparisons.

(f) When we advance `L` to `L'`, we need to ensure that one of the properties below holds *for all* LSNs `S` between `L` and `L'`. (Note that `S` can be `INCR(L)`, `INCR(INCR(L))`, ... up to `L'`. Also note that in the properties below, the new `L'` is used for evaluating the `REFLECTED` function.) Please select the correct property:

(1) All `S` should be smaller than the minimum LSN currently on a database block on disk.

(2) For all disk blocks with LSN `Y`, `REFLECTED(S,Y)` must be FALSE.

(3) For all disk blocks with LSN `Y`, `REFLECTED(S,Y)` must be TRUE.

(4) Let `Y` be the LSN of the latest valid checkpoint start record. Then `S` should be less than `Y`.

(5) Let `Y` be the LSN of the latest valid checkpoint start record. Then `REFLECTED(S,Y)` must be FALSE.

(6) Let `Y` be the LSN of the latest valid checkpoint start record. Then `REFLECTED(S,Y)` must be TRUE.

(7) Let `Y` be the LSN of the latest valid checkpoint end record. Then `S` should be less than `Y`.

(8) Let `Y` be the LSN of the latest valid checkpoint end record. Then `REFLECTED(S,Y)` must be FALSE.

(9) Let `Y` be the LSN of the latest valid checkpoint end record. Then `REFLECTED(S,Y)` must be TRUE.

The correct property is (Select one of 1-9): We gave credit for either (2) or (5)

Explanation: There are actually TWO properties that need to hold. To illustrate the properties, here are two examples of BAD things that could happen:

(i) Say the LSNs can be 0, 1, 2,..., 15 and say L is at 15. We want to advance L to say L'=4. Suppose there is block Z on disk that has LSN=3. The transaction that last updated Z, call it T_1, safely committed long ago, so we will not have to redo or undo this transaction in the future. If we advance L to L'=4, some new transaction T_2 may decide to modify Z and may get LSN = say 2. But now between the time that T_2's log record is written and the time Z is updated we have a problem: REFLECTED(2,3) is TRUE, yet Z does not reflect T_2's update yet! Property (2) in the list rules out this situation.

(ii) Again, say the LSNs can be 0, 1, 2,..., 15, L is at 15 and want to advance L to say L'=4. A transaction T_3 has modified block Z, generating a log record with LSN=3. A second transaction also modifies Z with LSN= 7. At this time say both T_3 and T_4 have committed, and both their Z actions have been flushed so that the LSN of Z is 7. Now assume that we advance L to L'=4 at this point. It now appears that the T_3 action has not been reflected on Z (with L=4, REFLECTED(3,7) is FALSE) and the T_4 action appears to have been reflected (REFLECTED(7,7) is TRUE). If the system fails, at recovery we will redo the T_3 action, which means that we perform the T_3 action twice! Property (5) avoids this situation.

We gave credit for either answer (2) or (5).

Note incidentally that both rules are needed. Property (2) does not avoid the problem in the example (ii) above (the LSN of Z is at 7 when we try moving L). Property (5) does not avoid the problem of example (i) since T_1's action could be before a checkpoint and T_2 has not even arrived when we advance L.