CS 245
Final Exam – Winter 2016

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:_____

Your SUNetID (username in stanford email):_____

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)

Your database system holds two important relations, R and S, that are frequently joined over a common attribute A. The relations are currently stored as rows, but you want to consider column storage to see if that improves join performance. Here is the information that applies to both parts of this problem:

- Relation R has three attributes, A, B, and C, each 10 bytes long.
- Relation S has three attributes, A, D, and E, each 10 bytes long.
- Each of R and S contain 42000 tuples. (Note that $42000 = 14 \times 30 \times 100$ which should simplify your work.)
- In addition to its header, each disk block can hold 4200 bytes. (Note that $4200 = 14 \times 30 \times 10$.)
- To perform $R \bowtie S$ we use a simple hash join algorithm as described in our class notes 7, pages 58-62. Assume there is enough main memory.
- The expected number of resulting tuples in $R \bowtie S$ is 10.

(a) We first evaluate the cost (in number of IOs) of performing $R \bowtie S$ with row storage. Here is the information for this scenario:

- The tuples of relation R are stored contiguously in blocks. They are not sorted.
- The S tuples are also contiguous, unsorted.

What is the number of IOs needed for the hash join? (As usual, do not include the cost of writing the final result to disk.)

Number of IOs (row storage):_____

(b) We now evaluate the join cost with column storage. Here is the information for this scenario:

- Relation R is stored as three columns, $R_1(I, A)$, $R_2(I, B)$, $R_3(I, C)$.
- Each R column contains an index I attribute that is 4 bytes in length. A tuple (a, b, c) from R is stored as (i, a), (i, b), (i, c) with the same i value.
- Columns R_1, R_2 and R_3 are separately, each column stored contiguously and unsorted (equivalent to three sequential files, one for each column).
- There is an in-memory index for the I attribute. Given an $I = i$ value, the index points to the R_1, R_2 and R_3 entries with that i value.
- Relation S is stored as three columns, $S_1(J, A)$, $S_2(J, D)$, $S_3(J, E)$.
- Attribute J is 4 bytes long, S columns are contiguous and unsorted, and there is a J in-memory index.

What is the number of IOs needed for the hash join in this case? Remember that in both parts (a) and (b) of this problem, the final result produced by the join should identical, i.e., $R \bowtie S$. (As usual, do not include the cost of writing the final result to disk.)

Number of IOs (column storage):_____

Problem 2 (10 points)

For each schedule below, determine if the schedule is recoverable, ACR (avoids cascading rollback), strict, or serial.

(a) Consider the following schedule:

$S_a: \ r_1(X) \ w_2(X) \ w_2(Y) \ c_2 \ r_1(Y) \ c_1$

- Recoverable (Yes or No)? _____
- ACR (Yes or No)? _____
- Strict (Yes or No)? _____
- Serial (Yes or No)? _____

(b) Consider the following schedule:

$S_b: \ w_1(X) \ w_2(Y) \ w_3(Z) \ c_1 \ r_3(X) \ c_2 \ r_3(Y) \ c_3$

- Recoverable (Yes or No)? _____
- ACR (Yes or No)? _____
- Strict (Yes or No)? _____
- Serial (Yes or No)? _____

(c) Consider the following schedule:

$S_c: \ w_1(X) \ w_2(Y) \ w_3(Z) \ r_3(X) \ c_2 \ r_3(Y) \ c_3 \ c_1$

- Recoverable (Yes or No)? _____
- ACR (Yes or No)? _____
- Strict (Yes or No)? _____
- Serial (Yes or No)? _____

(d) Consider the following schedule:

$S_d: \ w_2(X) \ w_2(Y) \ r_1(X) \ w_1(Y) \ w_2(Z) \ c_2 \ c_1 \ r_3(Y) \ w_3(Z) \ c_3$

- Recoverable (Yes or No)? _____
- ACR (Yes or No)? _____
- Strict (Yes or No)? _____
- Serial (Yes or No)? _____

Problem 3 (10 points)

For parts (a) and (b) of this question, consider a multi-granularity locking system, with lock modes S, X, IS, IX, and SIX. The object hierarchy is as follows:

- There is a root R with tables S and T;
- S has blocks A and B under it, T has block C under it;
- Block A contains record a_1, block B contains records b_1 and b_2, and C contains record c_1.

(a) Give the list of locks required to do the given operation. Start with the root lock and go downwards in the hierarchy. (To illustrate the style for answers, here is one possible answer: IS(R), IS(S), IS(A), S(a_1). This answer may or may not be the correct one for the questions below.)

Your answer should use the least restrictive locks possible, allowing the most access by other transactions. For example, the answer $X(R)$ avoids conflicts in all cases below, but is too restrictive.

Read all records in S: _____

Modify b_1: _____

Insert a record into C: _____

(b) For each of the following list of locks on our object hierarchy, write 'YES' if the series of locks is valid and non-redundant and '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.

In each sequence below, the root lock is first, followed by the table, block, and record locks respectively.

IS(R), IS(S), SIX(A), X(a_1): _____

IX(R), IX(S), SIX(B), S(b_2): _____

SIX(R), IX(S), IX(B), X(b_2): _____

(c) As mentioned earlier, this part is not about the particular hierarchy defined on the previous page.

For this part of the problem, consider two transactions that are simultaneously attempting to acquire locks on object O. Assume that each transactions has requested the appropriate lock on O's parent in the hierarchy. For each case below, write 'YES' if the transactions can obtain the specified locks (one each) at the same time, and 'NO' if one of the transactions will have to wait for the other to release its lock.

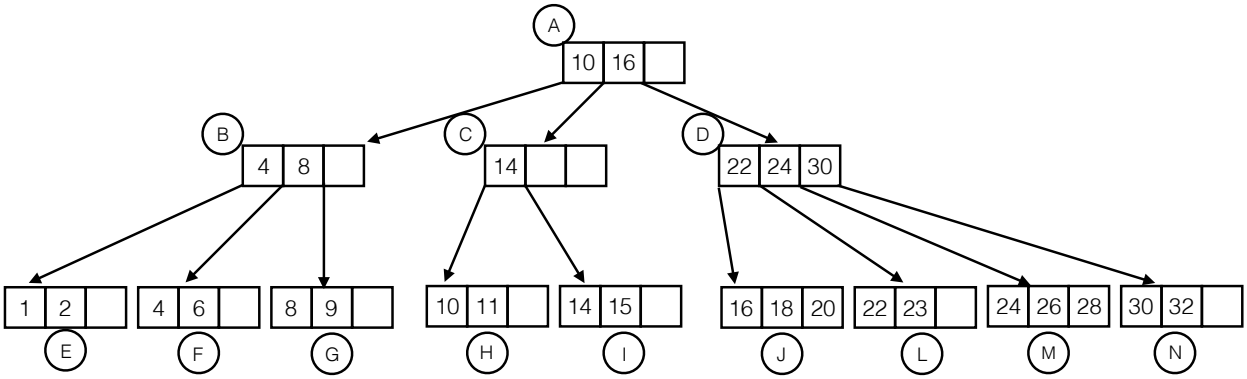
IS, IX: _____

IS, X: _____

SIX, IX: _____

Problem 4 (10 points)

Consider the following B+-tree having 13 nodes



We are interested in examining the order in which transactions acquire and release locks on nodes under the tree protocol discussed in class. Assume we are only using exclusive locks. Also assume that in each case, the transaction starts out with a pointer to node A (but not a lock on A) and does not have pointers to any other node.

For each of the following transactions, describe the order in which locks are acquired and released. Also, if the transaction modifies any B+-tree nodes, indicate where that occurs. For example, a possible answer may be

`Lock(A) Lock(B) Lock(E) Mod(E) Unlock(A) Unlock(B) Unlock(E)`

Note that you do *not* need to show the modified B+-tree, just the complete lock sequence. (The sample sequence above may or may not be the correct answer to any of the questions below, it is just an example.)

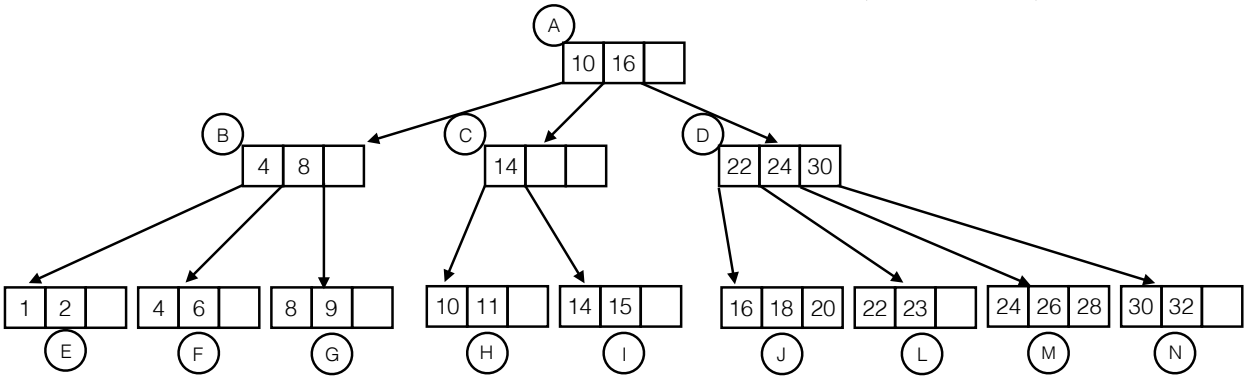
For the sequences below, a transaction should only acquire locks that it actually needs and should release locks as early as permitted under the tree protocol. (For example, it would be trivial to hold locks on all tree nodes until the end but that may delay other transactions unnecessarily.)

For simplicity, assume that during deletes, merge/rebalancing *always* happens with the *left* sibling, without even looking at the right sibling.

(a) Transaction T_1: Insert the key 3 into the B+ tree.

(b) Transaction T_2: Remove the key 6 from the B+ tree.

We reproduce a copy of the B+-tree here for your convenience (it is identical).



(c) Transaction T_3: Print all keys in the range 1-10.

Problem 5 (10 points)

Consider a system that uses logical action logging and the recovery algorithm discussed in class (you can refer to *Slides 72-73 in Notes-10*). Each log entry is of the form `<LSN, txn, action, obj, val1, val2>` , where

- `LSN` is the log sequence number
- `txn` is the transaction id
- `action` describes the action (one of `start`, `commit`, `abort`, `insert`, `delete`, `modify`)
- `obj` is the affected database logical object (if any)
- `val1`, `val2` describe the action (for `insert`, `val1` is the new value; for `modify`, `val1` is the old value, `val2` is the new value)

In addition, we have the following types of log records:

- `<LSN, begin ckpt>`: begin checkpoint
- `<LSN, end ckpt>`: end checkpoint
- `<LSN, CLR, x>`: compensation record for log record with `LSN=x`.

Assume that all the existing and new objects of the database that we consider reside on the same disk block Z. Initially, the database contains 5 objects - `A`, `B`, `C`, `D`, `E` - all with a value of zero. After a crash, the following log exists on disk:

```
<1, T1, start>
<2, T2, start>
<3, T3, start>
<4, T3, modify, C, 0, 9>
<5, T2, insert, F, 6>
<6, begin ckpt>
<7, T1, modify, A, 0, 1>
<8, T2, modify, B, 0, 2>
<9, end ckpt>
<10, T3, insert, G, 7>
<11, T1, delete, E>
<12, T3, commit>
<13, T2, modify, B, 2, 4>
<14, T1, delete, D>
<15, CLR, 13>
```

The `LSN` stored in the disk block Z at the time of crash is `12`.

(a) Compute the database state after the first step of the recovery algorithm completes (and the database reflects the state at the time of the crash). To describe the state, list all the *objects and their values* in the given table (an example entry has been filled for your reference). Do not show the deleted objects. (All the rows of the table may not be necessary.)

`A -> 1`

(b) Compute the state of the database after the recovery completes (both steps 1 and 2 are complete). Again, fill all existing objects and their stored values in the given table. (All the rows of the table may not be necessary.)

`A -> 0`

(c) Write all the log records that are appended to the given log during the recovery process. Please make sure to write the records in the formats specified in the problem statement.(All the rows of the table may not be necessary.)

Problem 6 (10 points)

(a) Consider following two transactions:

- T1: $r_1(Y), r_1(Z), w_1(Y), r_1(X), w_1(X)$
- T2: $r_2(X), r_2(Y), w_2(Y), w_2(X), w_2(Z)$

We want to count all possible conflict-serializable schedules of these two transactions.

Number of schedules equivalent to the serial schedule T_1T_2: _____

Number of schedules equivalent to the serial schedule T_2T_1: _____

Total number of conflict-serializable schedules: _____

(b) Consider the following schedule:

$S_b: \ w_3(Y) \ w_3(X) \ w_1(Y) \ r_2(Y) \ r_2(X) \ w_1(X)$

Draw the precedence graph for the schedule S_b:

Is S_b conflict serializable (Yes or No): _____

Problem 7 (10 points)

Consider three transactions - T_1, T_2, T_3 - that exclusive lock (L) and unlock (U) database items A, B and C in the following order:

- T_1 : $L(A)$ $L(B)$ $L(C)$ $U(A)$ $U(C)$ $U(B)$
- T_2 : $L(B)$ $L(C)$ $U(B)$ $U(C)$
- T_3 : $L(C)$ $L(A)$ $U(C)$ $U(A)$

The aim of the problem is to simulate how these transactions execute under the wait-die or wound-wait deadlock prevention strategies. Assume that the transactions have timestamps in the order of $T_1 < T_2 < T_3$ (T_1 is the oldest).

For the simulations, assume that the transactions execute in a round-robin fashion. That is, T_1 executes a step, then T_2 executes a step, then T_3 executes a step, then T_1 executes another step and so on. When it is a transaction's turn, it executes its next lock or unlock action if it can, and otherwise dies or waits or wounds the holder of the lock, as appropriate. If a transaction is waiting when it is its turn, it skips the step and continues waiting. When a waiting transaction can restart (due to the action of some other transaction), it can run its next action at its next round-robin step.

When a transaction T_X dies in step s, it is restarted and attempts its first action again at step $s + 3$ (its next round robin step).

When a transaction T_X in step r wounds another transaction T_Y, T_X can run its action as well in the same step r. The transaction that gets wounded, T_Y, performs a "Die" action in its next step (which would be step $r + 1$ or $r + 2$) and then proceeds as described in the previous paragraph.

Fill in the following tables to show the sequence of steps that the three transactions make. Row 1 shows steps 1, 2 and 3 (left to right), row 2 shows steps 4, 5 and 6, and so on. Use $L(X)$ to denote locking of an object X, $U(X)$ to denote unlocking of object X, "Wait" to denote the transaction waiting, "Die" to denote the transaction dying and "Wound T_i" to denote the transaction wounding another transaction T_i.

The first two rows have been filled to illustrate how you should fill out the rest of the table. Simulate actions until all the three transactions complete (not all cells in the table may be necessary).

(a) Simulation when the **wait-die** deadlock prevention strategy is used:

Row	$\mathbf{T_1}$	$\mathbf{T_2}$	$\mathbf{T_3}$
1	$L(A)$	$L(B)$	$L(C)$
2	Wait	Wait	Die
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

(b) Simulation when the **wound-wait** deadlock prevention strategy is used:

Row	$\mathbf{T_1}$	$\mathbf{T_2}$	$\mathbf{T_3}$
1	$L(A)$	$L(B)$	$L(C)$
2	Wound T_2; $L(B)$	Die	Wait
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			