CS 245
Final Exam – Winter 2017

This exam is open book and notes. You can use a calculator and your laptop to access course notes and videos (but not to communicate with other people or to browse the Internet). You have 180 minutes to complete the exam.

Best of luck!!!

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		20
Total		80

Please note: Problem 7 (true/false) is worth 20 points.

Problem 1: Recovery (10 Points)

Consider a database with two elements, X and Y. The initial values of X and Y are both 0. We consider three transactions U, V and W that modify these elements concurrently:

- T_1: $X := 42$
- T_2: $Y := 20$, $X := 10$
- T_3: $X := 100$, $Z := 101$

While the transactions execute, the database system crashes. We consider three recovery mechanisms in turn below; your task is to first complete the missing parts of the log and then answer the following questions:

(a) Using **pure undo logging**:

1. $< START\ \ T_2 >$
2. $< START\ \ T_3 >$
3. $< T_2, X, 0 >$
4. $< T_2, Y, 0 >$
5. $< COMMIT\ \ T_2 >$
6. $< START\ \ CKPT(T_3) >$
7. $< T_3, X, ??? >$
8. $< START\ \ T_1 >$
9. $< T_3, Z, 0 >$
10. $< T_1, X, ??? >$

(i) Please complete the undo log by providing in the appropriate values for the log below:

Line 6: _____

10

Line 10: _____

100

(ii) If the database crashes immediately after writing the above log entries, then subsequently performs recovery, which of the following statements is *true*:

A. After a successful recovery, the state of Y is 0.

B. At the time of crash, the state of X (on disk) must be 10.
C. At the time of crash, the state of Z (on disk) must be 101.
D. After a successful recovery, the state of X is 10.

True statement (one of A, B, C, or D): _____

D

(b) Using **pure redo logging**:

1. $< START \ \ T_2 >$
2. $< START \ \ T_3 >$
3. $< T_2, X, 10 >$
4. $< T_2, Y, 20 >$
5. $< COMMIT \ \ T_2 >$
6. $< START \ \ CKPT(T_3) >$
7. $< T_3, X, ??? >$
8. $< START \ \ T_1 >$
9. $< T_3, Z, ??? >$
10. $< T_1, X, 42 >$
11. $< END \ \ CKPT >$
12. $< COMMIT \ \ T_3 >$
13. $< START \ \ CKPT(T_1) >$
14. $< COMMIT \ \ T_1 >$

(i) Line 7: _____

100

Line 9: _____

101

(ii) If the database crashes immediately after writing the above log entries, then subsequently performs recovery, which of the following statements is *true*:

A. After a successful recovery, the state of X is 100.
B. At the time of crash, the state of X (on disk) must be 42.
C. At the time of crash, the state of Y (on disk) must be 20.
D. After a successful recovery, the state of Z is 0.

True statement (one of A, B, C, or D): _____

C

(c) Using **undo-redo logging**, with log format (txn, $item$, before-val, after-val):

1. $< START\ \ T_1 >$
2. $< START\ \ T_2 >$
3. $< START\ \ T_3 >$
4. $< T_3, X, 0, 100 >$
5. $< T_3, Z, 0, 101 >$
6. $< COMMIT\ \ T_3 >$
7. $< T_2, X, 100, 10 >$
8. $< START\ \ CKPT(T_1, T_2) >$
9. $< T_2, Y, ???, ??? >$
10. $< END\ \ CKPT >$
11. $< START\ \ CKPT(T_1, T_2) >$
12. $< COMMIT\ \ T_2 >$
13. $< T_1, X, ???, ??? >$

(i) Please complete the undo-redo log by providing in the appropriate values for the log below:

Line 9: _____ _____

0, 20

Line 13: _____ _____

10, 42

(ii) If the database crashes immediately after writing the above log entries, which of the following statements is *true*:

A. During recovery, the log entry in line 4 will be redone.
B. During recovery, the log entry in line 7 will be ignored.
C. During recovery, the log entry in line 9 will be ignored.
D. During recovery, the log entry in line 13 will be undone.

True statement (one of A, B, C, or D): _____

B

Problem 2: Serializability and Precedence Graphs (10 Points)

For each of the following schedules, draw the precedence graph and determine whether the schedule is conflict serializable. If the schedule is conflict serializable, write the conflict equivalent serial schedule of *transactions* (e.g., $T_5; T_7; T_2$). If the schedule is not conflict serializable, identify a set of conflicting actions that prevents it from being conflict serializable (e.g., $W_5(C)$, $W_7(D)$, $W_2(D)$).

To reiterate: for each, (a) draw the precedence graph, and, (b) if conflict serializable, give the conflict equivalent serial ordering of transactions or, if not, identify the conflicting operations.

(a) $R_1(A)W_2(B)R_2(A)W_1(B)W_3(B)W_1(C)R_3(B)W_1(A)$

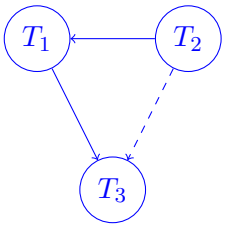
(b) $W_2(C)R_1(A)W_3(B)R_1(C)R_3(B)R_3(A)W_1(B)$

(c) $R_3(B)R_2(A)R_1(A)R_2(C)W_3(A)W_2(B)W_1(C)$

(d) $W_1(A)W_2(B)W_1(B)W_2(A)$

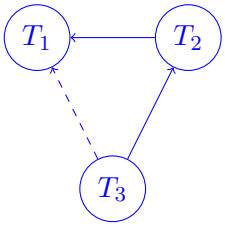
Solutions:

(a) $R_1(A)W_2(B)R_2(A)W_1(B)W_3(B)W_1(C)R_3(B)W_1(A)$



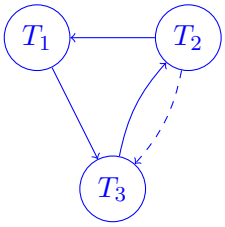
Conflict serializable, equivalent to $T_2T_1T_3$

(b) $W_2(C)R_1(A)W_3(B)R_1(C)R_3(B)R_3(A)W_2(B)W_1(B)$



Conflict serializable, equivalent to $T_3T_2T_1$

(c) $R_3(B)R_2(A)R_1(A)R_2(C)W_3(A)W_2(B)W_1(C)$



Not conflict serializable due to $R_3(B)$ before $W_2(B)$ and $R_2(A)$ before $W_3(A)$.

(d) $W_1(A)W_2(B)W_1(B)W_2(A)$

Not conflict serializable due to $W_1(A)$ before $W_2(A)$ and $W_2(B)$ before $W_1(B)$.

Problem 3: 2PL and Validation (10 points)

Consider the following schedules and decide which schedules are valid under a legal, two-phase-locking scheduler.(Assume that only exclusive locks are allowed.) If yes, augment the schedule with lock and unlock actions to show it can be generated by 2PL. For example, $l_1(A)$ and $u_1(A)$ for locking and unlocking object A by transaction 1. If not, circle the **first** action in the schedule that is problematic.

(a) $r_1(A); w_1(A); w_2(B); w_3(C); r_1(D); w_2(D)$
Yes. One possible augmented schedule is:

$l_1(A); r_1(A); w_1(A); l_2(B); w_2(B); l_3(C); w_3(C); l_1(D); r_1(D); u_1(D); l_2(D)$
$w_2(D); u_2(D)$

(b) $r_3(A); w_4(B); r_1(C); r_3(D); w_3(B); w_2(D); r_3(A); w_1(D); r_3(B); r_2(C); r_1(A)$
No. $r_2(C)$(the second to last operation) is problematic.

The following 5 transactions execute under a validation concurrency control protocol:

- T_1: Read Set: $\{B, E\}$, Write Set: $\{C, E, G\}$
- T_2: Read Set: $\{G\}$, Write Set: $\{A, B\}$
- T_3: Read Set: $\{A\}$, Write Set: $\{C\}$
- T_4: Read Set: $\{A, C\}$, Write Set: $\{B, D\}$
- T_5: Read Set: $\{G\}$, Write Set: $\{C, D\}$

Consider the following sequence of events:

1. T_1 starts
2. T_2 starts
3. T_2 begins validation
4. T_1 begins validation
5. T_3 starts
6. T_2 finishes validation
7. T_4 starts
8. T_5 starts
9. T_4 begins validation
10. T_1 finishes validation
11. T_5 begins validation
12. T_3 begins validation
13. T_4 finishes validation
14. T_3 finishes validation
15. T_5 finishes validation

State which transactions validate successfully and which do not. If a transaction successfully validates, write "YES." If not, write 'NO" and state the conflict by stating the transactions that conflict, and which item(s) they conflict on (e.g., "T_1 and T_2 have a read-write conflict on F") .

(a) Does T_1 validate successfully? If not, state the conflict as instructed above.
No. There is a read-write conflict on B between T_1 and T_2.

(b) Does T_2 validate successfully? If not, state the conflict as instructed above.
Yes.

(c) Does T_3 validate successfully? If not, state the conflict as instructed above.
No. There is a read-write conflict on A between T_3 and T_2.

(d) Does T_4 validate successfully? If not, state the conflict as instructed above.
Yes.

(e) Does T_5 validate successfully? If not, state the conflict as instructed above.
No. There is a write-write conflict on D between T_4 and T_5.

Problem 4: Multi-Granularity Locking (10 points)

Consider a multi-granularity locking system, with lock modes S, X, IS, IX, and SIX as in lecture, with a hierarchy from Relation, Table, Record, to Field. In this question, we consider a database with the following contents:

- Relation R has 2 underlying tables, A and B.
- Table A has 2 records t1 and t2, each with 3 fields E, F, and G.
- Table B has 2 records t3 and t4, each with 3 fields H, I, and J.

(a) Provide the sequence of lock requests required to perform the given operation. Your answer should use the least restrictive locks possible, allowing the greatest degree of concurrency. For example, the answer $X(R)$ avoids conflicts in all cases below, but is too restrictive. Provide your answers using the provided spaces, with one lock or unlock operation per space proceeding left to right (you need not use all the spaces provided). For example:

$X(R); SIX(t_1); IX(A); X(B)$
$SIX(t_4); SIX(I)$

i) Read all records in A:

IS(R), S(A)

ii) Modify the H field of t3:

IX(R),IX(B),IX(t3),X(t3.H)

iii) Perform the operation t1.E = t3.H (a read of t3.H and a write of t1.E are needed):

SIX(R),IX(A),IS(B),IX(t1),IS(t3),X(t1.E),S(t3.H) locks at the same level can be switched

(b) Consider the following four schedules, each with two transactions, U and V, executing under the multi-granularity locking system and database as above. Circle the schedules that can be be achieved under *strict* 2-phase locking. Below, $w_U(A)$ refers to write of table A by transaction U.

(i) $r_U(t1.E), w_V(A), commit(U), commit(V)$

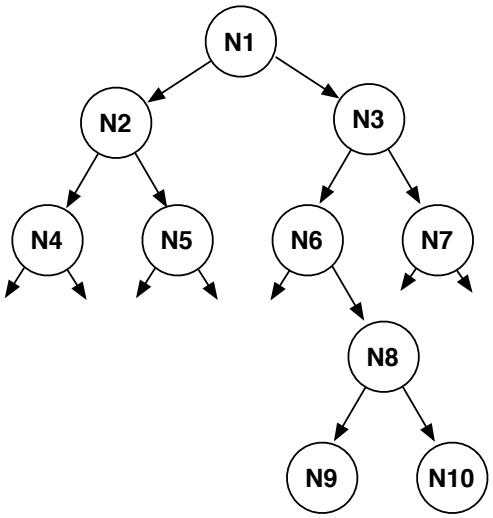
(ii) $r_V(A), w_U(t3), r_U(t1), commit(U), commit(V)$

(iii) $w_V(t1), r_U(A), r_V(A), commit(U), commit(V)$

(iv) $w_U(t1), r_U(A), w_V(t3), commit(U), commit(V)$

ii and iv should be circled.

(c) Consider the following index consisting of nodes $N1$ through $N10$:



What is the sequence of lock and unlock operations required to lock node 10? As above, provide your answer in the spaces below, with one lock or unlock operation per space proceeding left to right (you need not use all the spaces provided). Only depict lock or unlock operations (no intention locks). For example:

$L(N_5)U(N_5)L(N_1)U(N_1)$

$L(N_4)L(N_1)L(N_{10})$

hint: the above example is not correct.

L(N1); L(N2); U(N1); L(N3);
U(N1); L(N6); U(N3); L(N8);
U(N6); L(N10)

Problem 5: Deadlock Prevention (10 points)

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

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

For this problem we want to simulate how these transactions execute under either the wait-die or wound-wait deadlock prevention strategies.

For the simulation, assume that these transactions execute in a round-robin fashion (T_1 then T_2 then T_3). When it is a transaction's turn, it executes its next lock or unlock step 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 is restarted (due to the action of some other transaction) it becomes eligible to run at its very next turn.

Assume that the transactions have timestamps in the order of $T_1 < T_2 < T_3$ (T_1 is the oldest).

Fill in the following tables to show the sequence of steps that the three transactions make. Use $L(X)$ to denote locking an object X, $U(X)$ to denote unlocking X, "Die" to denote the transaction dying or being wounded, and "Wait" to denote the transaction waiting. If a transaction A is wounded by some other transaction B, transaction B's next action is 'Die'.

In the grid, the cell for $[i, j]$ (row i column T_j) represents the i^{th} action of T_j. We have filled out a few cells to illustrate how you should fill out the rest of the table. Simulate actions until all three transactions complete (not all cells in the grid may be necessary). If a transaction is already complete, its turn is skipped.

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

Step	T_1	T_2	T_3
1	L(B)	L(A)	L(C)
2	Wait	Wait	Die
3	L(C)	Die	Die
4	U(C)	L(A)	L(C)
5	U(B)	Wait	Die
6		L(C)	Die
7		U(A)	Die
8		U(C)	Die
9			L(C)
10			L(B)
11			U(C)
12			U(B)
13			
14			
15			

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

Step	T_1	T_2	T_3
1	L(B)	L(A)	L(C)
2	L(C)	Wait	Die
3	U(C)	L(C)	Wait
4	U(B)	U(A)	Wait
5		U(C)	L(C)
6			L(B)
7			U(C)
8			U(B)
9			
10			
11			
12			
13			
14			
14			

(c) Circle each of the following statements that are true:

(i) Wound-wait always outperforms wait-die.

(ii) If we know all records that each transaction will access, we can prevent deadlocks by imposing a total order on lock acquisitions.

(ii) Wait-die always outperforms wound-wait.

(iv) In general, locking uses fewer resources than optimistic methods that rely on validation.

ii and iv are true.

Problem 6: Pre-Midterm Potpourri (10 Points)

(a) Query Processing I/Os: Consider performing a natural join operation on two relations $R(A, B)$ and $S(B, C)$. Assume that R contains 36000 records stored contiguously on 1800 disk blocks (i.e., $B(R) = 1800$; 20 tuples per block), and that S contains 1000 records stored non-contiguously and that each block of memory and disk can store 20 tuples of S.

(i) Suppose we have just two blocks of memory. Ignoring output I/Os, what is the minimum number of I/Os required to perform the natural join via a iteration join if use our two blocks of memory to store one block each of R and S in memory at a time?:

Since S is not stored contiguously, we should hold S in memory to avoid I/Os, then do multiple passes over R. We will read in S one memory block at a time, then join with R. We have $1000/20 = 50$ passes of R. Therefore, we have 1000 I/Os to read in S, and $20 * 1800$ I/Os to join with S. $37,000$ I/Os total.

(ii) Suppose we have just enough memory to allow a two-pass hash join (*not* hybrid hash join!). Ignoring output I/Os, Using a two-pass hash join (as described in class notes 7, slides 61-64), what is the total number of I/Os required to perform the natural join via a hash join?:

Cost to read and write $R = 1800 * 2 = 3600$. Cost to read and write $S = 1000 + 50 = 1050$. Read R1, R2 = $50 + 1800$. Total: 6500

(b) Cardinality Estimation: Consider relations $M(A, B)$, $N(C, D)$, each containing 500 records.

(i) What is the expected number of records in $[M \bowtie N]$?

$500 * 500 = 250000$

(ii) Given a selection predicate $B = 2$ with expected selectivity 0.01 (i.e., 1 in 100 records will match the predicate). What is the expected number of records in $[\sigma_{B=2} M \bowtie N]$?
$0.01 * 500 * 500 = 2500$

(iii) Given a selection predicate $B = 2$ with expected selectivity 0.01. What is the expected number of records in $\sigma_{B=2}[M \bowtie N]$? $0.01 * 500 * 500 = 2500$

(iv) Given a selection predicate $B = 2$ with expected selectivity 0.01 and a selection predicate $C = 4$ with expected selectivity 0.1. What is the expected number of records in $[\sigma_{B=2}M \bowtie \sigma_{C=4}N]$? $0.01 * 500 * 500 = 250$

(c) Storage Costs: Consider relation $D(A, B, C)$ such that column A contains a 8 byte string, column B contains a 4-byte floating point number, and column C contains a binary blob of up to 2048 bytes represented by a 2-byte header followed by the data. If we store D on a disk with block size 512:

(i) What is the minimum number of blocks required to store 1000 records of D? $(8 + 4 + 2) * 1000/512 = 28$ blocks

(i) Ignoring the cost of additional headers required to mark records that span multiple blocks, what is the maximum number of blocks required to store 1000 records of D? $(8 + 4 + 2 + 2048) * 1000/512 = 4028$ blocks

Problem 7: True/False, Advanced Topics (20 points)

State if the following statements are true or false. Please write TRUE or FALSE in the space provided.

1. To maintain data durability in the presence of up to two fail-stop server failures, we must ensure modifications to data are persisted on three or more servers.

   ANSWER: _____

   True; we need $F + 1$ servers to survive F fail-stop failures.

2. Many NoSQL databases such as MongoDB provide flexible schemas, making it easy to add new columns and even nested columns to data.

   ANSWER: _____

   True

3. Serializability is always required to maintain application invariants in the presence of concurrent operations.

   ANSWER: _____

   False; see our example of "no copy of the database contains the key 'peter'" from lecture.

4. MapReduce was the first data processing system to allow users to write user-defined functions.

   ANSWER: _____

   False; as we discussed, databases like Postgres have supported UDFs for decades. MapReduce gave an easy way to parallelize certain classes of UDFs.

5. All transactional databases such as Oracle and SAP HANA guarantee serializability.

   ANSWER: _____

   False! See discussion, lecture 2/15 at 29:20.

6. In two-phase commit, it is always impossible for the coordinator to unblock the protocol if a single participant fails.

   ANSWER: _____

   False; unless the coordinator has already broadcasted commit, the coordinator can choose to abort the transaction.

7. In distributed model training, sequential execution always results in faster convergence time.

   ANSWER: _____

   False; see the section on HogWild! and the experimental results in the parameter server section of the notes (Notes 12, slide 84, covered in review lecture).

8. For read queries that access contiguous sets of keys (e.g., IDs 1000-2000), a distributed database that uses hash-based partitioning will have to contact fewer partitions than than a distributed database that uses range-based partitioning.

   ANSWER: _____

   False; range partitioning is better for range queries.

9. In the absence of failures, Spark enables faster execution than MapReduce because it avoids writing as much intermediate state to disk.

   ANSWER: _____

True; Spark relies on in-memory RDDs and relies on replay and lineage for fault tolerance.

10. Flexible schemas can make query processing less efficient because the data is less regularly structured and database has less control over how data is represented.

ANSWER: _____

True.

11. In the presence of unbounded message delays and fail-stop server failures, there exists at least one scenario for every atomic commitment protocol in which the protocol must block indefinitely.

ANSWER: _____

True; see slides.

12. To decide on a primary for each partition in a database cluster, we can use a consensus service like Zookeeper.

ANSWER: _____

True

13. In HogWild!-style asynchronous model training, increasing asynchrony is guaranteed to decrease convergence time.

ANSWER: _____

False; for example, in the parameter server section, we saw that training with bounded staleness of 16 was slower than training with bounded staleness of 8 (Notes 12, slide 84, covered at start of review lecture).

14. Consider the following protocol: write to servers R1, R2, and read from *either* servers R2, R3 or R4. Is this protocol a valid instance of quorum replication (as defined in class)?

ANSWER: _____

False; quorums (as described in lecture) must intersect.

15. Given that it takes 134ms for the speed of light to circle Earth's equator, two datacenters simultaneously executing conflicting serializable transactions on opposite sides of the planet can commit more than 7 transactions per second (without relying on quantum communication/entanglement).

ANSWER: _____

False; the two datacenters must exchange messages (at least 1 RTT) to commit transactions.

16. Instead of providing serializability, Amazon's Dynamo key-value store allows users to write their own conflict resolution policies in the presence of conflicting writes.

ANSWER: _____

True; we discussed this multiple times in the case of the shopping cart.

17. In distributed computing, it's always a good idea to assume the network is reliable.

ANSWER: _____

False; see the Fallacies of Distributed Computing!

18. In distributed computing, it's always a good idea to assume that latency is zero.

ANSWER: _____

False; see the Fallacies of Distributed Computing!

19. It is impossible to combine relational query processing with parameter server-style model training in a single system.

ANSWER: _____

False; this is exactly the model that TensorFlow provides.

20. As a Stanford student and CS245 graduate, you have the knowledge and the awesome power to build the next generation of data management systems.

ANSWER: _____

True!