CS245 Final Exam – Summer 1999

This exam is open book and notes. You have 180 minutes to complete the exam. There are four parts in the exam with a total of 150 points. Please read the instructions for each part carefully before starting to work on the questions. In Parts 1–3, if you wish to explain or demonstrate your solution to a problem for partial credit, you may use the space next to the question or the back of the pages (and warn us on the front). Please use this option sparingly, e.g., if you think the question is flawed or open to multiple interpretations, because we shall only be awarding partial credit in rare situations. In Part 4, clearness and conciseness will count in the score.

Print your name: ______________________________

In accordance with both the letter and spirit of the Honor Code, I have neither given or received assistance on this examination.

Signed: ______________________________

Part	Maximum	Points
I	18	
II	32	
III	40	
IV	60	
Total	150	

Part I. (18 points) Indicate whether each of the following statements is TRUE or FALSE. If you chose FALSE, please *correct* it or briefly *explain why.*

1. Data warehousing is more appealing than the mediation approach for OLAP systems which involve complex queries but few updates.

2. The real world actions in a transaction should be executed before the transaction commits.

3. Checkpointing can significantly reduce the recovery time when using undo logging.

4. The tree locking protocol can prevent deadlocks.

5. Owning an SIX lock is logically the same as owning both an S lock and an IX lock.

6. In a distributed locking system with data replications, each write action must write at least half of the copies of a data item to guarantee data consistency.

7. In a data warehouse, the more views we materialize, the better the system performance is.

8. Multi-key indexes are better than partitioned hash tables for answering range queries on multiple key attributes.

9. Double buffering is not useful when the time to read a block is less than the time to process a block.

Part II. (32 points) Circle the *one* correct answer from the listed multiple choices. Points $= 4 \times$ number right $- 1 \times$ number wrong.

1. Of the following three logging schemes:

   i. Undo logging
   ii. Redo logging
   iii. Undo/Redo logging

which ones can omit logging the ABORT record without degrading the recovery performance?
(a) i only (b) ii only (c) iii only (d) i, ii, and iii (e) none

---

2. Consider the following schedule of transactions T_1, T_2, and T_3.

$r_1(A)\ r_2(B)\ r_3(C)\ inc_2(C)$??? $inc_1(C)\ w_2(A)$

where *inc* stands for the increment action. Which of the following substitutions for ??? will make the schedule conflict serializable?
(a) $w_1(B)$ (b) $r_1(C)$ (c) $w_2(B)$ (d) $inc_3(B)$ (e) The schedule is not conflict serializable no matter what ??? is.

---

3. Consider two relations $R(A, B)$ and $S(B, C)$, and a materialized view $V = R \bowtie S$. Note that view V is *materialized* in the sense that its contents are stored in a table and kept consistent with the *base relations* R and S. We have the following pairs of queries (Q and Q'):

   i. $Q = \sigma_{A<C}(R \bowtie S)$, $Q' = \sigma_{A<C}(V)$
   ii. $Q = \sigma_{B=6}(R \bowtie S)$, $Q' = \sigma_{B=6}(V)$
   iii. $Q = \pi_A(V)$, $Q' = \pi_A(R)$

Suppose that we are answering query Q. In which of the above cases can we always improve the query performance by computing Q' instead?
(a) i only (b) iii only (c) i and ii only (d) i and iii only (e) ii and iii only

---

4. Consider two schedules S_1 and S_2 of the same set of transactions. Suppose we know that S_1 is conflict serializable, and the precedence graphs of the two schedules are the same, i.e., $P(S_1) = P(S_2)$. Of the following claims about S_2:

i. S_2 is a serial schedule.
ii. S_2 is conflict equivalent to S_1.
iii. S_2 is conflict serializable.

Which are true?
(a) i, ii, and iii (b) ii and iii only (c) ii only (d) iii only (e) none

---

5. Suppose we are using timestamp-based concurrency control, and there are two transactions T_1 and T_2 with time stamps 100 and 200, respectively. They try to execute the following sequence of actions in the (wall-clock-time) order shown below:

$r_1(A)\ w_1(A)\ r_2(A)\ r_2(B)\ w_1(B)\ c_1\ w_2(A)\ c_2$

Which of the following best characterizes the outcome of the transactions?
(a) All actions succeed.
(b) $w_2(A)$ fails and causes T_2 to abort.
(c) $w_1(B)$ fails and T_1 aborts; then $w_2(A)$ succeeds.
(d) $w_1(B)$ and $w_2(A)$ both fail and cause their transactions to abort.
(e) $w_1(B)$ fails and causes both T_1 and T_2 to abort.

---

6. Four hosts, C (the coordinator), X, Y, and Z are jointly running pieces of a transaction T, and attempt to engage in a two-phase commit of T. Unfortunately, after a crash both C and X become inaccessible from Y and Z. When Y and Z examine their logs to see what to do about T, consider the following situations:

i. Y's last log entry about T is $\langle T,$ `abort`$\rangle$, and Z's last log entry about T is $\langle T,$ `ready`$\rangle$.
ii. Y's last log entry about T is $\langle T,$ `abort`$\rangle$, and Z's log does not have an entry of $\langle T,$ `ready`$\rangle$ or $\langle T,$ `don't commit`$\rangle$.
iii. Y's last log entry about T is $\langle T,$ `commit`$\rangle$, and Z's log does not have an entry of $\langle T,$ `ready`$\rangle$ or $\langle T,$ `don't commit`$\rangle$.

Which of these situations could arise?
(a) i only (b) ii only (c) iii only (d) i and ii only (e) i, ii, and iii

---

Questions 7 and 8 below refer to the following sequence of actions of transactions T_1, T_2, and T_3:

$r_3(B)\ r_1(A)\ r_2(C)\ w_1(C)\ w_2(B)\ w_3(A)$

Transactions T_1, T_2, and T_3 initially arrive at time 100, 200, and 300, respectively. We make the following assumptions about timing:

i. A transaction requests the necessary lock (shared lock for read and exclusive lock for write) on a data item right before its action on that item is issued.
ii. If a transaction ever gets all the locks it needs, then it instantaneously completes work, commits, and releases its locks.
iii. If a transaction dies or is wounded, it instantaneously gives up its locks, and restarts only after all current transactions commit or abort.
iv. When a lock is released, it is instantaneously given to any transaction waiting for it (in a first-come-first-serve manner).

7. If a wait-die strategy is used to handle lock requests, in what order do the transactions finally commit?

(a) T_1, T_2, T_3 (b) T_1, T_3, T_2 (c) T_2, T_1, T_3 (d) T_2, T_3, T_1 (e) Deadlock occurs, and no one can commit.

---

8. If a wound-wait strategy is used to handle lock requests, in what order do the transactions finally commit?

(a) T_1, T_2, T_3 (b) T_1, T_3, T_2 (c) T_2, T_1, T_3 (d) T_2, T_3, T_1 (e) Deadlock occurs, and no one can commit.

Part III. (40 points) Write your final answer for each question.

1. Transactions should be "ACID". That is transactions should have the following properties:

A______________; C______________; I______________; D______________.

Fill in the above blanks, and briefly explain the first property.

Ans: ____

2. Consider the following schedule for the actions of transactions T_1, T_2, and T_3 on a data item A:

$r_1(A)\ w_2(A)\ r_3(A)\ w_1(A)\ w_3(A)$

Is this schedule view-serializable? If so, give the view-equivalent serial schedule. Otherwise, *briefly* explain why it is not view-serializable.

Ans: ____

Questions 3 and 4 refer to a relation R with 4000 records. Suppose a B+ tree dense index is built on R's key attribute. A block can either hold ten records of relation R or it can hold a B+ tree node with 20 keys and 21 pointers.

3. At least how many blocks are needed to hold the entire B+ tree index?

Ans: ____

4. Suppose the B+ tree index has exactly the number of blocks as computed above, and a 10-block buffer is available for holding some record blocks or index nodes in memory (in advance). We want to retrieve a tuple in R using its key value. What is the best query plan you can think of for this query? What is the expected number of I/Os needed for the query using your plan?

Ans: ____

____

____

5. Consider a file with 100,000 fixed-size records, each 100 bytes long. Suppose we want to store this file in 8KB disk blocks with no spanning records, and we want all the blocks to be at most 80% full in order to allow efficient insertions. At least how many blocks do we need to store the entire file?

Ans: ____________________________________

6. Consider a disk that has 10 surfaces, each with 100 tracks. Each track is divided into 36 sectors, and a sector holds 2048 bytes. Blocks consist of 2 sectors. 20% of the circumference of each track is occupied by gaps between the sectors. The disk rotates at 6000rpm, i.e., one rotation every 10 milliseconds. What is the transfer time for one block?

Ans: ____________________________________

7. Consider the validation concurrency control protocol. Suppose we have three transactions T_1, T_2, and T_3 and three database elements A, B, and C. The read sets and write sets of the transactions are listed below:

$RS(T_1) = \{A, B\}$; $WS(T_1) = \{B\}$
$RS(T_2) = \{A, C\}$; $WS(T_2) = \{A\}$
$RS(T_3) = \{C\}$; $WS(T_3) = \{B, C\}$

We use S_i, V_i, and F_i to represent that a transaction T_i starts, attempts to validate, and finishes, respectively. Given the following sequence of events:

$S_1\ S_2\ V_1\ S_3\ V_3\ F_1\ V_2\ F_2\ F_3$

what is the outcome of each transaction (commit, if successfully validated; abort, otherwise)?

Ans: ____________________________________

8. Consider a linear hashing structure with $i = 3$ and 6 actual buckets allocated. Suppose that each data block can hold at most r records, and the threshold for allocating new buckets is $T = 0.8r$. Assuming k-bit hash key with no duplicates, at most how many overflow blocks can a bucket have?

Ans: ____________________________________

Part IV. (60 points) Answer the following questions and *briefly* show your reasoning steps when necessary. Clearness and conciseness will be considered when grading this part.

1. (20 points) Consider a DBMS that uses *undo/redo logging* with non-quiescent checkpoints for recovery and uses the *basic two-phase locking protocol* (with the locking rules 1, 2, and 3 as described in class) for concurrency control. Suppose A and B are two database elements and their initial values are both 0. Three transactions T_1, T_2, and T_3 access the values of A and B. An incomplete log file right before a system crash is given as follows, in which the log entries are in the format ⟨Tid, Variable, Old value, New value⟩.

⟨T_1, `start`⟩
⟨T_2, `start`⟩
⟨T_1, A, 0, 1⟩
⟨T_2, ?, ?, 3⟩
⟨`checkpoint start`⟩
⟨?, B, ?, 4⟩
⟨T_3, `start`⟩
⟨T_1, B, 4, 6⟩
⟨`checkpoint end`⟩
⟨T_1, `commit`⟩
⟨T_3, A, ?, 8⟩
<== system crash (but no disk failure)

(NOTE: Read all three questions in this problem before answering one of them.)

(a) Suppose it is possible for a transaction to write a data item multiple times. Give all possible complete logs (up to the crash), and number them as $LG1$, $LG2$, ...

(b) Based on each log file LG_i, list all possible values of A and B on disk (separately) *before* recovery.

(c) Suppose each transaction always reads an item right before writing it. For each log file LG_i, answer the following questions. Is the schedule implied by that log recoverable? Does it avoid cascading rollback?

2. (18 points) Consider a DBMS that uses multi-granularity locks for concurrency control. Using this DBMS, we see two levels of database elements: the relation level and the tuple level. Suppose that we have a relation R containing some tuples, and a sequence of actions as follows:

(1) Transaction T_1 insert a tuple t_1 into R
(2) Transaction T_2 reads a tuple t_2 in R
(3) Transaction T_3 scans relation R
(4) Transaction T_4 writes tuple t_2 in R
(5) Transaction T_5 deletes tuple t_2 from R
(6) Transaction T_6 deletes relation R

We make the following assumptions about timing:

i. Action (1) is issued to the scheduler at time 0, and the other actions then arrive in the order shown above at an interval of τ.
ii. Each transaction requests the necessary lock right before its action is issued, and releases the lock right after its action completes.
iii. Whenever a new lock request arrives or a lock is released, the system examines the waiting queue for that lock and grants the lock to as many compatible requests as it can. The locks are granted to the transactions in a first-come-first-serve manner.
iv. Each action completes 10τ after it obtains the lock on relation R.

(a) For each transaction, list the *weakest* type of lock (S, X, IS, IX, SIX) that the transaction should hold on relation R before its action can be executed?

transaction	**lock on R**
T_1	
T_2	
T_3	
T_4	
T_5	
T_6	

(b) Draw the lock table structure as described in class or in the textbook for all the locked data elements in the database, after action(1) completes and before action (2) completes. Suppose all actions have arrived at the scheduler and requested their locks at this point.

(c) Fill out the following time table for the lock and unlock actions on relation R in the output schedule from the scheduler. You may use l_i and u_i to represent the actions that transaction i obtains and releases the lock on relation R, respectively.

time	**actions**
0	l_1

3. (10 points) Recall that an *outerjoin* of relations R and S is the union of the tuples from: (i) a regular natural join of R and S, (ii) the tuples from the left relation R that did not match any from the right relation S, padded with `NULL`s, (iii) tuples from the right relation S that did not match any from the left relation R padded with `NULL`s. For example, given R and S as follows:

R:

A	B
1	2
2	5

S:

B	C
1	4
2	6

R *outerjoin* S is

A	B	C
1	2	6
2	5	`NULL`
`NULL`	1	4

We have introduced a hash-join algorithm in class for computing the natural join of two relations. Can we use the same algorithm to compute outerjoins? If so, explain how. Otherwise, describe how you would extend the hash-join algorithm (and the data structures it uses, if necessary) to compute the outerjoin of two relations. You can use the above example relation schemas when explaining your answer.

4. (12 points) Consider a distributed system with two sites X and Y connected through a network. Relation $R(A, B, C)$ resides at site X and relation $S(C, D)$ resides at site Y. Suppose the statistics of the relations are as follows.

- $n_R = 1000$ (number of tuples in R)
- $n_S = 100,000$ (number of tuples in S)
- $s_A = s_B = s_C = s_D = 100$ bytes (size of each attribute)
- $V(C, R) = 200$ (image size of attribute C in R)
- $V(C, S) = 4000$ (image size of attribute C in S)

We want to compute the natural join of the two relations: $T = R \bowtie S$, and we want the resulting relation T to be output at site X. We are given two plans as follows.

Plan 1: Copy relation S from site Y to X, compute the join at site X, and output the result.

Plan 2: Copy relation R from site X to Y, compute the join at site Y, and move the resulting relation back to site X for output.

(a) Estimate the total number of bytes transfered between sites X and Y for plans 1 and 2. Which plan incurs less data movement?

(b) Can you design another plan that can further reduce the amount of data transfered between the two sites? *Briefly* describe your plan, and estimate the total number of bytes transfered between sites X and Y using your plan.