CS 441: Final Exam, Fall 1998

1. True or false: Justify.
   (a) Sort-merge join is pipelinable.
   (b) Using two-version locking based concurrency control, a query can always finish without any rollbacks.
   (c) Timestamp-based concurrency control avoids deadlocks.
   (d) Extendable hashing works fine as long as only insertions occur.
   (e) In commercial query languages, duplication elimination is the default.
   (f) Theta Join operations are commutative.
   (g) In a database where no new tuples are added, repeatable reads is equivalent to serializability.
   (h) Stable storage is a mirage.
   (i) Since checkpoints reduce recovery time, they should be takes as often as possible.
   (j) This course was not too bad! ...20

2. Give
   (a) one reason why (while computing $r_1 \bowtie r_2 \bowtie r_3$) join orders matters.
   (b) one use for intention locks.
   (c) one benefit of block interleaved distributed parity RAID.
   (d) one reason why cascading aborts should be avoided.
   (e) one optimization technique that pertains to the *select* operator.
   (f) one reason why a cohort logs its changes before responding positively to a master's `prepare to commit` message.
   (g) one advantage of *group commit*, i.e., committing several transactions together (the system waits until n transactions want to commit).
   (h) one disadvantage of group commit. ...16

3. What are
   (a) the three main indicators of disk speed?
   (b) the two main performance metrics in a transaction processing system?
   (c) two additional metrics in a parallel transaction processing system?
   (d) two differences between B-trees and B^+ trees?
   (e) two uses of the database log? ...10

4. Suppose we wish to evaluate $r_1 \bowtie r_2$ where r_1 and r_2 are stored at sites s_1 and s_2, respectively. The result of $r_1 \bowtie r_2$ is to be made available at s_1. R_1 and R_2 are the schemas for r_1 and r_2.

   This question relates to how $r_1 \bowtie r_2$ should be computed. If there are many tuples of r_2 that do not join with any tuple of r_1 then shipping r_2 to s_1 entails shipping tuples that fail to contribute to the result. It is desirable to remove such tuples before shipping data to s_1.

   Consider the following code that is based on the above idea.

   (a) At s_1, compute $temp_1 \leftarrow \Pi_{expr}(r_1)$,
   (b) Ship $temp_1$ from s_1 to s_2.
   (c) At s_2, compute $temp_2 \leftarrow r2 \bowtie temp_1$.
   (d) Ship $temp_2$ from s_2 to s_1.
   (e) At s_1, compute $r_1 \bowtie temp_2$.

   - What should *expr* (in step (a) above) be for the result of step (e) to be $r_1 \bowtie r2$? ...3
   - Demonstrate the correctness of your answer. ...3
   - What are the factors that contribute to the overall cost of $r_1 \bowtie r2$ in a 2-node system? ...2
   - Derive an expression for the total cost of $r_1 \bowtie r2$ in terms of these factors – given the above strategy. ...2
   - Explain how/why cost savings occur using the above (5-step) join strategy. ...1

5. A system desires to achieve serializability and failure atomicity. Still, it allows a transaction to the read the dirty data of another transaction. Answer the following questions assuming that a $force, no - steal$ protocol is used for transaction execution. (Give a short explanation for each answer.)
   (a) t_1 writes to x and t_2 reads x before t_1 commits. If t_1 were to abort, what should happen to t_2?
   (b) t_1 reads x and t_2 writes to x before t_1 commits. If t_1 aborts, what should happen to t_2?

(c) t_1 writes to x and t_2 writes to x before t_1 commits. What should the system do if t_2 wants to commit and t_1 has not yet committed? ...5

6. In the *nested transaction model*, a transaction T consists of a set S of subtransactions. A subtransaction s_1 in S can abort without causing the abortion of T – in this case, T might retry s_1 or try an alternative subtransaction. When a subtransaction s_1 commits, its changes are not made permanent, instead, T takes over those changes. T can abort even if s_1 commits.
   - The commit of a subtransaction is usually called *conditional* commitment. Why? ...2
   - Suppose concurrency control is done using locks. Indicate what the locking protocol would do to effect the above semantics of subtransaction commitment. ...4
   - Give an example of an application where it is preferable to code it as a nested transaction as opposed to a traditional atomic transaction. ...2
   - Give one disadvantage of nested transactions. ...1

7. ARIES incorporates a number of nifty techniques to perform efficient recovery. Some of these are listed below. For each indicate why/where/how the technique is useful
   - ARIES combines the analysis pass and the redo pass. ...1
   - After all the physical (page-level) operations corresponding to a logical operation are complete, it writes an *operation-complete* log record. ...2
   - ARIES maintains a *a backward chain* of pointers connecting the log records pertaining to a transaction. ...2
   - Whenever an undo (corresponding to log record L) is performed during recovery, an *undo log record* is added to the log. This log record points to the predecessor of L. ...3

8. When a transaction T_i wants to lock data item D which is held in a conflicting mode by some transaction T_j, T_i is added to the "wait queue" of D.
   A transaction T holding an S-lock on D may later on request X-lock on D by using the `Upgrade Lock` request.
   What is the problem that may arise if T is added to the normal "wait queue" on D ?
   Suggest a way to solve the problem ...5

9. Consider a database that includes *atomic* `Increment` and `Decrement` operations in addition to `Read` and `Write`.
   `Increment` (x, val) : add val to data item x.
   `Decrement` (x, val) : subtract val from data item x.
   Neither operation returns any value. The lock compatibility matrix is as given below.

	`Read`	`Write`	`Increment`	`Decrement`
`Read`	True	False	False	False
`Write`	False	False	False	False
`Increment`	False	False	True	True
`Decrement`	False	False	True	True

   - Give a real life example where use of `Increment/Decrement` operation instead of `Write` operation may greatly increase concurrency.
   - Under what conditions do `Increment` and `Decrement` commute?
   - If all the transactions lock data items adhering to above matrix, does 2PL ensure serializability? Explain your answer.
   - Consider that one of the conditions pointed out by you in (b) above does not hold. What problem may arise ? How would you solve it ? ...6

10. Consider a distributed database system. Give an example of two transactions which visit multiple sites to show that even if at each site they adhere to 2PL, unless the whole transaction uses 2PL globally, the transactions may not be conflict serializable. (Give the sequence in which locks are set and released by the transactions well as the sequence in which `Reads` and `Writes` are performed.) ...5

11. Suppose you use a cache to maintain a set of query results (for simplicity, assume they are of the same size). Give a situation where it is preferable to throw away the result of a frequently posed query in preference to a less frequently accessed query result. ...5

Total Marks = 100