Part A
DB Qual Exam – Spring 2002

Basic Part (20 points)

Problem A.1 (5 points)

Give two reasons why a database system should implement spanned records.

Problem A.2 (15 points)

Consider a database system that implements *group commit*. In this system, the commit process is divided into two phases: First, a transaction T_i performs an in-memory commit, m_i, where it releases all its locks and writes a commit record to the in-memory log buffer. Second, at some later time, T_i's commit record is flushed to disk, together with the records of other transactions j_1, j_2, ... We represent this group commit action by $g(i, j_1, j_2, ...)$ and we can assume the action is atomic. Note that this action is the true commit point of T_i, and $g(i, j_1, j_2, ...)$ is equivalent to c_i, c_{j_1}, c_{j_2}....

The group commit machinery guarantees two properties of the resulting schedule S:

1. For every transaction (that committed) T_i, $m_i <_S g(...i...)$.
2. If for any T_j, $m_i <_S m_j <_S g(...i...)$, then the i commit must also include j, i.e., the action is $g(...i, j...)$.

These properties are in addition to the "obvious" that state that a transaction T_i has at most one m_i action, and at most one $g(...i...)$ action, and that if a transaction aborts, then it performs no m_i or $g(...i...)$ actions. Also, transactions use two-phase locking, with shared and exclusive locks.

(a) Are the schedules produced recoverable (as per the definition in the Complete Book)? If YES, please give an informal proof. If NO, please give a simple counter-example and explain why the lack of recoverability will not be a problem in practice (for this system).

(b) Do the schedules produced avoid cascading rollbacks (ACR) (as per the definition in the Complete Book)? If YES, please give an informal proof. If NO, please give a simple counter-example and explain why the lack of the ACR property will not be a problem in practice (for this system).

Advanced Part (40 points)

Problem A.3 (7 points)

Consider two relations, $R(A, B, C)$ at site X and $S(C, D)$ at site Y. You want to compute the join $R \bowtie S$, and have the result stored at Y.

(a) Briefly describe how you would use a semijoin to perform the distributed join. Start processing at site Y.

(b) Under what conditions is the semijoin scheme better than one where R is sent to Y and the full join performed at Y.

(c) Briefly describe how you would use bit arrays (as described in one of the readings) to improve the semijoin scheme of Part (a). What would be the savings?

Problem A.4 (3 points)

As surely you noticed when tracking down the reading on the TPC-D benchmark, that benchmark is now obsolete. Please *very briefly* describe the benchmarks that replace it (one sentence per benchmark), and why the replacement was done (1 sentence).

Problem A.5 (15 points)

One of the papers on the qual reading list gives a formula for the probability that a transaction deadlocks.

(a) What is the formula given for a *single* database system (node)?

(b) Suppose that DB_Size is one. In such a case, variable Actions should also equal 1. Please discuss what the formula predicts for this case. If you think the prediction is accurate, then discuss its implications. If you think the formula fails in this case, please explain the main reason the formula does not work.

(c) Next, consider a wait-die deadlock prevention mechanism. Using a similar model and assumptions, give a formula for the probability that a transaction aborts (dies) in a single database system. Briefly explain how you derive the formula.

Problem A.6 (15 points)

(a) Consider two transactions, T_1 and T_2. Transaction T_1 consists of three actions, $w_1(A)$ and $w_1(B)$, $w_1(C)$ *not necessarily in this order.* Similarly, transaction T_2 consists of actions $w_2(A)$, $w_2(B)$ and $w_2(C)$ in any order. How many schedules of T_1 and T_2 are possible? (Do *not* consider commit or abort actions as part of the schedules, just the actions given.)

(b) For T_1 and T_2, how many of the possible schedules are serializable? Please explain. (Hint: Do *not* try enumerating all schedules; you will spend too much time.)

(c) Next consider two different transactions, T_3 and T_4. Transaction T_3 consists of two actions, $r_3(A)$ and $w_3(B)$, *not necessarily in this order.* Similarly, transaction T_4 consists of actions $r_4(A)$ and $w_4(B)$ in either order. How many schedules are there in this case, and how many of them are serializable? Please explain.