1. Question ID 25: Consider the following transaction schedule:

$r_1(X), r_2(X), r_3(X), r_1(Y), w_2(Z), r_3(Y), w_3(Z), w_1(Y)$

This schedule is conflict-equivalent to some or all serial schedules. Determine which serial schedules it is conflict-equivalent to, and then identify the true statement from the list below.

Question Explanation: First, observe that the schedule is not serial; there are several points of interleaving of transactions. However, it is conflict-serializable. If we construct the conflict graph, we find only that T_2 precedes T_3 because of the order in which Z is written, and T_3 precedes T_1 because the latter writes Y before the former reads Y. Thus, $T_2T_3T_1$ is the only possible serial order that is conflict-equivalent to the given schedule.

2. Question ID 27: Here are two transactions:

T_1: $r_1(X), w_1(X), r_1(Y), w_1(Y)$
T_2: $r_2(X), r_2(Y), w_2(X), w_2(Y)$

We wish to insert lock (l) and unlock (u) operations into the read and write steps of T_1 and T_2, and then schedule all of these operations using a two-phase-locking scheduler. Several different sequences of the 16 r, w, l, and u operations are allowed by this scheduler. Your problem is to find a simple way to recognize whether a schedule is allowable. Demonstrate your understanding by identifying the schedule that would be allowed by a two-phase-locking scheduler, from the list below.

Question Explanation: The desired schedule must obey the following rules:

1. The actions of each transaction must appear in the order given.
2. A transaction cannot read or write a data item (X or Y) without holding the lock on that item; i.e., the lock action occurs before and the unlock occurs after the read or write.
3. Both transactions cannot hold a lock on the same data item at the same time. That is, the second can only lock an item after the first unlocks it.
4. No transaction may lock an item after it has unlocked another item (the two-phase-locking condition).

3. Question ID 29: Which of the following schedules is serializable?

Question Explanation: A schedule is serializable if it is equivalent to some serial schedule. The usual test for serializability is to check for *conflict serializability*. We build a precedence graph in which transaction T precedes transaction S if T accesses some value before S, and at least one of those accesses is a write. The schedule is (conflict) serializable if and only if this graph has no cycles. Thus, we can check for serializability by looking at every data element accessed in the schedule, and seeing if its accesses imply any precedences among transactions. For example, one of the correct answers is:

$r_2(X),r_1(X),w_2(Y),r_2(Z),r_1(Y),w_2(Z),c_2,w_1(X),c_1$

If we look at accesses to X, we see that T_1 writes X after T_2 reads X. Thus, T_2 must precede T_1. Y is written by T_2 before T_1 reads Y, so again, T_2 must precede T_1. Finally, Z is accessed only by T_2, so it imposes no constraints. The serial order T_2, T_1 is thus equivalent to the given schedule, and the schedule is serializable.

NOTE: The c1, c2 in this question denote COMMIT actions (by T1 resp. T2). We did not include explicit commits in schedules so you can ignore these actions.