Homework 3 for Distributed Database Systems

1. Which of the following schedules are conflict equivalent? (Ignore the commit "C" and abort "A" commands)? Which of the following schedules (S1,S2,S3,S4) are serializable?
   S1 = W2(x), W1(x), R3(x), R1(x), C1, W2(y), R3(y), R3(z), C3, R2(z), C2
   S2 = R3(z), R3(y), W2(y), R2(z), W1(x), R3(x), W2(x), R1(x), C1, C2, C3
   S3 = R3(z), W2(x), W2(y), R1(x), R3(x), R2(z), R3(y), C3, W1(x), C2, C1
   S4 = R2(z), W2(x), W2(y), C2, W1(x), R1(x), A1, R3(x), R3(z), R3(y), C3

2. Compare and analyze the relative merits of centralized and hierarchical deadlock detection approaches in a distributed DBMS.

3. Consider the following modification to a local wait-for graph: Add a new node T_{ex}, and for every transaction T_i that is waiting for a lock at a certain external site, add the edge $T_i \rightarrow T_{ex}$. Also add an edge $T_{ex} \rightarrow T_i$ if a transaction executing at a certain external site is waiting for T_i to release a lock at this site. If there is a cycle in the modified local waits-for graph that does not involve T_{ex}, what can you conclude? If there exists a cycle involving T_{ex}, what can you conclude?

4. Whenever the local waits-for graph at a certain site suggests that there might be a global deadlock, send the local waits-for graph to the next site. At that site, combine the received graph with the local waits-for graph. If this combined graph does not indicate a deadlock, ship it on to the next, next site, and so on, until either a deadlock is detected or we are back at the site that originated this round of deadlock detection. Is this scheme guaranteed to find a global deadlock if one exists?