Exercise 1: Consider a database organized in the form of a rooted tree. Suppose that we insert a dummy vertex between each pair of vertices. Show that, if we follow the tree protocol on the new tree, we get better concurrency than if we follow the tree protocol on the original tree.

Answer. Consider two nodes A and B,where A is a parent of B. Let dummy vertex D be added between A and B. Consider a case where transaction T2 has a lock on B, and T1, which has a lock on A wishes to lock B, and T3 wishes to lock A. With the original tree, T1 cannot release the lock on A until it gets the lock on B. With the modified tree, T1 can get a lock on D, and release the lock on A, which allows T3 to proceed while T1 waits for T2. Thus, the protocol allows locks on vertices to be released earlier to other transactions, instead of holding them when waiting for a lock on a child.

Exercise 2: Show by example that there are schedules possible under the tree protocol that are not possible under the two-phase locking protocol, and vice versa.

Answer. Consider a tree-structured database graph (A -> B -> C), where A is the root node, and A is the parent of B. B is the parent of node C.

Schedule possible under tree protocol but not under 2PL.

T_1	T_2
Lock (A)	
Lock (B)	
Unlock (A)	
	Lock (A)
Lock (C)	
Unlock (B)	
	Lock (B)
	Unlock (A)
	Unlock (B)
Unlock (C)	

Schedule possible under 2PL but not tree protocol:

T_1	T_2
Lock (A)	
	Lock (B)
Lock (C)	
	Unlock (B)
Unlock (A)	
Unlock (C)	

Exercise 3: Consider a schedule of 3 transactions, T_1, T_2, and T_3 that access the database elements, A , B, and C.

Is the schedule feasible under **Timestamp protocol**? Will all the read and write operations run successfully? Explain your answer with proper justification.

The timestamps of transactions are: T1 = 200, T2 = 150 and T3 = 175. Initially, the elements A, B, C have read and write timestamps (denoted by RT and WT) as zero.

T_1	T_2	T_3
Read (B)		
	Read (A)	
		Read (C)
Write (B)		
Write (A)		
	Write (C)	
		Write (A)

Answer: In the first action, T_1 reads B. Since the write time of B is less than the timestamp of T_1 , this read is physically realizable and allowed to happen. The read time of B is set to 200, the

timestamp of T_1. The second and third read actions similarly are legal and result in the read time of each database element being set to the timestamp of the transaction that read it.

At the fourth step, T_1 writes B. Since the read time of B is not bigger than the timestamp of T_1, the write is physically realizable. Since the write time of B is no larger than the timestamp of T_1, we must actually perform the write. When we do, the write time of B is raised to 200, the timestamp of the writing transaction T_1.

Next, T_2 tries to write C. However, C was already read by transaction T_3, which theoretically executed at time 175, while T_2 would have written its value at time 150. Thus, T_2 is trying to do something that's physically unrealizable, and T_2 must be rolled back.

The last step is the write of A by T_3. Since the read time of A, 150, is less than the timestamp of T_3, 175, the write is legal. However, there is already a later value of A stored in that database element, namely the value written by T_1, theoretically at time 200. **Thus, T_3 is rolled back,**

T_1 (200)	**T_2 (150)**	**T_3 (175)**	**A**	**B**	**C**
Read (B)				RT=200	
	Read (A)		RT=150		
		Read (C)			RT=175
Write (B)				WT=200	
Write (A)			WT=200		
	Write (C)				
	Abort				
		Write (A)			

Exercise 4: Consider the timestamp-ordering protocol, and two transactions, one that writes two data items p and q, and another that reads the same two data items. Give a schedule whereby the timestamp test for a write operation fails and causes the first transaction to be restarted, in turn causing a cascading abort of the other transaction. Show how this could result in starvation of both transactions.

Answer: Consider two transactions T_1 and T_2 shown below.

T_1	T_2
Write (p)	
	Read (p)
	Read (q)
Write (q)	

Let $TS(T_1) < TS(T_2)$ and let the timestamp test at each operation except write(q) be successful. When transaction T_1 does the timestamp test for write(q) it finds that $TS(T_1) <$ R-timestamp(q), since $TS(T_1) < TS(T_2)$ and R-timestamp(q) = $TS(T_2)$. Hence the write operation fails and transaction T_1 rolls back. The cascading results in transaction T_2 also being rolled back as it uses the value for item p that is written by transaction T_1. If this scenario is exactly repeated every time the transactions are restarted, this could result in starvation of both transactions.