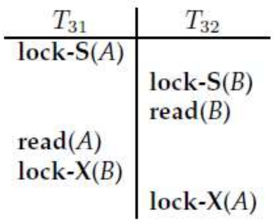
15.1 Show that the two-phase locking protocol ensures conflict serializability,and that transactions can be serialized according to their lock points.
Answer:

15.2 Consider the following two transactions:
*T*34: read(*A*);
read(*B*);
if *A* = 0 **then** *B* := *B* + 1;
write(*B*).
*T*35: read(*B*);
read(*A*);
if *B* = 0 **then** *A* := *A* + 1;
write(*A*).

Add lock and unlock instructions to transactions *T*31 and *T*32, so that they observe the two-phase locking protocol. Can the execution of these transactions result in a deadlock?

Answer:
a. Lock and unlock instructions:
*T*31: **lock-S**(*A*)
read(*A*)
lock-X(*B*)
read(*B*)
if *A* = 0
then *B* := *B* + 1
write(*B*)
unlock(*A*)
unlock(*B*)
*T*32: **lock-S**(*B*)
read(*B*)
lock-X(*A*)
read(*A*)
if *B* = 0
then *A* := *A* + 1
write(*A*)
unlock(*B*)
unlock(*A*)
b. Execution of these transactions can result in deadlock. For example, consider the following partial schedule:



T_{31}	T_{32}
lock-S(*A*)	
	lock-S(*B*)
	read(*B*)
read(*A*)	
lock-X(*B*)	
	lock-X(*A*)

The transactions are now deadlocked.

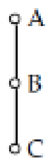
15.3 What benefit does rigorous two-phase locking provide? How does it compare with other forms of two-phase locking?
Answer: Rigorous two-phase locking has the advantages of strict 2PL. In addition it has the property that for two conflicting transactions, their commit order is their serializability order. In some systems users might expect this behavior.

15.4 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.

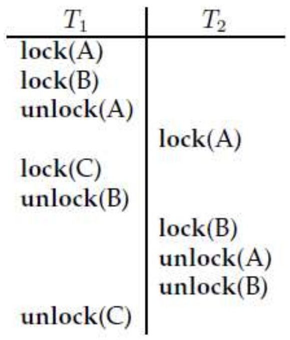
Answer: The proof is in Buckley and Silberschatz, "Concurrency Control in Graph Protocols by Using Edge Locks," Proc. ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems, 1984.

15.5 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 the tree-structured database graph given below.

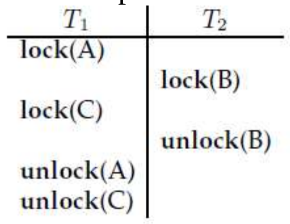


Schedule possible under tree protocol but not under 2PL:



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 under tree protocol:



T_1	T_2
lock(A)	
	lock(B)
lock(C)	
	unlock(B)
unlock(A)	
unlock(C)	

15.6 Consider the following extension to the tree-locking protocol, which allows both shared and exclusive locks:
• A transaction can be either a read-only transaction, in which case it can request only shared locks, or an update transaction, in which case it can request only exclusive locks.

- Each transaction must follow the rules of the tree protocol. Read-only transactionsmaylock any data itemfirst,whereas update transactions must lock the root first.

Show that the protocol ensures serializability and deadlock freedom.

Answer: The proof is in Kedem and Silberschatz, “Locking Protocols: From Exclusive to Shared Locks,” JACM Vol. 30, 4, 1983.

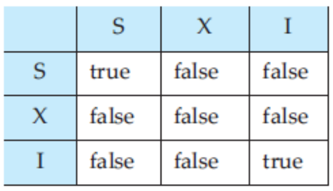
15.7 Consider the following graph-based locking protocol, which allows only exclusive lock modes, and which operates on data graphs that are in the form of a rooted directed acyclic graph.
- A transaction can lock any vertex first.
- To lock any other vertex, the transaction must be holding a lock on the majority of the parents of that vertex.Show that the protocol ensures serializability and deadlock freedom.

Show that the protocol ensures serializability and deadlock freedom.
Answer: The proof is in Kedem and Silberschatz, “Controlling Concurrency Using Locking Protocols,” Proc. Annual IEEE Symposium on Foundations of Computer Science, 1979.

15.8 Consider the following graph-based locking protocol, which allows only exclusive lock modes and which operates on data graphs that are in the form of a rooted directed acyclic graph.
•A transaction can lock any vertex first.
•To lock any other vertex, the transaction must have visited all the
parents of that vertex and must be holding a lock on one of the parents of the vertex. Show that the protocol ensures serializability and deadlock freedom.

Show that the protocol ensures serializability and deadlock freedom.
Answer: The proof is in Kedem and Silberschatz, “Controlling Concurrency Using Locking Protocols,” Proc. Annual IEEE Symposium on Foundations of Computer Science, 1979.

15.9 Locking is not done explicitly in persistent programming languages. Rather, objects (or the corresponding pages) must be locked when the objects are accessed. Most modern operating systems allow the user to set access protections (no access, read, write) on pages, and memory access that violate the access protections result in a protection violation (see the Unix `mprotect` command, for example). Describe how the accessprotection mechanism can be used for page-level locking in a persistent programming language.



	S	X	I
S	true	false	false
X	false	false	false
I	false	false	true

Figure 15.23 Lock-compatibility matrix.

Answer: The access protection mechanism can be used to implement page level locking. Consider reads first. A process is allowed to read a page only after it read-locks the page.

This is implemented by using `mprotect` to initially turn off read permissions to all pages, for the process. When the process tries to access an address in a page, a protection violation occurs. The handler associated with protection violation then requests a read lock on the page, and after the lock is acquired, it uses `mprotect` to allow read access to the page by the process, and finally allows the process to continue. Write access is handled similarly.

15.10 Consider a database system that includes an atomic **increment** operation, in addition to the read and write operations. Let *V* be the value of data item *X*. The operation

increment(*X*) by *C*

sets the value of*X*to*V*+ *C* in an atomic step. The value of*X*is not available to the transaction unless the latter executes a read(*X*). Figure 15.23 shows a lock-compatibility matrix for three lock modes: share mode, exclusive mode, and incrementation mode.
a. Show that, if all transactions lock the data that they access in the corresponding mode, then two-phase locking ensures serializability.
b. Show that the inclusion of **increment** mode locks allows for increased concurrency. (Hint: Consider check-clearing transactions in our bank example.)

	S	X	I
S	true	false	false
X	false	false	false
I	false	false	true

Answer: The proof is in Korth, "Locking Primitives in a Database System," JACM Vol. 30, 1983.

15.11 In timestamp ordering,**W-timestamp**(*Q*) denotes the largest timestamp of any transaction that executed write(*Q*) successfully. Suppose that, instead, we defined it to be the timestamp of the most recent transaction to execute write(*Q*) successfully.Would this change in wording make any difference? Explain your answer.

Answer: It would make no difference. The write protocol is such that the most recent transaction to write an item is also the one with the largest timestamp to have done so.

15.12 Use of multiple-granularity locking may require more or fewer locks than an equivalent system with a single lock granularity. Provide examples of both situations, and compare the relative amount of concurrency allowed.

Answer: If a transaction needs to access a large a set of items, multiple granularity locking requires fewer locks, whereas if only one item needs to be accessed, the single lock granularity system allows this with just one lock. Because all the desired data items are locked and unlocked together in the multiple granularity scheme, the locking overhead is low, but concurrency is also reduced.

15.13 Consider the validation-based concurrency-control scheme of Section 15.5. Show that by choosing Validation(*Ti*), rather than Start(*Ti*), as the timestamp of transaction *Ti* , we can expect better response time, provided that conflict rates among transactions are indeed low.

Answer: In the concurrency control scheme of Section 16.3 choosing **Start**(*Ti*) as the timestamp of *Ti* gives a subset of the schedules allowed by choosing **Validation**(*Ti*) as the timestamp. Using **Start**(*Ti*) means that whoever started first must finish first. Clearly transactions could enter the validation phase in the same order in which they began executing, but this is overly restrictive. Since choosing **Validation**(*Ti*) causes fewer nonconflicting transactions to restart, it gives the better response times.

15.14 For each of the following protocols, describe aspects of practical applications that would lead you to suggest using the protocol, and aspects that would suggest not using the protocol:

- Two-phase locking.
- Two-phase locking with multiple-granularity locking.
- The tree protocol.
- Timestamp ordering.
- Validation.
- Multiversion timestamp ordering.
- Multiversion two-phase locking.

Answer:

- Two-phase locking: Use for simple applications where a single granularity is acceptable. If there are large read-only transactions, multiversion protocols would do better. Also, if deadlocks must be avoided at all costs, the tree protocol would be preferable.
- Two-phase locking with multiple granularity locking: Use for an application mix where some applications access individual records and others access whole relations or substantial parts thereof. The drawbacks of 2PL mentioned above also apply to this one.
- The tree protocol: Use if all applications tend to access data items in an order consistent with a particular partial order. This protocol is free of deadlocks, but transactions will often have to lock unwanted nodes in order to access the desired nodes.
- Timestamp ordering: Use if the application demands a concurrent execution that is equivalent to a particular serial ordering (say, the order of arrival), rather than *any* serial ordering. But conflicts are handled by roll-back of transactions rather than waiting, and schedules are not recoverable. To make them recoverable, additional overheads and increased response time have to be tolerated. Not suitable if there are long read-only transactions, since they will starve. Deadlocks are absent.
- Validation: If the probability that two concurrently executing transactions conflict is low, this protocol can be used advantageously to get better concurrency and good response times with low overheads. Not suitable under high contention, when a lot of wasted work will be done.
- Multiversion timestamp ordering: Use if timestamp ordering is appropriate but it is desirable for read requests to never wait. Shares the other disadvantages of the timestamp ordering protocol.
- Multiversion two-phase locking: This protocol allows read-only transactions to always commit without ever waiting. Update transactions follow 2PL, thus allowing recoverable schedules with conflicts solved by waiting rather than roll-back. But the problem of deadlocks comes back, though read-only transactions cannot get involved in them. Keeping multiple versions adds space and time overheads though, therefore plain 2PL may be preferable in low conflict situations.

15.15 Explain why the following technique for transaction execution may provide better performance than just using strict two-phase locking: First execute the transaction without acquiring any locks and without performing any writes to the database as in the validation-based techniques, but unlike the validation techniques do not perform either validation or writes on the database. Instead, rerun the transaction using strict twophase locking. (Hint: Consider waits for disk I/O.)

Answer: TO BE SOLVED

15.16 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. (Such a situation, where two or more processes carry out actions, but are unable to complete their task because of interaction with the other processes, is called a **livelock**.)

Answer: TO BE SOLVED

15.17 Devise a timestamp-based protocol that avoids the phantom phenomenon.

Answer: In the text, we considered two approaches to dealing with the phantom phenomenon bymeans of locking. The coarser granularity approach obviously works for timestamps as well. The *B*+-tree index based approach can be adapted to timestamping by treating index buckets as data items with timestamps associated with them, and requiring that all read accesses use an index. We now show that this simple method works. Suppose a transaction *Ti* wants to access all tuples with a particular range of search-key values, using a *B*+- tree index on that search-key. *Ti* will need to read all the buckets in that index which have key values in that range. It can be seen that any delete or insert of a tuple with a key-value in the same range will need to write one of the index buckets read by *Ti*. Thus the logical conflict is converted to a conflict on an index bucket, and the phantom phenomenon is avoided.

15.18 Suppose that we use the tree protocol of Section 15.1.5 to manage concurrent access to a B+-tree. Since a split may occur on an insert that affects the root, it appears that an insert operation cannot release any locks until it has completed the entire operation. Under what circumstances is it possible to release a lock earlier?

Answer: Note: The tree-protocol of Section 16.1.5 which is referred to in this question, is different from the multigranularity protocol of Section 16.4 and the *B*+-tree concurrency protocol of Section 16.9. One strategy for early lock releasing is given here. Going down the tree from the root, if the currently visited node's child is not full, release locks held on all nodes except the current node, request an X-lock on the child node, after getting it release the lock on the current node, and then descend to the child. On the other hand, if the child is full, retain all locks held, request an X-lock on the child, and descend to it after getting the lock. On reaching the leaf node, start the insertion procedure. This strategy results in holding locks only on the full index tree nodes from the leaf upwards, uptil and including the first nonfull node. An optimization to the above strategy is possible. Even if the current node's child is full, we can still release the locks on all nodes but the current one. But after getting the X-lock on the child node, we split it right away. Releasing the lock on the current node and retaining just the lock on the appropriate split child, we descend into it making it the current node.With this optimization, at any time at most two locks are held, of a parent and a child node.

15.19 The snapshot isolation protocol uses a validation step which, before performing a write of a data item by transaction *T*, checks if a transaction concurrent with *T* has already written the data item.

a. A straightforward implementation uses a start timestamp and a commit timestamp for each transaction, in addition to an *update set*, that is the set of data items updated by the transaction. Explain how to perform validation for the first-committer-wins scheme by using the transaction timestamps along with the update sets. You may assume that validation and other commit processing steps are executed serially, that is for one transaction at a time,

b. Explain how the validation step can be implemented as part of commit processing for the first-committer-wins scheme, using a modification of the above scheme, where instead of using update sets, each data item has a write timestamp associated with it. Again, you may assume that validation and other commit processing steps are executed serially.

c. The first-updater-wins scheme can be implemented using timestamps as described above, except that validation is done immediately after acquiring an exclusive lock, instead of being done at commit time.

i. Explain how to assign write timestamps to data items to implement the first-updater-wins scheme.
ii. Show that as a result of locking, if the validation is repeated at commit time the result would not change.
iii. Explain why there is no need to perform validation and other commit processing steps serially in this case.

Answer:

15.20 What benefit does strict two-phase locking provide? What disadvantages result?
Answer: Because it produces only cascadeless schedules, recovery is very easy. But the set of schedules obtainable is a subset of those obtainable from plain two phase locking, thus concurrency is reduced.

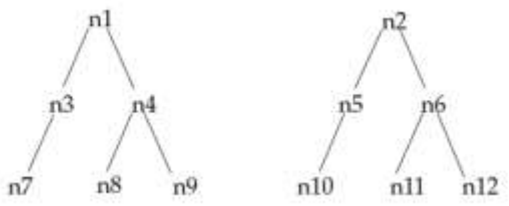
15.21 Most implementations of database systems use strict two-phase locking.Suggest three reasons for the popularity of this protocol.

Answer: It is relatively simple to implement, imposes low rollback overhead because of cascadeless schedules, and usually allows an acceptable level of concurrency.

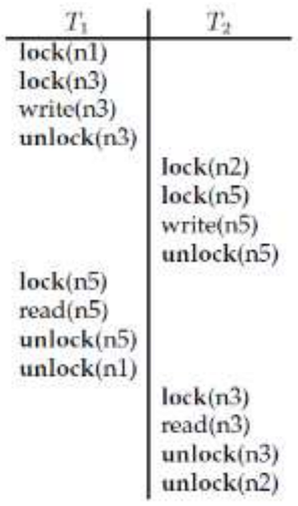
15.22 Consider a variant of the tree protocol called the *forest* protocol. The database is organized as a forest of rooted trees. Each transaction *Ti* must follow the following rules:
• The first lock in each tree may be on any data item.
• The second, and all subsequent, locks in a treemay be requested only if the parent of the requested node is currently locked.
• Data items may be unlocked at any time.
• A data item may not be relocked by *Ti* after it has been unlocked by *Ti* .

Show that the forest protocol does *not* ensure serializability.

Answer: Take a system with 2 trees:



We have 2 transactions, *T*1 and *T*2. Consider the following legal schedule:



T_1	T_2
lock(n1)	
lock(n3)	
write(n3)	
unlock(n3)	
	lock(n2)
	lock(n5)
	write(n5)
	unlock(n5)
lock(n5)	
read(n5)	
unlock(n5)	
unlock(n1)	
	lock(n3)
	read(n3)
	unlock(n3)
	unlock(n2)

This schedule is not serializable.

15.23 Under what conditions is it less expensive to avoid deadlock than to allow deadlocks to occur and then to detect them?

Answer: Deadlock avoidance is preferable if the consequences of abort are serious (as in interactive transactions), and if there is high contention and a resulting high probability of deadlock.

15.24 If deadlock is avoided by deadlock-avoidance schemes, is starvation still possible? Explain your answer.

Answer: A transaction may become the victim of deadlock-prevention rollback arbitrarily many times, thus creating a potential starvation situation.

15.25 In multiple-granularity locking, what is the difference between implicit and explicit locking?

Answer: When a transaction explicitly locks a node in shared or exclusive mode, it implicitly locks all the descendents of that node in the same mode. The transaction need not explicitly lock the descendent nodes. There is no difference in the functionalities of these locks, the only difference is in the way they are acquired, and their presence tested.

15.26 Although SIX mode is useful in multiple-granularity locking, an exclusive and intention-shared (XIS) mode is of no use.Why is it useless?

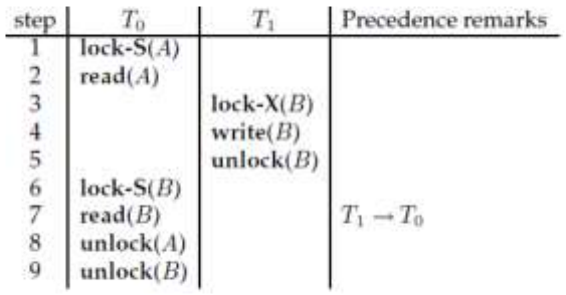
Answer: An exclusive lock is incompatible with any other lock kind. Once a node is locked in exclusive mode, none of the descendents can be simultaneously accessed by any other transaction in any mode. Therefore an exclusive and intend-shared declaration has no meaning.

15.27 The multiple-granularity protocol rules specify that a transaction *Ti* can lock a node *Q* in S or IS mode only if *Ti* currently has the parent of *Q* locked in either IX or IS mode. Given that SIX and S locks are stronger than IX or IS locks, why does the protocol not allow locking a node in S or IS mode if the parent is locked in either SIX or S mode?
Answer:

15.28 When a transaction is rolled back under timestamp ordering, it is assigned a new timestamp. Why can it not simply keep its old timestamp?

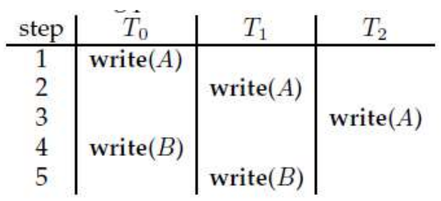
Answer: A transaction is rolled back because a newer transaction has read or written the data which it was supposed to write. If the rolled back transaction is re-introduced with the same timestamp, the same reason for rollback is still valid, and the transaction will have be rolled back again. This will continue indefinitely.

15.29 Show that there are schedules that are possible under the two-phase locking protocol, but are not possible under the timestamp protocol, and vice versa.
Answer: A schedule which is allowed in the two-phase locking protocol but not in the timestamp protocol is:



step	T_0	T_1	Precedence remarks
1	lock-S(A)		
2	read(A)		
3		lock-X(B)	
4		write(B)	
5		unlock(B)	
6	lock-S(B)		
7	read(B)		$T_1 \rightarrow T_0$
8	unlock(A)		
9	unlock(B)		

This schedule is not allowed in the timestamp protocol because at step 7, the W-timestamp of *B* is 1. A schedule which is allowed in the timestamp protocol but not in the twophase locking protocol is:



step	T_0	T_1	T_2
1	write(A)		
2		write(A)	
3			write(A)
4	write(B)		
5		write(B)	

This schedule cannot have lock instructions added to make it legal under two-phase locking protocol because *T*1 must unlock (*A*) between steps 2 and 3, and must lock (*B*) between steps 4 and 5.

15.30 Under a modified version of the timestamp protocol, we require that a commit bit be tested to see whether a read request must wait. Explain how the commit bit can prevent cascading abort. Why is this test not necessary for write requests?
Answer: Using the commit bit, a read request is made to wait if the transaction which wrote the data item has not yet committed. Therefore, if the writing transaction fails before commit, we can abort that transaction alone. The waiting read will then access the earlier version in case of a multiversion system, or the restored value of the data item after abort in case of a single-version system. For writes, this commit bit checking is unnecessary. That is because either the write is a "blind" write and thus independent of the old value of the data item or there was a prior read, in which case the test was already applied.

15.31 As discussed in Exercise 15.19, snapshot isolation can be implemented using a form of timestamp validation. However, unlike the multiversion timestamp-ordering scheme, which guarantees serializability, snapshot isolation does not guarantee serializability. Explain what is the key difference between the protocols that results in this difference.

Answer:

15.32 Outline the key similarities and differences between the timestamp based implementation of the first committer-wins version of snapshot isolation, described in Exercise 15.19, and the optimistic-concurrency-controlwithout- read-validation scheme, described in Section 15.9.3

Answer:

15.33 Explain the phantom phenomenon. Why may this phenomenon lead to an incorrect concurrent execution despite the use of the two-phase locking protocol?

Answer: The phantom phenomenon arises when, due to an insertion or deletion, two transactions logically conflict despite not locking any data items in common. The insertion case is described in the book. Deletion can also lead to this phenomenon. Suppose *Ti* deletes a tuple from a relation while *Tj* scans the relation. If *Ti* deletes the tuple and then *Tj* reads the relation, *Ti* should be serialized before *Tj* . Yet there is no tuple that both *Ti* and *Tj* conflict on. An interpretation of 2PL as just locking the accessed tuples in a relation is incorrect. There is also an index or a relation data that has information about the tuples in the relation. This information is read by any transaction that scans the relation, and modified by transactions that update, or insert into, or delete from the relation. Hence locking must also be performed on the index or relation data, and this will avoid the phantom phenomenon.

15.34 Explain the reason for the use of degree-two consistency.What disadvantages does this approach have?

Answer: TO BE SOLVED

15.35 Give example schedules to show that with key-value locking, if any of lookup, insert, or delete do not lock the next-key value, the phantom phenomenon could go undetected.

Answer: TO BE SOLVED

15.36 Many transactions update a common item (e.g., the cash balance at a branch), and private items (e.g., individual account balances). Explain how you can increase concurrency (and throughput) by ordering the operations of the transaction.

Answer:

15.37 Consider the following locking protocol: All items are numbered, and once an item is unlocked, only higher-numbered items may be locked. Locks may be released at any time. Only X-locks are used. Show by an example that this protocol does not guarantee serializability.

Answer: