Practice Exercises

14.1 Suppose that there is a database system that never fails. Is a recovery manager required for this system? 5

14.2 Consider a file system such as the one on your favorite operating system.

a. What are the steps involved in creation and deletion of files, and in writing data to a file? 10

b. Explain how the issues of atomicity and durability are relevant to the creation and deletion of files and to writing data to files.

14.3 Database-system implementers have paid much more attention to the ACID properties than have file-system implementers. Why might this be the case? 10

14.4 Justify the following statement: Concurrent execution of transactions is more important when data must be fetched from (slow) disk or when transactions are long, and is less important when data are in memory and transactions are very short. 10

14.10 Consider a database for an airline where the database system uses snapshot isolation. Describe a particular scenario in which a nonserializable execution occurs, but the airline may be willing to accept it in order to gain better overall performance. 10

Exercises

14.12 List the ACID properties. Explain the usefulness of each. 5

14.13 During its execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through 5 which a transaction may pass. Explain why each state transition may occur.

14.14 Explain the distinction between the terms *serial schedule* and *serializable schedule*. 5

14.15 Consider the following two transactions: 10

```
T_13: read(A);
      read(B);
      if A = 0 then B := B + 1;
      write(B).
T_14: read(B);
      read(A);
      if B = 0 then A := A + 1;
      write(A).
```

Let the consistency requirement be $A = 0 \vee B = 0$, with $A = B = 0$ the initial values.

a. Show that every serial execution involving these two transactions preserves the consistency of the database.

b. Show a concurrent execution of T_{13} and T_{14} that produces a nonserializable schedule.

c. Is there a concurrent execution of T_{13} and T_{14} that produces a serializable schedule?

14.16 Give an example of a serializable schedule with two transactions such that the order in which the transactions commit is different from the serialization order. 5

14.17 What is a recoverable schedule? Why is recoverability of schedules desirable? Are there any circumstances under which it would be desirable to allow nonrecoverable schedules? Explain your answer. 5

14.18 Why do database systems support concurrent execution of transactions, in spite of the extra programming effort needed to ensure that concurrent execution does not cause any problems? 5

14.19 Explain why the read-committed isolation level ensures that schedules are cascade-free. 5

14.20 For each of the following isolation levels, give an example of a schedule that respects the specified level of isolation, but is not serializable: 5

a. Read uncommitted

b. Read committed

c. Repeatable read

14.21 Suppose that in addition to the operations read and write, we allow an operation pred_read(r, P), which reads all tuples in relation r that satisfy 10
predicate P.

a. Give an example of a schedule using the pred_read operation that exhibits the phantom phenomenon, and is nonserializable as a result.

b. Give an example of a schedule where one transaction uses the pred_read operation on relation r and another concurrent transactions deletes a tuple from r, but the schedule does not exhibit a phantom conflict. (To do so, you have to give the schema of relation r, and show the attribute values of the deleted tuple.)

Practice Exercises

15.1 Show that the two-phase locking protocol ensures conflict serializability, and that transactions can be serialized according to their lock points. 5

15.2 Consider the following two transactions: 5

```
$T_{34}$: 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?

15.3 What benefit does rigorous two-phase locking provide? How does it compare with other forms of two-phase locking? 5

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. 5

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. 5

15.6 Consider the following extension to the tree-locking protocol, which allows both shared and exclusive locks: 10

- 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 transactions may lock any data item first, whereas update transactions must lock the root first.

Show that the protocol ensures serializability and deadlock freedom.

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. 10

- 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.

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. 10

- 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.

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 access-protection mechanism can be used for page-level locking in a persistent programming language. 10

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

Figure 15.23 Lock-compatibility matrix.

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: 10

- Two-phase locking.
- Two-phase locking with multiple-granularity locking.

- The tree protocol.
- Timestamp ordering.
- Validation.
- Multiversion timestamp ordering.
- Multiversion two-phase locking.

[illegible]

Exercises

15.20 What benefit does strict two-phase locking provide? What disadvantages result? 5

15.21 Most implementations of database systems use strict two-phase locking. Suggest three reasons for the popularity of this protocol. 5

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 T_i must follow the following rules: 5

- The first lock in each tree may be on any data item.
- The second, and all subsequent, locks in a tree may 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 T_i after it has been unlocked by T_i.

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

15.23 Under what conditions is it less expensive to avoid deadlock than to allow deadlocks to occur and then to detect them? 5

15.24 If deadlock is avoided by deadlock-avoidance schemes, is starvation still possible? Explain your answer. 5

15.25 In multiple-granularity locking, what is the difference between implicit and explicit locking? 5

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? 5

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? 5

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. 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? 5