Practice Exercises

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

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?
   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?

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.

14.5 Since every conflict-serializable schedule is view serializable, why do we emphasize conflict serializability rather than view serializability?

14.6 Consider the precedence graph of Figure 14.16. Is the corresponding schedule conflict serializable? Explain your answer.

14.7 What is a cascadeless schedule? Why is cascadelessness of schedules desirable? Are there any circumstances under which it would be desirable to allow noncascadeless schedules? Explain your answer.

14.8 The lost update anomaly is said to occur if a transaction $T_j$ reads a data item, then another transaction $T_k$ writes the data item (possibly based on a previous read), after which $T_j$ writes the data item. The update performed by $T_k$ has been lost, since the update done by $T_j$ ignored the value written by $T_k$. 
a. Give an example of a schedule showing the lost update anomaly.

b. Give an example schedule to show that the lost update anomaly is possible with the read committed isolation level.

c. Explain why the lost update anomaly is not possible with the repeatable read isolation level.

14.9 Consider a database for a bank where the database system uses snapshot isolation. Describe a particular scenario in which a nonserializable execution occurs that would present a problem for the bank.

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.

14.11 The definition of a schedule assumes that operations can be totally ordered by time. Consider a database system that runs on a system with multiple processors, where it is not always possible to establish an exact ordering between operations that executed on different processors. However, operations on a data item can be totally ordered. Does the above situation cause any problem for the definition of conflict serializability? Explain your answer.

Exercises

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

14.13 During its execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through
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*.

14.15 Consider the following two transactions:

\[ T_{13}: \text{read}(A); \]
\[ \text{read}(B); \]
\[ \quad \text{if } A = 0 \text{ then } B := B + 1; \]
\[ \text{write}(B). \]
\[ T_{14}: \text{read}(B); \]
\[ \text{read}(A); \]
\[ \quad \text{if } B = 0 \text{ then } A := A + 1; \]
\[ \text{write}(A). \]

Let the consistency requirement be \( A = 0 \lor B = 0 \), 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.

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.

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?

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

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:

- 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 \texttt{pred\_read}(r, P), which reads all tuples in relation \( r \) that satisfy predicate \( P \).

a. Give an example of a schedule using the \texttt{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 \texttt{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.)

Bibliographical Notes

Gray and Reuter [1993] provides detailed textbook coverage of transaction-processing concepts, techniques and implementation details, including concurrency control and recovery issues. Bernstein and Newcomer [1997] provides textbook coverage of various aspects of transaction processing.

The concept of serializability was formalized by Eswaran et al. [1976] in connection to work on concurrency control for System R.

References covering specific aspects of transaction processing, such as concurrency control and recovery, are cited in Chapters 15, 16, and 26.