Chapter 10

Transaction Management and Concurrency Control

Answers to Review Questions

1. Explain the following statement: a transaction is a logical unit of work.

A transaction is a logical unit of work that must be entirely completed of aborted; no intermediate states are accepted. In other words, a transaction, composed of several database requests, is treated by the DBMS as a *unit* of work in which all transaction steps must be fully completed if the transaction is to be accepted by the DBMS.

Acceptance of an incomplete transaction will yield an inconsistent database state. To avoid such a state, the DBMS ensures that all of a transaction's database operations are completed before they are committed to the database. For example, a credit sale requires a minimum of three database operations:

1. An invoice is created for the sold product.
2. The product's inventory quantity on hand is reduced.
3. The customer accounts payable balance is increased by the amount listed on the invoice.

If only parts 1 and 2 are completed, the database will be left in an inconsistent state. Unless all three parts (1, 2, and 3) are completed, the entire sales transaction is canceled.

2. **What is a consistent database state, and how is it achieved?**

   A consistent database state is one in which all data integrity constraints are satisfied. To achieve a consistent database state, a transaction must take the database from one consistent state to another. (See the answer to question 1.)

3. **The DBMS does not guarantee that the semantic meaning of the transaction truly represents the real-world event. What are the possible consequences of that limitation? Give an example.**

   The database is designed to verify the syntactic accuracy of the database commands given by the user to be executed by the DBMS. The DBMS will check that the database exists, that the referenced attributes exist in the selected tables, that the attribute data types are correct, and so on. Unfortunately, the DBMS is not designed to guarantee that the syntactically correct transaction accurately represents the real-world event.

   For example, if the end user sells 10 units of product 100179 (Crystal Vases), the DBMS cannot detect errors such as the operator entering 10 units of product 100197 (Crystal Glasses). The DBMS will execute the transaction, and the database will end up in a *technically consistent state* but in a *real-world inconsistent state* because the wrong product was updated.

4. **List and discuss the five transaction properties.**

   The five transaction properties are:

Atomicity	requires that *all* parts of a transaction must be completed or the transaction is aborted. This property ensures that the database will remain in a consistent state.
Consistency	Indicates the permanence of the database consistent state**.**
Isolation	means that the data required by an executing transaction cannot be accessed by any other transaction until the first transaction finishes. This property ensures data consistency for concurrently executing transactions.
Durability	indicates that the database will be in a *permanent* consistent state after the execution of a transaction. In other words, once a consistent state is reached, it cannot be lost.
Serializability	means that a series of concurrent transactions will yield the same result as if they were executed one after another.

   All five transaction properties work together to make sure that a database maintains data integrity and consistency for either a single-user or a multi-user DBMS.

5. **What does serializability of transactions mean?**

   Serializability of transactions means that a series of concurrent transactions will yield the same result as if they were executed one after another

6. **What is a transaction log, and what is its function?**

   The transaction log is a special DBMS table that contains a description of all the database transactions executed by the DBMS. The database transaction log plays a crucial role in maintaining database concurrency control and integrity.

   The information stored in the log is used by the DBMS to recover the database after a transaction is aborted or after a system failure. The transaction log is usually stored in a different hard disk or in a different media (tape) to prevent the failure caused by a media error.

7. **What is a scheduler, what does it do, and why is its activity important to concurrency control?**

   The scheduler is the DBMS component that establishes the order in which concurrent database operations are executed. The scheduler interleaves the execution of the database operations (belonging to several concurrent transactions) to ensure the *serializability* of transactions. In other words, the scheduler guarantees that the execution of concurrent transactions will yield the same result as though the transactions were executed one after another. The scheduler is important because it is the DBMS component that will ensure transaction serializability. In other words, the scheduler allows the concurrent execution of transactions, giving end users the impression that they are the DBMS's only users.

8. **What is a lock, and how, in general, does it work?**

   A lock is a mechanism used in concurrency control to guarantee the exclusive use of a data element to the transaction that owns the lock. For example, if the data element X is currently locked by transaction T1, transaction T2 will not have access to the data element X until T1 releases its lock.

   Generally speaking, a data item can be in only two states: locked (being used by some transaction) or unlocked (not in use by any transaction). To access a data element X, a transaction T1 first must request a lock to the DBMS. If the data element is not in use, the DBMS will lock X to be used by T1 exclusively. No other transaction will have access to X while T1 is executed.

9. **What are the different levels of lock granuality?**

   Lock granularity refers to the size of the database object that a single lock is placed upon. Lock granularity can be:

Database-level, meaning the entire database is locked by one lock.
Table-level, meaning a table is locked by one lock.
Page-level, meaning a diskpage is locked by one lock.
Row-level, meaning one row is locked by one lock.
Field-level, meaning one field in one row is locked by one lock.

10. Why might a page-level lock be preferred over a field-level lock?

Smaller lock granualarity improves the concurrency of the database by reducing contention to lock database objects. However, smaller lock granularity also means that more locks must be maintained and managed by the DBMS, requiring more processing overhead and system resources for lock management. Concurrency demands and system resource usage must be balanced to ensure the best overall transaction performance. In some circumstances, page-level locks, which require fewer system resources, may produce better overall performance than field-level locks, which require more system resources.

11. What is concurrency control, and what is its objective?

Concurrency control is the activity of coordinating the simultaneous execution of transactions in a multiprocessing or multi-user database management system. The objective of concurrency control is to ensure the serializability of transactions in a multi-user database management system. (The DBMS's scheduler is in charge of maintaining concurrency control.)

Because it helps to guarantee data integrity and consistency in a database system, concurrency control is one of the most critical activities performed by a DBMS. If concurrency control is not maintained, three serious problems may be caused by concurrent transaction execution: lost updates, uncommitted data, and inconsistent retrievals.

12. What is an exclusive lock, and under what circumstances is it granted?

An exclusive lock is one of two lock types used to enforce concurrency control. (A lock can have three states: unlocked, shared (read) lock, and exclusive (write) lock. The "shared" and "exclusive" labels indicate the nature of the lock.)

An exclusive lock exists when access to a data item is specifically reserved for the transaction that locked the object. The exclusive lock must be used when a potential for conflict exists, e.g., when one or more transactions must update (WRITE) a data item. Therefore, an exclusive lock is issued only when a transaction must WRITE (update) a data item *and no locks are currently held on that data item by any other transaction.*

To understand the reasons for having an exclusive lock, look at its counterpart, the shared lock. Shared locks are appropriate when concurrent transactions are granted

READ access on the basis of a common lock, because concurrent transactions based on a READ cannot produce a conflict.

A shared lock is issued when a transaction must read data from the database *and no exclusive locks are held* on the data to be read.

13. What is a deadlock, and how can it be avoided? Discuss several strategies for dealing with deadlocks.

Base your discussion on Chapter 10's Section 10-3d, Deadlocks. Start by pointing out that, although locks prevent serious data inconsistencies, their use may lead to two major problems:

1. The transaction schedule dictated by the locking requirements may not be serializable, thus causing data integrity and consistency problems.
2. The schedule may create *deadlocks*. Database deadlocks are the equivalent of a traffic gridlock in a big city and are caused by two transactions waiting for each other to unlock data.

Use Table 10.13 in the text to illustrate the scenario that leads to a deadlock.) The table has been reproduced below for your convenience.

TABLE 10.13 How a Deadlock Condition is Created

TIME	TRANSACTION	REPLY	LOCK STATUS	
0			**Data X**	**Data Y**
1	T1:LOCK(X)	OK	Unlocked	Unlocked
2	T2: LOCK(Y)	OK	Locked	Unlocked
3	T1:LOCK(Y)	WAIT	Locked	Locked
4	T2:LOCK(X)	WAIT	Locked	**Deadlock** Locked
5	T1:LOCK(Y)	WAIT	Locked	Lock
6	T2:LOCK(X)	WAIT	Locked	Lock
7	T1:LOCK(Y)	WAIT	Locked	Lock
8	T2:LOCK(X)	WAIT	Locked	Lock
9	T1:LOCK(Y)	WAIT	Locked	Lock
…	……..	……..	……..	…….
…	……..	……..	……..	…….
…	……..	……..	……..	…….
…	……..	……..	……..	…….

In a real world DBMS, many more transactions can be executed simultaneously, thereby increasing the probability of generating deadlocks. Note that deadlocks are

possible only if one of the transactions wants to obtain an exclusive lock on a data item; no deadlock condition can exist among shared locks.

Three basic techniques exist to control deadlocks:

Deadlock Prevention

A transaction requesting a new lock is aborted if there is a possibility that a deadlock may occur. If the transaction is aborted, all the changes made by this transaction are rolled back and all locks are released. The transaction is then re-scheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.

Deadlock Detection

The DBMS periodically tests the database for deadlocks. If a deadlock is found, one of the transactions (the "victim") is aborted (rolled back and rescheduled) and the other transaction continues. Note particularly the discussion in Section 10-4a, Wait/Die and Wound/Wait Schemes.

Deadlock Avoidance

The transaction must obtain all the locks it needs before it can be executed. This technique avoids rollback of conflicting transactions by requiring that locks be obtained in succession. However, the serial lock assignment required in deadlock avoidance increases the response times.

The best deadlock control method depends on the database environment. For example, if the probability of deadlocks is low, deadlock detection is recommended. However, if the probability of deadlocks is high, deadlock prevention is recommended. If response time is not high on the system priority list, deadlock avoidance may be employed.

14. What are some disadvantages of time-stamping methods for concurrency control?

The disadvantages are: 1) each value stored in the database requires two additional time stamp fields – one for the last time the field was read and one for the last time it was updated, 2) increased memory and processing overhead requirements, and 3) many transactions may have to be stopped, rescheduled, and restamped.

15. Why might it take a long time to complete transactions when an optimistic approach to concurrency control is used?

Because the optimistic approach makes the assumption that conflict from concurrent transactions is unlikely, it does nothing to avoid conflicts or control the conflicts. The only test for conflict occurs during the validation phase. If a conflict is detected, then the entire transaction restarts. In an environment with few conflicts from concurrency, this type of single checking scheme works well. In an environment where conflicts are

common, a transaction may have to be restarted numerous times before it can be written to the database.

16. What are the three types of database critical events that can trigger the database recovery process? Give some examples for each one.

Backup and recovery functions constitute a very important component of today's DBMSs. Some DBMSs provide functions that allow the database administrator to perform and schedule automatic database backups to permanent secondary storage devices, such as disks or tapes.

Critical events include:

- *Hardware/software failures*. hard disk media failure, a bad capacitor on a motherboard, or a failing memory bank. Other causes of errors under this category include application program or operating system errors that cause data to be overwritten, deleted, or lost.
- *Human-caused incidents*. This type of event can be categorized as unintentional or intentional.
  - An unintentional failure is caused by carelessness by end-users. Such errors include deleting the wrong rows from a table, pressing the wrong key on the keyboard, or shutting down the main database server by accident.
  - Intentional events are of a more severe nature and normally indicate that the company data are at serious risk. Under this category are security threats caused by hackers trying to gain unauthorized access to data resources and virus attacks caused by disgruntled employees trying to compromise the database operation and damage the company.
- *Natural disasters*. This category includes fires, earthquakes, floods, and power failures.

17. What are the four ANSI transaction isolation levels? What type of reads does each level allow?

The four ANSI transaction isolation levels are 1) read uncommitted, 2) read committed, 3) repeatable read, and 4) Serializable. These levels allow different "questionable" reads. A read is questionable if it can produce inconsistent results. Read uncommitted isolation will allow dirty reads, non-repeatable reads and phantom reads. Read committed isolation will allow non-repeatable reads and phantom reads. Repeatable read isolation will allow phantom reads. Serializable does not allow any questionable reads.