6.7 Explain the trade-off that exists in concurrency control.

A high level of control granularity means easy administration for the DBMS but poor throughput. A low level of control granularity means difficult administration for the DBMS, but better throughput.

6.8 Describe what an atomic transaction is and explain why atomicity is important.

An atomic transaction is one in which either all of the database actions are committed to the database or none of them are. Without it, there is a danger that partially processed transactions will be committed to the database.

6.9 Explain the difference between concurrent transactions and simultaneous transactions. How many CPUs are required for simultaneous transactions?

With concurrent transactions, two or more users access the database using a single CPU on the database server. The CPU executes some instructions from one, and then executes some from the other, switching back and forth between them. The actions may appear simultaneous to the two users. For transactions to be simultaneous, they would have to be actually processed at the same time. For transactions to be processed simultaneously, two or more CPUs are required. With modern server computers, such processing is possible.

6.10 Give an example, other than the one in this text, of the lost update problem.

The lost update problem occurs when two transactions attempt to update the same data resource simultaneously. Because each transaction copies the record into its work area, each can effect different changes and then rewrite the record. The problem is that all updates except the last one to be rewritten are lost. Example: At a music store, two salespeople rent the same B-flat clarinet to two different customers.

6.11 Define the terms dirty read, nonrepeatable read, *and* phantom read.

- *A **dirty read** reads data that has been changed but not committed by another user.*
- *A **nonrepeatable read** is a read in which multiple reads of the same record may not obtain the same values because of changes made by other users.*
- *A **phantom read** occurs when new rows are found on a second or subsequent read of a file.*

6.12 Explain the difference between an explicit and an implicit lock.

Explicit locks are set by the application program, while implicit locks are set by the DBMS on behalf of the application or users.

6.13 What is lock granularity?

Lock granularity is the size of the resource that is locked. Large grained locks are easy for the DBMS to administer but result in frequent conflicts, whereas small grained locks are more difficult and expensive to administer but result in fewer conflicts.

6.14 Explain the difference between an exclusive lock and a shared lock.

An exclusive lock allows no other access, whereas a shared lock allows other transactions to read but not update the data.

6.15 Explain two-phased locking.

The two-phase locking strategy is used to obtain a number of locks until a lock is released, and after that no more locks can be obtained until all are released. Thus, the transaction has a lock growing phase and then a lock shrinking phase. Most DBMS products implement a special case of this in which all locks are released when a transaction commits.

6.16 How does releasing all locks at the end of a transaction relate to two-phase locking?

It is a special case of two-phase locking—during the life of the transaction, locks are only acquired (the growing phase). When the transaction is committed, all locks are released. The shrinking phase occurs at one point.

6.17 What is deadlock? How can it be avoided? How can it be resolved when it occurs?

Deadlock occurs when each of two transactions is waiting for a resource the other has locked. Both transactions are placed in the wait state, but neither one can get out of it. A DBMS can either prevent the deadly embrace from occurring (via control over locking) or it can allow it and then detect and break it (by aborting one of the transactions, thus releasing its locks and enabling the other transaction to continue).

6.18 Explain the difference between optimistic and pessimistic locking.

The assumption with optimistic locking is that conflict will not occur. Therefore, no locks are placed until the transaction is completed. At that point, locks are obtained and records are updated only if they have not been changed since last read. With pessimistic locking, the assumption is made that conflict will occur. Therefore, records are locked before they're read. When the transaction is complete, all locks are released.

6.19 Explain the benefits of marking transaction boundaries, declaring lock characteristics, and letting a DBMS place locks.

It's more flexible and puts less demand on the application programmer. The programmer defines the boundaries of the transaction and then defines the locking behavior that is wanted. The DBMS then figures out where to set locks. If the locking behavior declaration is subsequently changed, the DBMS will then place locks in accordance with the new behavior. This reduces the likelihood of programmer errors as well.

6.20 Explain the use of the SQL transaction control statements BEGIN TRANSACTION, COMMIT TRANSACTION, and ROLLBACK TRANSACTION.

These statements are used to mark transaction boundaries. The application programmer must set these boundaries because the DBMS cannot automatically determine what DBMS operations are to be atomic.

6.21 Explain the meaning of the expression ACID transaction.

An ACID transaction is atomic, consistent, insolated, and durable.

6.22 Describe statement-level consistency.

An update will apply to a set of rows as they existed at the time the statement was first executed.

6.23 Describe transaction-level consistency. What disadvantage can exist with it?

All updates in a transaction will be to rows as they existed at the time the transaction began. The possible disadvantage is that, depending on how this is implemented, transaction cannot view its own changes.

6.24 What is the purpose of transaction isolation levels?

They exist to provide flexibility in the degree of isolation between the works of different transactions.

6.25 Explain what read uncommitted isolation level is. Give an example of its use.

The read uncommitted isolation level allows dirty reads, nonrepeatable reads, and phantom reads. It might be used when making a list of Web sites in response to a search for articles on a movie.

6.26 Explain what read committed isolation level is. Give an example of its use.

The read committed isolation level disallows dirty reads, but allows nonrepeatable reads and phantom reads. It might be used when making a list of stock prices that are expected to be current only as of the time the stock price was read.

6.27 Explain what repeatable read isolation level is. Give an example of its use.

The repeatable read isolation level disallows dirty reads and nonrepeatable reads, but allows phantom reads. It might be used when making a list of inventory quantities on a list of inventory items that were known to be in the database when the read began.

6.28 Explain what serializable isolation level is. Give an example of its use.

The serializable isolation level disallows dirty reads, nonrepeatable reads, and phantom reads. It might be used when rank ordering a list of commodities, where the ranking process requires several passes through the table.

6.29 Explain the term **cursor***.*

A **cursor** is a pointer into a set of rows.

6.30 Explain why a transaction may have many cursors. Also, how is it possible that a transaction may have more than one cursor on a given table?

The transaction may need to process several tables at one time. A cursor can be opened on two different views of a table (these are SQL views, not application views), in which case there will be two cursors open on the same table for that transaction. Furthermore, there are some transactions in which the logic requires that two cursors process the same table.

6.31 What is the advantage of using different types of cursors?

Because cursors require considerable memory, having many cursors open at the same time for, say, a thousand concurrent transactions, can consume considerable memory and CPU time. One way to reduce cursor overhead is to define reduced-capability cursors and use them when a full-capability cursor is not needed.

6.32 Explain forward-only cursors. Give an example of their use.

The simplest cursor is the **forward-only cursor**. With it, the application can only move forward through the recordset. Changes made by other cursors in this transaction and by other transactions will be visible only if they occur in rows ahead of the cursor. They would be used when processing a report sequentially and you use a forward-only cursor when you are sequentially reading a recordset.

6.33 Explain static cursors. Give an example of their use.

With a **static cursor**, the application sees the data as it was at the time the cursor was opened. Changes made by this cursor are visible. Changes from other sources are not visible. Both backward and forward scrolling are allowed. Use this when processing a recordset and you need to see any changes you have made but not changes other users have made.

6.34 Explain keyset cursors. Give an example of their use.

Dynamic changes of any type and from any source are visible. When the cursor is opened, a primary key value is saved for each row in the recordset. When the application accesses a row, the key is used to fetch the current values for the row. All inserts, updates, deletions, and changes in the recordset order are visible. If the isolation level is dirty read, then uncommitted changes are visible. Otherwise, only committed changes are visible. Updates from any source

are visible. Inserts from sources outside this cursor are not visible. (There is no key for them in the keyset.) Inserts from this cursor appear at the bottom of the recordset. Deletions from any source are visible. Use a keyset cursor when you need to see updates from any source and your own inserts, but not inserts from other sources.

6.35 Explain dynamic cursors. Give an example of their use.

With a **dynamic cursor**, changes in row order are not visible. If the isolation level is dirty read, then committed updates and deletions are visible; otherwise only committed updates and deletions are visible. Use a dynamic cursor when you need to see all changes being made. Use with dirty read isolation level to see uncommitted changes, otherwise only committed changes will be seen.

6.36 What happens if you do not declare transaction-isolation level and cursor type to the DBMS? Is not declaring the isolation level and cursor type good or bad?

If you do not specify the isolation level of a transaction or do not specify the type of cursors you open, the DBMS will use a default level and type. These defaults may be perfect for your application, but they also may be terrible. Thus, even though these issues can be ignored, the consequences of them cannot be avoided.

6.37 Explain the necessity of defining processing rights and responsibilities. How are such responsibilities enforced?

We need to define processing rights and responsibilities to bring order to the processing of the database, which is a shared resource. While rights can be enforced by the DBMS and application programs, responsibilities must be documented and understood by users. The upholding of responsibilities cannot be automated; it's a matter of user training and behavior.

6.38 Explain the relationships of users, groups, permission, and objects for a generic database security system.

Permissions give roles to users and database objects themselves the ability to interact with database objects. [NOTE: Although permissions can be granted to users, a basic principle of systems administration is that, unless absolutely necessary, no users should have permissions granted directly to them. Instead, users should be placed in roles (which will hold groups of users), and the roles should be granted permissions to the database objects].

Each user can be in many roles, and each role can have many users. Each user, role or object may have many permissions. Each permission gives one user, role or object to one database object.

6.39 Describe the advantages and disadvantages of DBMS-provided security.

Advantages:

- Easier to implement
- Done regardless of the source of data changes and activities
- Probably more consistent

Disadvantages:

- May not suffice for particular needs
- Works best for vertical security

6.40 Describe the advantages and disadvantages of application-provided security.

Advantages:

- Easier to implement
- Done regardless of the source of data changes and activities
- Probably more consistent

Disadvantages:

- May not suffice for particular needs
- Works best for vertical security

6.41 Explain how a database could be recovered via reprocessing. Why is this generally not feasible?

Reprocessing means reapplying all transactions since the latest database save to that saved copy of the database. Because reprocessing takes as much time as original processing, reprocessing is usually infeasible. Additionally, initially asynchronous events may occur in a different order.

6.42 Define the terms **rollback** *and* **rollforward***.*

Rollback means applying before images (before changes were made) to the current database. This takes the database backwards in time.

Rollforward means loading the latest database backup then applying all after images (after changes were made) to it. This brings the database forward in time.

6.43 Why is it important to write to a log before changing the database values?

When database values are written to the log before they are changed in the database, the log record is correct in the event a database failure. Otherwise the database could be behind the log and changes that were committed might not be recovered.

6.44 Describe the rollback process. Under what conditions should rollback be used?

Start with a current copy of the database, and then back out changes. This method is used when database has not been lost; there is just some update activity that needs to be removed.

6.45 Describe the rollforward process. Under what conditions should rollforward be used?

Start with a restored database from a prior saved backup, then apply after images. This method is used when database has been lost.