Question 1:

If a failure occurs after the INSERT operation but before the UPDATE operation, the database might be left in an inconsistent state. To ensure consistency, the DBMS uses the concept of atomicity from the ACID properties. The entire transaction will be treated as a single unit, meaning all operations must be completed successfully for the transaction to be committed. If a failure occurs, the DBMS will roll back the transaction, undoing any changes made by the INSERT operation. This rollback ensures that partial updates are not saved in the database, maintaining consistency.

Question 2:

For a banking application transferring money from one account to another, the transaction should follow these steps:

1. **Start the transaction.**
2. **Check the balance** of the account from which the money is being withdrawn to ensure sufficient funds.
3. **Deduct the amount** from the sender's account.
4. **Add the amount** to the recipient's account.
5. **Commit the transaction.** If all steps succeed, the changes are saved to the database.
6. **Rollback the transaction** if any step fails, ensuring no partial changes are applied.

Following the ACID properties is crucial:

- **Atomicity:** Ensures all steps are completed successfully, or none at all.
- **Consistency:** Guarantees that the transaction leaves the database in a valid state.
- **Isolation:** Prevents other transactions from interfering with this transfer.
- **Durability:** Ensures that once the transaction is committed, the changes are permanently recorded.

Question 3:

When two concurrent transactions attempt to update the same record, the DBMS uses concurrency control techniques to ensure consistency:

- **Locking Mechanisms:** The DBMS may apply locks on the record being updated. For example, if one transaction holds an exclusive lock on the record, the second transaction must wait until the first transaction completes and releases the lock.
- **Timestamp Ordering:** Each transaction is given a timestamp, and the DBMS schedules operations based on these timestamps to ensure consistency.
- **Optimistic Concurrency Control:** The DBMS allows transactions to execute without locks initially but checks for conflicts before committing the changes. If a conflict is detected, the conflicting transaction may be rolled back.

Question 4:

- **COMMIT:** A COMMIT operation saves all changes made by the transaction to the database. Once committed, the changes are permanent and visible to other transactions.

- **ROLLBACK:** A ROLLBACK operation undoes all changes made during the transaction, reverting the database to its state before the transaction began.

If a transaction fails before the COMMIT statement is executed, the DBMS will automatically roll back the transaction, ensuring no partial updates are saved.

Example:

1. Start a transaction.
2. Update a row in a table.
3. Perform a ROLLBACK before committing the changes.

The database will revert to its original state, and the update will not be saved.

Question 5:

The two-phase commit protocol (2PC) in a distributed database involves two phases:

1. **Prepare Phase:** The coordinator asks all participating nodes if they are ready to commit the transaction. Each node responds with a "yes" (ready) or "no" (abort).
2. **Commit Phase:** If all nodes respond positively, the coordinator sends a commit command. Otherwise, it sends a rollback command.

If a failure occurs during the prepare phase, the coordinator will abort the transaction, ensuring that none of the changes are committed. This prevents inconsistencies across the distributed system.

Question 6:

A "dirty read" occurs when a transaction reads data that has been modified by another uncommitted transaction. If the second transaction is rolled back, the first transaction has read data that never became part of the permanent database state.

Example: Transaction A updates a record's value, and Transaction B reads the updated value before Transaction A commits. If Transaction A is rolled back, the value read by Transaction B becomes invalid.

Prevention: Using higher isolation levels like "READ COMMITTED" or "SERIALIZABLE" can prevent dirty reads by ensuring that only committed data is visible to other transactions.

Question 7:

- **READ UNCOMMITTED:** Allows a transaction to read uncommitted changes made by other transactions. This can lead to dirty reads, inconsistent data, and is not suitable for applications requiring high data integrity.
- **SERIALIZABLE:** Ensures that transactions are executed in a way that their effect is as if they were executed sequentially. This prevents dirty reads, non-repeatable reads, and phantom reads, ensuring maximum consistency.

Scenarios:

- **READ UNCOMMITTED:** Suitable for non-critical applications like generating rough reports.
- **SERIALIZABLE:** Suitable for financial transactions where data consistency is crucial.

Question 8:

A **deadlock** occurs when two or more transactions wait indefinitely for each other to release locks on resources.

Detection and Resolution:

- **Detection:** The DBMS can use wait-for graphs to detect circular waits.
- **Resolution:** Once a deadlock is detected, the DBMS can terminate one of the transactions to break the cycle.

Prevention: Techniques such as setting timeouts on locks or using lock ordering can help prevent deadlocks.

Question 9:

A transaction log system should include:

- **Transaction ID:** To uniquely identify each transaction.
- **Before and After Values:** To keep track of changes made by each transaction.
- **Operation Type:** Indicates whether the operation was an INSERT, UPDATE, or DELETE.
- **Timestamp:** To log when the changes occurred.

These components help in recovering from failures by allowing the DBMS to "undo" or "redo" transactions.

Question 10:

When a power outage occurs, the DBMS uses the transaction log to recover:

- **Undo Operations:** Roll back changes made by incomplete transactions to restore the database to its previous consistent state.
- **Redo Operations:** Reapply changes from committed transactions that were not fully written to the database.

The use of "undo" and "redo" ensures that the database can be brought back to a consistent state.