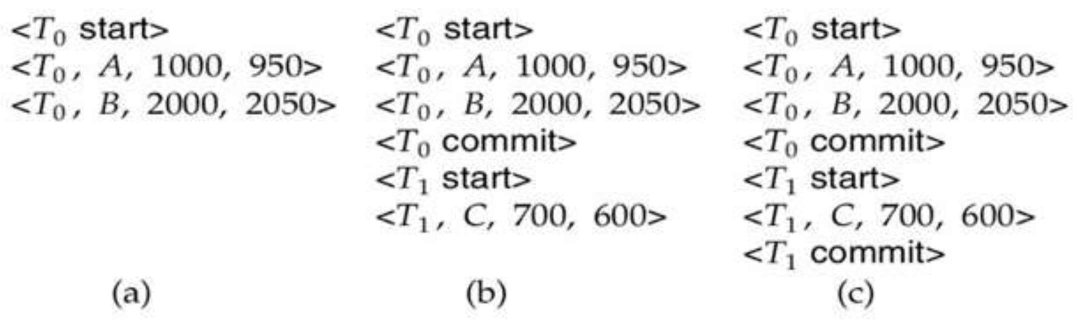
RECOVERY SYSTEM



(a)	(b)	(c)
$<T_0$ start>	$<T_0$ start>	$<T_0$ start>
$<T_0, A, 1000, 950>$	$<T_0, A, 1000, 950>$	$<T_0, A, 1000, 950>$
$<T_0, B, 2000, 2050>$	$<T_0, B, 2000, 2050>$	$<T_0, B, 2000, 2050>$
	$<T_0$ commit>	$<T_0$ commit>
	$<T_1$ start>	$<T_1$ start>
	$<T_1, C, 700, 600>$	$<T_1, C, 700, 600>$
		$<T_1$ commit>

Elaborate the Recovery actions given the log as it appears at three instances of time

Certainly! Let's break down the recovery actions based on the log at three different instances of time:

1. **Instance (a):**
   - At this point, transaction T0 has started and made changes to data items A and B. However, it has not yet committed. If a failure occurs now, the changes made by T0 would need to be undone.
   - Recovery Action:
     - If a failure occurs, the system should perform an **undo** operation for transaction T0. This means reverting the changes made by T0 on data items A and B.
2. **Instance (b):**
   - Here, transaction T0 has committed its changes to data items A and B. If a failure occurs after this point, the system would need to ensure that the changes made by T0 are permanent.
   - Transaction T1 has started but not yet committed.
   - Recovery Actions:
     - For T0:
       - If a failure occurs, the system should perform a **redo** operation for transaction T0. This means reapplying the changes made by T0 on data items A and B.

        - For T1:
            - If a failure occurs, the system should perform an **undo** operation for transaction T1. This means reverting the changes made by T1 on data item C.

3. **Instance (c):**
    - Both transactions T0 and T1 have committed their changes. In case of a failure occurring after this point, the recovery process should ensure that all changes made by both transactions are permanent.
    - Recovery Actions:
        - For T0:
            - No further action needed since T0 has already committed.
        - For T1:
            - No further action needed since T1 has also committed.

- Undo: Revert changes made by a transaction.
- Redo: Reapply changes made by a transaction.

Elaborate the Deferred Database Modification schemes. Also mention the recovery mechanism.

The deferred database modification scheme is a log-based recovery approach where all modifications to the database are first recorded in the log, but the actual writes to the database are deferred until after the transaction partially commits.Here's how the deferred database modification scheme works:

1. When a transaction Ti starts, a <Ti start> log record is written to the log.
2. Whenever a write(X) operation is performed, a log record <Ti, X, V> is written, where V is the new value for data item X. However, the actual write to the database is not performed at this time.
3. When the transaction Ti partially commits, a <Ti commit> log record is written to the log.
4. After the <Ti commit> record is written, the log records are read and used to actually execute the previously deferred writes to the database.

Recovery Mechanism:During recovery after a system crash, the recovery procedure needs to redo only those transactions that have both the <Ti start> and <Ti commit> log records present in the log. This is because the presence of the <Ti commit> record indicates that the transaction had partially committed and its updates need to be redone.The recovery procedure works as follows:

1. Scan the log backwards from the end to find the most recent checkpoint record.
2. Continue scanning backwards until a <Ti start> record is found, where Ti is the most recent transaction that started before the checkpoint.
3. For all transactions (starting from Ti or later) that have a <Ti start> record but no <Ti commit> record, execute undo(Ti) to restore the database to a consistent state.
4. For all transactions that have both the <Ti start> and <Ti commit> records, execute redo(Ti) to apply their updates to the database.

The key advantage of the deferred database modification scheme is that it simplifies the recovery process, as there is no need to undo any partially completed transactions. The trade-off is that the commit latency is higher since the actual database updates are deferred until after the transaction partially commits.

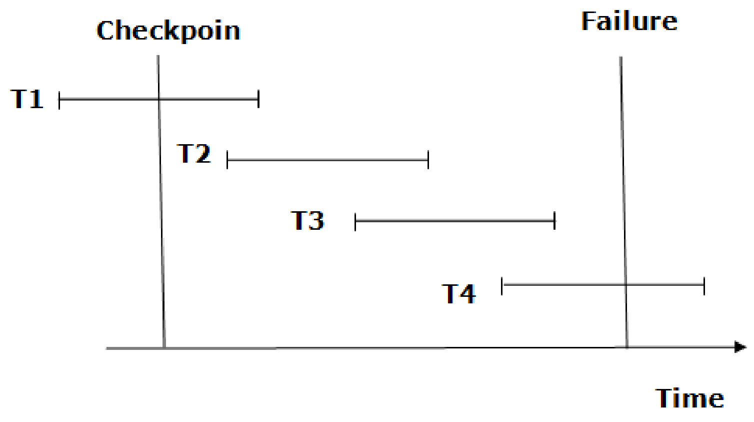
Explain the purpose of Checkpoint mechanism. Explain the steps for performing a checkpoint.

Checkpoint

- The checkpoint is a type of mechanism where all the previous logs are removed from the system and permanently stored in the storage disk.
- The checkpoint is like a bookmark. While the execution of the transaction, such checkpoints are marked, and the transaction is executed then using the steps of the transaction, the log files will be created.
- When it reaches to the checkpoint, then the transaction will be updated into the database, and till that point, the entire log file will be removed from the file. Then the log file is updated with the new step of transaction till next checkpoint and so on.
- The checkpoint is used to declare a point before which the DBMS was in the consistent state, and all transactions were committed.

Recovery using Checkpoint

In the following manner, a recovery system recovers the database from this failure:



- The recovery system reads log files from the end to start. It reads log files from T4 to T1.
- Recovery system maintains two lists, a redo-list, and an undo-list.

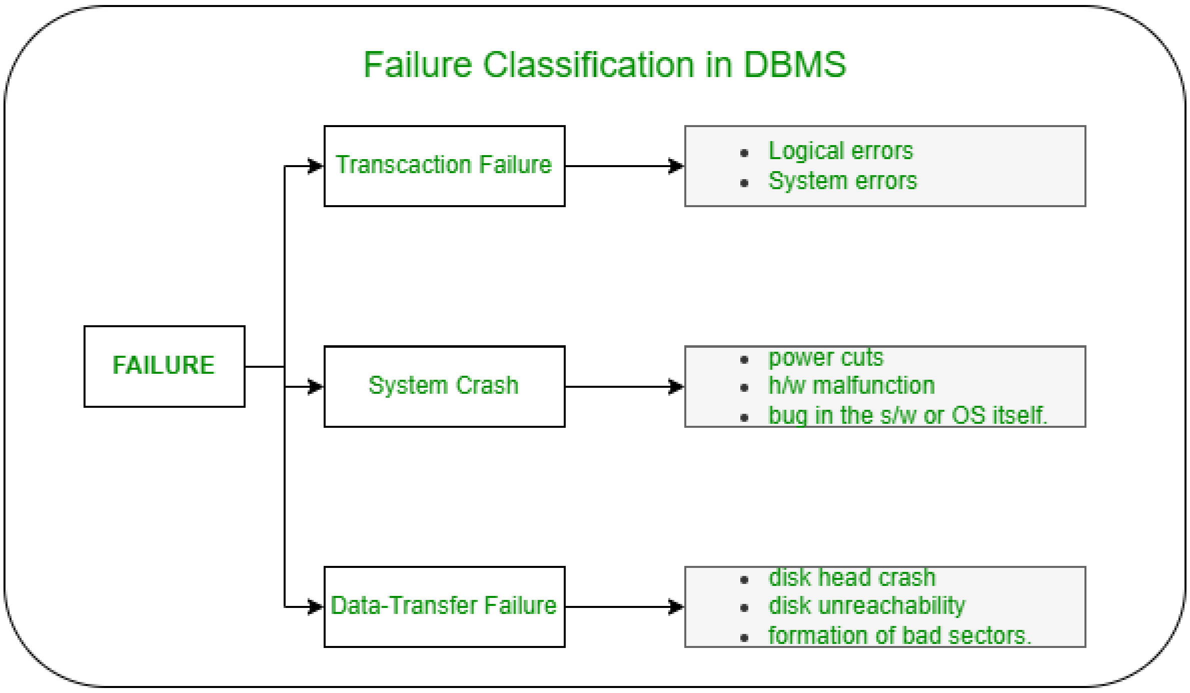
- The transaction is put into redo state if the recovery system sees a log with <Tn, Start> and <Tn, Commit> or just <Tn, Commit>. In the redo-list and their previous list, all the transactions are removed and then redone before saving their logs.
- **For example:** In the log file, transaction T2 and T3 will have <Tn, Start> and <Tn, Commit>. The T1 transaction will have only <Tn, commit> in the log file. That's why the transaction is committed after the checkpoint is crossed. Hence it puts T1, T2 and T3 transaction into redo list.
- The transaction is put into undo state if the recovery system sees a log with <Tn, Start> but no commit or abort log found. In the undo-list, all the transactions are undone, and their logs are removed.
- **For example:** Transaction T4 will have <Tn, Start>. So T4 will be put into undo list since this transaction is not yet complete and failed amid.

State and explain various classes of failure in database system.

Failure in terms of a database can be defined as its inability to execute the specified transaction or loss of data from the database. A DBMS is vulnerable to several kinds of failures and each of these failures needs to be managed differently. There are many reasons that can cause database failures such as network failure, system crash, natural disasters, carelessness, sabotage(corrupting the data intentionally), software errors, etc.

Failure Classification in DBMS

A failure in DBMS can be classified as:



Failure Classification in DBMS

Transaction Failure:

If a transaction is not able to execute or it comes to a point from where the transaction becomes incapable of executing further then it is termed as a failure in a transaction.

Reason for a transaction failure in DBMS:

1. **Logical error:** A logical error occurs if a transaction is unable to execute because of some mistakes in the code or due to the presence of some internal faults.

2. **System error:** Where the termination of an active transaction is done by the database system itself due to some system issue or because the database management system is unable to proceed with the transaction. *For example*– The system ends an operating transaction if it reaches a deadlock condition or if there is an unavailability of resources.

System Crash:

A system crash usually occurs when there is some sort of hardware or software breakdown. Some other problems which are external to the system and cause the system to abruptly stop or eventually crash include failure of the transaction, operating system errors, power cuts, main memory crash, etc.

These types of failures are often termed soft failures and are responsible for the data losses in the volatile memory. It is assumed that a system crash does not have any effect on the data stored in the non-volatile storage and this is known as the *fail-stop assumption*.

Data-transfer Failure:

When a disk failure occurs amid data-transfer operation resulting in loss of content from disk storage then such failures are categorized as data-transfer failures. Some other reason for disk failures includes disk head crash, disk unreachability, formation of bad sectors, read-write errors on the disk, etc.

In order to quickly recover from a disk failure caused amid a data-transfer operation, the backup copy of the data stored on other tapes or disks can be used. Thus it’s a good practice to backup your data frequently.

Explain Shadow paging in detail.

Shadow Paging is recovery technique that is used to recover database. In this recovery technique, database is considered as made up of fixed size of logical units of storage which are referred as **pages.** pages are mapped into physical blocks of storage, with help of the **page table** which allow one entry for each logical page of database. This method uses two page tables named **current page table** and **shadow page table**. The entries which are present in current page table are used to point to most recent database pages on disk. Another table i.e., Shadow page table is used when the transaction starts which is copying current page table. After this, shadow page table gets saved on disk and current page table is going to be used for transaction. Entries present in current page table may be changed during execution but in shadow page table it never get changed. After transaction, both tables become identical. This technique is also known as **Cut-of-Place updating.** - to enable efficient recovery after a system crash.Here's how shadow paging works:

1. Initially, the current page table and the shadow page table are identical, both pointing to the same database pages.
2. During the execution of a transaction, updates are made only to the current page table, while the shadow page table remains unchanged.
3. Whenever a page is about to be written for the first time during the transaction's execution, a copy of that page is made onto an unused page. The current page table is then updated to point to the new copy, while the shadow page table still points to the original page.
4. **To commit a transaction:**
   a. All modified pages in the main memory buffer are flushed to disk.
   b. The current page table is written to disk, becoming the new shadow page table.
   c. A pointer to the new shadow page table is stored at a fixed location on disk.
5. After a system crash, the recovery process is trivial - it simply uses the shadow page table stored on disk to access the database, as the shadow

page table represents the state of the database prior to the failed transaction's execution.

The key advantages of shadow paging are:

- No overhead of writing log records
- Recovery is very simple and fast, as it only requires using the shadow page table

However, shadow paging also has some disadvantages:

- Copying the entire page table can be very expensive, especially for large databases.
- The commit overhead is high, as the entire current page table needs to be written to disk.
- Data fragmentation can occur, as related pages may get separated on disk.
- It is harder to extend shadow paging to support concurrent transactions compared to log-based schemes.

List and elaborate the Drawbacks of Shadow Paging.

The drawbacks of Shadow Paging include:

- During commit operation, changed blocks are going to be pointed by shadow page table which have to be returned to collection of free blocks otherwise they become accessible.
- The commit of single transaction requires multiple blocks which decreases execution speed.
- To allow this technique to multiple transactions concurrently it is difficult.

    1. **High Overhead of Copying Entire Page Table**: Shadow Paging involves copying the entire page table, which can be very expensive, especially for large databases. This copying process can lead to increased memory and processing requirements.
    2. **Data Fragmentation**: As pages are copied and stored separately during the transaction's execution, related pages can get separated on disk, leading to data fragmentation. This fragmentation can impact the efficiency of data access and storage.
    3. **Commit Overhead**: The commit overhead in Shadow Paging is high. This is because, during the commit phase, all modified pages in the main memory buffer need to be flushed to disk, the current page table needs to be written to disk, and the current page table becomes the new shadow page table. This process can be time-consuming and resource-intensive.
    4. **Garbage Collection**: After every transaction completion, the database pages containing old versions of modified data need to be garbage collected. This process of freeing up pages not pointed to from the current or shadow page table adds complexity and overhead to the system.
    5. **Concurrency Limitations**: Shadow Paging is harder to extend to support concurrent transactions compared to log-based recovery schemes. Enabling multiple transactions to run concurrently while maintaining the

integrity of the shadow page table can be challenging and may require complex modifications to the system.

6. **Difficult to implement for some systems**: Shadow paging can be difficult to implement for some systems that have complex data structures or use a lot of shared memory. In these cases, it may not be possible to maintain a shadow copy of the entire database.
7. **Limited fault tolerance**: While shadow paging does provide improved fault tolerance in some cases, it does not provide complete fault tolerance. In the event of a crash, there is still a risk of data loss if the changes made during a transaction are not properly copied to the actual database.

Elaborate the Immediate Database Modification with its Recovery mechanism

Immediate Database Modification is a recovery technique that allows database updates of an uncommitted transaction to be made as the writes are issued. This approach ensures that the updates are applied to the database immediately, without waiting for the transaction to commit. Here is an elaboration of Immediate Database Modification along with its recovery mechanism:

Immediate Database Modification:

- **Database Updates**: In Immediate Database Modification, updates made by a transaction are directly applied to the database as the writes are issued, even before the transaction commits.
- **Logging**: Update logs must contain both the old value and the new value of the data item being modified. These logs are written before the database item is updated.
- **Output Operations**: The output of updated blocks can take place at any time before or after the transaction commits. The order in which blocks are output can be different from the order in which they are written.
- **Data Access**: Transactions transfer data items between system buffer blocks and their private work-areas using read(X) and write(X) operations. These operations ensure that the data is accessed and updated correctly.

Recovery Mechanism:

- **Undo and Redo Operations**: The recovery procedure in Immediate Database Modification involves two operations: undo(Ti) and redo(Ti).
- **Undo Operation**: The undo(Ti) operation restores the value of all data items updated by transaction Ti to their old values. This operation is executed by scanning the log records of the transaction and reverting the changes made by it.

- **Redo Operation**: The redo(Ti) operation sets the value of all data items updated by transaction Ti to the new values. This operation is performed after the undo operation to reapply the changes made by the transaction.
- **Idempotent Operations**: Both undo and redo operations must be idempotent, meaning that even if they are executed multiple times, the effect remains the same as if executed once. This ensures consistency during recovery.
- **Recovery Process**: During recovery after a system crash, transactions are undone if the log contains the record <Ti start> but does not contain the record <Ti commit>. Transactions are redone if the log contains both the <Ti start> and <Ti commit> records. The recovery process involves scanning the log, performing undo and redo operations as needed to restore the database to a consistent state.

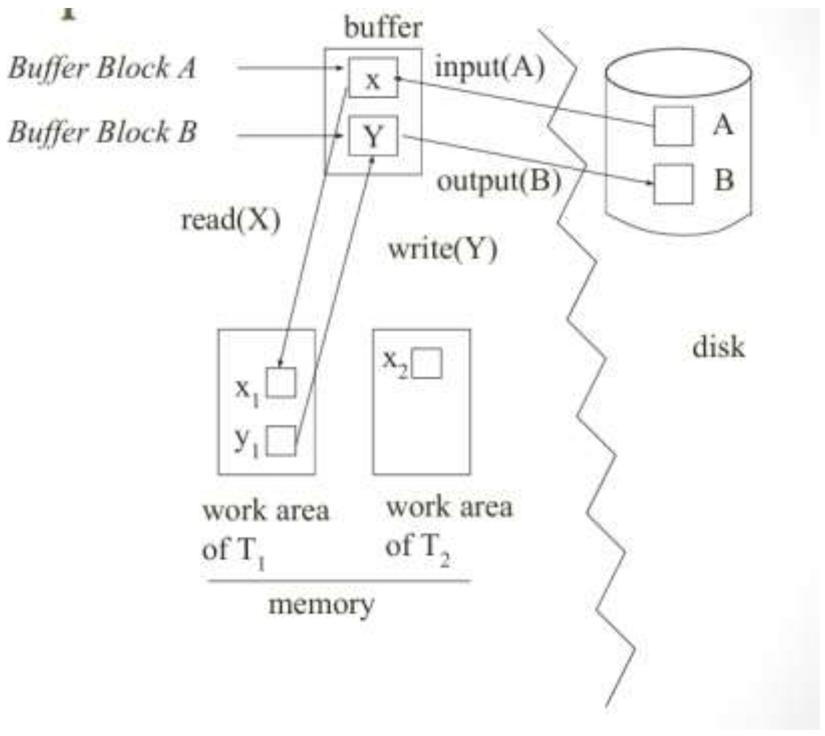
Immediate Database Modification offers a straightforward approach to database updates, ensuring that changes are applied immediately while maintaining the ability to recover the database to a consistent state in case of failures.

Elaborate with appropriate diagram the process through which the transactions read data residing in permanent storage.

The process through which transactions read data residing in permanent storage (i.e., the database) is as follows:

1. **Data Access Operations**: Transactions use the following operations to access data items in the database:
    - read(X): This operation assigns the value of data item X to the local variable xi in the transaction's private work area.
    - write(X): This operation assigns the value of the local variable xi to the data item X in the database.
2. **Buffer Management**: The database maintains an in-memory buffer of data blocks. When a transaction needs to access a data item X, the following steps occur:
    - If the block containing X is already present in the buffer, the read(X) operation can be performed directly on the buffer block.
    - If the block containing X is not in the buffer, an input(BX) operation is issued to bring the block BX containing X from the permanent storage (disk) into the buffer.
    - After the read(X) operation, the transaction can perform subsequent accesses to X using the local copy xi without needing to access the disk again.
3. **Write-Ahead Logging (WAL) Rule**: Before a modified buffer block is output to the permanent storage, all log records pertaining to the updates in that block must first be written to the log on stable storage. This ensures that undo information is available for recovery purposes.
4. **Latch Acquisition**: Before a transaction can perform a write(X) operation, it must acquire an exclusive latch on the block containing X. This latch is held only for the duration of the write operation and is then released. These short-duration locks are called latches.

5. **Deferred Database Modification**: In the deferred database modification scheme, the actual writes to the database are deferred until after the transaction partially commits. The updates are first recorded in the log, and the database is modified later by reading the log records.



Compare Deferred Database Modification and Immediate Database Modification

1. Deferred Update:

it is a technique for the maintenance of the transaction log files of the DBMS. It is also called NO-UNDO/REDO technique. It is used for the recovery of transaction failures that occur due to power, memory, or OS failures. Whenever any transaction is executed, the updates are not made immediately to the database. They are first recorded on the log file and then those changes are applied once the commit is done. This is called the "Re-doing" process. Once the rollback is done none of the changes are applied to the database and the changes in the log file are also discarded. If the commit is done before crashing the system, then after restarting the system the changes that have been recorded in the log file are thus applied to the database.

2. Immediate Update:

It is a technique for the maintenance of the transaction log files of the DBMS. It is also called UNDO/REDO technique. It is used for the recovery of transaction failures that occur due to power, memory, or OS failures. Whenever any transaction is executed, the updates are made directly to the database and the log file is also maintained which contains both old and new values. Once the commit is done, all the changes get stored permanently in the database, and records in the log file are thus discarded. Once rollback is done the old values get restored in the database and all the changes made to the database are also discarded. This is called the "Un-doing" process. If the commit is done before crashing the system, then after restarting the system the changes are stored permanently in the database.

Difference between Deferred update and Immediate update:

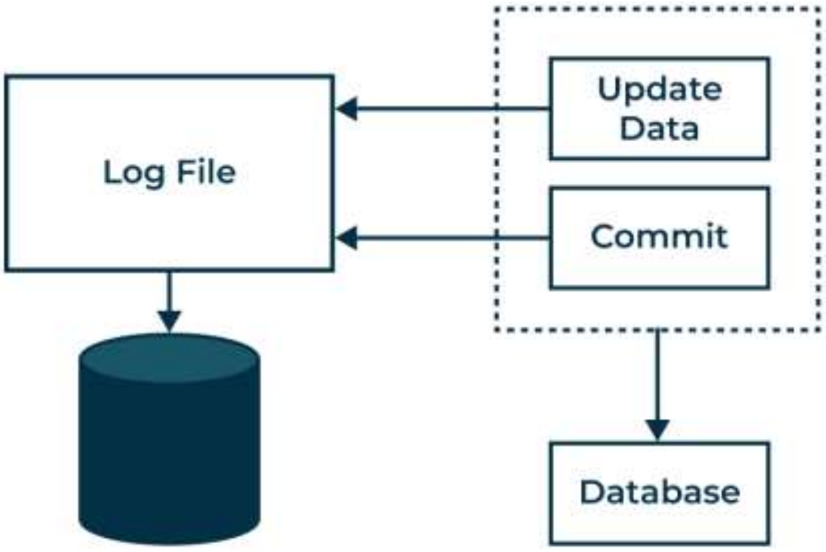
Deferred Update	**Immediate Update**
In a deferred update, the changes are not applied immediately to the database.	In an immediate update, the changes are applied directly to the database.
The log file contains all the changes that are to be applied to the database.	The log file contains both old as well as new values.
In this method once rollback is done all the records of log file are discarded and no changes are applied to the database.	In this method once rollback is done the old values are restored to the database using the records of the log file.
Concepts of buffering and caching are used in deferred update method.	Concept of shadow paging is used in immediate update method.
The major disadvantage of this method is that it requires a lot of time for recovery in case of system failure.	The major disadvantage of this method is that there are frequent I/O operations while the transaction is active.
In this method of recovery, firstly the changes carried out by a transaction on the data are done in the log file and then applied to the database on commit. Here, the maintained record gets discarded **on rollback** and thus, not applied to the database.	In this method of recovery, the database gets directly updated after the changes made by the transaction and the log file keeps the old and new values. In the case of **rollback**, these records are used to restore old values.

Feature	**Deferred Update**	**Immediate Update**
Update timing	Updates occur after instruction execution	Updates occur during instruction execution
Processor speed	May be faster: update occurs after instruction execution, allowing for multiple updates to be performed at once	May be slower: update occurs during instruction execution, potentially causing the processor to stall
Complexity	More complex: requires additional instructions or mechanisms to handle updates after instruction execution	Less complex: updates occur immediately during instruction execution
Consistency	May result in temporary inconsistency between data in registers and memory	Data in registers and memory are always consistent
Flexibility	May be more flexible: allows for more complex data manipulations and algorithms	Less flexible: immediate updates can limit the range of data manipulations and algorithms that can be performed

Explain log-based recovery technique.

Log and Log Records:

- The log is a sequential record of all the activities and modifications made to the database.
- Each transaction's actions are recorded in log records, including updates, inserts, and deletes.
- A log record typically contains information such as the transaction identifier, data item modified, old value, and new value.



Undo and Redo Operations:

- The system utilizes the information in the log records to perform undo and redo operations during recovery.
- Undo operation: Reverts the database to its state before the transaction was executed, using the old value recorded in the log.
- Redo operation: Reapplies the changes made by a completed transaction to the database, using the new value recorded in the log.

Deferred vs. Immediate Modification Techniques:

- Deferred Modification Technique: Transactions modify the database only after partial commitment, reducing the risk of data inconsistency.

- Immediate Modification Technique: Transactions modify the database while still active, potentially leading to data inconsistencies.

Recovery Using Log Records:

- After a system crash, the DBMS consults the log to determine which transactions need to be redone or undone.
- Transactions need to be undone if they were active but not committed or aborted at the time of the crash.
- Transactions need to be redone if they were active and committed or aborted at the time of the crash.

Use of Checkpoints:

- Checkpoints are used to reduce recovery overhead by marking a point in the log where all transactions up to that point have been completed.
- During recovery, the system starts from the last checkpoint to identify the set of transactions that need to be redone or undone.
- This reduces the search space and speeds up the recovery process.

Advantages of Log-Based Recovery:

1. **Durability:** Ensures that committed transactions are not lost, even in the event of a system crash.
2. **Faster Recovery:** Recovery can be performed by replaying committed transactions from the log, leading to faster recovery times.
3. **Incremental Backup:** Allows for incremental backups by storing only the changes made since the last backup in the log.
4. **Data Integrity:** Reduces the risk of data corruption by ensuring that all transactions are correctly committed or rolled back.

Disadvantages of Log-Based Recovery:

1. **Additional Overhead:** Maintaining the log file adds overhead to the database system, potentially impacting performance.

2. **Complexity:** Log-based recovery is a complex process that requires careful management and administration.
3. **Storage Space:** The log file can consume significant storage space, especially in databases with large transaction volumes.
4. **Time-Consuming:** Recovering transactions from the log file can be time-consuming, particularly for databases with numerous transactions.