UNIT-6

Database Recovery

Introduction:

- ➔ It refers to the process of restoring a database to a consistent state after a failure.
- ➔ Failures can occur due to various reasons such as hardware malfunction, software errors, power outages, or even natural disasters.
- ➔ The goal of database recovery is to ensure the database remains consistent and that no data is lost or corrupted.

Types of Failures in DBMS:

1. Transaction Failure:

   Occurs when a transaction cannot complete successfully due to logical errors, deadlocks, or system errors.

2. System Crash:

   Happens when the database server crashes due to software, operating system, or hardware failures.

3. Media Failure:

   This involves a failure of the storage media, such as a disk crash, which can cause the loss of the entire database or parts of it.

4. Catastrophic Failure:

   Disasters like fires, floods, or earthquakes that destroy the database and related infrastructure.

Example:

- Suppose a system crashes after a transaction that transfers money between two bank accounts. In this case, during recovery:
  - The system uses **undo logging** to revert any incomplete transactions (if the transaction was not committed).
  - The system uses **redo logging** to reapply the transaction if it was committed but not written to the database yet.

Faculty : Shubham Upadhyay

Log Based Recovery:

- ➔ The log is a sequence of records. Log of each transaction is maintained in some stable storage so that if any failure occurs, then it can be recovered from there.
- ➔ If any operation is performed on the database, then it will be recorded in the log.
- ➔ Let's assume there is a transaction to modify the City of a student. The following logs are written for this transaction.

  When the transaction is initiated, then it writes 'start' log.

  - ◆ <Ti, Start>

  When the transaction modifies the City from 'Noida' to 'Bangalore', then another log is written to the file.

  - ◆ <Ti, City, 'Noida', 'Bangalore' >

  When the transaction is finished, then it writes another log to indicate the end of the transaction.

  - ◆ <Ti, Commit>

Types of log based recovery method:

1. **Immediate database modification**
2. **Deferred database modification**

1. Immediate database modification :

- ➔ The Immediate modification technique occurs if database modification occurs while the transaction is still active.
- ➔ In this technique, the database is modified immediately after every operation. It follows an actual database modification.

2. Deferred database modification :

- ➔ The deferred modification technique occurs if the transaction does not modify the database until it has committed.
- ➔ **Faculty : Shubham Upadhyay**

➔ In this method, all the logs are created and stored in the stable storage, and the database is updated when a transaction commits.

Recovery using Log records

When the system is crashed, then the system consults the log to find which transactions need to be undone and which need to be redone.

1. If the log contains the record <Ti, Start> and <Ti, Commit> or <Ti, Commit>, then the Transaction Ti needs to be redone.
2. If log contains record<Tn, Start> but does not contain the record either <Ti, commit> or <Ti, abort>, then the Transaction Ti needs to be undone.

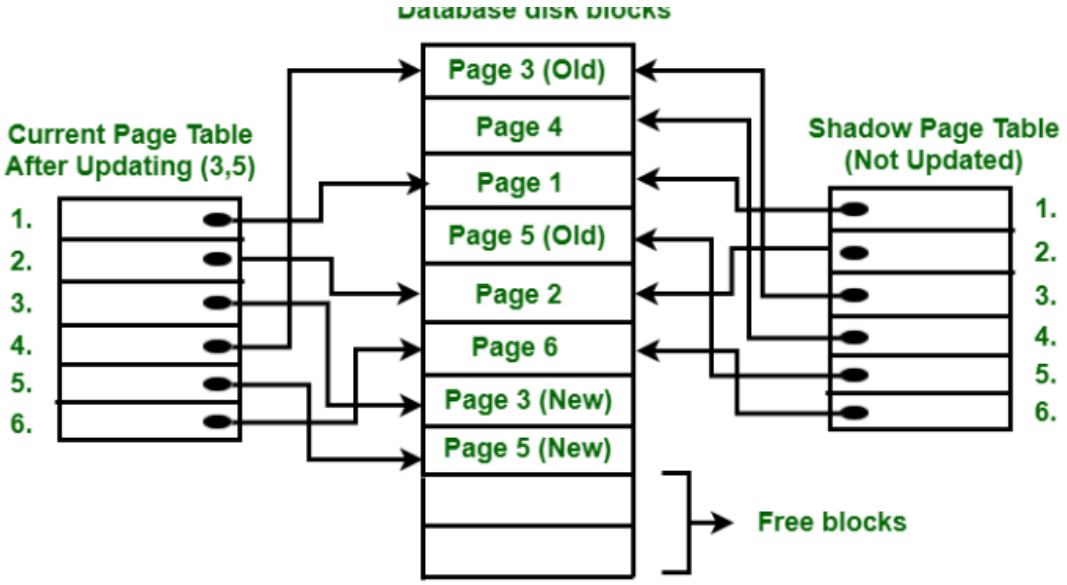
Shadow Paging:

➔ Shadow Paging is a recovery technique where the database pages (units of storage) are not modified directly during a transaction.
➔ Instead, a shadow copy of the page is created, and all updates are made on this shadow page.
➔ The actual database page is only replaced with the shadow page when the transaction is committed.
➔ If the transaction fails, the shadow page is discarded, and the original database page remains intact, thus preserving the consistency of the database.

How Does Shadow Paging Work?

1. **Page Table**:
    - The database is divided into fixed-size pages, and a page table is maintained to keep track of the location of each page in storage.
    - During normal operation, the page table points to the current database pages.
2. **Shadow Page Table**:
    - When a transaction begins, a copy of the page table, called the **shadow page table**, is created. This shadow page table points to the original pages in the database.
    - Any modifications during the transaction are made on new pages, and the shadow page table is updated to point to these new pages.

3. **Commit Phase**:
    - When the transaction is committed, the original page table is replaced with the shadow page table. This makes the new pages part of the actual database.
    - The old pages that were not modified are now considered "shadow pages" and can be discarded.
4. **Failure Recovery**:
    - If a transaction fails before commit, the original page table is still in use, so the database remains in a consistent state. No changes are written to the database, and the shadow pages are discarded.



In this 2 write operations are performed on page 3 and 5. Before start of write operation on page 3, current page table points to old page 3. When write operation starts following steps are performed :

1. Firstly, search start for available free block in disk blocks.
2. After finding free block, it copies page 3 to free block which is represented by Page 3 (New).
3. Now current page table points to Page 3 (New) on disk but shadow page table points to old page 3 because it is not modified.
4. The changes are now propagated to Page 3 (New) which is pointed by current page table.
5. **Faculty : Shubham Upadhyay**

Checkpoints:

- ➔ A checkpoint is a mechanism used to reduce the time required for recovery in case of a system failure.
- ➔ It is part of the recovery process in transactional systems, especially those that support transaction logging.
  Key Concepts:

1. **Transaction Logging**: In a DBMS, every transaction is logged (using a write-ahead log, or WAL) to ensure that the system can recover from failures by replaying or undoing logged actions. This ensures the **ACID** properties of transactions (Atomicity, Consistency, Isolation, Durability).
2. **Need for Checkpoints**: In a large system, the log may grow very large, and during recovery, it would take a long time to read the entire log from the beginning to determine which transactions to redo or undo. **Checkpoints** help limit the amount of log data that needs to be scanned during recovery.

How Checkpoints Work:

1. **Initiating a Checkpoint**:
   - The DBMS periodically creates a checkpoint in the log file.
   - A checkpoint forces all changes in the database buffers to be written to disk, ensuring that the database is consistent up to that point.
2. **Checkpoint Process**:
   - The **DBMS stops accepting new transactions temporarily**.
   - All dirty pages (pages that have been modified but not yet written to disk) are flushed from memory to disk.
   - A checkpoint record is written to the log, indicating that all previous transactions have been committed, and the data is now consistent up to that point.
3. **After a Checkpoint**:
   - Only log records after the checkpoint need to be read during recovery. The DBMS knows that data prior to the checkpoint is already written and consistent on disk.
   - This reduces the recovery time, as the system only needs to process a smaller part of the log.

Types of Checkpoints:

1. **Consistent Checkpoint**:
   - The system ensures that all transactions before the checkpoint are fully committed and reflected on disk.
   - This is the simplest form of a checkpoint and provides quick recovery.
2. **Fuzzy Checkpoint**:

    - In this method, the system does not wait for all transactions to complete before writing a checkpoint. It writes part of the in-progress transactions to disk but logs the state of those transactions.
    - Fuzzy checkpoints are faster to create but can involve more complex recovery processes, as some incomplete transactions need to be rolled back during recovery.

Checkpoint in the Recovery Process:

During recovery, after a failure:

1. The system reads the last checkpoint from the log.
2. It starts reading the log from the checkpoint forward, identifying transactions that need to be redone (if they were committed) or undone (if they were not committed).
3. Any transactions that were committed before the checkpoint are considered already applied and are ignored in recovery.

Advantages of Checkpoints:

- **Reduced Recovery Time**: Checkpoints limit how far back the DBMS must go in the log file to recover from a failure.
- **Improved Performance**: Since recovery scans a shorter log after a checkpoint, system downtime due to failures is minimized.
- **Prevents Log Overflow**: By regularly creating checkpoints, the log size can be managed, preventing unbounded growth.

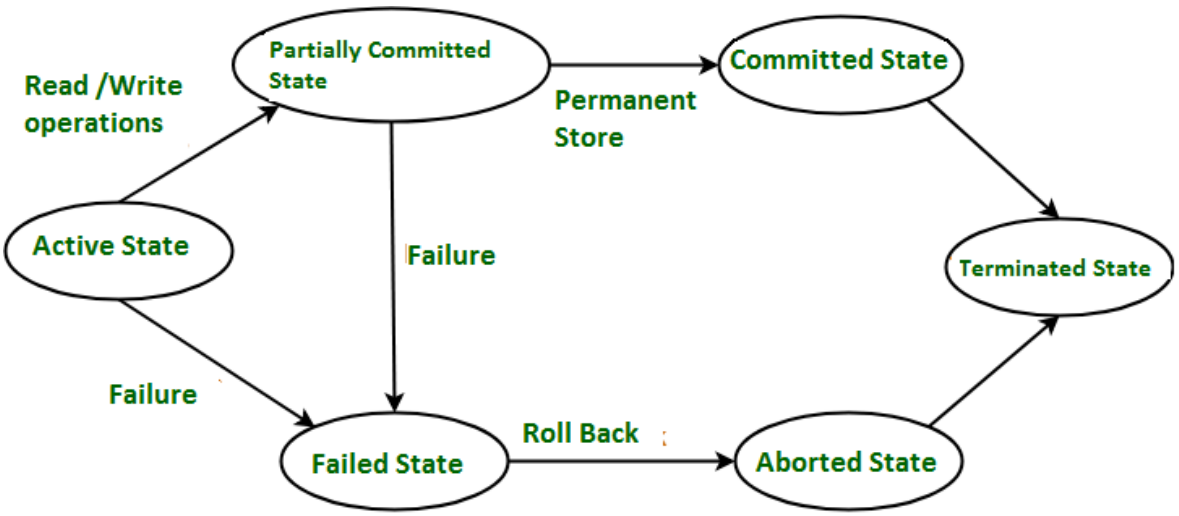
Transaction:

➔ Transaction in DBMS is a set of logically related operations executed as a single unit.
➔ A database transaction must be atomic, meaning that it must be either entirely completed or aborted.
➔ Transfer of money from one account to another in a bank management system is the best example of Transaction.

Transaction States in DBMS

A Transaction log is a file maintained by the recovery management component to record all the activities of the transaction. After the commit is done, the transaction log file is removed.

Faculty : Shubham Upadhyay



Transaction States in DBMS

These are different types of Transaction States :

1. Active State – When the instructions of the transaction are running then the transaction is in active state. If all the 'read and write' operations are performed without any error then it goes to the "partially committed state"; if any instruction fails, it goes to the "failed state".

2. Partially Committed – After completion of all the read and write operations the changes are made in the main memory or local buffer. If the changes are made permanent on the DataBase then the state will change to "committed state" and in case of failure it will go to the "failed state".

3. Failed State – When any instruction of the transaction fails, it goes to the "failed state" or if failure occurs in making a permanent change of data on Database.

4. **Aborted State –** After having any type of failure the transaction goes from "failed state" to "aborted state" and since in previous states, the changes are only made to local buffer or main memory and hence these changes are deleted or rolled-back.

5. Committed State – It is the state when the changes are made permanent on the Database and the transaction is complete and therefore terminated in the "terminated state".

Faculty : Shubham Upadhyay

6. Terminated State – If there isn't any roll-back or the transaction comes from the "committed state", then the system is consistent and ready for new transaction and the old transaction is terminated.

Properties of Transaction:

Transactions follow 4 properties, namely, **Atomicity, Consistency, Isolation, and Durability**.

i) Atomicity

- This property ensures that either all operations of a transaction are executed or it is aborted. In any case, a transaction can never be completed partially.
- Each transaction is treated as a single unit.
- Atomicity is achieved through commit and rollback operations, i.e. changes are made to the database only if all operations related to a transaction are completed, and if it gets interrupted, any changes made are rolled back using rollback operation to bring the database to its last saved state.

ii) Consistency

- This property of a transaction keeps the database consistent before and after a transaction is completed.

iii) Isolation

- This property states that two transactions must not interfere with each other, i.e. if some data is used by a transaction for its execution, then any other transaction can not concurrently access that data until the first transaction has completed.
- It ensures that the integrity of the database is maintained.

Faculty : Shubham Upadhyay

iv) Durability

- This property ensures that the changes made to the database after a transaction is completely executed, are durable.
- It indicates that permanent changes are made by the successful execution of a transaction.

Scheduling:

➔ A schedule is a process of grouping the transactions into one and executing them in a predefined order.
➔ Is a process of lining the transactions and executing them one by one.
➔ When there are multiple transactions that are running in a concurrent manner and the order of operation is needed to be set so that the operations do not overlap each other.
➔ There are two types of scheduling:
    1. Serial
    2. Non-Serial

Serial Scheduling:

➔ A serial schedule is a schedule in which no transaction starts until a running transaction has ended.
➔ A serial schedule is a schedule in which one transaction is executed completely before starting another transaction.

Faculty : Shubham Upadhyay

Example of Serial Schedule

Serial Schedule

T1	T2
Read(A) Write(A) Read(B) Write(B)	Read(A) Write(A) Read(B) Write(B)

Non-Serial Schedule:

➔ This is a type of Scheduling where the operations of multiple transactions are interleaved.

Example of Non Serial Schedule

Non-serial schedule

T1	T2
Read(A) Write(A) Read(B) Write(B)	 Read(A) Write(A) Read(B) Write(B)

Faculty : Shubham Upadhyay

Serializability:

- ➔ Serializability refers to the property that ensures the correctness of transactions in a concurrent database environment.
- ➔ It ensures that the final outcome of concurrent transaction execution is the same as if the transactions were executed sequentially, one after another, without any overlap.

Why is Serializability Important?

In a database, multiple transactions may occur at the same time. These transactions might access or modify the same data items. If proper controls are not in place, this could lead to problems like:

- **Lost updates**: Two transactions update the same data, and one update is lost.
- **Dirty reads**: A transaction reads uncommitted data from another transaction.
- **Inconsistent data**: A transaction sees an inconsistent state in the middle of another transaction's execution.

Types (forms) of serializability

1. **Conflict serializability**
2. **View Serializability**

Conflict serializability:

- ➔ If a given schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called as a conflict serializable schedule.

Conflicting Operations : A pair of operations are said to be conflicting operations if they follow the set of conditions given below:-

➔ Each operation is a part of different transactions
➔ Both operations are performed on the same data item
➔ One of the performed must be a write operation

Concurrency Control:

➔ Concurrency control is a set of mechanisms and techniques used to ensure that multiple transactions can access and modify the database concurrently without leading to data inconsistency.
➔ Here are the key concurrency problems:

1. Lost Update Problem

- **Description**: This occurs when two or more transactions read the same data and then update it based on the value they read. The last update overwrites the previous ones, leading to loss of data.
- **Example**:
    - Transaction T1 and T2 both read the same account balance of Rs500.
    - T1 adds Rs100, making the balance Rs600.
    - T2 adds Rs200, making the balance Rs700.
    - T2 writes Rs700 to the database, but T1 had already writtenRs600, resulting in a lost update and incorrect final balance.

2. Dirty Read (Uncommitted Data)

- **Description**: A dirty read happens when a transaction reads data that has been modified by another transaction that has not yet committed. If the second transaction is rolled back, the first transaction ends up with incorrect data.
- **Example**:
    - Transaction T1 updates a product's price from Rs10 to Rs15 but has not yet committed.
    - Transaction T2 reads the uncommitted price of Rs15 and makes further calculations.
    - If T1 rolls back (e.g., due to an error), T2 has based its work on incorrect, temporary data.

3. Non-Repeatable Read (Inconsistent Retrievals)

- **Description**: This occurs when a transaction reads the same data multiple times and gets different results because another transaction modifies the data between the reads.

- **Example**:
    - Transaction T1 reads the balance of an account as Rs500.
    - Transaction T2 modifies the balance to Rs700 and commits.
    - When T1 reads the balance again, it sees Rs700, resulting in inconsistent results within the same transaction.

Lock Based Protocol:

➔ Lock-Based Protocols in DBMS are concurrency control mechanisms that ensure transactions are executed in a serializable manner to prevent data inconsistencies during concurrent transaction execution.
➔ These protocols use locks to restrict access to database resources (like rows or tables) when multiple transactions are operating on the same data.
➔ There are two types of locks used in databases.

1.Shared Lock : Shared lock is also known as read lock which allows multiple transactions to read the data simultaneously. The transaction which is holding a shared lock can only read the data item but it can not modify the data item.

2.Exclusive Lock : Exclusive lock is also known as the write lock. Exclusive lock allows a transaction to update a data item. Only one transaction can hold the exclusive lock on a data item at a time. While a transaction is holding an exclusive lock on a data item, no other transaction is allowed to acquire a shared/exclusive lock on the same data item.

Two Phase Lock Protocol:

➔ The Two-Phase Locking Protocol (2PL) is one of the most widely used concurrency control mechanisms in Database Management Systems (DBMS).
➔ It ensures serializability, which means that the execution of transactions will be equivalent to some serial execution, maintaining the database's consistency and integrity.
➔ Two phase locking protocol works in two phases.

Faculty : Shubham Upadhyay

1. **Growing Phase :** In this phase, the transaction starts acquiring locks before performing any modification on the data items.
    ➔ Once a transaction acquires a lock, that lock can not be released until the transaction reaches the end of the execution.
2. **Shrinking Phase :** In this phase, the transaction releases all the acquired locks once it performs all the modifications on the data item.
    ➔ Once the transaction starts releasing the locks, it can not acquire any locks further.

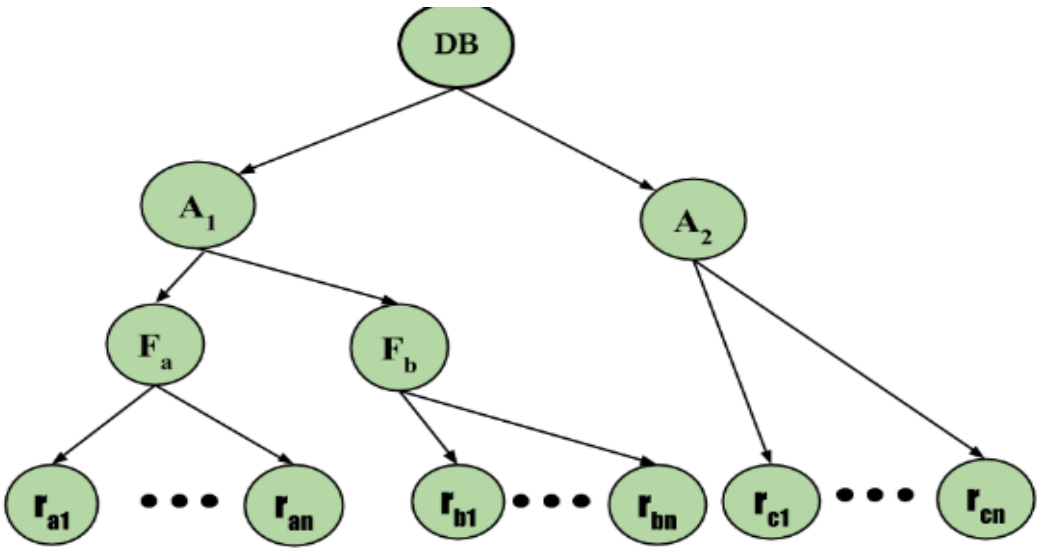
Multiple Granularity:

➔ It is the size of the data item allowed to lock.
➔ that allows transactions to lock different levels of a database hierarchy (such as database, table, page, row) with different types of locks.

Granularity Hierarchy:

In a DBMS, data can be organized in a hierarchical structure, such as:

- **Database**: The highest level of granularity.
- **File**: A specific file within the database.
- **Table**: A table within the file.
- **Page**: A portion of data within the table.
- **Row**: The smallest unit of data, typically a row or record in the table.



Faculty : Shubham Upadhyay

➔ Locks can be applied at different levels of this hierarchy.
➔ By using multiple granularity locking, a transaction can lock a higher-level object (like a table) or a lower-level object (like a row) depending on the required operations.

Intention Locking:

➔ Intention Locking in DBMS is a type of locking mechanism that allows for better concurrency control, especially in hierarchical data structures like tables and rows.
➔ It is primarily used to manage the locking of larger data granularity levels (such as a table) while ensuring that finer-grained locks (such as row-level locks) can be applied without conflict.
➔ Intention locks indicate that a transaction intends to acquire a more specific type of lock at a finer granularity in the hierarchy.

Types of Intention Locks: There are three main types of intention locks:

- **Intention Shared (IS)**: A transaction intends to acquire **shared locks** at a finer granularity (e.g., a row in a table). The IS lock is placed at a higher level (e.g., table) to signal that the transaction will read data from finer levels.

- **Intention Exclusive (IX)**: A transaction intends to acquire **exclusive locks** at a finer granularity. The IX lock signals that the transaction will modify data at the finer level (e.g., rows).

- **Shared-Intention Exclusive (SIX)**: A transaction holds a **shared lock** on the entire data structure but also intends to acquire **exclusive locks** at a finer granularity. This lock mode allows a transaction to read all data at the higher level (e.g., table) but modify individual data items (e.g., rows) with exclusive locks.

Deadlock:

- ➔ The Deadlock is a condition in a multi-user database environment where transactions are unable to the complete because they are each waiting for the resources held by other transactions.
- ➔ This creates a circular wait, where each process is waiting for a resource that is held by another process, leading to a complete blockage of the system.

Conditions for Deadlock:

There are four necessary conditions for a deadlock to occur:

1. **Mutual Exclusion:** At least one resource must be held in a non-shareable mode, meaning only one transaction can use the resource at a time.
2. **Hold and Wait:** A transaction holding at least one resource is waiting to acquire additional resources held by other transactions.
3. **No Preemption:** Resources cannot be forcibly taken away from a transaction; the transaction must release them voluntarily.
4. **Circular Wait:** A circular chain of transactions exists, where each transaction holds a resource and waits for another resource held by the next transaction in the chain.