## ****1. Purpose of Database Recovery****

**Define database recovery:** The process of restoring the database to a correct and consistent state after a failure.  
Database recovery ensures the database can return to a consistent state after failures.  
Failures include system crashes, transaction errors, disk failures, and natural disasters.

**Why recovery is needed:**

The necessity of database recovery arises from the multitude of potential failures that can compromise data.

**Hardware failures:** Disk crashes, power outages.

**Software failures:** Operating system errors, DBMS bugs.

**Application errors:** Incorrect program logic.

**Human errors:** Accidental deletion of data.

**Natural disasters:** Fires, floods.

**Database recovery** is a critical mechanism that ensures:

* Data consistency and integrity after failures
* System reliability and availability
* Transaction durability (ACID property)
* Protection against various failure types:

System crashes

Hardware failures

Software errors

Human errors

Natural disasters

****Key Objectives:****

* Restore the database to a consistent state after failure
* Preserve committed transactions
* Undo uncommitted transactions
* Minimize downtime and data loss

## ****Components of Database Recovery (Based on Diagram)****

The diagram shows the interaction between the following components.

**- Query Processor:** Accepts SQL queries and passes instructions to the transaction manager.  
**- Transaction Manager:** Coordinates the execution of transactions. Ensures ACID properties.  
**- Log Manager:** Maintains a log of all activities before they're applied to the database.  
**- Buffer Manager:** Manages the buffer pool and controls reading/writing to disk.  
**- Recovery Manager:** Uses logs to undo or redo transactions after failure.

We will focus mainly on: **Logging, Log Manager, and Recovery Manager**.

## ****3. Logging****

### Definition:

**Logging** is the process of recording all changes made to the database by a transaction before they are committed. These logs are stored in a special file called the **log file**.

### Purpose:

To record transaction activities for recovery purposes.

To ensure atomicity and durability (ACID properties).

To enable the system to **redo or undo** transactions.

**The characteristics of a log:**

**Sequential:** Log records are written in chronological order.

**Persistent:** Stored on stable storage (e.g., disk) to survive failures.

**Append-only:** New records are added to the end.

### What is Logged:

Detail the contents of a log record. For example:

**Transaction ID:** Identifies the transaction that made the change.

**Data item ID**: Identifies the specific data that was modified.

**Before image (old value):** The value of the data *before* the change.

**After image (new value):** The value of the data *after* the change.

**Log record type:** e.g., START TRANSACTION, WRITE, COMMIT, ABORT.

**The importance of the Write-Ahead Logging (WAL) principle:**

The Write-Ahead Logging (WAL) principle is fundamental to ensuring database recoverability. It states:

The log record for a change must be written to stable storage *before* the change is applied to the database itself.

This ensures that if a crash occurs, the system can use the log to either undo the change (if the transaction didn't commit) or redo the change (if the transaction did commit).

Example:

Consider a transaction T1 that transfers $50 from account A to account B.

The log might contain the following records:

<T1, START>

<T1, A, 100, 50> (A was $100, now $50)

<T1, B, 200, 250> (B was $200, now $250)

<T1, COMMIT>

If a crash occurs before <T1, COMMIT> is written, the Recovery Manager can use the log to undo the changes to A and B. If the crash occurs after <T1, COMMIT>, the Recovery Manager can use the log to redo the changes if they weren't written to disk before the crash.

### Types of Logs:

**Undo Logs** – To revert incomplete transactions.

**Redo Logs** – To reapply committed transactions during recovery.

## ****4. Log Manager****

### Definition:

The **Log Manager** is responsible for creating, maintaining, and managing the log files. It ensures that all necessary actions are logged properly and persistently.

### Responsibilities:

**Log record creation:** Generates log records for every database modification.

**Log buffer management:** Temporarily stores log records in an in-memory buffer to improve performance. Explain the concept of flushing the buffer to disk.

**Log file management:** Allocates and manages the physical log file(s) on stable storage. This includes issues like log file size, and how new log files are created.

**Write-ahead logging enforcement:** Ensures that log records are written to disk before the corresponding data changes are written to disk .

### Interaction (Based on Diagram):

The **Transaction Manager** sends transaction information to the **Log Manager**.

The **Log Manager** writes this data to the log file before any update is made to the **Buffer Manager** or actual **Data**.

In case of failure, the Log Manager provides data to the **Recovery Manager**.

## ****5. Recovery Manager****

### Definition:

The **Recovery Manager** is responsible for restoring the database to a consistent state after a crash or failure.

**The types of failures that the Recovery Manager handles:**

**Transaction failure:** A single transaction fails (e.g., due to an error in the application).

**System failure:** The entire DBMS or operating system crashes.

**Media failure:** A disk crash (the most serious type of failure).

### Phases of Recovery:

**Analysis phase:** The Recovery Manager examines the log to determine:

* Which transactions were committed before the crash.
* Which transactions were active (not committed) at the time of the crash.
* The point in the log where recovery should start.

**Redo phase:** The Recovery Manager re-applies the changes of all committed transactions. It reads the log forward from the appropriate point and reapplies the 'after images' of the data.

**Undo phase:** The Recovery Manager undoes the changes of all uncommitted transactions. It reads the log backward and restores the 'before images' of the data.

### Interaction:

Uses the logs maintained by the **Log Manager**.

Works with the **Buffer Manager** to reload consistent data into memory.

Ensures that only committed transactions remain in the final database state.