DBMS QB – UNIT – 5

[4-marks]

1. What is Database Recovery in DBMS?

- **Database Recovery** is the process of restoring the database to a **correct and consistent state** after a failure occurs. A failure may happen due to system crash, transaction error, or disk problem. The main goal of recovery is to ensure that the database does not remain in an incorrect or partially updated state.
- In DBMS, every transaction is treated as a **logical unit of work**. This means either all operations of a transaction must be completed, or none of them should affect the database. Recovery ensures this rule is followed even after failures.
- Recovery uses a **system log**, which stores information about all changes made by transactions. By checking the log, the DBMS decides which operations need to be undone or redone to restore consistency.

2. List any four types of failures in a database system

- **Transaction Failure** occurs when a transaction cannot complete its execution. This may happen due to logical errors, invalid input, division by zero, or when a user manually cancels the transaction. In such cases, partial changes must be undone.
- **System Crash** happens due to hardware, software, or power failure. The contents of main memory are lost, but the database stored on disk usually remains safe. Recovery is required to restore the database to a consistent state.
- **Disk Failure** occurs when disk blocks lose data due to read/write errors or disk head crash. This type of failure can cause loss of database data stored on disk and may require backup restoration.
- **Physical or Catastrophic Failure** includes events like fire, flood, power failure, theft, or accidental overwriting of disks. These failures can damage a large portion of the database and require restoring from backup.

3. What is a System Crash? Give one example

- A **System Crash** is a type of failure where the database system stops working due to problems in hardware, software, or power supply. During a system crash, the main memory contents are lost, but the data stored on disk is usually not damaged.
- System crashes interrupt transaction execution. Some transactions may have completed, while others may be partially executed. This can leave the database in an inconsistent state if recovery is not applied.
- **Example:** A sudden **power failure** while transactions are executing causes the system to shut down immediately. When the system restarts, recovery is required to undo incomplete transactions and redo committed ones using the log.

4. What is the purpose of REDO in transaction recovery?

- The **REDO operation** is used to **reapply the effects of committed transactions** whose updates were not written to disk before a system failure occurred. This ensures that all committed transactions are reflected correctly in the database.
- Sometimes a transaction commits successfully, but the system crashes before its updated data pages are written from memory to disk. In such cases, the database on disk does not show the committed changes.
- During recovery, the DBMS scans the log and performs REDO on all committed transactions after the last checkpoint. This guarantees durability and ensures that committed work is not lost.

5. What is the purpose of UNDO operation?

- The **UNDO operation** is used to **reverse the effects of incomplete or failed transactions**.
- If a transaction fails before reaching its commit point, any changes it made must be removed from the database.
- UNDO ensures that partial updates of a transaction do not remain in the database. This helps maintain database consistency and follows the all-or-nothing rule of transactions.
- For UNDO to work, the system log stores **old values** of data items before they are modified. During recovery, these old values are restored to roll back the transaction safely.

6. Define Deferred Update

- **Deferred Update** is a recovery technique in which **actual updates to the database on disk are postponed until the transaction commits**. During transaction execution, all changes are kept only in memory buffers and recorded in the log.
- If a transaction fails before committing, **no UNDO is required** because the database on disk was never changed. The system simply discards the in-memory changes.
- After a transaction commits, the updates are written to the database. If a crash occurs after commit but before writing data to disk, **REDO** is used to reapply the changes from the log.
- Because it requires **NO-UNDO and REDO**, this method is also called the **NO-UNDO/REDO protocol**.

7. Define Immediate Update with an example

- **Immediate Update** is a recovery technique in which **database pages on disk can be updated before a transaction commits**. This means changes made by a transaction may reach disk even if the transaction is not completed.
- Since uncommitted changes can be written to disk, **UNDO may be required** if the transaction fails. The system log must store old values so changes can be reversed.
- **Example:**
  A transaction transfers money between two accounts and updates the balance on disk. If the system crashes before commit, the updated balance must be undone to restore the original value.
- Depending on the strategy used, immediate update may require **UNDO only** or **both UNDO and REDO** during recovery.

8. What is Shadow Paging?

- **Shadow Paging** is a recovery technique that does **not require a log in a single-user environment**. It uses two page directories: a **shadow directory** and a **current directory**.
- When a transaction starts, the current directory is copied to create a shadow directory. The shadow directory points to the old database pages, while the current directory is used for updates.
- When a page is updated, the system creates a **new copy of the page** instead of overwriting the old one. The current directory points to the new page, while the shadow directory still points to the old page.
- If a crash occurs before commit, the system restores the shadow directory. This provides **NO-UNDO and NO-REDO** recovery.

9. What is a Checkpoint?

- A **Checkpoint** is a special record written to the transaction log that marks a point where the database and log are in a **consistent state**. It helps reduce recovery time after a system crash.
- During checkpointing, all modified memory buffers are **force-written to disk**, and a checkpoint record is added to the log. This record also stores a list of active transactions.
- After a crash, the DBMS does not need to scan the entire log. It only needs to process log records **after the last checkpoint**, which makes recovery faster.
- Checkpoints may be taken after a fixed time interval or after a certain number of transactions.

10. Define Transaction Log

- A **Transaction Log** is a **sequential file** that records all actions performed by transactions in the database system. It is stored on stable storage such as disk.
- The log contains records like **start transaction, write operations, commit, abort, and checkpoint**. For write operations, it may store old values, new values, or both.
- The main purpose of the transaction log is to support **recovery operations such as UNDO and REDO**. By analyzing the log, the DBMS can restore the database to a consistent state after failure.
- The transaction log is essential for ensuring **atomicity and durability** of transactions.

11. What is Media Failure?

- **Media Failure** occurs when the **physical storage media** of the database, such as a disk, gets damaged and loses data. This type of failure affects the database stored on disk rather than only the main memory.
- Media failure can happen due to **disk read/write head crash**, bad sectors, or hardware malfunction. In such cases, some disk blocks may lose their stored data permanently.
- This failure is more serious than a system crash because actual database files may be destroyed. Recovery usually requires **restoring the database from a backup** and then applying REDO operations using the transaction log.

12. What is meant by Backup in a DBMS?

- A Backup in DBMS is a saved copy of the database stored on external or archival storage such as tapes, external disks, or cloud storage. It is used to restore data when major failures occur.
- Backup is mainly required during media failures or catastrophic failures, where the database on disk is damaged or completely lost. In such situations, the live database cannot be recovered using logs alone.
- After restoring the backup, the DBMS uses the transaction log to reapply committed transactions. This helps reconstruct the database to the most recent consistent state before failure.

13. Differentiate between REDO and UNDO

- REDO is used to reapply changes of committed transactions whose updates were not written to disk before a crash. It ensures that committed work is not lost.
- UNDO is used to reverse changes of uncommitted or failed transactions. It removes partial updates that should not affect the database.
- REDO uses new values stored in the log, while UNDO uses old values recorded before modification. Both operations help maintain database consistency.
- REDO ensures durability, whereas UNDO ensures atomicity of transactions.

14. What is meant by a Write-Ahead Log (WAL)?

- Write-Ahead Logging (WAL) is a recovery rule that states log records must be written to disk before actual database updates are written to disk. This ensures recovery is always possible.
- Before a transaction commits, all its REDO and UNDO log entries must be force-written to the log file. Only after that is the transaction considered committed.
- WAL guarantees that in case of a crash, the DBMS can undo uncommitted changes or redo committed changes using the log. It is a fundamental requirement for safe database recovery.

15. What is Soft Crash and Hard Crash?

- A Soft Crash occurs when the system crashes but the database disk is not physically damaged. Examples include power failure or operating system crash.
- In a soft crash, only main memory contents are lost, while disk data remains intact. Recovery is done using transaction logs, UNDO, and REDO operations.
- A Hard Crash occurs when the database disk itself is damaged due to media failure or physical disaster. In this case, data stored on disk is lost.
- Hard crashes require backup restoration followed by log-based recovery, making them more serious than soft crashes.

[5-marks]

1. Explain different types of database failures with examples

Database failures occur when the DBMS cannot complete transactions correctly due to unexpected problems. These failures are mainly classified into different types.

- **Transaction Failure** happens when a transaction cannot finish execution. This may occur due to logical errors, invalid input, division by zero, or when a user cancels the transaction. For example, a bank transaction fails due to insufficient balance.
- **System Failure (System Crash)** occurs due to hardware, software, or power failure. In this case, main memory contents are lost, but disk data usually remains safe. For example, a sudden power cut shuts down the database server.
- **Media Failure** happens when storage devices like disks are damaged. Disk head crash or bad sectors can destroy stored data. For example, a hard disk crash causes loss of database files.
- **Physical or Catastrophic Failure** includes fire, flood, theft, or accidental overwriting of disks. These failures affect large parts of the database and require backup restoration.

2. Describe REDO and UNDO operations and explain when they are used

REDO and UNDO are recovery operations used to restore database consistency after failures.

- **REDO operation** is used to reapply the changes of **committed transactions** whose updates were not written to disk before a crash. This ensures that committed work is not lost. REDO is performed during recovery by reading new values from the transaction log.
- REDO is mainly used when a transaction commits successfully, but the system crashes before its updates are saved to disk. In such cases, the log is used to redo the operations.
- **UNDO operation** is used to reverse the changes made by **uncommitted or failed transactions**. This prevents partial updates from remaining in the database.
- UNDO is applied when a transaction fails before commit or during system crash. Old values stored in the log are restored to maintain consistency.

3. Explain Deferred Update recovery technique with an example

The **Deferred Update recovery technique** delays actual updates to the database until the transaction commits.

- In deferred update, all changes made by a transaction are stored only in **memory buffers and the transaction log** during execution. The database on disk remains unchanged until commit.
- If a transaction fails before commit, **no UNDO is required** because no changes were written to disk. The system simply discards the in-memory changes.
- After commit, updates are written to disk. If a crash occurs after commit but before disk write, **REDO is required** to apply changes from the log.
- **Example:**
  Transaction T updates account balance but crashes before commit. Since the database was not updated, changes are ignored. If T commits and crash occurs, REDO applies updates using the log.

This method is also called **NO-UNDO/REDO** recovery.

4. Explain Immediate Update and how it helps in recovery

The **Immediate Update recovery technique** allows database updates to be written to disk **before the transaction commits**.

- In this method, when a transaction updates data, the changes may immediately reach the disk. Therefore, the database may contain updates of uncommitted transactions.
- Because of this, **UNDO may be required** if a transaction fails. The transaction log stores old values so changes can be rolled back.
- Immediate update follows the **Write-Ahead Logging (WAL)** rule, which ensures log records are written to disk before data pages.
- Recovery may involve:
  - **UNDO**, to remove effects of failed transactions
  - **REDO**, to reapply committed changes not saved to disk
- This method improves performance because it does not wait until commit to write data, but requires careful recovery handling.

5. Write short notes on:

a) System Failure

- A **System Failure** occurs due to hardware malfunction, software crash, or power failure. It causes loss of main memory contents, including buffers and caches.
- Disk data usually remains intact, making recovery possible using transaction logs. Some transactions may be partially executed at the time of crash.
- During recovery, the DBMS performs **UNDO for uncommitted transactions** and **REDO for committed transactions** to restore consistency.
- System failures are also called **soft crashes** and are common in database environments.

b) Media Failure

- **Media Failure** occurs when the physical storage device storing the database is damaged. This includes disk crashes, read/write errors, or corrupted disk blocks.
- Media failures are more serious than system failures because actual database files may be lost permanently.
- Recovery from media failure requires restoring the **database backup** from archival storage such as tape or external disk.
- After restoring the backup, **REDO operations** are applied using the transaction log to bring the database to the latest consistent state.

6. Explain Shadow Paging and its working mechanism

Shadow Paging is a database recovery technique that avoids using a transaction log, especially in a single-user environment.

- In shadow paging, the database is divided into fixed-size disk pages. A **page directory** is maintained, which contains pointers to all database pages on disk.
- When a transaction starts, the current page directory is copied to create a **shadow directory**. The shadow directory represents the stable state of the database before the transaction begins.
- During updates, the original pages are **not overwritten**. Instead, new copies of updated pages are written to free disk blocks, and the current directory is updated to point to them.
- If a system crash occurs before commit, the system discards the current directory and restores the shadow directory. This automatically brings the database back to its old consistent state.
- Since old pages are never overwritten, **no UNDO and no REDO** are required.

7. What is Checkpointing? Explain its advantages

Checkpointing is a recovery technique used to reduce the time required for database recovery after a system crash.

- A **checkpoint** is a special entry written into the transaction log at a moment when the database and log are in a consistent state. At this point, all modified memory buffers are force-written to disk.
- The checkpoint record also stores a list of **active transactions**, making it easier to identify which transactions need UNDO or REDO during recovery.
- During recovery, the DBMS does not need to scan the entire log from the beginning. It only processes log records **after the last checkpoint**.

Advantages of Checkpointing:

- It **reduces recovery time** significantly.
- It limits the amount of log data that must be scanned.
- It improves overall system efficiency.
- It helps the DBMS quickly restore database consistency after failure.

8. Describe the concept of Write-Ahead Logging (WAL)

Write-Ahead Logging (WAL) is a fundamental recovery rule used in DBMS to ensure safe recovery after failures.

- According to WAL, **log records must be written to disk before the actual data pages are written to disk**. This ensures that recovery information is never lost.
- Before a transaction commits, all its log entries, including REDO and UNDO information, must be **force-written to the log file**.
- If a crash occurs, the DBMS uses the log to:
    - **UNDO** changes of uncommitted transactions
    - **REDO** changes of committed transactions
- WAL guarantees **atomicity and durability** of transactions. Even if data pages are partially written, the log always contains enough information to recover the database correctly.

9. Explain the need for database backup

A **database backup** is a stored copy of the database used to recover data after serious failures.

- Backup is required mainly during **media failures** or **catastrophic failures**, where disk data is damaged or completely lost.
- In such cases, recovery using logs alone is not sufficient because the actual database files no longer exist.
- The DBMS restores the database from the backup stored on archival media such as tape, external disk, or cloud storage.
- After restoring the backup, the system applies **REDO operations** from the transaction log to bring the database to the most recent consistent state.
- Backup ensures **data safety**, **business continuity**, and protection against hardware failure, disasters, or accidental data loss.

10. Differentiate between Deferred and Immediate Update with suitable examples

Deferred Update and **Immediate Update** are two different recovery techniques based on when database updates are written to disk.

Deferred Update

- Database updates are **not written to disk until the transaction commits**.
- All changes are stored in memory buffers and the log.
- If a transaction fails before commit, **no UNDO is required**.
- REDO is required if a crash occurs after commit.
- **Example:**
  A transaction updates salary data but crashes before commit. Since disk data was unchanged, updates are simply ignored.

Immediate Update

- Database updates can be written to disk **before commit**.
- UNDO is required if the transaction fails.
- Both **UNDO and REDO** may be needed during recovery.
- **Example:**
  A bank transfer updates balance on disk, but crashes before commit. The update must be undone using the log.

11. Explain the role of checkpoint in reducing recovery time

A **checkpoint** plays a very important role in reducing the time needed for database recovery after a system crash.

- A checkpoint is a special record written into the **transaction log** when the DBMS ensures that all modified data pages in memory are written to disk. This means the database and log are synchronized at that moment.
- At the time of checkpointing, the DBMS also records a **list of active transactions**. This helps identify which transactions were incomplete during a crash.
- During recovery, the system does **not need to scan the entire log from the beginning**. It only needs to process log records written **after the last checkpoint**.
- This significantly reduces recovery time, especially for large databases with long transaction histories.
- Checkpoints make recovery faster, simpler, and more efficient by limiting the amount of REDO and UNDO operations.

12. Describe the process of logging in transaction recovery

Logging is the process of recording all important actions of transactions in a **transaction log**, which is stored on stable storage.

- Whenever a transaction starts, a **start record** is written in the log. This marks the beginning of the transaction.
- When a transaction updates a data item, the log records the operation. Depending on the recovery method, the log may store the **old value**, the **new value**, or both.
- When a transaction successfully completes, a **commit record** is written to the log. If the transaction fails, an **abort record** is written instead.
- The log also stores **checkpoint records**, which help speed up recovery.
- During recovery, the DBMS reads the log to decide which transactions need **UNDO** and which need **REDO**, ensuring database consistency.

13. Explain how UNDO works using the transaction log

The **UNDO operation** is used to reverse the effects of transactions that did not commit successfully.

- When a transaction updates data, the **old value of the data item** is stored in the transaction log before the update occurs. This information is essential for UNDO.
- If a transaction fails or the system crashes before commit, the DBMS identifies the uncommitted transactions using the log.
- During recovery, the system scans the log **backward** and restores the old values for all write operations performed by uncommitted transactions.
- This rollback process ensures that **partial updates are removed** from the database.
- UNDO guarantees **atomicity**, meaning a transaction either completes fully or has no effect on the database.

14. Explain the concepts of stable storage and its role in recovery

Stable storage refers to storage media that is **highly reliable** and unlikely to lose data even during failures.

- Examples of stable storage include **disk storage with backup copies**, mirrored disks, or replicated storage systems.
- The transaction log and database backups are stored on stable storage to ensure recovery information is not lost.
- During a system crash, main memory contents are lost, but stable storage remains intact. This allows the DBMS to access log records for recovery.
- Stable storage ensures that **log records survive crashes**, making UNDO and REDO operations possible.
- Without stable storage, recovery would not be reliable, and committed transactions could be lost permanently.

15. Discuss advantages and disadvantages of Shadow Paging

Shadow Paging is a recovery technique that maintains two page directories: a shadow directory and a current directory.

Advantages

- No need for UNDO or REDO because original pages are never overwritten.
- Recovery is fast and simple because the system only switches back to the shadow directory.
- Provides strong consistency, as the database always remains in a valid state.

Disadvantages

- Updated pages are written to new disk locations, which causes **loss of physical locality** and reduces performance.
- Copying and storing page directories is expensive and time-consuming.
- Requires **garbage collection** to free old pages after commit.
- Difficult to implement efficiently in **multi-user environments**, where logging and checkpoints are still required.

[10-marks]

1. Explain the various types of failures (transaction failure, system crash, media failure, application failure) in detail with examples

In a database system, failures may occur due to many reasons and can affect the correctness of data. DBMS classifies failures into different types so that proper recovery techniques can be applied.

1. Transaction Failure

- A transaction failure occurs when a transaction cannot complete its execution successfully.
- This may happen due to **logical errors** such as division by zero, invalid input, or incorrect program logic.
- A transaction may also fail if the **user cancels it manually** during execution.
- In some cases, concurrency control mechanisms abort transactions to resolve **deadlocks or serializability violations**.
- **Example:** A bank withdrawal transaction fails because the account balance is insufficient. The transaction is aborted and all its changes must be undone.

2. System Crash

- A system crash occurs due to **hardware failure, software error, or power failure**.
- In this type of failure, the **main memory contents are lost**, but the database stored on disk usually remains safe.
- Some transactions may have committed, while others may be partially executed at the time of crash.
- **Example:** A sudden power cut shuts down the database server while transactions are executing.

3. Media Failure

- Media failure occurs when the **physical storage device** such as a hard disk is damaged.
- This may happen due to disk head crash, read/write errors, or corrupted disk blocks.
- Media failure can result in **permanent loss of database data** stored on disk.
- **Example:** A hard disk crash destroys database files, making recovery from logs alone impossible.

4. Application Failure

- Application failure happens due to **programming errors or incorrect application logic**.
- Even though the DBMS works correctly, the application may issue wrong operations that cause transaction failure.
- **Example:** A payroll application calculates salary incorrectly due to a logic bug and aborts the transaction.

2. Describe REDO and UNDO operations in detail. Explain how DBMS uses logs to recover from failures

REDO and UNDO are the two most important recovery operations used by a DBMS to restore database consistency after failures.

UNDO Operation

- UNDO is used to **reverse the effects of uncommitted or failed transactions**.
- When a transaction modifies a data item, the **old value** of the item is first recorded in the transaction log.
- If a transaction fails before commit or the system crashes, the DBMS identifies such transactions using the log.
- During recovery, the system scans the log backward and restores old values to roll back the transaction.
- UNDO ensures **atomicity**, meaning a transaction either completes fully or has no effect on the database.

REDO Operation

- REDO is used to **reapply the effects of committed transactions** whose updates were not written to disk before a crash.
- Even after a transaction commits, its updated pages may still be in memory and not on disk.
- If a crash occurs, REDO uses the **new values stored in the log** to reapply changes.
- REDO ensures **durability**, meaning committed transactions are not lost.

Role of Logs in Recovery

- The **transaction log** is a sequential file stored on stable storage.
- It records transaction start, write operations, commit, abort, and checkpoints.
- During recovery, the DBMS analyzes the log to determine:
    - Which transactions need UNDO
    - Which transactions need REDO
- By using logs with UNDO and REDO operations, the DBMS restores the database to a **consistent and correct state** after failures.

3. Explain the Immediate Update and Deferred Update recovery techniques with clear examples and diagrams

In DBMS, recovery techniques decide **when database updates are written to disk** and how recovery is handled if a failure occurs. The two main techniques are **Deferred Update** and **Immediate Update**.

Deferred Update Recovery Technique

- In **Deferred Update**, the database is **not updated on disk until the transaction commits**.
- During execution, all updates are stored only in **main memory buffers** and recorded in the **transaction log**.
- If the transaction fails before commit, the database on disk remains unchanged. Hence, **no UNDO is required**.
- After commit, if the system crashes before data pages are written to disk, **REDO is required** to reapply changes from the log.
- This technique follows the **NO-UNDO / REDO** policy.

Example:
Transaction T updates account balance from 5000 to 4000.

- Before commit → changes are only in memory.
- If crash occurs → memory is cleared, disk data remains correct.
- If commit occurs and crash happens → REDO applies update from log.

Immediate Update Recovery Technique

- In **Immediate Update**, database updates may be **written to disk before the transaction commits**.
- Because uncommitted changes may reach disk, **UNDO is required** if a transaction fails.
- The system log stores **old values** before updates, enabling rollback.
- Depending on timing of crash, both **UNDO and REDO** may be required.
- This technique follows **UNDO / REDO** or **UNDO / NO-REDO** policies.

Example:
A bank transfer updates balance on disk immediately.

- If crash occurs before commit → UNDO restores old balance.
- If commit occurred but data wasn't fully saved → REDO reapplies updates.

[I didn't get any diagram , sry :)]

4. Discuss Shadow Paging in detail. Explain its advantages, disadvantages, and step-by-step working

Shadow Paging is a recovery technique that avoids using a traditional transaction log, mainly suitable for **single-user systems**.

Concept of Shadow Paging

- The database is divided into **fixed-size pages**.
- A **page directory** contains pointers to all database pages.
- Two directories are used:
    - **Shadow Directory** → stable version of database
    - **Current Directory** → modified version during transaction

Step-by-Step Working of Shadow Paging

1. When a transaction starts, the **current directory is copied** to create a shadow directory.
2. The shadow directory is saved on disk and remains unchanged.
3. When a page is updated:
    - The original page is **not overwritten**.
    - A **new copy** of the page is written to a free disk block.
    - The current directory points to the new page.
4. If a crash occurs before commit:
    - The current directory is discarded.
    - The shadow directory is restored.
    - Database returns to the exact previous state.
5. On commit:
    - The shadow directory is discarded.
    - The current directory becomes the new shadow directory.
    - Old pages are freed using garbage collection.

Advantages of Shadow Paging

- No need for **UNDO or REDO** operations.
- Recovery is **fast and simple**.
- Database always remains in a consistent state.
- No log is required in single-user systems.

Disadvantages of Shadow Paging

- Loss of **physical locality**, reducing performance.
- Copying directories is expensive.
- Requires **garbage collection**.
- Difficult to implement in **multi-user environments**.
- Atomic switching of directories is complex.

5. What is Checkpointing? Explain how it improves recovery time. Discuss its working with the help of an example

What is Checkpointing

- **Checkpointing** is a recovery technique in DBMS where the system periodically saves a **consistent state of the database** and records this point in the transaction log.
- At a checkpoint, all modified data pages present in **main memory buffers are written to disk**.
- A special **checkpoint record** is written into the log, along with the list of **active transactions** at that moment.
- This creates a known safe point from where recovery can start in case of a system crash.

How Checkpointing Improves Recovery Time

- Without checkpointing, the DBMS would need to scan the **entire transaction log** from the beginning during recovery.
- With checkpointing, the system only scans the log **after the last checkpoint**, which greatly reduces recovery time.
- It limits the number of **REDO and UNDO operations** required.
- This makes recovery faster, efficient, and suitable for large databases.

Working of Checkpointing

1. The DBMS temporarily pauses transaction execution.
2. All modified memory buffers are **force-written to disk**.
3. A **checkpoint record** is written to the transaction log.
4. The log is force-written to disk.
5. Transaction execution resumes.

Example

- Suppose transactions T1, T2, and T3 are running.
- A checkpoint is taken after T1 commits, but T2 and T3 are still active.
- If a crash occurs later, recovery starts from this checkpoint.
- Only T2 and T3 are examined for UNDO or REDO.
- Transactions before the checkpoint (like T1) are ignored, saving recovery time.

6. Explain the full recovery process of a DBMS after a crash using logging, checkpoints, REDO, and UNDO principles

The DBMS recovery process ensures that the database returns to a **consistent and correct state** after a crash.

Step 1: Use of Transaction Log

- The transaction log records all important operations such as transaction start, updates, commit, abort, and checkpoints.
- For write operations, the log stores old values (for UNDO) and/or new values (for REDO).
- The log is stored on stable storage, so it survives system crashes.

Step 2: Locate the Last Checkpoint

- After a crash, the DBMS first finds the most recent checkpoint in the log.
- Recovery starts from this checkpoint instead of the beginning of the log.
- This reduces the number of log records that need to be processed.

Step 3: Identify Transactions

- Transactions are divided into:
    - Committed transactions → need REDO
    - Uncommitted or active transactions → need UNDO
- The list of active transactions at the checkpoint helps in this identificatio

Step 4: Perform UNDO

- For transactions that did not commit, the DBMS performs UNDO.
- Using old values from the log, the system reverses partial updates.
- This ensures atomicity, meaning incomplete transactions leave no effect.

Step 5: Perform REDO

- For committed transactions whose updates were not written to disk, REDO is applied.
- New values from the log are reapplied to the database.
- This ensures durability, meaning committed data is not lost.

Final Result

- After completing UNDO and REDO operations, the database reaches a consistent state.
- All committed transactions are reflected correctly.
- All incomplete transactions are completely removed.

7. Discuss the difference between Deferred Update, Immediate Update, and Shadow Paging recovery techniques

DBMS uses different recovery techniques to handle failures depending on **when database updates are written to disk** and **how recovery is performed**. The three main techniques are **Deferred Update**, **Immediate Update**, and **Shadow Paging**.

Deferred Update

- In **Deferred Update**, changes made by a transaction are **not written to the database on disk until the transaction commits**.
- All updates are stored temporarily in **main memory buffers** and recorded in the **transaction log**.
- If a transaction fails before commit, the database remains unchanged, so **UNDO is not required**.
- If a crash happens after commit but before disk write, **REDO is required**.
- This technique follows **NO-UNDO / REDO** policy.
- It is simple and safe but may cause memory overflow if transactions are long.

Immediate Update

- In **Immediate Update**, changes can be **written to disk before the transaction commits**.
- Because uncommitted data may reach disk, **UNDO is required** if the transaction fails.
- The log stores **old values and new values**, enabling rollback and reapplication.
- Recovery may need **both UNDO and REDO**.
- It follows **UNDO / REDO** or **UNDO / NO-REDO** policies.
- It improves performance but increases recovery complexity.

Shadow Paging

- Shadow Paging avoids logs and maintains **two page directories**: shadow and current.
- Updates are written to **new pages**, never overwriting old ones.
- If a crash occurs, the system restores the shadow directory.
- It requires **NO-UNDO and NO-REDO**.
- It is fast in recovery but inefficient for large, multi-user systems.

8. Explain the concept of Write-Ahead Logging (WAL). Describe its rules and importance in recovery

Concept of Write-Ahead Logging (WAL)

Write-Ahead Logging (WAL) is a fundamental recovery rule in DBMS that ensures **recovery information is always safely stored before database changes are written to disk**.

- WAL states that **log records must be written to stable storage before the actual data pages are written to disk**.
- This guarantees that if a crash occurs, the DBMS can always use the log to recover data.

Rules of Write-Ahead Logging

1. **Before a data item is updated on disk**, its corresponding log record must be written to disk.
2. **Before a transaction commits**, all its log records (UNDO and REDO information) must be force-written to disk.
3. Only after the log is safely stored can the transaction be considered committed.

Importance of WAL in Recovery

- WAL makes **UNDO possible**, because old values are safely stored in the log.
- WAL makes **REDO possible**, because new values are also available.
- It ensures **atomicity**, meaning incomplete transactions can be rolled back.
- It ensures **durability**, meaning committed transactions are never lost.
- WAL is essential for **Immediate Update recovery techniques**.
- Without WAL, recovery after crashes would be unreliable and unsafe.

Conclusion

- WAL acts as the **foundation of log-based recovery**.
- It guarantees that the DBMS can always restore the database to a **consistent state** after failures.
- Almost all modern DBMS systems strictly follow WAL principles.

9. Explain the purpose of database backup. Describe different types of backups and how they are used in crash recovery

Purpose of Database Backup

- A **database backup** is a stored copy of the database kept on **archival or stable storage** such as tape, external disk, or cloud.
- The main purpose of backup is to **protect data from permanent loss** due to serious failures like media failure or catastrophic failure.
- In some failures, such as disk crashes, the actual database files are destroyed and **transaction logs alone are not sufficient**.
- Backup allows the DBMS to **restore the database to a previous consistent state** and then reconstruct recent changes using logs.
- It ensures **data safety, business continuity, and disaster recovery**.

Types of Backups

1. Full Backup

- A full backup stores a **complete copy of the entire database**.
- It is usually taken at regular intervals.
- During recovery, the database is first restored from the full backup.

2. Incremental Backup

- Incremental backup stores **only the changes made after the last backup**.
- It requires less storage and is faster to create.
- During recovery, the DBMS restores the full backup first, then applies incremental backups in order.

3. Differential Backup

- Differential backup stores all changes made **since the last full backup**.
- It is larger than incremental but faster to restore.

Use of Backup in Crash Recovery

- In **media failure**, the DBMS restores the database from the latest backup.
- After restoration, **REDO operations** from the transaction log are applied.
- This brings the database to the most recent consistent state before failure.
- Backup + log together ensure complete recovery.

10. Describe the complete recovery mechanism using Transaction Logs, Checkpoints, REDO, and UNDO (with diagram and example schedule)

The DBMS recovery mechanism ensures that the database returns to a **consistent and correct state** after a system crash.

1. Transaction Logs

- A **transaction log** records all important operations such as:
    - Start transaction
    - Write operations (old and/or new values)
    - Commit or abort
    - Checkpoints
- The log is stored on **stable storage**, so it survives crashes.
- It is the main source of information for recovery.

2. Checkpoints

- A **checkpoint** marks a position in the log where:
    - All modified memory buffers are written to disk
    - A list of active transactions is recorded
- Recovery starts from the **last checkpoint**, not from the beginning of the log.
- This greatly reduces recovery time.

3. UNDO Operation

- UNDO is applied to **uncommitted transactions**.
- The DBMS scans the log backward and restores **old values**.
- This removes partial effects of failed transactions.
- UNDO ensures **atomicity**.

4. REDO Operation

- REDO is applied to **committed transactions** whose updates were not written to disk.
- The DBMS reapplies **new values** from the log.
- REDO ensures **durability**.

Example Schedule

- T1 commits before checkpoint
- T2 commits after checkpoint
- T3 is active and crashes

Recovery actions:

- T1 → ignored
- T2 → REDO
- T3 → UNDO