Chapter 7

Database Recovery Technique

- It is the process of restoring the database to the most recent consistent state when a failure occurred. Recovery manager is responsible for database recovery.
- **Failure classification:** There are various types of failure that may occur in a system, each of which needs to be dealt with in a different manner. Following failure may occur in DBMS

1. Transaction failure
    a. Logical error
    b. System error
2. System crash
3. Disk failure

1. **Transaction failure**: A transaction needs to abort once it fails to execute or once it reaches to any further extent from wherever it can't go to any extent further. This is known as transaction failure. Reasons for transaction failure are :
    a. **Logical error**: The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow or limited resources.
    b. **System errors**: The system has entered an undesirable state (e.g. deadlock), as a result of which a transaction cannot continue with its normal execution. The transaction however can be re-executed at a later time.
2. **System crash**: there is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage and brings transaction processing to a halt.
3. **Disk failure**: A disk block loses its content as a result of either a head crash or failure during a data transfer operation.

   Recovery Technique: It is the method by which system restore its most recent consistent state just before the time of failure. Following methods are used for recovery:
    a. Log based recovery
        i. Write-Ahead logging
        ii. Checkpointing
        iii. Caching (buffering) of disk blocks

1. **Log based recovery**: A log is maintained, in which all the modifications of the database are kept. A log consists of log records. Each log records contains following fields

- Transaction identifier: A unique number given to each transaction.
- Data-item identifier: a unique number given to data item written.
- Date and time of updates.
- Old values: value of data item before write.
- New values: value of data item after write.

Logs must be written on the non-volatile storage. In log-based recovery, following two operations for recovery are required:

a. **Redo**: it means, the work of the transactions that completed successfully before crash is to be performed again, i.e. updates new values.
b. **Undo**: It means, all the work done by transactions that did not complete due to crash is to be undone, i.e. restores old values
    i. **Write-Ahead logging**: Log is written before any update is made to the database. Transaction is not allowed to modify the physical database until UNDO portion of log is written to stable storage.
    ii. **Checkpoints**: it is a mechanism where all the previous logs are removed from the system and stored permanently in storage disk. Advantages of checkpoints are:
    No need to redo successfully completed transactions before most recent checkpoints, less searching required, old records can be deleted.

Transaction Rollback and cascading Rollback: If a transaction fails for whatever reason after updating the database, but before the transaction commits, it may be necessary to roll back the transaction. If any data item values have been changed by the transaction and written to the database, they must be restored to their previous values. The undo-type log entries are used to restore the old values of the data items that must be rolled back.

If a transaction T is rolled back, any transaction S that has read the value of some data item written by T must be rolled back. Once S is rolled back, any transaction R that has read the values of some data item written by S must also be rolled back and so on. This phenomenon is called cascading roll-back.

Recovery based on Deferred Update (No-UNDO/REDO):

In deferred update technique, it defers (stops) all the write operations of any transaction T_i until it practically commits. All the activities are recorded in log. Log records are used to modify actual database. Suppose a transaction T_i wants to write on data item A_j, then a log records [T_i A_j, V_1, V_2] is saved in log and it is used to modify database after actual modification T_i enters in committed state. In this technique, the old value field is not needed.

Example: To illustrate, consider transaction T_{31} and T_{32}. Suppose you want to transfer Rs 200 from account A to B in transaction T31 and deposit Rs 200 to account C in T_{32}.

T_{31}	T_{32}
read(A);	read(C)
A:=A-200;	C:=C+200;
write(A);	write(c)
read(B);	
B:=B+200;	
Write(B);	

Suppose, the initial values of A, B and C accounts are Rs 500, Rs 1000 and Rs 600 respectively various log records of T_{31} and T_{32} are shown in figures:

[T_{31} start]

[T_{31}, A]

[T_{31}, A, 300]

[T_{31}, B]

[T_{31}, B, 1200]

[T_{31} commit]

[T_{32} start]

[T_{32}, C]

[T_{32}, C, 800]

[T_{32} commit]

Fig: log records for transaction T_{31} and T_{32}

For a redo operation, log must contain [T_i start] and [T_i commit] log records.

[T_{31} start]	[T_{31} start]
[T_{31} , A]	[T_{31}, A]
[T_{31}, A, 300]	[T_{31}, A, 300]
(a)	[T_{31}, B]
	[T_{31}, B, 1200]
	[T_{31} commit]
	[T_{32} start]
	[T_{32} start]
	[T_{32}, C]
	[T_{32}, C, 800]
	(b)

Crash will happen at any time of execution of transactions. Suppose crash happened

a. After write (A) of T_{31}: at that time log records in long are shown in figure (a), there is no need to redo operation because no commit record appears in the log. Log records of T_{31} can be deleted.
b. After write(C) of T_{32}: at that time log records in log are shown in figure (b). in this situation, you have to redo T_{32} because both [T_{31} start] and T_{31} commit] appears in log. After redo operation, value of A and B are 300 and 1200 respectively, values remain same because redo is idempotent.

Recovery based on immediate update (undo/redo)

In immediate update technique, database is modified by any transaction T_i during its active state. It means, real database is modified just after the write operation but after log record is written to stable storage. This is because log records are used during recovery. Use both undo and redo operations in this method. Old value field is also needed (for undo operation).

To illustrate consider again the transaction T_{31} and T_{32} of following figure. Suppose, the initial values of A, B and C accounts are Rs 500, Rs 1000, and Rs 600 respectively, corresponding log records after successful completion of T_{31} and T_{32} are shown in below

[T_{31} start]

[$T_{31,}$ A]

[$T_{31, A,}$ 500, 300]

[$T_{31,}$ B]

[$T_{31,}$ B, 1000, 1200]

[T_{31} commit]

[T_{32} start]

[$T_{32,}$ C]

[$T_{32,}$ C, 600, 800]

[T_{32} commit]

Log records for transactions T_{31} and T_{32}

For a transaction T_i to be redone, log must contain both [$T_{i,}$ start] and [T_i commit] records. For a transaction T_i to be undone, log must contain only [T_i start] record.

[T_{31} start]	[T_{31} start]
[T_{31} , A]	[$T_{31,}$ A]
[$T_{31,}$ A, 500, 300]	[$T_{31,}$ A, 500, 300]
(a)	[$T_{31,}$ B]

$[T_{31,}$ B, 100, 1200]

$[T_{31}$ commit]

$[T_{32}$ start]

$[T_{31,}$ C]

$[T_{32,}$ C, 600, 800]

(b)

Crash will happen at any time of execution of transaction, suppose crash happened:

a. After write(A) of T_{31}: at that time log records in long are shown in figure in figure (a), here only $[T_{31}$ start] exists so undo transaction $T_{31.}$ Accountant A restores its old value 500.
b. After write(c) of T_{32}: at that time log records in log are shown in figure (b). during backup record $[T_{32}$ start] appears but there is no $[T_{32}$ commit], so undo transaction $T_{32.}$ As a result, account C restores its old values 600. When you found both $[T_{31,}$ start] and T_{32} commit] records in log, redo transaction T_{31} and account A and B both keep their new values.
c. During recovery: if system is crashed at the time of recovery, simply starts the recovery again.

2. **Shadow paging**: shadow paging is an alternative technique for recovery to overcome the disadvantages of log based recovery techniques. The main idea behind the shadow paging is to keep two pages tables in database, one is used for current operations and other is used in case of recovery.
   Database is partitioned into fixed length blocks called pages. For memory management, adopt any paging technique.
   Page table: to keep record of each page in database, maintain a page table. Total number of entries in page table is equal to the number of pages in database. Each entry in page table contains a pointer to the physical location of pages. The two page table in shadow paging are:

- Shadow page table: This table cannot be changed during any transaction.
- Current page table: this table may be changed during transaction.
  Both page tables are identical at the start of transaction. Suppose a transaction T_i performs a write operation on data item V that resides in page j. the write operation procedure is as follows:

1. If j^{th} page is in main memory then ok otherwise first transfer it from secondary memory to main memory by instruction input (V).
2. Suppose page is used first time then,
   a. System first finds a free page on disk and deletes its entry from free page list
   b. Then modify the current page table so that it points to the page found in step (a).
3. Modify the contents of page or assign new value to V.

After performing all the above operations, the following steps are needed to commit T_i.

1. Modified pages are transferred from main memory to disk.
2. Save current page table on disk.
3. Change current page table into shadow page table.

Shadow page table must be stored in non-volatile or stable storage because it is used at the time of recovery.

Advantages:

1. The overhead of maintaining the transaction log file is eliminated.
2. Since there is no need for undo and redo operations, recovery is significant faster.

Disadvantages:

1. Data gets fragmented or scattered.
2. Need for periodic garbage collection to reclaim in accessible blocks.
3. Hard to extend algorithm to allow transaction to run concurrently.