CHANDIGARH ENGINEERING COLLEGE

B.TECH.(CSE 5th Sem.)

Notes Subject: Database Management System

Subject Code: ((BTCS 501-18))

Unit 4

Concurrency Control in DBMS

Concurrency control concept comes under the Transaction in database management system (DBMS). It is a procedure in DBMS which helps us for the management of two simultaneous processes to execute without conflicts between each other, these conflicts occur in multi user systems.

Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data.

For example, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.

Advantages

The advantages of concurrency control are as follows −

- Waiting time will be decreased.
- Response time will decrease.
- Resource utilization will increase.
- System performance & Efficiency is increased.

Control concurrency

The simultaneous execution of transactions over shared databases can create several data integrity and consistency problems.

For example, if too many people are logging in the ATM machines, serial updates and synchronization in the bank servers should happen whenever the transaction is done, if not it gives wrong information and wrong data in the database.

Main problems in using Concurrency

The problems which arise while using concurrency are as follows −

- Updates will be lost − One transaction does some changes and another transaction deletes that change. One transaction nullifies the updates of another transaction.

- Uncommitted Dependency or dirty read problem − On variable has updated in one transaction, at the same time another transaction has started and deleted the value of the variable there the variable is not getting updated or committed that has been done on the first transaction this gives us false values or the previous values of the variables this is a major problem.
- Inconsistent retrievals − One transaction is updating multiple different variables, another transaction is in a process to update those variables, and the problem occurs is inconsistency of the same variable in different instances.

Concurrency control techniques

The concurrency control techniques are as follows −

Locking

Lock guaranties exclusive use of data items to a current transaction. It first accesses the data items by acquiring a lock, after completion of the transaction it releases the lock.

Types of Locks

The types of locks are as follows −

1. Shared Lock [Transaction can read only the data item values]

It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.

It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive Lock [Used for both read and write data item values]

In the exclusive lock, the data item can be both reads as well as written by the transaction.

This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

There are four types of lock protocols available:

1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

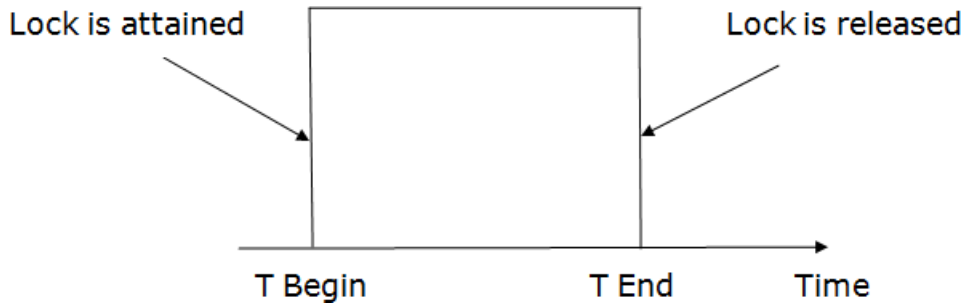
2. Pre-claiming Lock Protocol

Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.

Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.

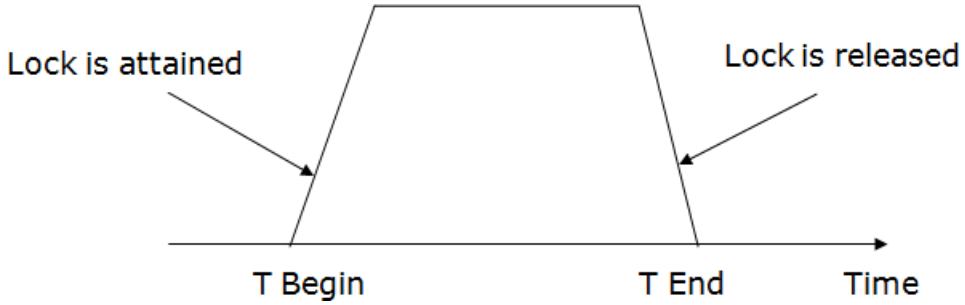
If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.

If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.



3. Two-phase locking (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.



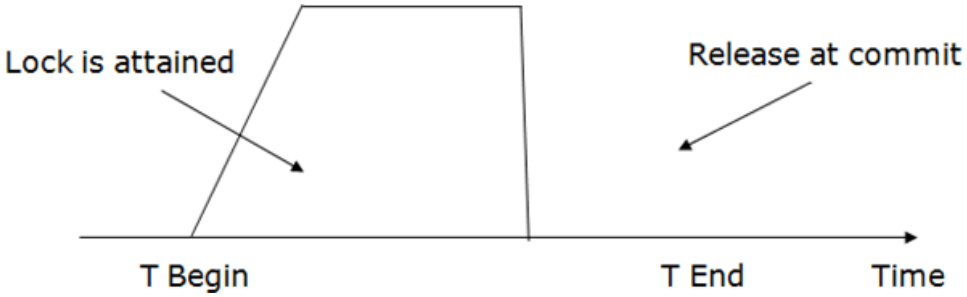
There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

In the below example, if lock conversion is allowed then the following phase can happen:

1. Upgrading of lock (from S(a) to X (a)) is allowed in growing phase.
2. Downgrading of lock (from X(a) to S(a)) must be done in shrinking phase.



It does not have cascading abort as 2PL does.

Time Stamping

Time stamp is a unique identifier created by DBMS that indicates relative starting time of a transaction. Whatever transaction we are doing it stores the starting time of the transaction and denotes a specific time.

This can be generated using a system clock or logical counter. This can be started whenever a transaction is started. Here, the logical counter is incremented after a new timestamp has been assigned.

- The Timestamp Ordering Protocol is used to order the transactions based on their Timestamps. The order of transaction is nothing but the ascending order of the transaction creation.
- The priority of the older transaction is higher that's why it executes first. To determine the timestamp of the transaction, this protocol uses system time or logical counter.
- The lock-based protocol is used to manage the order between conflicting pairs among transactions at the execution time. But Timestamp based protocols start working as soon as a transaction is created.
- Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 007 times and transaction T2 has entered the system at 009 times. T1 has the higher priority, so it executes first as it is entered the system first.
- The timestamp ordering protocol also maintains the timestamp of last 'read' and 'write' operation on a data.

Basic Timestamp ordering protocol works as follows:

1. Check the following condition whenever a transaction Ti issues a Read

(X) operation:

- If W_TS(X) >TS(Ti) then the operation is rejected.
- If W_TS(X) <= TS(Ti) then the operation is executed.
- Timestamps of all the data items are updated.

2. Check the following condition whenever a transaction Ti issues a Write(X) operation:

If TS(Ti) < R_TS(X) then the operation is rejected.

If TS(Ti) < W_TS(X) then the operation is rejected and Ti is rolled back otherwise the operation is executed.

Where

TS(TI) denotes the timestamp of the transaction Ti. R_TS(X) denotes the Read time-stamp of data-item X. W_TS(X) denotes the Write time-stamp of data-item X.

Advantages and Disadvantages of TO protocol:

- TO protocol ensures serializability since the precedence graph is as follows:

Image: Precedence Graph for TS ordering

ACID Properties in DBMS

The commands that is executed and followed to access and modify the information in databases are referred to as transactions. Transactions access data using read and write operations. In order to maintain consistency in a database, before and after the transaction, certain properties are followed. These are called ACID properties: Atomicity By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations. —Abort: If a transaction aborts, changes made to database are not visible. —Commit: If a transaction commits, changes made are visible.

Atomicity: is also known as the 'All or nothing rule'. Consider the following transaction T consisting of T1 and T2: Transfer of 100 from account X to account Y. If the transaction fails after completion of T1 but before completion of T2.(say, after write(X) but before write(Y)), then amount has been deducted from X but not added to Y. This results in an inconsistent database state. Therefore, the transaction must be executed in entirety in order to ensure correctness of database state. s

Consistency This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above, The total amount before and after the transaction must be maintained. Total before T occurs = 500 + 200 = 700. Total after T occurs = 400 + 300 = 700. Therefore, database is consistent. Inconsistency occurs in case T1 completes but T2 fails. As a result T is incomplete.

Isolation: This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state. Transactions occur independently without interference. Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed. This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in some order. Let X= 500, Y = 500. Consider two transactions T and T". Suppose T has been executed till Read (Y) and then T'' starts. As a result , interleaving of operations takes place due to which T'' reads correct value of X but incorrect value of Y and sum computed by T'': (X+Y = 50, 000+500=50, 500) is thus not consistent with the sum at end of transaction: T: (X+Y = 50, 000 + 450 = 50, 450). This results in database inconsistency, due to a loss of 50 units. Hence, transactions must take place in isolation and changes should be visible only after they have been made to the main memory.

Durability: This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. These updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost. The ACID properties, in totality, provide a mechanism to ensure correctness and consistency of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.

Serializability in DBMS

If a non-serial schedule can be transformed into its corresponding serial schedule, it is said to be serializable. Simply said, a non-serial schedule is referred to as a serializable schedule if it yields the same results as a serial timetable.

Non-serial Schedule

A schedule where the transactions are overlapping or switching places. As they are used to carry out actual database operations, multiple transactions are running at once. It's possible that these transactions are focusing on the same data set. Therefore, it is crucial that non-serial schedules can be serialized in order for our database to be consistent both before and after the transactions are executed.

Example:

Transaction-1	Transaction-2
R(a)	
W(a)	
	R(b)
	W(b)

Transaction-1	Transaction-2
R(b)	
	R(a)
W(b)	
	W(a)

We can observe that Transaction-2 begins its execution before Transaction-1 is finished, and they are both working on the same data, i.e., "a" and "b", interchangeably. Where "R"-Read, "W"-Write

Serializability testing

We can utilize the Serialization Graph or Precedence Graph to examine a schedule's serializability. A schedule's full transactions are organized into a Directed Graph, what a serialization graph is.

Precedence Graph

It can be described as a Graph G(V, E) with vertices V = "V1, V2, V3,..., Vn" and directed edges E = "E1, E2, E3,..., En". One of the two operations—READ or WRITE—performed by a certain transaction is contained in the collection of edges. Where Ti -> Tj, means Transaction-Ti is either performing read or write before the transaction-Tj.

Types of Serializability

There are two ways to check whether any non-serial schedule is serializable.

Types of Serializability - Conflict & View

1. Conflict serializability

Conflict serializability refers to a subset of serializability that focuses on maintaining the consistency of a database while ensuring that identical data items are executed in an order. In a DBMS each transaction has a value and all the transactions, in the database rely on this uniqueness. This uniqueness ensures that no two operations with the conflict value can occur simultaneously.

For example lets consider an order table and a customer table as two instances. Each order is associated with one customer even though a single client may place orders. However there are restrictions for achieving conflict serializability in the database. Here are a few of them.

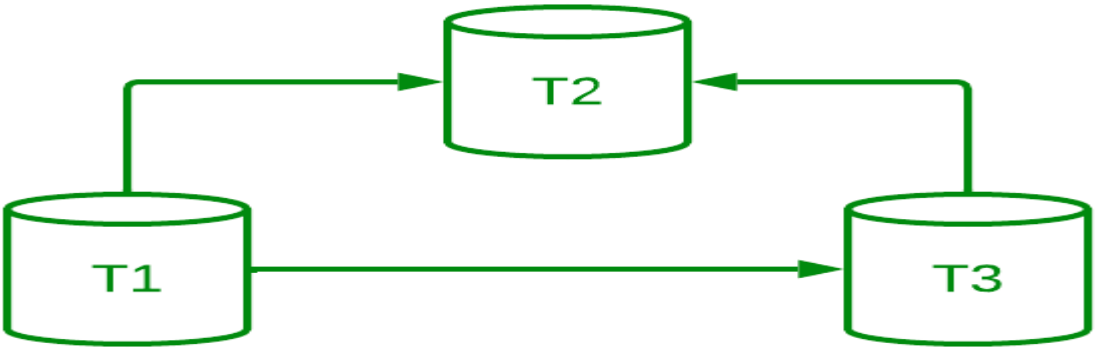
1. Different transactions should be used for the two procedures.
2. The identical data item should be present in both transactions.
3. Between the two operations, there should be at least one write operation.

Example

Three transactions—t1, t2, and t3—are active on a schedule "S" at once. Let's create a graph of precedence.

Transaction - 1 (t1)	Transaction - 2 (t2)	Transaction - 3 (t3)
R(a)		
	R(b)	
		R(b)
	W(b)	
W(a)		
		W(a)
	R(a)	
	W(a)	

It is a conflict serializable schedule as well as a serial schedule because the graph (a DAG) has no loops. We can also determine the order of transactions because it is a serial schedule.



DAG of transactions

As there is no incoming edge on Transaction 1, Transaction 1 will be executed first. T3 will run second because it only depends on T1. Due to its dependence on both T1 and T3, t2 will finally be executed.

Therefore, the serial schedule's equivalent order is: t1 --> t3 --> t2

Note: A schedule is unquestionably consistent if it is conflicting serializable. A non-conflicting serializable schedule, on the other hand, might or might not be serial. We employ the idea of View Serializability to further examine its serial behavior.

2. View Serializability

View serializability is a kind of operation in a serializable in which each transaction should provide some results, and these outcomes are the output of properly sequentially executing the data item. The view serializability, in contrast to conflict serialized, is concerned with avoiding database inconsistency. The view serializability feature of DBMS enables users to see databases in contradictory ways.

To further understand view serializability in DBMS, we need to understand the schedules S1 and S2. The two transactions T1 and T2 should be used to establish these two schedules. Each schedule must follow the three transactions in order to retain the equivalent of the transaction. These three circumstances are listed below.

1. The first prerequisite is that the same kind of transaction appears on every schedule. This requirement means that the same kind of group of transactions cannot appear on both schedules S1 and S2. The schedules are not equal to one another if one schedule commits a transaction but it does not match the transaction of the other schedule.

2. The second requirement is that different read or write operations should not be used in either schedule. On the other hand, we say that two schedules are not similar if schedule S1 has two write operations whereas schedule S2 only has one. The number of the write operation must be the same in both schedules, however there is no issue if the number of the read operation is different.

3. The second to last requirement is that there should not be a conflict between either timetable. execution order for a single data item. Assume, for instance, that schedule S1's transaction is T1, and schedule S2's transaction is T2. The data item A is written by both the transaction T1 and the transaction T2. The schedules are not equal in this instance. However, we referred to the schedule as equivalent to one another if it had the same number of all write operations in the data item.

Timestamp based Concurrency Control

The main idea for this protocol is to order the transactions based on their Timestamps. A schedule in which the transactions participate is then serializable and the only *equivalent serial schedule permitted* has the transactions in the order of their Timestamp Values. Stating simply, the schedule is equivalent to the particular *Serial Order* corresponding to the *order of the Transaction timestamps*. An algorithm must ensure that, for each item accessed by *Conflicting Operations* in the schedule, the order in which the item is accessed does not violate the ordering. To ensure this, use two Timestamp Values relating to each database item X.

- W_TS(X) is the largest timestamp of any transaction that executed write(X) successfully.
- R_TS(X) is the largest timestamp of any transaction that executed read(X) successfully.

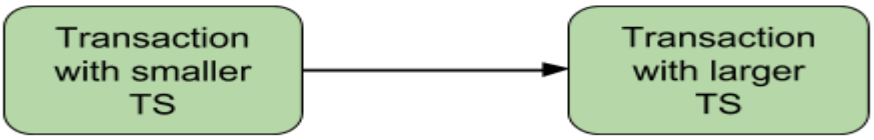
Basic Timestamp Ordering

Every transaction is issued a timestamp based on when it enters the system. Suppose, if an old transaction Ti has timestamp TS(Ti), a new transaction Tj is assigned timestamp TS(Tj) such that TS(Ti) < TS(Tj). The protocol manages concurrent execution such that the timestamps determine the serializability order. The timestamp ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. Whenever some Transaction *T* tries to issue a R_item(X) or a W_item(X), the Basic TO algorithm compares the timestamp of *T* with R_TS(X) & W_TS(X) to ensure that the Timestamp order is not violated. This describes the Basic TO protocol in the following two cases.

- Whenever a Transaction *T* issues a W_item(X) operation, check the following conditions:
    - If *R_TS(X) > TS(T)* and if *W_TS(X) > TS(T)*, then abort and rollback T and reject the operation. else,
    - Execute W_item(X) operation of T and set W_TS(X) to TS(T).
- Whenever a Transaction *T* issues a R_item(X) operation, check the following conditions:
    - If *W_TS(X) > TS(T)*, then abort and reject T and reject the operation, else
    - If W_TS(X) <= TS(T), then execute the R_item(X) operation of T and set R_TS(X) to the larger of TS(T) and current R_TS(X).

Whenever the Basic TO algorithm detects two conflicting operations that occur in an incorrect order, it rejects the latter of the two operations by aborting the Transaction that issued it. Schedules produced by Basic TO are guaranteed to be *conflict serializable*. Already discussed that using Timestamp can ensure that our schedule will be *deadlock free*.
One drawback of the Basic TO protocol is that Cascading Rollback is still possible. Suppose we have a Transaction T1 and T2 has used a value written by T1. If T1 is aborted and resubmitted to the system then, T2 must also be aborted and rolled back. So the problem of Cascading aborts still prevails. Let's gist the Advantages and Disadvantages of Basic TO protocol:

- Timestamp Ordering protocol ensures serializability since the precedence graph will be of the form:



Precedence Graph for TS ordering

- Timestamp protocol ensures freedom from deadlock as no transaction ever waits.
- But the schedule may *not be cascade free*, and may not even be recoverable.

Strict Timestamp Ordering

A variation of Basic TO is called Strict TO ensures that the schedules are both Strict and Conflict Serializable. In this variation, a Transaction T that issues a R_item(X) or W_item(X) such that TS(T) > W_TS(X) has its read or write operation delayed until the Transaction T‘ that wrote the values of X has committed or aborted.

Advantages of Timestamp Ordering Protocol

- High Concurrency: Timestamp-based concurrency control allows for a high degree of concurrency by ensuring that transactions do not interfere with each other.
- Efficient: The technique is efficient and scalable, as it does not require locking and can handle a large number of transactions.
- No Deadlocks: Since there are no locks involved, there is no possibility of deadlocks occurring.
- Improved Performance: By allowing transactions to execute concurrently, the overall performance of the database system can be improved.

Disadvantages of Timestamp Ordering Protocol

- Limited Granularity: The granularity of timestamp-based concurrency control is limited to the precision of the timestamp. This can lead to situations where transactions are unnecessarily blocked, even if they do not conflict with each other.
- Timestamp Ordering: In order to ensure that transactions are executed in the correct order, the timestamps need to be carefully managed. If not managed properly, it can lead to inconsistencies in the database.
- Timestamp Synchronization: Timestamp-based concurrency control requires that all transactions have synchronized clocks. If the clocks are not synchronized, it can lead to incorrect ordering of transactions.
- Timestamp Allocation: Allocating unique timestamps for each transaction can be challenging, especially in distributed systems where transactions may be initiated at different locations.

Database Recovery Techniques in DBMS

Database Systems like any other computer system, are subject to failures but the data stored in them must be available as and when required. When a database fails it must possess the facilities for fast recovery. It must also have atomicity i.e. either transactions are completed successfully and committed (the effect is recorded permanently in the database) or the transaction should have no effect on the database.

Types of Recovery Techniques in DBMS

Database recovery techniques are used in database management systems (DBMS) to restore a database to a consistent state after a failure or error has occurred. The main goal of recovery techniques is to ensure data integrity and consistency and prevent data loss.

There are mainly two types of recovery techniques used in DBMS

- Rollback/Undo Recovery Technique
- Commit/Redo Recovery Technique
- CheckPoint Recovery Technique

Database recovery techniques ensure data integrity in case of system failures. Understanding how these techniques work is crucial for managing databases effectively. The GATE CS Self-Paced Course covers recovery strategies in DBMS, providing practical insights into maintaining data consistency

Rollback/Undo Recovery Technique

The rollback/undo recovery technique is based on the principle of backing out or undoing the effects of a transaction that has not been completed successfully due to a system failure or error. This technique is accomplished by undoing the changes made by the transaction using the log records stored in the transaction log. The transaction log contains a record of all the transactions that have been performed on the database. The system uses the log records to undo the changes made by the failed transaction and restore the database to its previous state.

Commit/Redo Recovery Technique

The commit/redo recovery technique is based on the principle of reapplying the changes made by a transaction that has been completed successfully to the database. This technique is accomplished by using the log records stored in the transaction log to redo the changes made by the transaction that was in progress at the time of the failure or error. The system uses the log records to reapply the changes made by the transaction and restore the database to its most recent consistent state.

Checkpoint Recovery Technique

Checkpoint Recoveryis a technique used to improve data integrity and system stability, especially in databases and distributed systems. It entails preserving the system's state at regular intervals, known as checkpoints, at which all ongoing transactions are either completed or not initiated. This saved state, which includes memory and CPU registers, is kept in stable, non-volatile storage so that it can withstand system crashes. In the event of a breakdown, the system can be restored to the most recent checkpoint, which reduces data loss and downtime. The frequency of checkpoint formation is carefully regulated to decrease system overhead while ensuring that recent data may be restored quickly.

Overall, recovery techniques are essential to ensure data consistency and availability in Database Management System, and each technique has its own advantages and limitations that must be considered in the design of a recovery system.

Database Systems

There are both automatic and non-automatic ways for both, backing up data and recovery from any failure situations. The techniques used to recover lost data due to system crashes, transaction errors, viruses, catastrophic failure, incorrect command execution, etc. are database recovery techniques. So to prevent data loss recovery techniques based on deferred updates and immediate updates or backing up data can be used. Recovery techniques are heavily dependent upon the existence of a special file known as a system log. It contains information about the start and end of each transaction and any updates which occur during the transaction. The log keeps track of all transaction operations that affect the values of database items. This information is needed to recover from transaction failure.

- The log is kept on disk start_transaction(T): This log entry records that transaction T starts the execution.
- read item(T, X): This log entry records that transaction T reads the value of database item X.
- write item(T, X, old_value, new_value): This log entry records that transaction T changes the value of the database item X from old_value to new_value. The old value is sometimes known as a before an image of X, and the new value is known as an afterimage of X.
- Commit (T): This log entry records that transaction T has completed all accesses to the database successfully and its effect can be committed (recorded permanently) to the database.
- Abort (T): This records that transaction T has been aborted.
- Checkpoint: A checkpoint is a mechanism where all the previous logs are removed from the system and stored permanently in a storage disk. Checkpoint declares a point before which the DBMS was in a consistent state, and all the transactions were committed.

A transaction T reaches its commit point when all its operations that access the database have been executed successfully i.e. the transaction has reached the point at which it will not abort (terminate without completing). Once committed, the transaction is permanently recorded in the database. Commitment always involves writing a commit entry to the log and writing the log to disk. At the time of a system crash, the item is searched back in the log for all transactions T that have written a start_transaction(T) entry into the log but have not written a commit(T) entry yet; these transactions may have to be rolled back to undo their effect on the database during the recovery process.

- Undoing: If a transaction crashes, then the recovery manager may undo transactions i.e. reverse the operations of a transaction. This involves examining a transaction for the log entry write_item(T, x, old_value, new_value) and setting the value of item x in the database to old-value. There are two major techniques for recovery from non-catastrophic transaction failures: deferred updates and immediate updates.
- Deferred Update: This technique does not physically update the database on disk until a transaction has reached its commit point. Before reaching commit, all transaction updates are recorded in the local transaction workspace. If a transaction fails before reaching its commit point, it will not have changed the database in any way so UNDO is not needed. It may be

necessary to REDO the effect of the operations that are recorded in the local transaction workspace, because their effect may not yet have been written in the database. Hence, a deferred update is also known as the No-undo/redo algorithm.

- Immediate Update: In the immediate update, the database may be updated by some operations of a transaction before the transaction reaches its commit point. However, these operations are recorded in a log on disk before they are applied to the database, making recovery still possible. If a transaction fails to reach its commit point, the effect of its operation must be undone i.e. the transaction must be rolled back hence we require both undo and redo. This technique is known as undo/redo algorithm.
- Caching/Buffering: In this one or more disk pages that include data items to be updated are cached into main memory buffers and then updated in memory before being written back to disk. A collection of in-memory buffers called the DBMS cache is kept under the control of DBMS for holding these buffers. A directory is used to keep track of which database items are in the buffer. A dirty bit is associated with each buffer, which is 0 if the buffer is not modified else 1 if modified.
- Shadow Paging: It provides atomicity and durability. A directory with n entries is constructed, where the ith entry points to the ith database page on the link. When a transaction began executing the current directory is copied into a shadow directory. When a page is to be modified, a shadow page is allocated in which changes are made and when it is ready to become durable, all pages that refer to the original are updated to refer new replacement page.
- Backward Recovery: The term" Rollback" and" UNDO" can also refer to backward recovery. When a backup of the data is not available and previous modifications need to be undone, this technique can be helpful. With the backward recovery method, unused modifications are removed and the database is returned to its prior condition. All adjustments made during the previous traction are reversed during the backward recovery. In other words, it reprocesses valid transactions and undoes the erroneous database updates.
- Forward Recovery: "Roll forward "and" REDO" refers to forwarding recovery. When a database needs to be updated with all changes verified, this forward recovery technique is helpful. Some failed transactions in this database are applied to the database to roll those modifications forward. In other words, the database is restored using preserved data and valid transactions counted by their past saves.