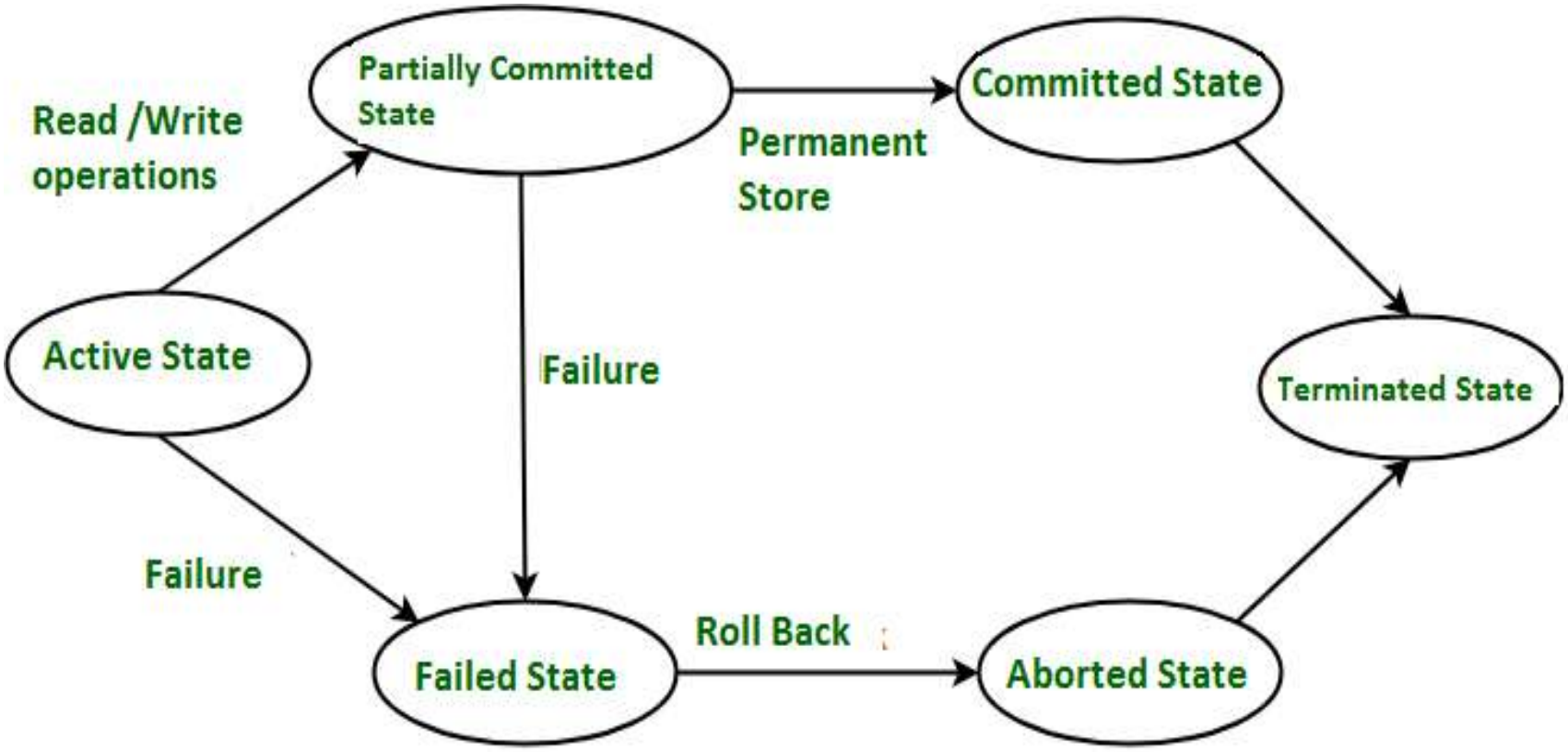
TRANSACTION

A collection of actions that transforms the DB from one consistent state to another consistent state, during the execution.

States of Transaction



Transaction States in DBMS

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 operation the changes are made in 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 Data Base and the transaction is complete and therefore terminated in the "terminated state".

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.

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 the database are not visible.

Commit : If a transaction commits, changes made are visible.

Atomicity is also known as the 'All or nothing rule'.

Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

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.

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, the 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 the 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.

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.

What is a Schedule?

A schedule is a series of operations from one or more transactions. A schedule can be of two types:

Serial Schedule: When one transaction completely executes before starting another transaction, the schedule is called a serial schedule.

A serial schedule is always consistent.

Example: If a schedule S has debit transaction T1 and credit transaction T2, possible serial schedules are T1 followed by T2 (T1->T2) or T2 followed by T1 ((T2->T1).

A serial schedule has low throughput and less resource utilization.

Concurrent Schedule: When operations of a transaction are interleaved with operations of other transactions of a schedule, the schedule is called a Concurrent schedule.

Example: The Schedule of debit and credit transactions shown in Table 1 is concurrent.

But concurrency can lead to inconsistency in the database.

The above example of a concurrent schedule is also inconsistent.

Concurrency Control in DBMS

Executing a single transaction at a time will increase the waiting time of the other transactions which may result in delay in the overall execution.

Concurrency control provides a procedure that is able to control concurrent execution of the operations in the database.

The fundamental goal of database concurrency control is to ensure that concurrent execution of transactions does not result in a loss of database consistency.

The concept of serializability can be used to achieve this goal, since all serializable schedules preserve consistency of the database.

Concurrency Control Protocols

Concurrency control protocols are the set of rules which are maintained in order to solve the concurrency control problems in the database.

It ensures that the concurrent transactions can execute properly while maintaining the database consistency.

- Lock based concurrency control protocol
- Timestamp based concurrency control protocol

Locked based Protocol

In locked based protocol, each transaction needs to acquire locks before they start accessing or modifying the data items.

There are two types of locks used in databases.

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.

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.

There are two kind of lock based protocol mostly used in database:

Two Phase Locking Protocol :

Two phase locking is a widely used technique which ensures strict ordering of lock acquisition and release.

Two phase locking protocol works in two phases.

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.

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.

Strict Two Phase Locking Protocol :

It is almost similar to the two phase locking protocol the only difference is that in two phase locking the transaction can release its locks before it commits, but in case of strict two phase locking the transactions are only allowed to release the locks only when they performs commits.

Timestamp based Protocol

In this protocol each transaction has a timestamp attached to it.

Timestamp is nothing but the time in which a transaction enters into the system.

The conflicting pairs of operations can be resolved by the timestamp ordering protocol through the utilization of the timestamp values of the transactions.

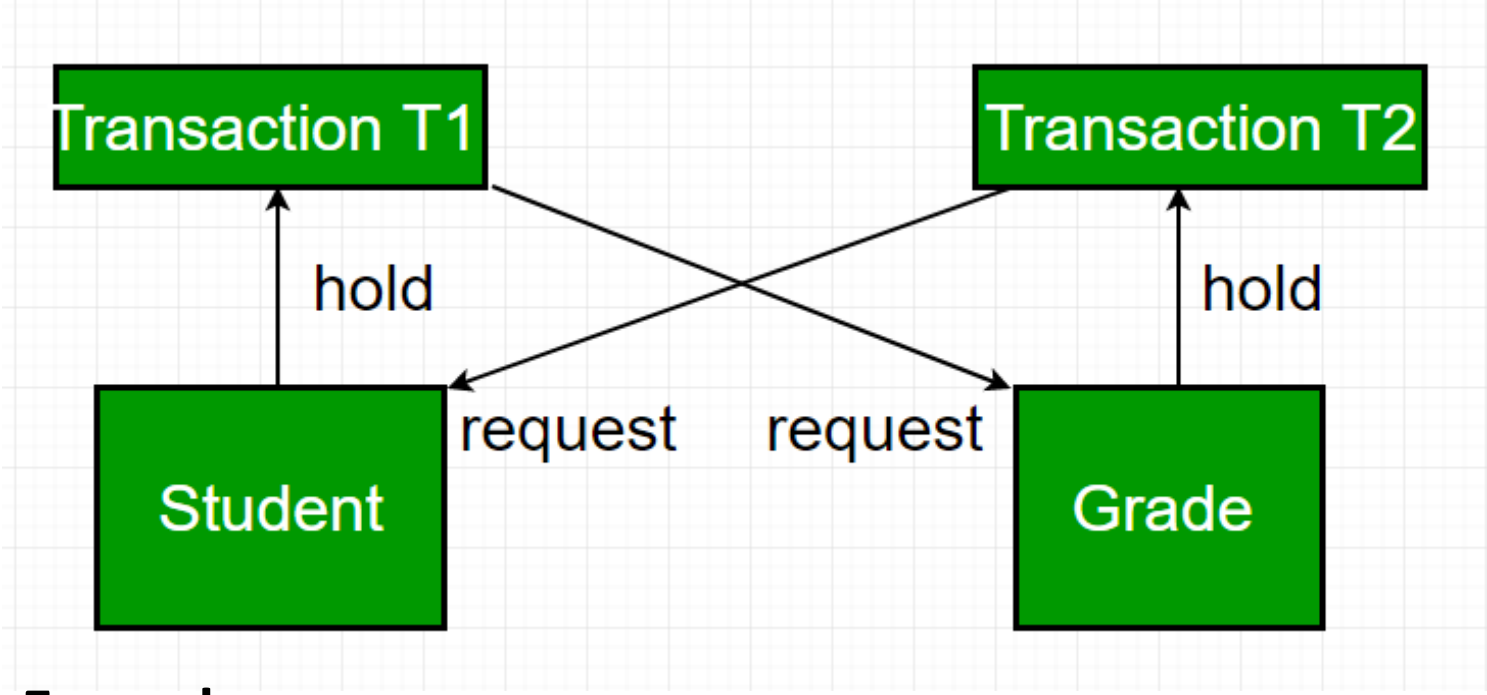
Therefore, guaranteeing that the transactions take place in the correct order.

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 results in a cycle of the dependencies where no transaction can proceed.

Basically, **Deadlocks occur when two or more transactions wait indefinitely for resources held by each other.**



Example:

Transaction T1 holds a lock on some rows in the Students table and **needs to update** some rows in the Grades table.

Simultaneously, Transaction **T2 holds** locks on those very rows (Which T1 needs to update) in the Grades table **but needs** to update the rows in the Student table **held by Transaction T1**.

What is Deadlock Avoidance?

When a database is stuck in a deadlock, It is always better to avoid the deadlock rather than restarting or aborting the database.

The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases.

Transaction T1 simply waits for transaction T2 to release the lock on Grades before it begins.

When transaction T2 releases the lock, Transaction T1 can proceed freely.

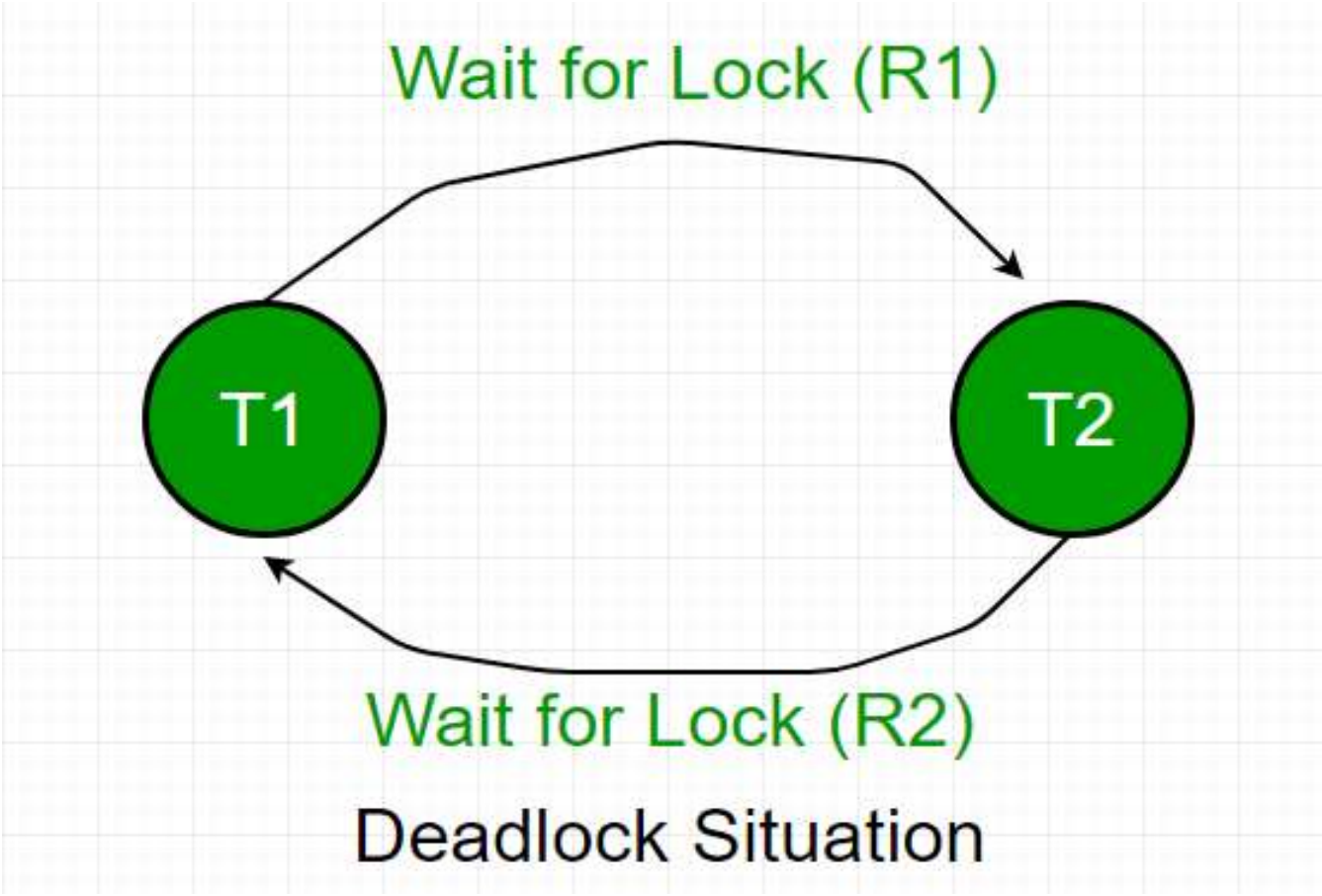
What is Deadlock Detection?

When a transaction waits indefinitely to obtain a lock, The database management system should detect whether the transaction is involved in a deadlock or not.

Wait-for-graph is one of the methods for detecting the deadlock situation.

This method is suitable for smaller databases. In this method, a graph is drawn based on the transaction and its lock on the resource.

If the graph created has a closed loop or a cycle, then there is a deadlock.



Deadlock Situation

What is Deadlock Prevention?

For a large database, the deadlock prevention method is suitable.

A deadlock can be prevented if the resources are allocated in such a way that a deadlock never occurs.

The DBMS analyzes the operations whether they can create a deadlock situation or not, If they do, that transaction is never allowed to be executed.

Deadlock prevention mechanism proposes two schemes:

- **Wait-Die Scheme**
- **Wound Wait Scheme**

Wait-Die Scheme:

In this scheme, If a transaction requests a resource that is locked by another transaction, then one of the two possibilities may occur:

- if TS(Ti)<TS(Tj)- Ti which is requesting a conflicting lock, is older than Tj. Then Ti is allowed to wait until the data item is available.

- if TS(Ti)>TS(Tj)- Ti is younger than Tj, Then Ti dies. Ti is restarted later with a random delay but with same time stamp.

Wound Wait Scheme:

In this scheme, if an older transaction requests for a resource held by a younger transaction, then an older transaction forces a younger transaction to kill the transaction and release the resource.

The younger transaction is restarted with a minute delay but with the same timestamp.

If the younger transaction is requesting a resource that is held by an older one, then the younger transaction is asked to wait till the older one releases it.

Optimistic Concurrency Control (OCC)

Optimistic Concurrency Control is a technique used in Database Management Systems (DBMS) to manage concurrent transactions.

It is based on the idea that conflicts between transactions will be rare, so instead of locking data for the duration of a transaction, transactions are allowed to execute without restrictions and are only checked for conflicts at the end of their execution.

Pessimistic Locking

Pessimistic locking is a more **conservative** approach to concurrency control, where a transaction assumes that **conflicts will happen** and therefore locks the data it needs to access or modify for the entire duration of its execution.

Locking Mechanism: A transaction acquires a lock on data before performing any operation (read, write, or update).

This lock **prevents other transactions from accessing** the data until the transaction holding the lock commits or aborts.

Optimistic Locking

Optimistic locking takes a **more optimistic** approach, assuming that **conflicts between transactions will be rare**.

Instead of acquiring locks ahead of time, transactions proceed with their operations without locking data.

Conflicts are checked only when the transaction is about to commit.

Starvation

Starvation or Livelock is the situation when a transaction has to wait for an indefinite period of time to acquire a lock.

Reasons for Starvation:

- If the waiting scheme for locked items is unfair. (priority queue)
- Victim selection (the same transaction is selected as a victim repeatedly)
- Resource leak.
- Via denial-of-service attack.

Basic Time Stamping

Basic Time Stamping is a concurrency control mechanism that eliminates deadlock.

A unique time stamp is assigned to each transaction, usually showing when it was started.

This effectively allows an age to be assigned to transactions and an order to be assigned.

Database Recovery

Database Systems like any other computer system, are subject to failures but the data stored in them must be available as and when required.

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**

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.

Backup Techniques

There are different types of Backup Techniques. Some of them are listed below.

Full database Backup: In this full database including data and database, Meta information needed to restore the whole database, including full-text catalogs are backed up in a predefined time series.

Differential Backup: It stores only the data changes that have occurred since the last full database backup.

When some data has changed many times since the last full database backup, a differential backup stores the most recent version of the changed data. For this first, we need to restore a full database backup.

Transaction Log Backup: In this, all events that have occurred in the database, like a record of every single statement executed is backed up.

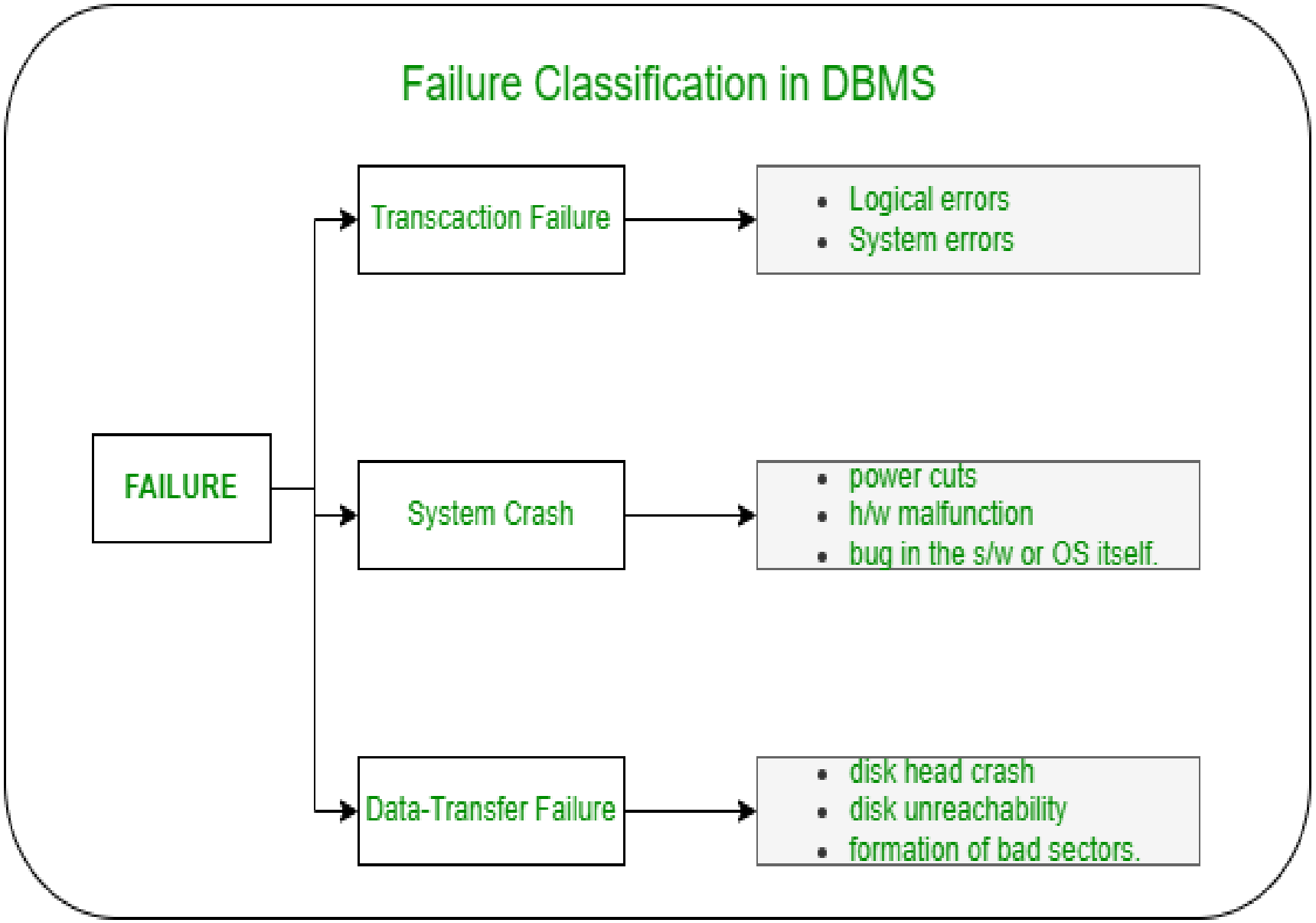
It is the backup of transaction log entries and contains all transactions that had happened to the database.

Failure:

Failure in terms of a database can be defined as its inability to execute the specified transaction or loss of data from the database.

A DBMS is vulnerable to several kinds of failures and each of these failures needs to be managed differently.

There are many reasons that can cause database failures such as network failure, system crash, natural disasters, carelessness, sabotage (corrupting the data intentionally), software errors, etc.


Failure Classification in DBMS
FAILURE
Transcaction Failure
Logical errors
System errors
System Crash
power cuts
h/w malfunction
bug in the s/w or OS itself.
Data-Transfer Failure
disk head crash
disk unreachability
formation of bad sectors.

Database Security

Database Security means keeping sensitive information safe and prevent the loss of data.

Security of data base is controlled by Database Administrator (DBA).

The following are the main control measures are used to provide security of data in databases:

- Authentication
- Access control
- Encryption

Authentication

Authentication is the process of confirmation that whether the user log in only according to the rights provided to him to perform the activities of data base.

A particular user can login only up to his privilege but he can’t access the other sensitive data.

The privilege of accessing sensitive data is restricted by using Authentication.

Access Control:

The security mechanism of DBMS must include some provisions for restricting access to the data base by unauthorized users.

Access control is done by creating user accounts and to control login process by the DBMS.

So, that database access of sensitive data is possible only to those people (database users) who are allowed to access such data and to restrict access to unauthorized persons.

Encryption

This method is mainly used to protect sensitive data (such as credit card numbers, OTP numbers) and other sensitive numbers.

The data is encoded using some encoding algorithms.

An unauthorized user who tries to access this encoded data will face difficulty in decoding it, but authorized users are given decoding keys to decode data.