Decomposition of a Relation:

The process of breaking up or dividing a single relation into two or more sub relations is called the decomposition of a relation.

Properties of Decomposition:

- **Lossless Decomposition -** Lossless decomposition ensures ○ No information is lost from the original relation during decomposition. ○ When the sub relations are joined back, the same relation is obtained that was decomposed.
- **Dependency Preservation -** Dependency preservation ensures ○ None of the functional dependencies that hold on the original relation are lost. ○ The sub relations still hold or satisfy the functional dependencies of the original relation.

Types of Decomposition:

- **Lossless Join Decomposition:**
  - ○ Consider there is a relation R which is decomposed into sub relations R1, R2, …., Rn. ○ This decomposition is called lossless join decomposition when the join of the sub relations results in the same relation R that was decomposed.
  - ○ For lossless join decomposition, we always have- R1R2 R3 ……. Rn = R

    where is a natural join operator ⋈ ⋈ ⋈

- **Lossy Join Decomposition:**⋈ ○ Consider there is a relation R which is decomposed into sub relations R1, R2, …., Rn.
  - ○ This decomposition is called lossy join decomposition when the join of the sub relations does not result in the same relation R that was decomposed.
  - ○ For lossy join decomposition, we always have- R1R2 R3 ……. Rn R

    where ⋈ is a natural join operator ⋈ ⋈ ⋈ ⊃

Normalization:

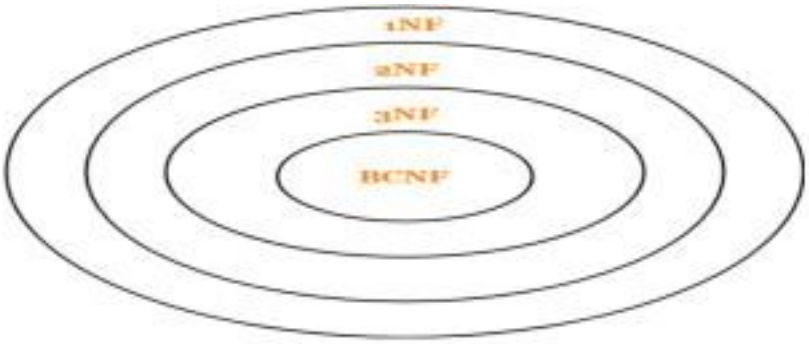
In DBMS, database normalization is a process of making the database consistent by-

- Reducing the redundancies

- Ensuring the integrity of data through lossless decomposition

Normal Forms:

- **First Normal Form (1NF) -** A given relation is called in First Normal Form (1NF) if each cell of the table contains only an atomic value i.e. if the attribute of every tuple is either single valued or a null value.
- **Second Normal Form (2NF) -** A given relation is called in Second Normal Form (2NF) if and only if ○ Relation already exists in 1NF.
    - ○ No partial dependency exists in the relation.
      A ⟶ B is called a **partial dependency** if and only if- A is a subset of some candidate key and B is a non-prime attribute.
- **Third Normal Form (3NF) -** A given relation is called in Third Normal Form (3NF) if and only if ○ Relation already exists in 2NF.
    - ○ No transitive dependency exists for non-prime attributes.
      A ⟶ B is called a **transitive dependency** if and only if- A is not a super key and B is a non-prime attribute.
- **Boyce-Codd Normal Form -** A given relation is called in BCNF if and only if ○ Relation already exists in 3NF.
    - ○ For each non-trivial functional dependency 'A ⟶ B', A is a super key of the relation.



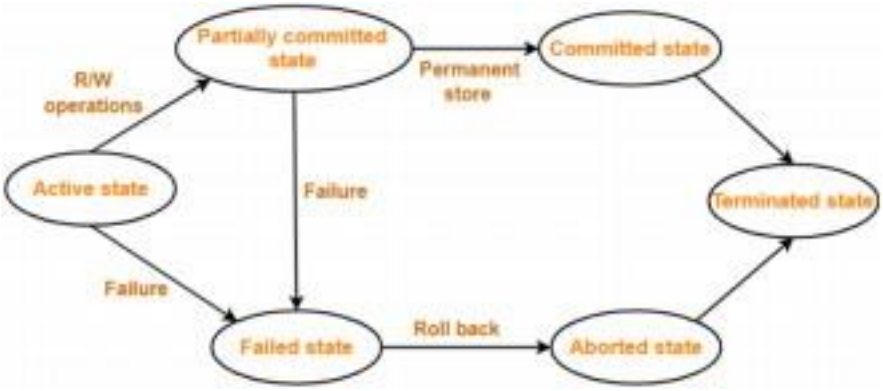
<u>Transaction:</u>

Transaction is a single logical unit of work formed by a set of operations.

Operations in Transaction:

- **Read Operation - Read(A)** instruction will read the value of 'A' from the database and will store it in the buffer in main memory.
- **Write Operation – Write(A)** will write the updated value of 'A' from the buffer to the database.

Transaction States:



- **Active State –** ○ This is the first state in the life cycle of a transaction.
    - ○ A transaction is called in an active state as long as its instructions are getting executed.
    - ○ All the changes made by the transaction now are stored in the buffer in main memory.
- **Partially Committed State –** ○ After the last instruction of the transaction has been executed, it enters into a partially committed state.
    - ○ After entering this state, the transaction is considered to be partially committed.
    - ○ It is not considered fully committed because all the changes made by the transaction are still stored in the buffer in main memory.
- **Committed State –** ○ After all the changes made by the transaction have been successfully stored into the database, it enters into a committed state.
    - o Now, the transaction is considered to be fully committed.
- **Failed State –** ○ When a transaction is getting executed in the active state or partially committed state and some failure occurs due to which it becomes impossible to continue the execution, it enters into a failed state.
- **Aborted State –** ○ After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
    - ○ To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
    - ○ After the transaction has rolled back completely, it enters into an aborted state.
- **Terminated State –**
    - ○ This is the last state in the life cycle of a transaction.
    - ○ After entering the committed state or aborted state, the transaction finally enters into a terminated state where its life cycle finally comes to an end

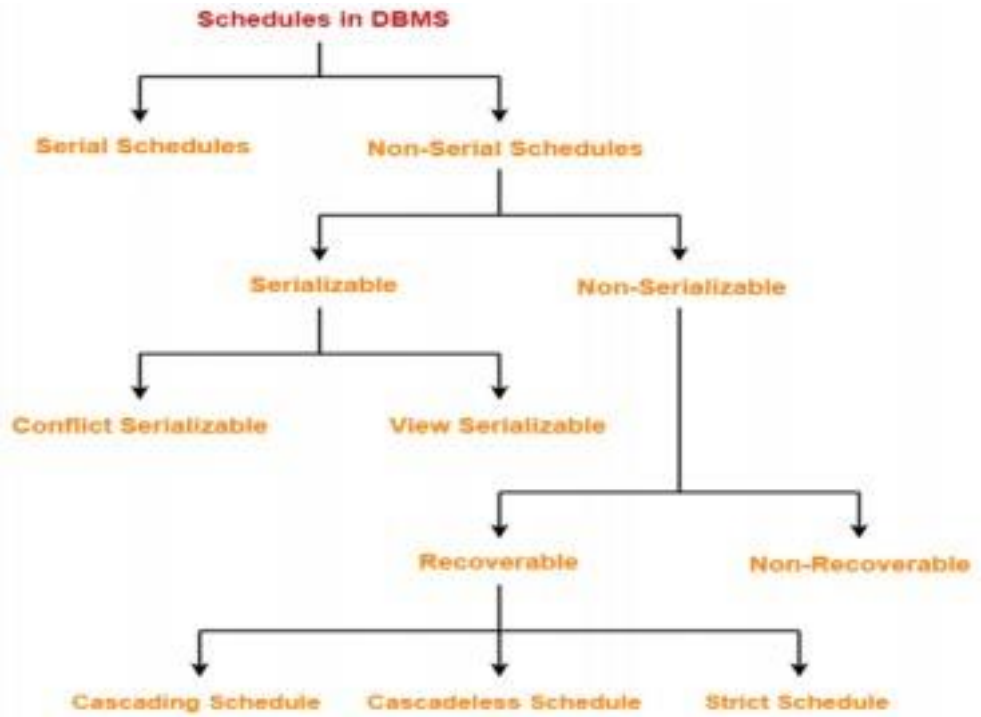
ACID Properties:

To ensure the consistency of the database, certain properties are followed by all the transactions occurring in the system. These properties are called as **ACID Properties** of a transaction.

- **Atomicity –** ○ This property ensures that either the transaction occurs completely or it does not occur at all.
    - ○ In other words, it ensures that no transaction occurs partially.
- **Consistency –** ○ This property ensures that integrity constraints are maintained.
    - ○ In other words, it ensures that the database remains consistent before and after the transaction.
- **Isolation –** ○ This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
    - ○ The resultant state of the system after executing all the transactions is the same as the state that would be achieved if the transactions were executed serially one after the other.
- **Durability –** ○ This property ensures that all the changes made by a transaction after its successful execution are written successfully to the disk.
    - ○ It also ensures that these changes exist permanently and are never lost even if there occurs a failure of any kind.

Schedules:

The order in which the operations of multiple transactions appear for execution is called as a schedule.



- **Serial Schedules –** ○ All the transactions execute serially one after the other.
  ○ When one transaction executes, no other transaction is allowed to execute.
  ○ Serial schedules are always- Consistent, Recoverable, Cascadeless and Strict. ●
  Non-Serial Schedules – ○ Multiple transactions execute concurrently. o
  Operations of all the transactions are inter leaved or mixed with each other.
  ○ Non-serial schedules are **not** always- Consistent, Recoverable, Cascadeless and Strict.

Serializability –

- Some non-serial schedules may lead to inconsistency of the database. ● Serializability is a concept that helps to identify which non-serial schedules are correct and will maintain the consistency of the database.
- **Serializable Schedules –** ○ If a given non-serial schedule of 'n' transactions is equivalent to some serial schedule of 'n' transactions, then it is called as a serializable schedule.
  - Serializable schedules are always- Consistent, Recoverable, Cascadeless and Strict.

Types of Serializability –

- **Conflict Serializability** - If a given non-serial schedule can be converted into a serial schedule by swapping its non-conflicting operations, then it is called a conflict serializable schedule.
- **View Serializability -** If a given schedule is found to be viewed as equivalent to some serial schedule, then it is called a view serializable schedule.

Non-Serializable Schedules –

- A non-serial schedule which is not serializable is called a non-serializable schedule. ● A non-serializable schedule is not guaranteed to produce the same effect as produced by some serial schedule on any consistent database.
- Non-serializable schedules- may or may not be consistent, may or may not be recoverable.
- **Irrecoverable Schedules –**
  If in a schedule, ○ A transaction performs a dirty read operation from an uncommitted transaction
  ○ And commits before the transaction from which it has read the value then such a schedule is known as an Irrecoverable Schedule.
- **Recoverable Schedules –**
  If in a schedule, ○ A transaction performs a dirty read operation from an uncommitted transaction ○ And its commit operation is delayed till the uncommitted transaction either commits or roll backs
  then such a schedule is known as a Recoverable Schedule.

Types of Recoverable Schedules –

- **Cascading Schedule -** If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Schedule or Cascading Rollback or Cascading Abort.
- **Cascadeless Schedule -** If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Cascadeless Schedule.
- **Strict Schedule -** If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Strict Schedule.