Normalization in DBMS-

- Reducing the redundancies
- Ensuring the integrity of data through lossless decomposition

Normalization is done through normal forms.

1. First Normal Form (1NF)
2. Second Normal Form (2NF)
3. Third Normal Form (3NF)
4. Boyce-Codd Normal Form (BCNF)

First Normal Form-

A given relation is called in First Normal Form (1NF) if each cell of the table contains only an atomic value.

OR

A given relation is called in First Normal Form (1NF) if the attribute of every tuple is either single valued or a null value.

Example-

Student_id	Name	Subjects
100	Akshay	Computer Networks, Designing
101	Aman	Database Management System
102	Anjali	Automata, Compiler Design

Relation is not in 1NF

- This relation can be brought into 1NF.
- This can be done by rewriting the relation such that each cell of the table contains only one value.

Student_id	Name	Subjects
100	Akshay	Computer Networks
100	Akshay	Designing
101	Aman	Database Management System
102	Anjali	Automata
102	Anjali	Compiler Design

This relation is in First Normal Form (1NF).

NOTE-

- By default, every relation is in 1NF.
- This is because formal definition of a relation states that value of all the attributes must be atomic.

Second Normal Form-

A given relation is called in Second Normal Form (2NF) if and only if-

1. Relation already exists in 1NF.
2. No partial dependency exists in the relation.

Partial Dependency

A partial dependency is a dependency where few attributes of the candidate key determines non-prime attribute(s).

OR

A partial dependency is a dependency where a portion of the candidate key or incomplete candidate key determines non-prime attribute(s).

In other words,

$A \rightarrow B$ is called a partial dependency if and only if-

1. A is a proper subset of some candidate key
2. B is a non-prime attribute.

If any one condition fails, then it will not be a partial dependency.

Example-

Consider a relation- R (V , W , X , Y , Z) with functional dependencies-

$$VW \rightarrow XY$$

$$Y \rightarrow V$$

$$WX \rightarrow YZ$$

The possible candidate keys for this relation are-

VW , WX , WY

From here,

- Prime attributes = { V , W , X , Y }
- Non-prime attributes = { Z }

Now, if we observe the given dependencies-

- There is no partial dependency.
- This is because there exists no dependency where incomplete candidate key determines any non-prime attribute.

Thus, we conclude that the given relation is in 2NF.

Third Normal Form-

A given relation is called in Third Normal Form (3NF) if and only if-

1. Relation already exists in 2NF.
2. No transitive dependency exists for non-prime attributes.

Transitive Dependency

$A \rightarrow B$ is called a transitive dependency if and only if-

1. A is not a super key.
2. B is a non-prime attribute.

If any one condition fails, then it is not a transitive dependency.

A relation is called in Third Normal Form (3NF) if and only if-

Any one condition holds for each non-trivial functional dependency $A \rightarrow B$

1. A is a super key
2. B is a prime attribute

Example-

Consider a relation- R (A , B , C , D , E) with functional dependencies-

$$A \rightarrow BC$$

$$CD \rightarrow E$$

$$B \rightarrow D$$

$$E \rightarrow A$$

The possible candidate keys for this relation are-

A , E , CD , BC

From here,

- Prime attributes = { A , B , C , D , E }
- There are no non-prime attributes

Now,

- It is clear that there are no non-prime attributes in the relation.
- In other words, all the attributes of relation are prime attributes.
- Thus, all the attributes on RHS of each functional dependency are prime attributes.

Thus, we conclude that the given relation is in 3NF.

Boyce-Codd Normal Form-

A given relation is called in BCNF if and only if-

1. Relation already exists in 3NF.
2. For each non-trivial functional dependency $A \rightarrow B$, A is a super key of the relation.

Example-

Consider a relation- R (A , B , C) with the functional dependencies-

$$A \rightarrow B$$

$$B \rightarrow C$$

$$C \rightarrow A$$

The possible candidate keys for this relation are-

A , B , C

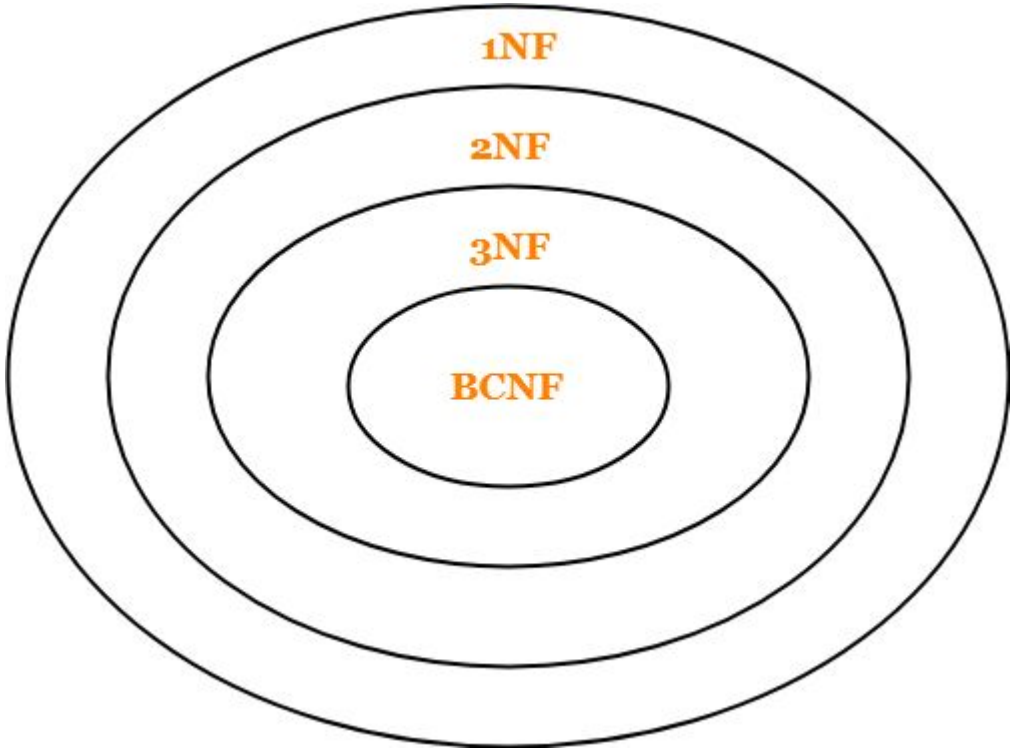
Now, we can observe that RHS of each given functional dependency is a candidate key.

Thus, we conclude that the given relation is in BCNF.

Normal Forms in DBMS-

Remember the following diagram which implies-

- A relation in BCNF will surely be in all other normal forms.
- A relation in 3NF will surely be in 2NF and 1NF.
- A relation in 2NF will surely be in 1NF.


1NF
2NF
3NF
BCNF

Point-03:

While determining the normal form of any given relation,

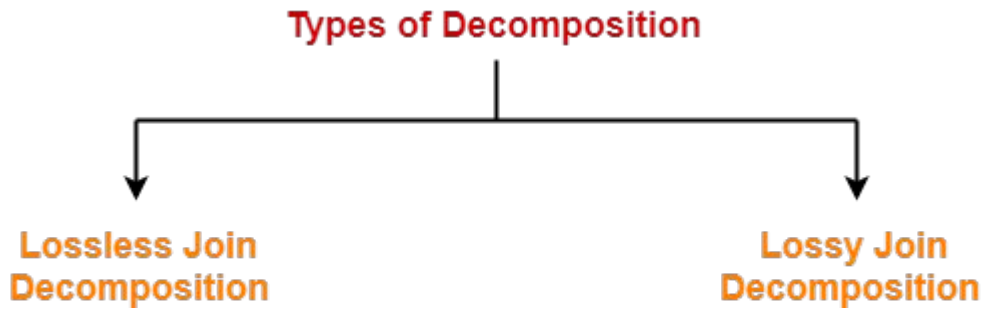
- Start checking from BCNF.
- This is because if it is found to be in BCNF, then it will surely be in all other normal forms.
- If the relation is not in BCNF, then start moving towards the outer circles and check for other normal forms in the order they appear.

Properties of Decomposition-

1. Lossless decomposition- No information is lost from the original relation during decomposition.

Dependency Preservation- None of the functional dependencies that holds on the original relation are lost.

Types of Decomposition-



1. Lossless Join Decomposition-

Consider there is a relation R which is decomposed into sub relations R_1 , R_2 , …. , $R_{n.}$

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-

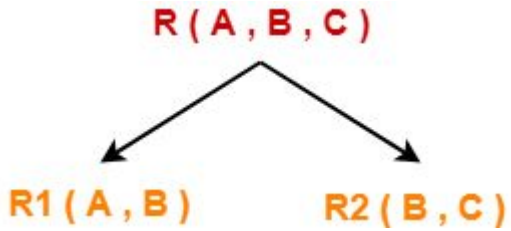
$$R_1 \bowtie R_2 \bowtie R_3 \ldots\ldots \bowtie R_n = R$$

where $\bowtie$ is a natural join operator

Example-

A	B	C
1	2	1
2	5	3
3	3	3

Consider this relation is decomposed into two sub relations R_1(A , B) and R_2(B , C)-



The two sub relations are-

A	B
1	2
2	5
3	3

R_1(A , B)

B	C
2	1
5	3
3	3

R_2(B , C)

Now, let us check whether this decomposition is lossless or not.

For lossless decomposition, we must have-

$$R_1 \bowtie R_2 = R$$

Now, if we perform the natural join ($\bowtie$) of the sub relations R_1 and R_2 , we get-

A	B	C
1	2	1
2	5	3
3	3	3

This relation is same as the original relation R.

Thus, we conclude that the above decomposition is lossless join decomposition.

NOTE-

- Lossless join decomposition is also known as **non-additive join decomposition.**
- This is because the resultant relation after joining the sub relations is same as the decomposed relation.
- No extraneous tuples appear after joining of the sub-relations.

2. Lossy Join Decomposition-

- Consider there is a relation R which is decomposed into sub relations R_1 , R_2 , …. , R_n.
- 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.
- The natural join of the sub relations is always found to have some extraneous tuples.
- For lossy join decomposition, we always have-

$$R_1 \bowtie R_2 \bowtie R_3 \ldots\ldots \bowtie R_n \supset R$$

where $\bowtie$ is a natural join operator

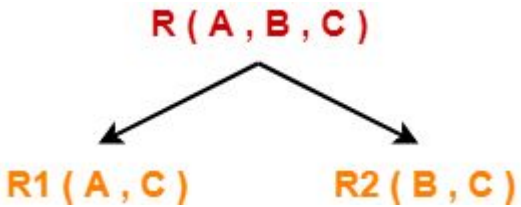
Example-

Consider the following relation R(A , B , C)-

A	B	C
1	2	1
2	5	3
3	3	3

R(A , B , C)

Consider this relation is decomposed into two sub relations as R_1(A , C) and R_2(B , C)-



The two sub relations are-

A	C
1	1
2	3
3	3

R_1(A , B)

B	C
2	1
5	3
3	3

R_2(B , C)

Now, let us check whether this decomposition is lossy or not.

For lossy decomposition, we must have-

$$R_1 \bowtie R_2 \supset R$$

Now, if we perform the natural join ($\bowtie$) of the sub relations R_1 and R_2 we get-

A	B	C
1	2	1
2	5	3
2	3	3
3	5	3
3	3	3

This relation is not same as the original relation R and contains some extraneous tuples.

Clearly, $R_1 \bowtie R_2 \supset R$.

Thus, we conclude that the above decomposition is lossy join decomposition.

Determining Whether Decomposition Is Lossless Or Lossy-

Consider a relation R is decomposed into two sub relations R_1 and R_2.

Then,

- If all the following conditions satisfy, then the decomposition is lossless.
- If any of these conditions fail, then the decomposition is lossy.

Condition-01:

Union of both the sub relations must contain all the attributes that are present in the original relation R.

Thus,

$$R_1 \cup R_2 = R$$

Condition-02:

- Intersection of both the sub relations must not be null.
- In other words, there must be some common attribute which is present in both the sub relations.

Thus,

$$R1 \cap R2 \neq \varnothing$$

Condition-03:

Intersection of both the sub relations must be a super key of either R_1 or R_2 or both.

Thus,

$$R_1 \cap R_2 = \text{Super key of } R_1 \text{ or } R_2$$

Transaction in DBMS-

Transaction is a set of operations which are all logically related

Operations in Transaction-

The main operations in a transaction are-

1. Read Operation
2. Write Operation

1. Read Operation-

- Read operation reads the data from the database and then stores it in the buffer in main memory.
- For example- **Read(A)** instruction will read the value of A from the database and will store it in the buffer in main memory.

2. Write Operation-

- Write operation writes the updated data value back to the database from the buffer.
- For example- **Write(A)** will write the updated value of A from the buffer to the database.

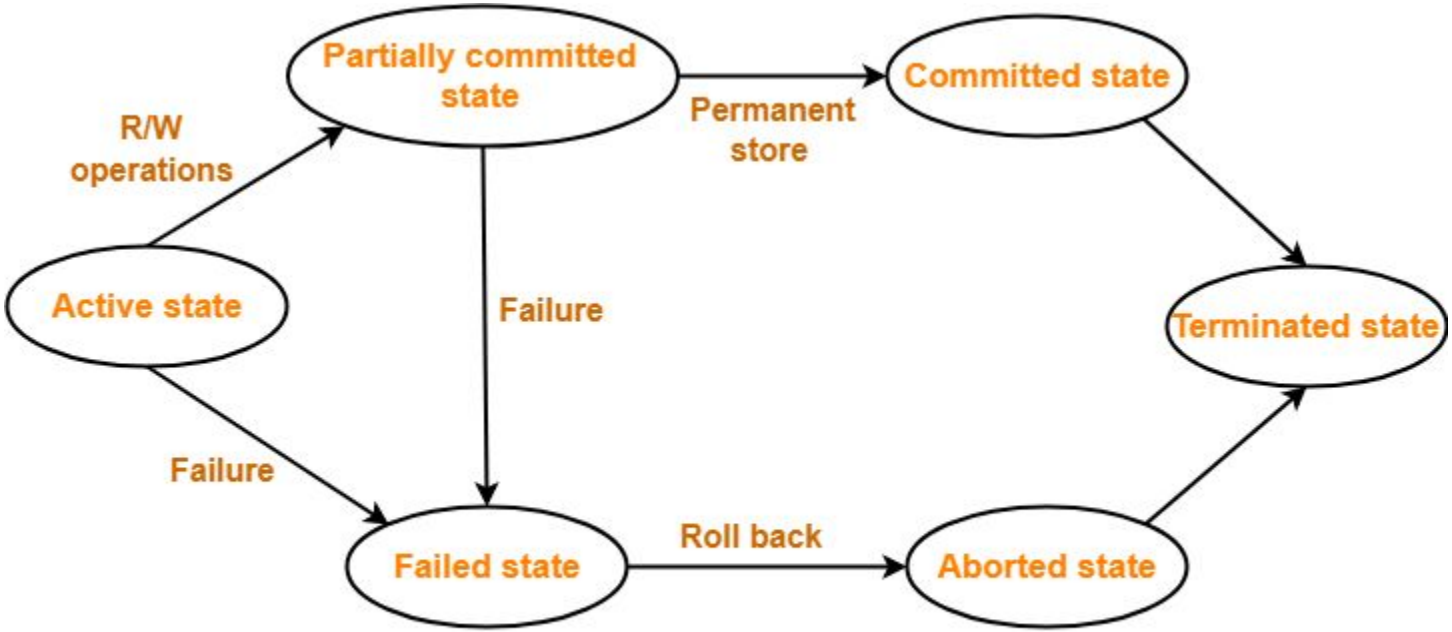
Transaction States-

A transaction goes through many different states throughout its life cycle.

These states are called as **transaction states**.

Transaction states are as follows-

1. Active state
2. Partially committed state
3. Committed state
4. Failed state
5. Aborted state
6. Terminated state



Transaction States in DBMS

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

2. Partially Committed State-

- After the last instruction of transaction has 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.

3. Committed State-

- After all the changes made by the transaction have been successfully stored into the database, it enters into a **committed state**.
- Now, the transaction is considered to be fully committed.

NOTE-

- After a transaction has entered the committed state, it is not possible to roll back the transaction.
- In other words, it is not possible to undo the changes that has been made by the transaction.
- This is because the system is updated into a new consistent state.
- The only way to undo the changes is by carrying out another transaction called as **compensating transaction** that performs the reverse operations.

4 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**.

5. Aborted State-

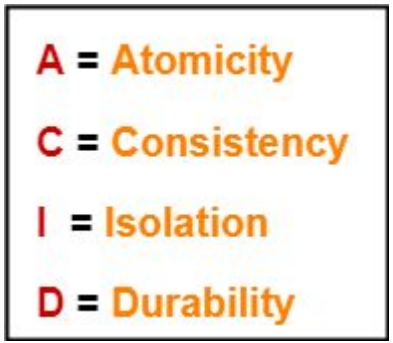
- After the transaction has failed and entered into a failed state, all the changes made by it have to be undone.
- To undo the changes made by the transaction, it becomes necessary to roll back the transaction.
- After the transaction has rolled back completely, it enters into an **aborted state**.

6. 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-

- It is important to ensure that the database remains consistent before and after the transaction.
- To ensure the consistency of database, certain properties are followed by all the transactions occurring in the system.
- These properties are called as **ACID Properties** of a transaction.



1. 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.
- That is why, it is also referred to as “**All or nothing rule**“.
- **It is the responsibility of Transaction Control Manager to ensure atomicity of the transactions.**

2. Consistency-

- This property ensures that integrity constraints are maintained.
- In other words, it ensures that the database remains consistent before and after the transaction.
- It is the responsibility of DBMS and application programmer to ensure consistency of the database.

3. Isolation-

- This property ensures that multiple transactions can occur simultaneously without causing any inconsistency.
- During execution, each transaction feels as if it is getting executed alone in the system.
- A transaction does not realize that there are other transactions as well getting executed parallely.
- Changes made by a transaction becomes visible to other transactions only after they are written in the memory.
- The resultant state of the system after executing all the transactions is same as the state that would be achieved if the transactions were executed serially one after the other.
- **It is the responsibility of concurrency control manager to ensure isolation for all the transactions.**

4. 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.
- It is the responsibility of recovery manager to ensure durability in the database.