Normalization :

Database Normalization is a technique of organizing the data in the database. It is a systematic approach of decomposing tables to eliminate redundancy. It is a multi-step process that puts data into tabular form, removing the duplicated data from its relational tables.
It is necessary to normalize the table that is present on the database. Every table in the database has to be in normal form. So, normalization is used for mainly two purposes:

1. It is used to eliminate repeated data. Having repeated data in the system not only makes the process slow but will cause trouble during the later part of transactions.
2. To ensure the data dependencies make some logical sense. Usually, the data is stored in database with certain logic. Huge datasets without any purposes are completely waste, it's like having an abundant resource without any application. The data that we have should make some logical sense.

Normalization came into existence because of the problems that occurred in the data. These problems are known as data anamolies. If a table is not properly normalized and has data redundancy, then it will not only eat up the extra memory space but will also make it difficult to handle and update the database. The anamolies are:

1. **Insertion anomaly:** It occurs when we cannot insert data to the table without the presence of another attribute.
2. **Update anomaly:** It is a data inconsistency that results from data redundancy and a partial update of data.
3. **Deletion Anomaly:** It occurs when certain attributes are lost because of the deletion of other attributes.

So these are some of the problems that occurred while managing the data. To eliminate all these anomalies, normalization came into existence. There are many normal forms that are still under development, but let's focus on the very basic and essential ones only.

❖ *1st Normal Form (1NF) :*

In first normal form, we tackle the problem of atomicity here atomicity, which means values in the table should not be further divided. In simple terms, a single cell cannot hold multiple values. If a table contains a composite or multivalued attributes, it violates the first normal form. The functions performed in the first normal form are :

1. It removes repeating groups from the table.
2. It creates a separate table for each set of related data.
3. It identifies each set of related data with the primary key.

Example:

Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553206126 +91 9449424949	60,131
1EDU002	Barry	+91 8762989672	48,302
1EDU003	Clair	+91 9916255225	22,900
1EDU004	David	+91 6363625811 +91 8762055007	81,538

In the above table, we can clearly see that the Phone Number column has two values. Thus it violated the 1st NF. Now if we apply the 1st NF to the above table we get the below table as the result.

Employee ID	Employee Name	Phone Number	Salary
1EDU001	Alex	+91 8553206126	60,131
1EDU001	Alex	+91 9449424949	60,131
1EDU002	Barry	+91 8762989672	48,302
1EDU003	Clair	+91 9916255225	22,900
1EDU004	David	+91 6363625811	81,538
1EDU004	David	+91 8762055007	81,538

By this, we have achieved atomicity and also each and every column have unique values.

❖ *2nd Normal Form (2NF) :*

The first condition in the 2nd NF is that the table has to be in 1st NF. The table also should not contain partial dependency. Here partial dependency means the proper subset of candidate keys determines a non-prime attribute.

Example:

Employee ID	Department ID	Office Location
1EDU001	ED-T1	Pune
1EDU002	ED-S2	Bengaluru
1EDU003	ED-M1	Delhi
1EDU004	ED-T3	Mumbai

This table has a composite primary key Emplyoee ID, Department ID. The non-key attribute is Office Location. In this case, Office Location only depends on Department ID, which is only part of the primary key. Therefore, this table does not satisfy the second Normal Form.
To bring this table to Second Normal Form, we need to break the table into two parts. Which will give us the below tables:

Employee ID	Department ID
1EDU001	ED-T1
1EDU002	ED-S2
1EDU003	ED-M1
1EDU004	ED-T3

Department ID	Office Location
ED-T1	Pune
ED-S2	Bengaluru
ED-M1	Delhi
ED-T3	Mumbai

As you can see we have removed the partial functional dependency that we initially had. Now, in the table, the column Office Location is fully dependent on the primary key of that table, which is Department ID.

❖ *3rd Normal Form (3NF) :*

The same rule applies as before i.e, the table has to be in 2NF before proceeding to 3NF. The other condition is there should be no transitive dependency for non-prime attributes. That means non-prime attributes (which doesn't form a candidate key) should not be dependent on other non-prime attributes in a given table. So a transitive dependency is a functional dependency in which X → Z (X determines Z) indirectly, by virtue of X → Y and Y → Z (where it is not the case that Y → X)

Example:

Student ID	Student Name	Subject ID	Subject	Address
1DT15ENG01	Alex	15CS11	SQL	Goa
1DT15ENG02	Barry	15CS13	JAVA	Bengaluru
1DT15ENG03	Clair	15CS12	C++	Delhi
1DT15ENG04	David	15CS13	JAVA	Kochi

In the above table, Student ID determines Subject ID, and Subject ID determines Subject. Therefore, Student ID determines Subject via Subject ID. This implies that we have a transitive functional dependency, and this structure does not satisfy the third normal form.
Now in order to achieve third normal form, we need to divide the table as shown below:

Student ID	Student Name	Subject ID	Address
1DT15ENG01	Alex	15CS11	Goa
1DT15ENG02	Barry	15CS13	Bengaluru
1DT15ENG03	Clair	15CS12	Delhi
1DT15ENG04	David	15CS13	Kochi

Subject ID	Subject

15CS11	SQL
15CS13	JAVA
15CS12	C++
15CS13	JAVA

As you can see from the above tables all the non-key attributes are now fully functional dependent only on the primary key. In the first table, columns Student Name, Subject ID and Address are only dependent on Student ID. In the second table, Subject is only dependent on Subject ID.

❖ *Boyce Codd Normal Form (BCNF) :*

This is also known as 3.5 NF. Its the higher version 3NF and was developed by Raymond F. Boyce and Edgar F. Codd to address certain types of anomalies which were not dealt with 3NF.
Before proceeding to BCNF the table has to satisfy 3rd Normal Form.
In BCNF if every functional dependency A → B, then A has to be the Super Key of that particular table.

Example:

Student ID	Subject	Professor
1DT15ENG01	SQL	Prof. Mishra
1DT15ENG02	JAVA	Prof. Anand
1DT15ENG02	C++	Prof. Kanthi
1DT15ENG03	JAVA	Prof. Anand
1DT15ENG04	DBMS	Prof. Lokesh

One student can enrol for multiple subjects. There can be multiple professors teaching one subject. And, For each subject, a professor is assigned to the student. In this table, all the normal forms are satisfied except BCNF. Why?
As you can see Student ID, and Subject form the primary key, which means the Subject column is a prime attribute. But, there is one more dependency, Professor → Subject.
And while Subject is a prime attribute, Professor is a non-prime attribute, which is not allowed by BCNF. Now in order to satisfy the BCNF, we will be dividing the table into two parts. One table will hold Student ID which already exists and newly created column Professor ID.

Student ID	Professor ID
1DT15ENG01	1DTPF01
1DT15ENG02	1DTPF02
1DT15ENG02	1DTPF03
:	:

And in the second table, we will have the columns Professor ID, Professor and Subject.

Professor ID	Professor	Subject
1DTPF01	Prof. Mishra	SQL
1DTPF02	Prof. Anand	JAVA
1DTPF03	Prof. Kanthi	C++
:	:	:

By doing this we are satisfied the Boyce Codd Normal Form.