Normalization in DBMS (Database Management System) is the process of organizing data in a database in a structured and efficient manner to minimize data redundancy, improve data integrity and reduce data anomalies. Normalization helps to ensure that the data is logically consistent, accurate, and easy to maintain.

There are several levels of normalization, known as normal forms. The most commonly used normal forms are:

First Normal Form (1NF): A relation is in first normal form if and only if it has no repeating groups or arrays. The attributes in each tuple must be atomic (indivisible) and there must be no duplicate tuples.
Example: Consider a table with the following data about customers and their orders:

Customer_ID	Customer_Name	Order_1	Order_2	Order_3
1	John	DVD	CD	DVD
2	Jane	DVD		
3	Peter	CD	CD	

This table violates 1NF because it has repeating groups (Order_1, Order_2, Order_3). To bring it to 1NF, we need to split the table into two tables: Customers and Orders.
Customers Table:

Customer_ID	Customer_Name
1	John
2	Jane
3	Peter

Orders Table:

Customer_ID	Order
1	DVD
1	CD
1	DVD
2	DVD
3	CD
3	CD

Second Normal Form (2NF): A relation is in second normal form if and only if it is in 1NF and every non-key attribute is fully functionally dependent on the primary key.
Example: Consider a table with the following data about students and their courses:

Student_ID	Course_ID	Course_Name	Instructor	Instructor_Office
001	100	Math	Smith	Room 101
002	100	Math	Jones	Room 102
003	101	Science	Brown	Room 103

This table violates 2NF because Course_Name and Instructor are dependent on the Course_ID, but only partially dependent on the primary key (Student_ID and Course_ID). To bring it to 2NF, we need to split the table into two tables: Students and Courses.

Students Table:

Student_ID	Course_ID
001	100
002	100
003	101

Courses Table:

Course_ID	Course_Name	Instructor	Instructor_Office
100	Math	Smith	Room 101
100	Math	Jones	Room 102
101	Science	Brown	Room 103

Third Normal Form (3NF): A relation is in third normal form if and only if it is in 2NF and every non-key attribute is non-transitively dependent on the primary key.

Example: Consider a table with the following data about employees and their departments:

Employee_ID	Employee_Name	Department_Name	Department_Location
001	John	Marketing	New York
002	Jane	Finance	London
003	Peter	Marketing	Paris

This table violates 3NF because Department_Location is dependent on Department_Name, which is not a candidate key. To bring it to 3NF, we need to split the table into three tables: Employees, Departments, and Locations.

Employee (Employee_ID, Employee_Name, Department_ID)

Department (Department_ID, Department_Name, Department_Location)

Second Normal Form (2NF) is a level of database normalization that requires a table to be in First Normal Form (1NF) and every non-key attribute to be fully functionally dependent on the entire primary key. In other words, a table is in 2NF if all its non-key attributes depend on the primary key and not on any part of the primary key.
To understand 2NF better, let us consider an example of a table that violates 2NF:

Order_Details Table

Order_ID	Product_ID	Product_Name	Product_Description	Quantity	Unit_Price
1001	2001	Mouse	Optical, wired	2	10
1002	2001	Mouse	Optical, wired	1	10
1002	2002	Keyboard	Wireless	1	30
1003	2002	Keyboard	Wireless	3	30

In the above table, the primary key is the combination of Order_ID and Product_ID. The table has two non-key attributes, Product_Name and Product_Description, that depend only on the Product_ID and not on the entire primary key. This means that these attributes are not fully functionally dependent on the primary key, and hence the table violates 2NF.
To bring the table to 2NF, we need to split it into two tables. One table will have the information about orders and another table will have information about products.

Orders Table

Order_ID	Product_ID	Quantity	Unit_Price
1001	2001	2	10
1002	2001	1	10
1002	2002	1	30
1003	2002	3	30

Products Table

Product_ID	Product_Name	Product_Description
2001	Mouse	Optical, wired
2002	Keyboard	Wireless

Now, we can see that the Products table has no redundancy, and every attribute in the Orders table is fully functionally dependent on the primary key. Hence, both tables are in 2NF

Third Normal Form (3NF) is a level of database normalization that requires a table to be in Second Normal Form (2NF) and every non-key attribute to be dependent only on the primary key and not on any other non-key attribute. In other words, a table is in 3NF if all its non-key attributes are dependent only on the primary key and not on any other non-key attribute.

To understand 3NF better, let us consider an example of a table that violates 3NF:
Employees Table

Employee_ID	Employee_Name	Department	Manager_Name	Manager_Department
1001	John Smith	Sales	Tom Johnson	Marketing
1002	Jane Doe	Marketing	Tom Johnson	Marketing

In the above table, the primary key is the Employee_ID. The table has two non-key attributes, Manager_Name and Manager_Department, which are dependent on the Manager's Employee_ID rather than the Employee's Employee_ID. This means that these attributes are not fully functionally dependent on the primary key, and hence the table violates 3NF.

To bring the table to 3NF, we need to split it into two tables. One table will have the information about employees, and another table will have information about managers.

Employees Table

Employee_ID	Employee_Name	Department	Manager_ID
1001	John Smith	Sales	2001
1002	Jane Doe	Marketing	2001
2001	Tom Johnson	Marketing	2001

Managers Table

Manager_ID	Manager_Name	Manager_Department
2001	Tom Johnson	Marketing

Now, we can see that the Managers table has no redundancy, and every attribute in the Employees table is dependent only on the primary key. Hence, both tables are in 3NF

Boyce-Codd Normal Form (BCNF) is a level of database normalization that requires a table to be in Third Normal Form (3NF) and every determinant (i.e., every attribute that uniquely determines another attribute) to be a candidate key. In other words, a table is in BCNF if all its determinants are candidate keys.

To understand BCNF better, let us consider an example of a table that violates BCNF:
Students_Courses Table

Student_ID	**Course_ID**	**Course_Name**	**Instructor**	**Instructor_Office**
1001	**2001**	**Math**	**John Smith**	**Room 101**
1001	**2002**	**Physics**	**Jane Doe**	**Room 102**
1002	**2002**	**Physics**	**John Smith**	**Room 101**

In the above table, the primary key is the combination of Student_ID and Course_ID. The table has two non-key attributes, Instructor and Instructor_Office, which are dependent only on the Instructor rather than the combination of Student_ID and Course_ID. This means that these attributes are not fully functionally dependent on the primary key, and hence the table violates BCNF.

To bring the table to BCNF, we need to split it into two tables. One table will have the information about students and courses, and another table will have information about instructors.

Students_Courses Table

Student_ID	**Course_ID**
1001	**2001**
1001	**2002**
1002	**2002**

|

Courses_Instructors Table

Course_ID	**Course_Name**	**Instructor_ID**
2001	**Math**	**1001**
2002	**Physics**	**1002**
2002	**Physics**	**1001**

Instructors Table

Instructor_ID	**Instructor_Name**	**Instructor_Office**
1001	**John Smith**	**Room 101**
1002	**Jane Doe**	**Room 102**

Now, we can see that the Courses_Instructors table has no redundancy, and every attribute in the Students_Courses table is dependent only on the primary key. The Courses_Instructors table is in BCNF because every determinant is a candidate key. Hence, both tables are in BCNF.

Fourth Normal Form (4NF) is a level of database normalization that requires a table to be in Boyce-Codd Normal Form (BCNF) and have no multi-valued dependencies. In other words, a table is in 4NF if it has no independent multi-valued dependencies.

To understand 4NF better, let us consider an example of a table that violates 4NF:
Customers_Orders Table

Customer_ID	Order_ID	Customer_Name	Order_Date	Product_ ID	Product_Name	Product_Category
1001	2001	John Smith	2022-01-01	3001	Laptop	Electronics
1001	2001	John Smith	2022-01-01	3002	Printer	Electronics
1002	2002	Jane Doe	2022-02-01	3001	Laptop	Electronics
1002	2003	Jane Doe	2022-03-01	3003	Smartphone	Electronics

In the above table, the primary key is the combination of Customer_ID, Order_ID, and Product_ID. The table has multiple independent multi-valued dependencies. For example, we can see that Customer_Name, Order_Date, and Product_Category are independent of each other, and their values depend only on specific combinations of the primary key. This means that the table violates 4NF.

To bring the table to 4NF, we need to split it into three tables. One table will have information about customers, another table will have information about orders, and a third table will have information about products.

Customers Table

Customer_ID	Customer_Name
1001	John Smith
1002	Jane Doe

Orders Table

Order_ID	Customer_ID	Order_Date
2001	1001	2022-01-01
2002	1002	2022-02-01
2003	1002	2022-03-01

Products Table

Product_ID	Product_Name	Product_Category
3001	Laptop	Electronics

3002	Printer	Electronics
3003	Smartphone	Electronics

Order_Products Table

Order_ID	Product_ID
2001	3001
2001	3002
2002	3001
2003	3003

Now, we can see that all the independent multi-valued dependencies are eliminated, and every attribute in the Order_Products table is dependent only on the primary key. The Order_Products table is in 4NF because it has no independent multi-valued dependencies. Hence, all four tables are in 4NF.

Fifth Normal Form (5NF) is the highest level of database normalization that aims to eliminate redundant data from a database. It is also known as Project-Join Normal Form (PJNF) and is achieved by decomposing a table into smaller tables that are free of any kind of redundancy.

To understand 5NF better, let us consider an example of a table that violates 5NF:
Students_Courses Table

Student_ID	Course_ID	Course_Name	Instructor
001	101	Math	Smith
001	101	Math	Johnson
002	101	Math	Smith
002	102	Science	Kim

In the above table, we can see that the combination of Student_ID and Course_ID is not a candidate key because a student can take the same course multiple times with different instructors. This leads to redundant data in the table, and it violates 5NF.

To bring the table to 5NF, we need to decompose it into smaller tables. First, we create a table that contains information about courses, instructors, and the relationship between them:

Courses Table

Course_ID	Course_Name	Instructor
101	Math	Smith
101	Math	Johnson
102	Science	Kim

Next, we create a table that contains information about students and the courses they take:

Students_Courses Table

Student_ID	Course_ID
001	101
002	101
002	102

Finally, we create a table that contains information about the instructors that teach each course:

Courses_Instructors Table

Course_ID	Instructor
101	Smith
101	Johnson
102	Kim

Now, we can see that all the redundant data has been eliminated, and every table is in 5NF. In the Students_Courses table, each student takes a unique course, and each course has a unique instructor. In the Courses_Instructors table, each course has a unique instructor. In the Courses table, each course has a unique name and instructor.

Domain-Key Normal Form (DKNF) is the highest level of database normalization, which ensures that all constraints on a database are expressed as domain constraints or key constraints. A table is said to be in DKNF if all its constraints are in either domain or key constraints.

To understand DKNF better, let us consider an example of a table that violates DKNF:

Employee_Project Table

Emp_ID	Project_Name	Start_Date	End_Date	Project_Manager
001	Project 1	01/01/2022	06/01/2022	John
002	Project 2	01/02/2022	06/02/2022	Jane
001	Project 3	01/03/2022	06/03/2022	John

In the above table, we can see that there is no unique key for the table. If we consider the combination of Emp_ID and Project_Name as a primary key, it still violates DKNF. This is because the table has a functional dependency between Project_Name and Project_Manager, which is not expressed as a domain or key constraint.

To bring the table to DKNF, we need to decompose it into smaller tables. First, we create a table that contains information about employees:

Employees Table

Emp_ID	Employee_Name
001	John
002	Jane

Next, we create a table that contains information about projects:

Projects Table

Project_Name	Start_Date	End_Date	Project_Manager
Project 1	01/01/2022	06/01/2022	John
Project 2	01/02/2022	06/02/2022	Jane
Project 3	01/03/2022	06/03/2022	John

Finally, we create a table that establishes a relationship between employees and projects:

Employee_Project Table

Emp_ID	Project_Name
001	Project 1
002	Project 2
001	Project 3

Now, we can see that all constraints are either domain constraints or key constraints, and the table is in DKNF. In the Projects table, Project_Name is a unique key, and Project_Manager is a domain constraint. In the Employees table, Emp_ID is a unique key, and Employee_Name is a domain constraint. In the Employee_Project table, Emp_ID and Project_Name are both unique keys.