• Normalization prevents inconsistencies in data quality • Different forms: 1NF, 2NF, 3NF, BCNF • Normalized tables prevents data redundancy. Denormalized tables better for faster execution

- Normalization is the process of eliminating data redundancy and enhancing data integrity.
- There are many normalization techniques, including the **Boyce-Codd Normal Form (BCNF)**.

—------

Normalization in SQL and database design is a process used to organize a database into tables and columns. The main idea with normalization is to divide a database into two or more tables and define relationships between the tables. The objective is to isolate data so that additions, deletions, and modifications of a field can be made in just one table and then propagated through the rest of the database via the defined relationships.

Levels of Normalization:

Normalization is typically done in several steps, each of which corresponds to a "normal form," a condition that a table must meet to be considered normalized under that level. The most commonly used normal forms are:

First Normal Form (1NF): Each table cell should contain a single value, and each record needs to be unique.
Second Normal Form (2NF): Meet all the requirements of the 1st normal form, and all non-key attributes are fully functional and depend on the primary key.
Third Normal Form (3NF): Meet all the requirements of the 2nd normal form, and all the attributes must be directly dependent on the primary key.
Boyce-Codd Normal Form (BCNF): A stronger version of the Third Normal Form (3NF). A table is in BCNF if it is in 3NF and for every one of its dependencies X → Y, X is a super key.

Example of Normalization:

Consider a table `Employee`:

EmployeeID	Name	Address	City	DepartmentID	DepartmentName
1	Alice	123 Oak St	Anytown	D1	Sales
2	Bob	456 Maple St	Anytown	D2	Engineering

This table is not normalized because:

- Non-prime attributes (`DepartmentName`) depend on other non-prime attributes (`DepartmentID`).
- The `Address` and `City` could be considered repeating groups based on employee residence.

After normalization, this might be divided into three tables:

`Employee` Table:

EmployeeID	Name	AddressID	DepartmentID
1	Alice	A1	D1
2	Bob	A2	D2

`Address` Table:

AddressID	Address	City
A1	123 Oak St	Anytown
A2	456 Maple St	Anytown

`Department` Table:

DepartmentID	DepartmentName
D1	Sales
D2	Engineering

In this example, the tables are more normalized: each table represents one entity type, and non-key attributes depend on the key (the ID fields).

When are Unnormalized Tables Better?

Despite the benefits of normalization, there are scenarios where unnormalized tables might be preferred, especially for performance reasons:

Performance: In systems where read speed is crucial and the data rarely changes (such as in data warehousing and reporting databases), unnormalized tables (or denormalization) can reduce the number of joins needed and improve query performance.
Simplicity: For very small databases, or in cases where the database is subjected to mostly read operations, the complexity introduced by multiple joins due to normalization can be unnecessary.
Aggregation: In analytical processes, having pre-aggregated data can be more efficient. Denormalized tables can store aggregated data, such as sums and averages, to speed up query processing.

Denormalization is typically applied after normalization in circumstances where the benefits of data integrity and avoidance of redundancy are outweighed by the need for speed and simplicity in query operations. However, this comes at the cost of increased storage and potential for data anomalies, so it should be done judiciously.

While normalization typically aims to reduce redundancy and ensure data integrity in transactional databases, the structuring into fact and dimension tables in data warehousing seeks to optimize for query speed and analytical clarity, often accepting some redundancy (denormalization) for these purposes.