* **Normalization:**
* **Definition:** Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. The main goals of normalization are to eliminate redundant data and ensure data dependencies make sense. This is achieved by dividing large tables into smaller, related tables and defining relationships between them.
* Normalization is usually carried out in several stages called normal forms. The most common normal forms are:

**1. First Normal Form (1NF)**: Ensures that the table has only atomic (indivisible) values, and each record is unique.

**2. Second Normal Form (2NF)**: Achieved when the table is in 1NF, and all non-key attributes are fully functional dependent on the primary key.

**3. Third Normal Form (3NF)**: Achieved when the table is in 2NF, and all non-key attributes are non-transitively dependent on the primary key (i.e., no transitive dependency).

* Advanced normal forms include Boyce-Codd Normal Form (BCNF), Fourth Normal Form (4NF), and Fifth Normal Form (5NF), each addressing specific types of redundancy and dependency issues.
* **Advantages:**
* Normalization reduces data redundancy.
* Improved database organization in general
* Improved data inconsistency
* Greater flexibility in database design
* Maintains relational integrity
* **Disadvantages:**
* Before creating the database, you must first determine the demands of the user.
* When the relationships are normalized to higher normal forms, the performance degrades.
* Careless data decomposition may lead to data loss.
* It is a time-consuming process to decompose into higher normalization forms.
* **When Use:**

Use normalization when data integrity and minimizing redundancy are the primary concerns. This is typical for transactional databases where write operations are frequent.

* **Example:**

Suppose you have a table ‘Employee with the following columns:

* EmployeeID
* EmployeeName
* CourseID
* CourseName
* InstructorName

To normalize this table, you might divide it into two tables:

* **Employee Table:**
* EmployeeID
* EmployeeName
* **Courses Table:**
* CourseID
* CourseName
* InstructorName

And a third table to establish the relationship:

* **Enrollment Table:**
* EmployeeID
* CourseID

-----------------------------------------------------------------------------------------

* **Denormalization:**
* **Definition:** Denormalization is the process of combining normalized tables to improve read performance. This is done by adding redundancy back into the database. Denormalization is typically used in scenarios where read performance is more critical than write performance and data integrity.

While denormalization can speed up read operations by reducing the number of joins needed, it can also lead to increased redundancy and potential data anomalies. It requires careful consideration and balance.

* **Advantages:**
* Increased query execution speed.
* Writing queries is much easier.
* No need to obtain data from dictionary tables where the values are constant over time.
* Ability to add aggregate data, which can be used for more efficient reporting.
* Reduction of the number of tables in a relational database.
* **Disadvantages:**
* Increased pricessing size.
* Increased table sizes.
* Increased costs of updating tables and inserts.
* Data may be inconsistent.
* **When Use:**

Use denormalization when read performance is critical, such as in data warehousing or read-heavy applications.

* **Example:**

Continuing with the previous example, you might combine the ‘Students’, ‘Courses’, and ‘Enrollment’ tables back into a single table for denormalization:

* StudentID
* StudentName
* CourseID
* CourseName
* InstructorName

This single table will improve read performance for queries that require data from all three original tables but will introduce redundancy.