***Normalization and Denormalization***

Normalization and denormalization are two contrasting database design strategies, each with its own purposes and benefits. Here’s an overview of their differences:

**Normalization**

**Definition**:  
Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity by dividing large tables into smaller ones and defining relationships among them.

**Objectives**:

* Eliminate redundant data.
* Ensure data dependencies make sense (i.e., only related data is stored together).
* Improve data integrity and consistency.

**Benefits**:

* Reduced storage space due to elimination of duplicates.
* Improved data integrity, as updates only need to be made in one place.
* Easier to maintain data accuracy.

**Disadvantages**:

* More complex queries may be needed to retrieve data due to the need for joins.
* Performance can be slower for read-heavy applications because of multiple tables and joins.

**Example**: A normalized database might have separate tables for customers, orders, and products, ensuring that each piece of information is stored only once.

**Denormalization**

**Definition**:  
Denormalization is the process of combining normalized tables back into larger tables to improve read performance, often at the expense of redundancy.

**Objectives**:

* Optimize read performance by reducing the number of joins.
* Simplify queries for reporting and data retrieval.

**Benefits**:

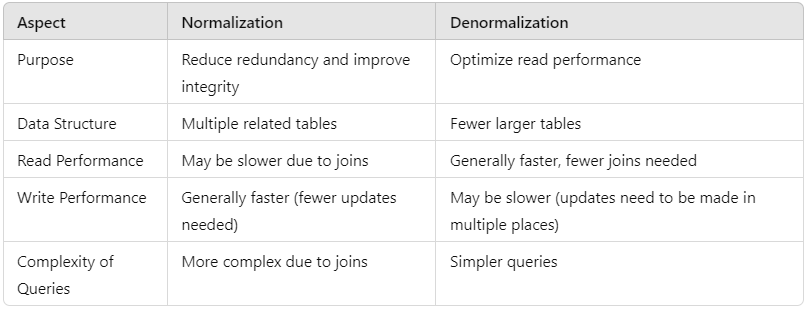
* Faster read operations, as fewer joins are needed.
* Simplified query structure, making it easier for developers to write and understand queries.

**Disadvantages**:

* Increased storage space due to redundancy.
* Higher risk of data anomalies (inconsistencies) because the same data may be stored in multiple places.
* More complex data updates, as changes need to be made in multiple locations.

**Example**: A denormalized database might combine customer and order information into a single table, reducing the need for joins when retrieving order details.

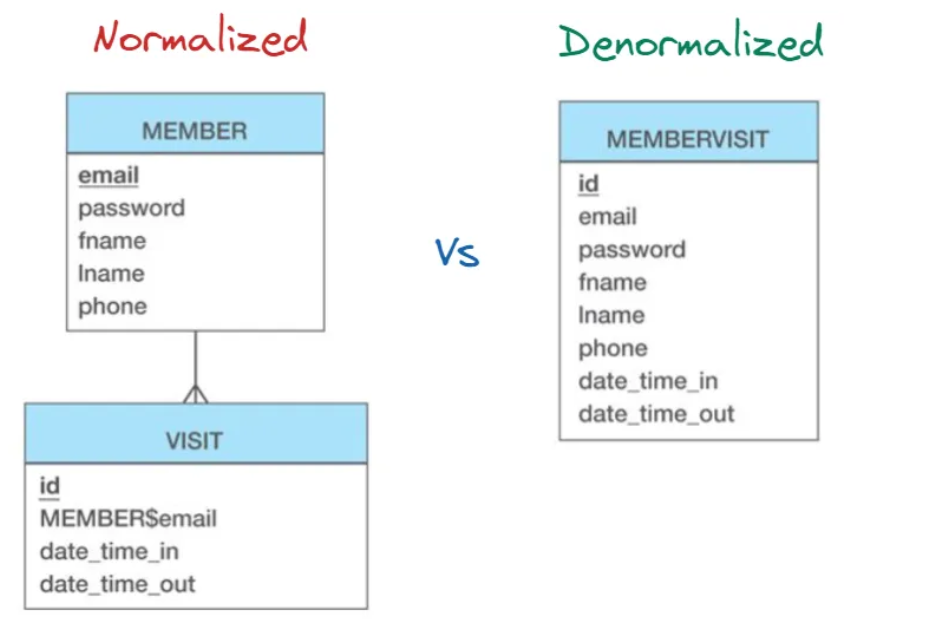
**Summary of Key Differences:**

****

**When to Use Each**

* **Normalization: Ideal for transaction-oriented systems where data integrity and consistency are crucial, such as banking or e-commerce applications.**
* **Denormalization: Beneficial for read-heavy applications, such as reporting and analytical systems, where performance is prioritized over data redundancy.**

**By understanding these differences, you can make informed decisions about your database design based on the specific needs of your application.**

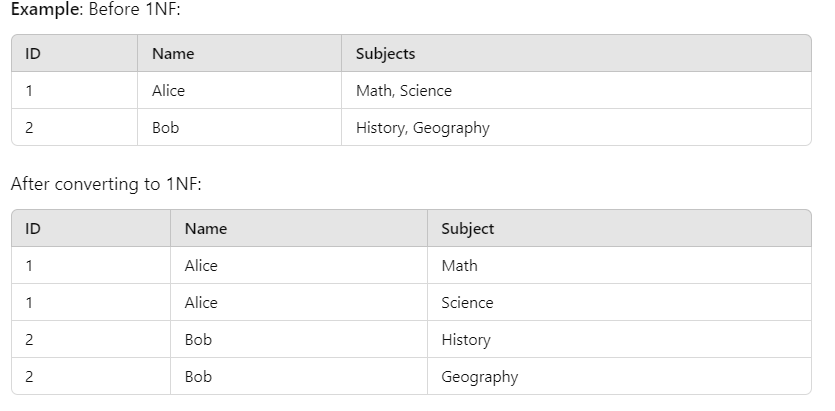
****

**Here’s a breakdown of the first three normal forms (1NF, 2NF, and 3NF) along with diagrams to illustrate each stage:**

**First Normal Form (1NF)**

**Definition:** A table is in 1NF if:

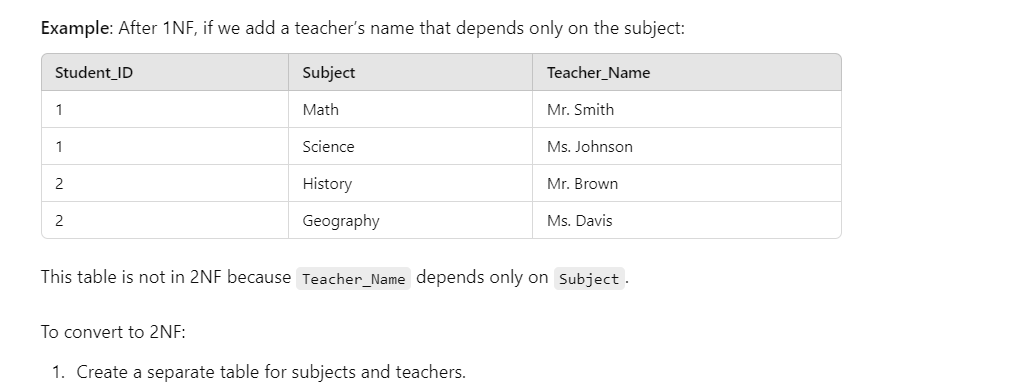
* All columns contain atomic (indivisible) values.
* Each column contains values of a single type.
* Each column has a unique name.
* The order of data does not matter.

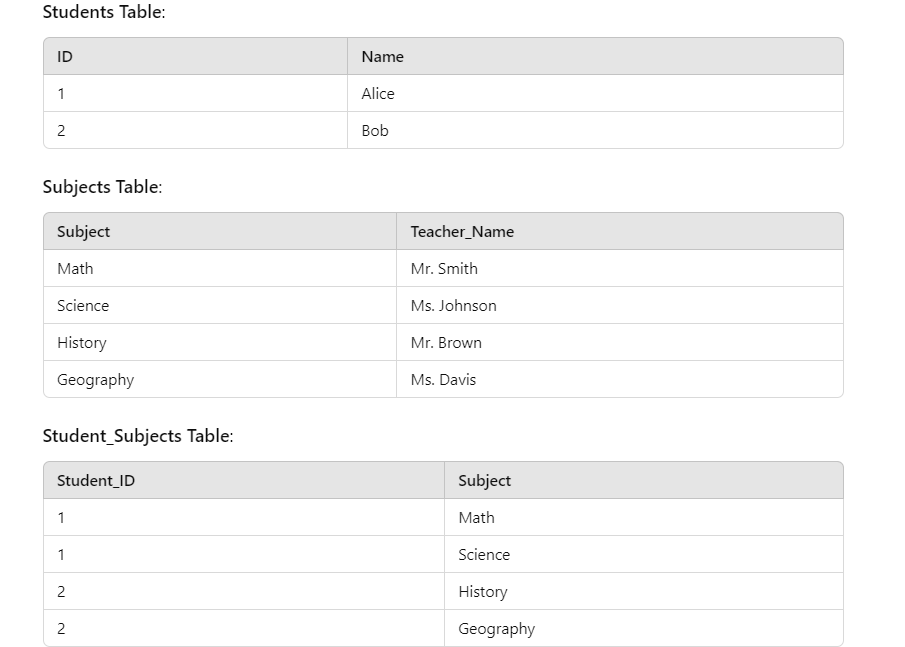


**Second Normal Form (2NF)**

**Definition**: A table is in 2NF if:

* It is in 1NF.
* All non-key attributes are fully functionally dependent on the primary key (no partial dependencies).



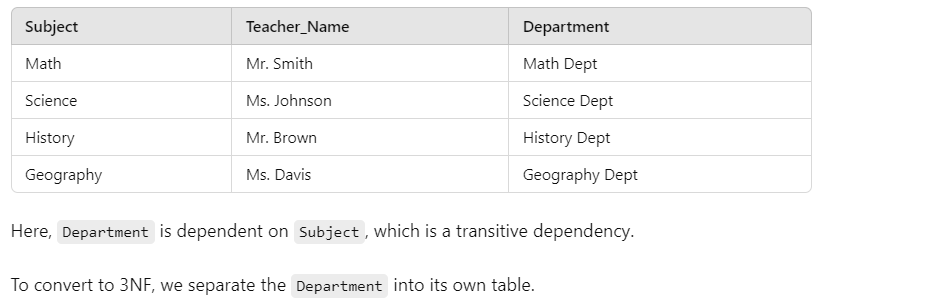


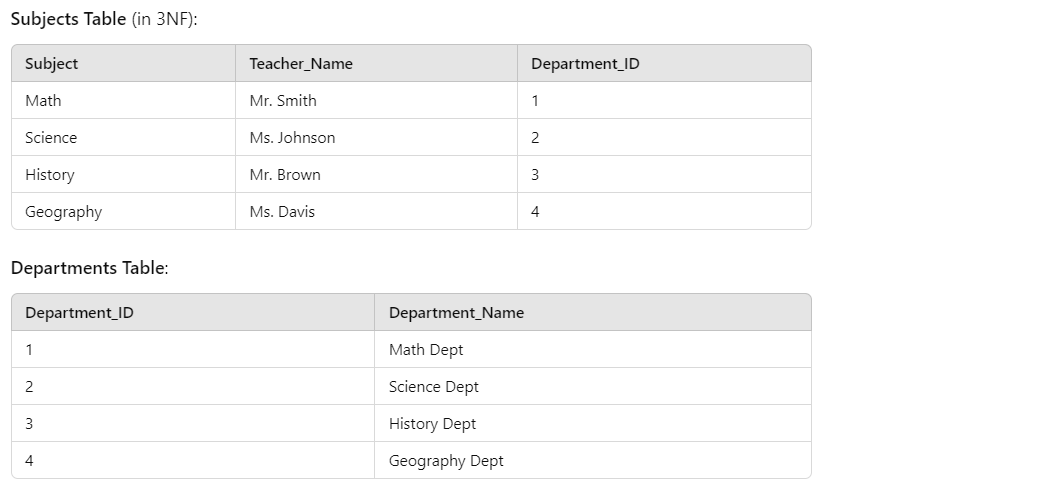
**Third Normal Form (3NF)**

**Definition**: A table is in 3NF if:

* It is in 2NF.
* There are no transitive dependencies (no non-key attribute depends on another non-key attribute).

**Example**: In the Subjects table, if we also include a department:



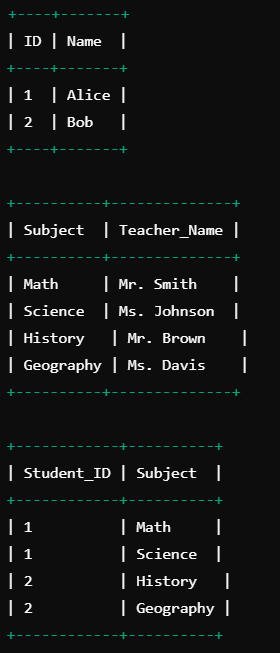


Summary Diagrams

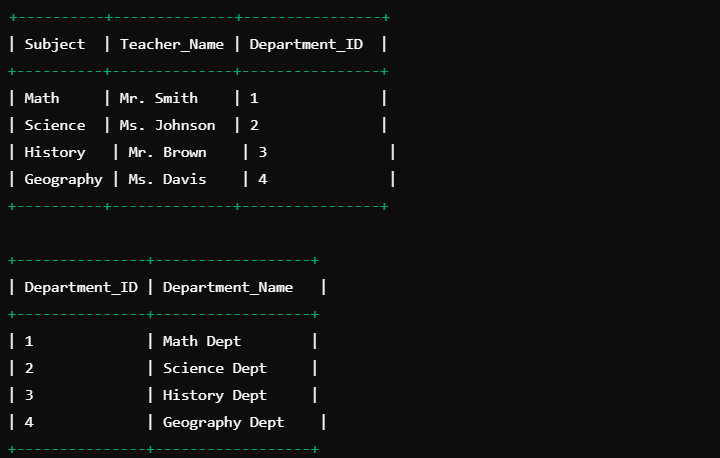
**1NF**:



**2NF**:



**3NF**:



This structured approach helps maintain data integrity and improves query efficiency, especially in large databases.