# **Normalization**

A large database defined as a single relation may result in data duplication. This repetition of data may result in:

* Making relations very large.
* It isn't easy to maintain and update data as it would involve searching many records in relation.
* Wastage and poor utilization of disk space and resources.
* The likelihood of errors and inconsistencies increases.

So to handle these problems, we should analyze and decompose the relations with redundant data into smaller, simpler, and well-structured relations that are satisfy desirable properties. Normalization is a process of decomposing the relations into relations with fewer attributes.

What is Normalization?

* Normalization is the process of organizing the data in the database.
* Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate undesirable characteristics like Insertion, Update, and Deletion Anomalies.
* Normalization divides the larger table into smaller and links them using relationships.
* The normal form is used to reduce redundancy from the database table.

Why do we need Normalization?

The main reason for normalizing the relations is removing these anomalies. Failure to eliminate anomalies leads to data redundancy and can cause data integrity and other problems as the database grows. Normalization consists of a series of guidelines that helps to guide you in creating a good database structure.

**Data modification anomalies can be categorized into three types:**

* **Insertion Anomaly:** Insertion Anomaly refers to when one cannot insert a new tuple into a relationship due to lack of data.
* **Deletion Anomaly:** The delete anomaly refers to the situation where the deletion of data results in the unintended loss of some other important data.
* **Updatation Anomaly:** The update anomaly is when an update of a single data value requires multiple rows of data to be updated.

## Types of Normal Forms:

Normalization works through a series of stages called Normal forms. The normal forms apply to individual relations. The relation is said to be in particular normal form if it satisfies constraints.

|  |  |
| --- | --- |
| **Normal Form** | **Description** |
| [1NF](https://www.javatpoint.com/dbms-first-normal-form) | A relation is in 1NF if it contains an atomic value. |
| [2NF](https://www.javatpoint.com/dbms-second-normal-form) | A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key. |
| [3NF](https://www.javatpoint.com/dbms-third-normal-form) | A relation will be in 3NF if it is in 2NF and no transition dependency exists. |
| BCNF | A stronger definition of 3NF is known as Boyce Codd's normal form. |

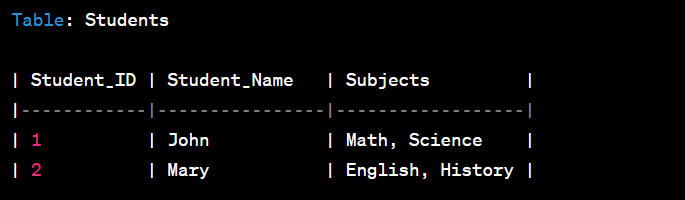
1. First normal form (1NF):

Each column in the table must contain atomic values (no multivalued attributes or repeating groups).

Each row must have a unique identifier (primary key).

Example:

Consider the following table, where each cell contains multiple values:



To bring the table to 1NF, we need to split the multivalued attribute into separate rows:

A computer screen shot of a black background

Description automatically generated

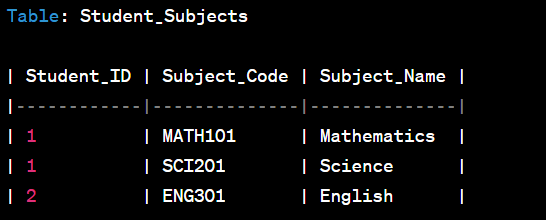
1. Second Normal Form (2NF):

The table must already be in 1NF.

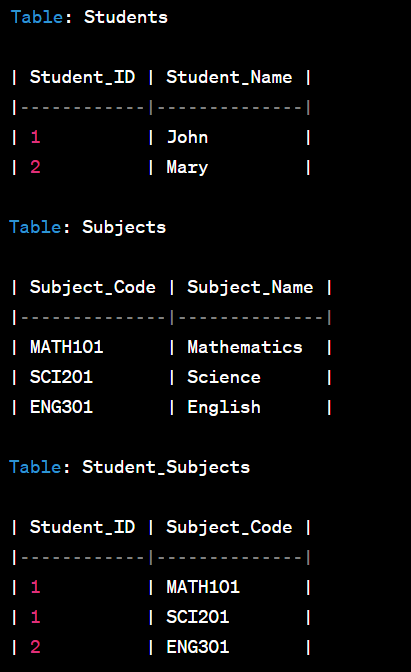
All non-key attributes must depend on the entire primary key (no partial dependencies).

Example:

Consider the following table with a composite key (student\_id, subject\_code):



Here, Subject\_Name depends on the Subject\_Code, which is a part of the composite key, not the entire primary key (Student\_ID, Subject\_Code). To achieve 2NF, we split the table into two separate tables:



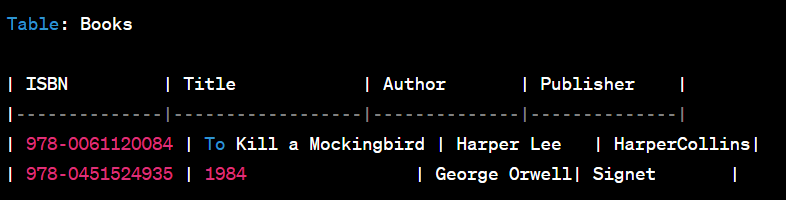
1. Third Normal Form (3NF):

The table must already be in 2NF.

There should be no transitive dependencies, i.e., non-key attributes should not depend on other non-key attributes.

Example:

Consider the following table with a transitive dependency:



Here, Publisher depends on Author, not on the primary key (ISBN). To achieve 3NF, we create a separate Authors table:

A screenshot of a computer

Description automatically generated

In 3NF, a relation is in 2NF and all the non-key attributes are dependent only on the primary key. Any non-key attribute should not be transitively dependent on another non-key attribute. This normalization form helps in avoiding transitive dependencies and eliminates redundant data.

3NF allows for some level of redundancy, but it generally works well for most practical applications and simplifies data storage. However, it may not completely eliminate all types of anomalies, such as certain types of update anomalies (e.g., insertion and deletion anomalies).

1. BCNF (Boyce-Codd Normal Form):

The table must already be in 3NF.

Every determinant (attribute on which other attributes depend) must be a candidate key.

Example:

Consider the following table with a transitive dependency:

A screen shot of a computer

Description automatically generated

Here, Department\_Name depends on Department\_ID, which is not a candidate key. To achieve BCNF, we split the table into two separate tables:

A screenshot of a computer

Description automatically generated

BCNF is a more stringent normal form than 3NF. A relation is in BCNF if and only if, for every non-trivial functional dependency X -> Y (where X is a candidate key or superkey), X must be a superkey. BCNF eliminates all types of anomalies, including transitive dependencies, and ensures that the data is well-structured with no redundancy.

While BCNF provides a higher level of data integrity and avoids certain anomalies that may still exist in 3NF, it can also result in the creation of many additional tables and can complicate data retrieval and querying.

In summary, normalization is a process of organizing data in relational databases to eliminate redundancy and improve data integrity. Each level of normalization ensures specific rules to reduce data duplication and anomalies in the database.