**Database Normalization:**

**As a Developer, you regularly work with enormous amounts of data stored in different tables that are present inside databases**.

**It often becomes difficult to extract the information if the data is unorganized**.

**We can solve this problem using Normalization by structuring the database in different forms or stages.**

What Is Normalization in DBMS?

**Normalization**is a technique for organizing the data into multiple related tables to minimize Data Redundancy and Data Inconsistency. It aims to eliminate anomalies in data.

Why Do We Need Normalization?

Data inconsistency results from anything that affects data integrity. This can cause the data to be correct in one place and wrong elsewhere it is stored. This can lead to unreliable and meaningless information. It occurs between tables when similar data is stored in different formats in two different tables.

For example, consider the following tables:

**LibraryVisitors (StudentID, Student\_Name, Student\_Address, InTime, OutTime);  
Students (StudentID, Student\_Name, Student\_Address, Department, RollNo, CourseRegistered);**

In the above tables, Student\_Address is stored in both tables. For each student\_id, the address must be the same in those two tables. Both these relations must be considered to retrieve or update the correct address. The issues mentioned arise due to poorly designed/structured databases.

We can eliminate data inconsistency in databases by using constraints on the relations.

**Data Redundancy**is the condition where the same data is stored at different locations leading to the wastage of storage space.

Examples:

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Id** | **Student Name** | **Course ID** | **Course Name** |
| 111 | John | C08 | English |
| 112 | Alice | C08 | English |
| 111 | John | C02 | French |

In the above table, we have stored student name John twice as he registered for two different courses and course name English twice as two students registered for it. This is called data redundancy. Data redundancy causes many problems in databases.

We can eliminate data redundancy in the databases by the normalization of relations.

Functional Dependency

An attribute is dependent on another attribute if another attribute uniquely identifies it.

It is denoted by  A –> B, meaning A determines B, and B depends upon A.

Example: We can find the Student’s name using the Student\_ID.

What Is an Anomaly?

An anomaly is an unexpected side effect of trying to insert, update, or delete a row. Essentially more data must be provided to accomplish an operation than expected.

Consider the following relation:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Retail\_Outlet\_ID** | **Outlet\_Location** | **Item\_Code** | **Description** | **Qty\_Available** | **Retail\_Unit\_Price** |
| R1001 | King Street, Hyderabad, 540001 | I1001 | Britannia Marie Gold | 25 | 1600 |
| R1002 | Rajaji Nagar, Bangalore, 600341 | I1106 | Cookies | 58 | 1289 |
| R1003 | MVP Colony, Visakhapatnam, 500021 | I1200 | Best Rice | 22 | 2000 |
| R1001 | King street, Hyderabad | I1309 | Dal | 20 | 1500 |

Types of Anomalies

Here are some of the most common anomalies that happen in database management.

**Insertion anomalies**

These occur when we cannot insert a new tuple into the table due to a lack of data.

What happens if we try to **insert**(add) the details of a new retail outlet with no items in its stock?

* NULL values would be inserted into the item details columns, which is not preferable.

**Deletion anomalies**

They happen when the deletion of some data deletes the other required data also (Unintended data loss)

What happens if we try to **delete** the item of item code I1106?

* The details of the retail outlet R1002 will also be deleted from the database.

**Update anomalies:**

These happen when an update of a single record requires an update in multiple records.

How many rows will be **update**d if the retail outlet location of R1002 is changed from King Street to Victoria Street?

* 2 Rows will be updated

**Data redundancy**

This happens when new items are supplied to a retail outlet.

What details do we need to insert?

* Apart from all necessary details, retail\_outlet\_location will also be inserted, which is redundant.

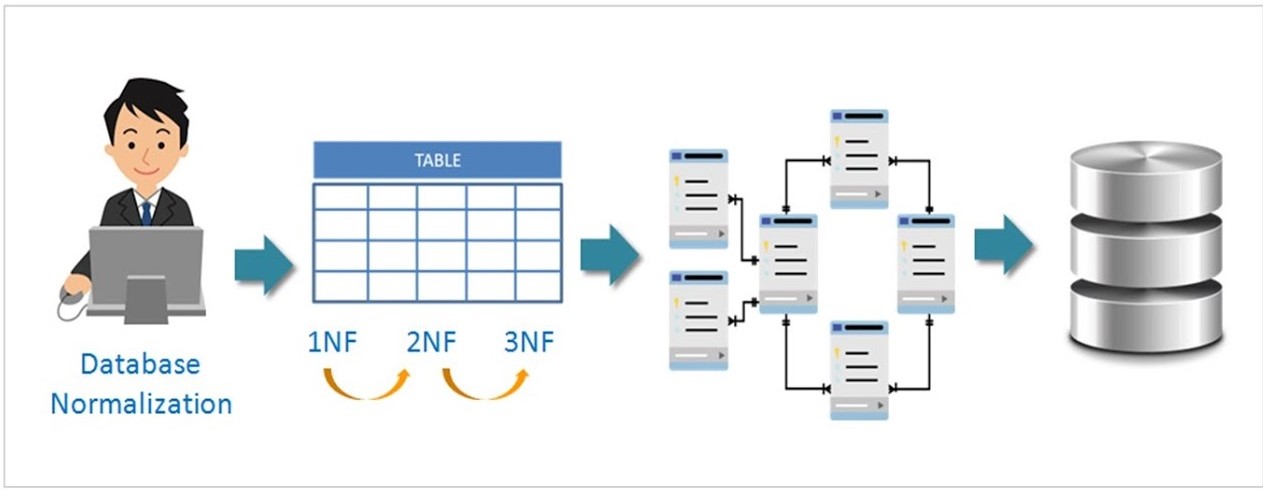
What Is Normalization?

Database normalization is the process of organizing a relational database in accordance with a series of so-called normal forms in order to reduce data redundancy and improve data integrity. It was first proposed by Edgar F. Codd.

**“Normal Forms”** (NF) are the different stages of Normalization in DBMS:

* 1 NF (First Normal Form)
* 2 NF (Second Normal Form)
* 3 NF (Third Normal Form)
* BCNF (Boyce -Codd Normal Form)
* 4 NF (Fourth Normal Form)
* 5 NF (Fifth Normal Form)
* 6 NF (Sixth Normal Form)

4NF to 6NF applies to multivalued dependencies and complex table scenarios



1 NF: First Normal Form

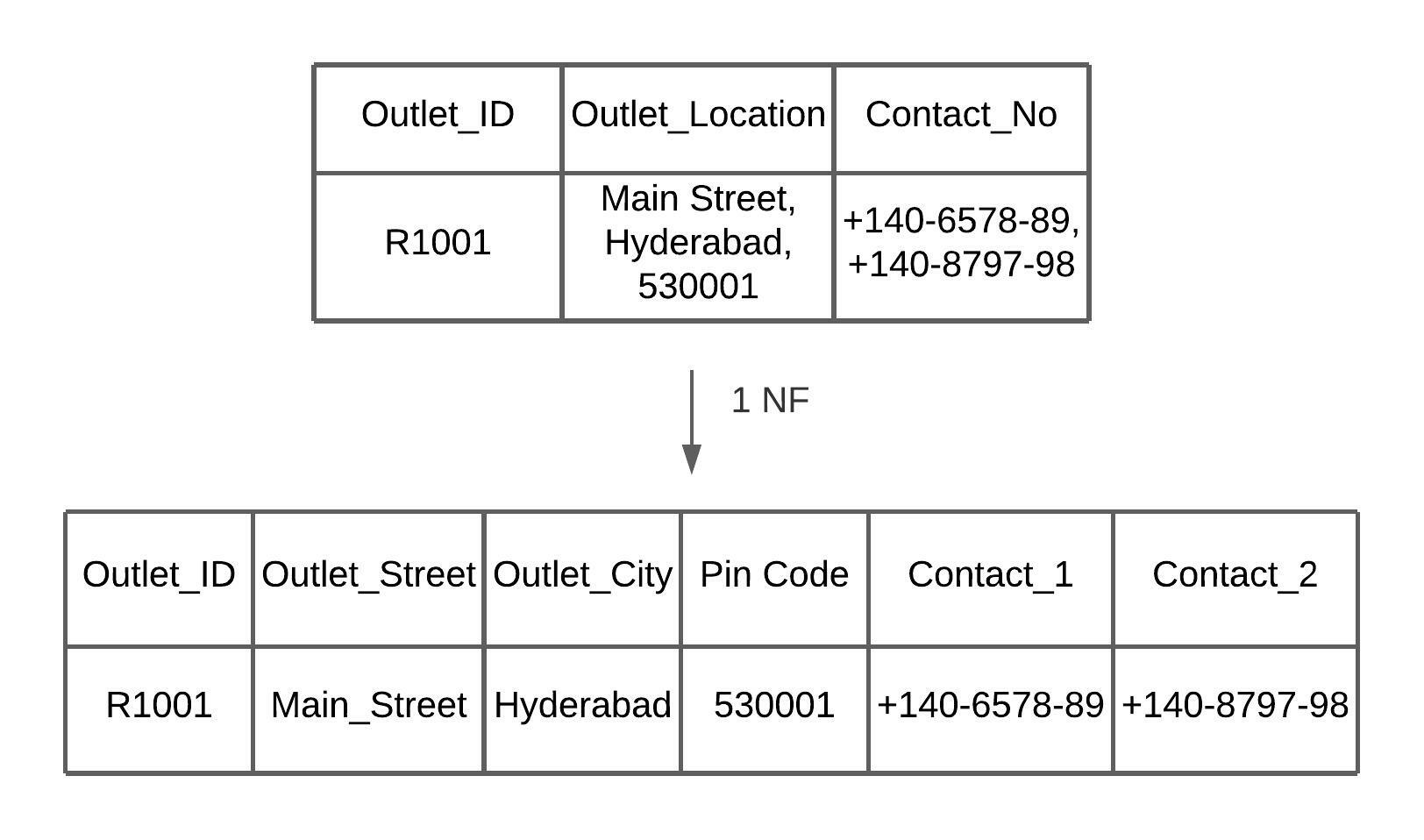
A relation R is said to be in 1 NF (First Normal) if and only if:

* All the attributes of R are **atomic**.
* It does not contain**any multi-valued** attributes.

In the above-taken example of the Retail\_Outlets table, we have stored multiple values in an address field, such as street name, city name, and pin code.

What if we want to know about all retail outlets in a given city? We may need to perform some string operations on the address field, which is not preferable. So we need to store all these **atomic** values in separate fields.

A multi-valued attribute is an attribute that can have multiple values like Contact numbers. They should also be separated like ContactNo1, ContanctNo2,.. to achieve 1st Normal form.



1st Normal Form

**Advantage:** 1 NF allows users to use the database queries effectively as it removes ambiguity by removing the non-atomic and multi-valued attributes, which creates major issues in the future while updating and extracting the data from the database.

**Limitation:** Data redundancy still exists even after 1st Normal form, so we need further normalization in DBMS.

2 NF: Second Normal Form

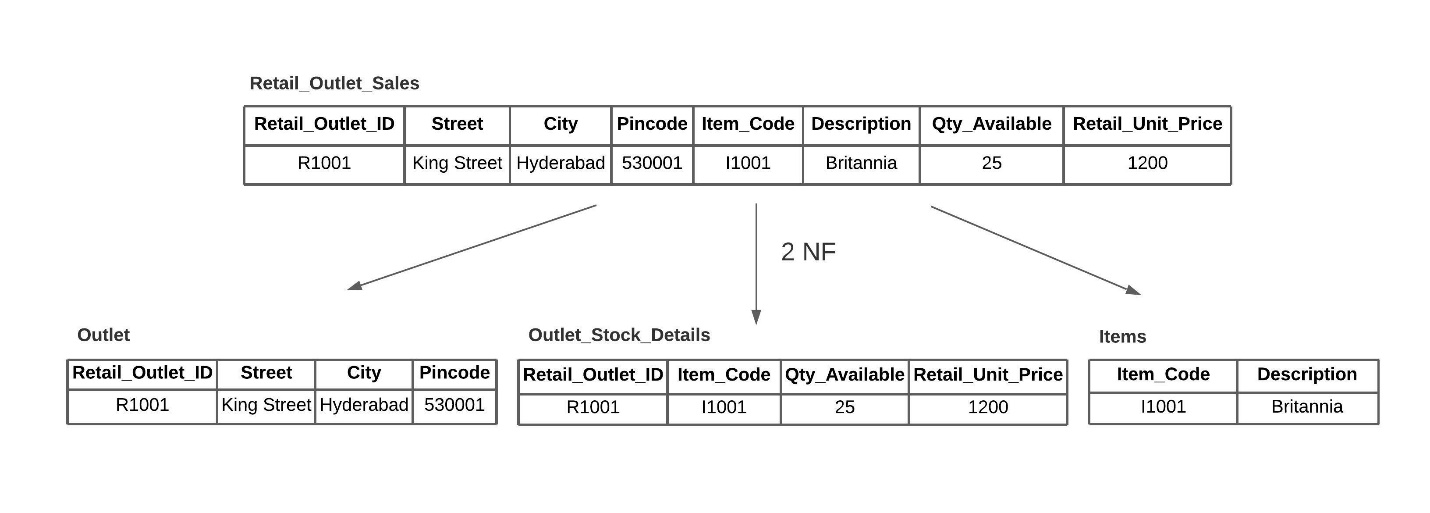
A relation R is said to be in 2 NF (Second Normal) form if and only if:

* R is **already in 1 NF**
* There is **no partial dependency** in R between non-key attributes and key attributes.

Suppose we have a composite primary or candidate key in our table. Partial dependency occurs when a part of the primary key (Key attribute) determines the non-key attribute.

In the Retail Outlets table, the Item\_Code and Retail\_Outlet\_ID are key attributes. The item description is partially dependent on Item\_Code only. Outlet\_Location depends on Retail\_Outlet\_ID. These are partial dependencies.

To achieve normalization in DBMS, we need to eliminate these dependencies by decomposing the relations.



2nd Normal Form

From the above decomposition, we eliminated the partial dependency.

**Advantage:** 2 NF attempts to reduce the amount of redundant data in a table by extracting it, placing it in a new table(s), and creating relationships between those tables.

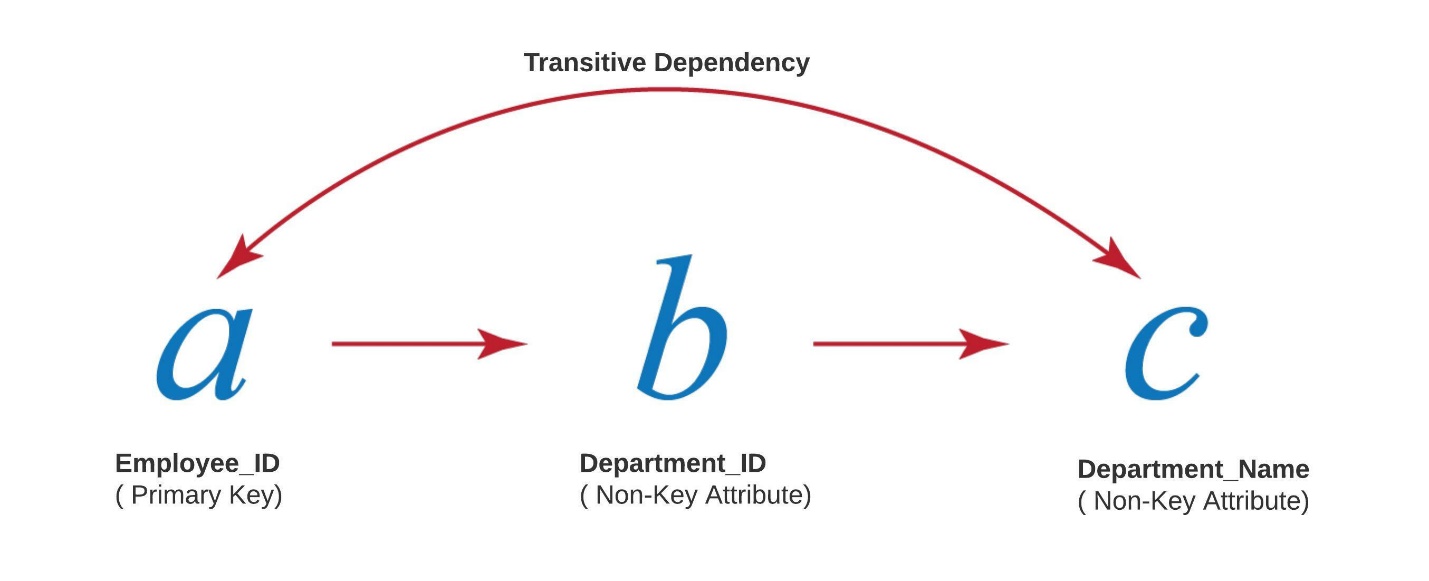
**Limitation:** There are still some anomalies, as there might be some indirect dependencies between Non-Key attributes, leading to redundant data.

3 NF: Third Normal Form

A relation R is said to be in 3 NF (Third Normal Form) if and only if:

1. R is **already in 2 NF**
2. There is no **transitive dependency** that exists between key attributes and non-key attributes through other non-key attributes.

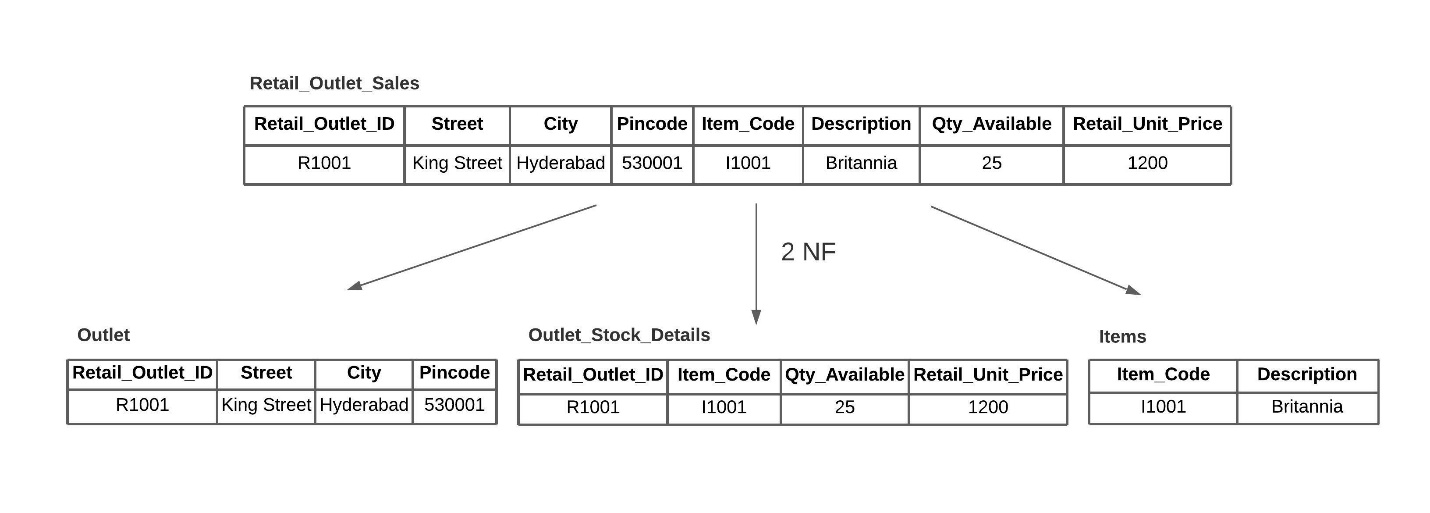
A transitive dependency exists when another non-key attribute determines a non-key attribute. In other words, If A determines B and B determines C, then automatically, A determines C.

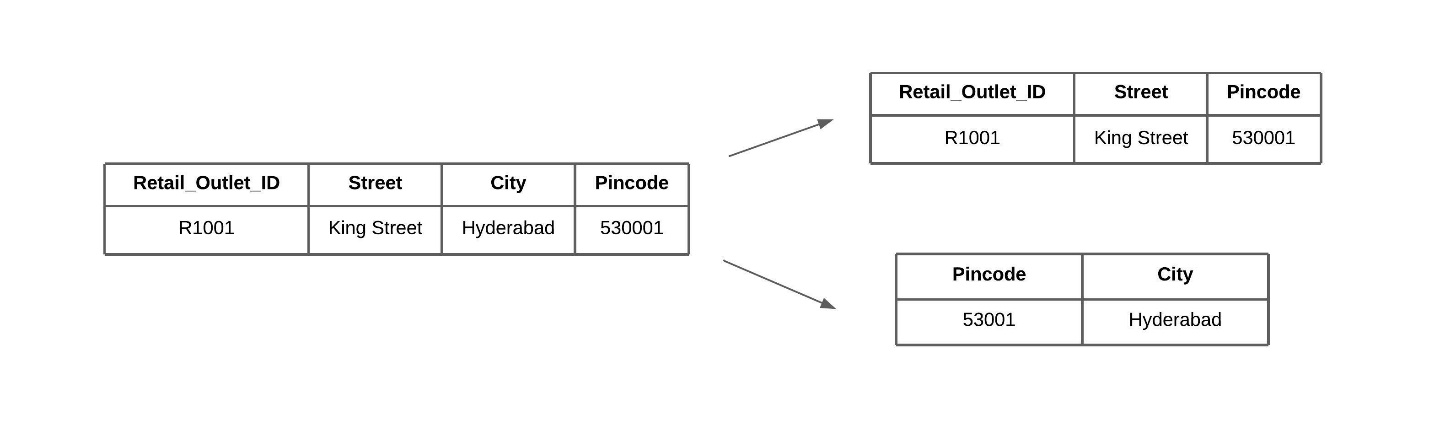


Transitive Dependency

Some other examples:

* The Year of birth determines the Age of the person
* The price of an Item determines the class of the Item
* The ZIP code of a city determines the City’s Name



* 

3rd Normal Form

**Advantage:** 3 NF ensures data integrity. It also reduces the amount of data duplication.

Boyce-Codd Normal Form

It is an upgraded version of the 3rd Normal form. It is also called as 3.5 Normal Form.

A relation R is said to be in 3 NF (Third Normal Form) if and only if:

* R is **already in 3 NF**
* For any dependency A –> B, then A should be the **Super key.**

In simple words, if A –> B, then A cannot be a non-prime Attribute if B is a prime attribute which means that A non-prime**attribute cannot determine a prime attribute.**

You must be wondering how’s this possible. but Yes, there can be some cases in which the Non-Prime attribute will determine the prime attributes even if the relationship was in the 3rd Normal form. BCNF does not allow this kind of dependency.

Sample Table

Let us understand this better with an example. Look at the below Relation of Student Enrollments table.

|  |  |  |
| --- | --- | --- |
| **Student\_ID** | **Course\_Name** | **Professor** |
| 101 | JAVA | Prof. Java |
| 102 | C++ | Prof. CPP |
| 101 | Python | Prof. Python |
| 103 | JAVA | Prof. Java\_2 |
| 104 | Python | Prof. Python\_2 |

In the above relation:

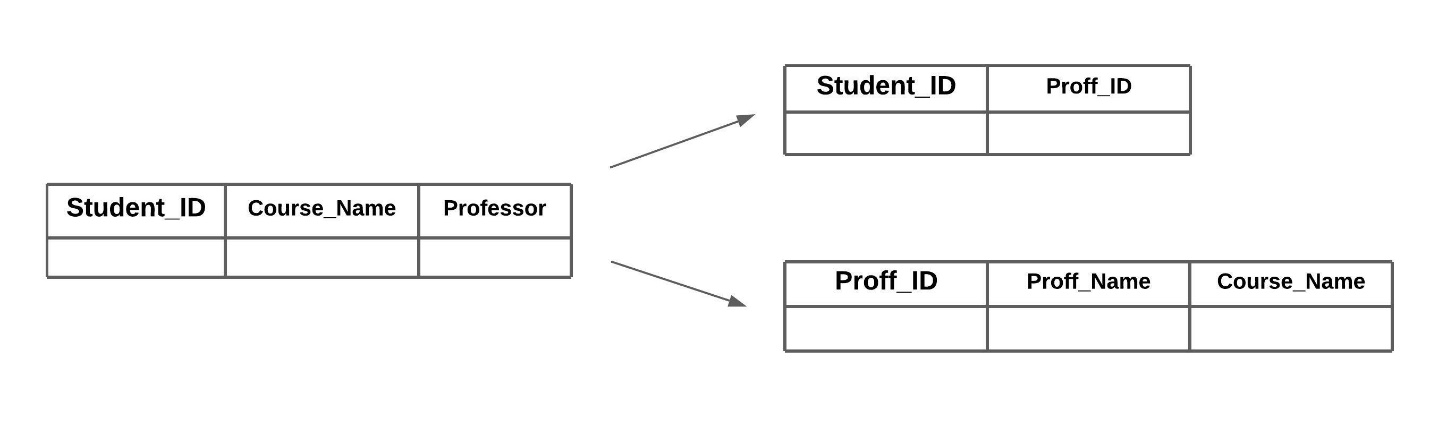
* One student can enroll in multiple courses.
* Multiple professors can teach one course.
* One professor can be assigned only one course.

So the (Student\_ID & Course\_Name) will form the primary key. These 2 will compositely determine all other attributes in the relation. In our case, it is only the professor.

* The Relation is clearly in 1st Normal Form as there are No Multivalued attributes, and all attributes have atomic values.
* The Relation is in 2nd Normal Form as there are No Partial dependencies.
* Student\_Id cannot determine Course\_Name as one student can enroll in multiple courses.
* Course\_Name cannot determine the professor, as multiple professors may teach the same course.
* The relation is in 3rd normal form as there are no transitive dependencies.

If we observe here, the “Professor” attribute, a non-prime attribute, can determine the Course\_Name as each professor teaches only one course. But Course\_Name is a prime attribute, and Professor is not a Super Key. That means a non-prime attribute determines the prime attribute.

This is not allowed in BCNF. So, how do we decompose this relation?



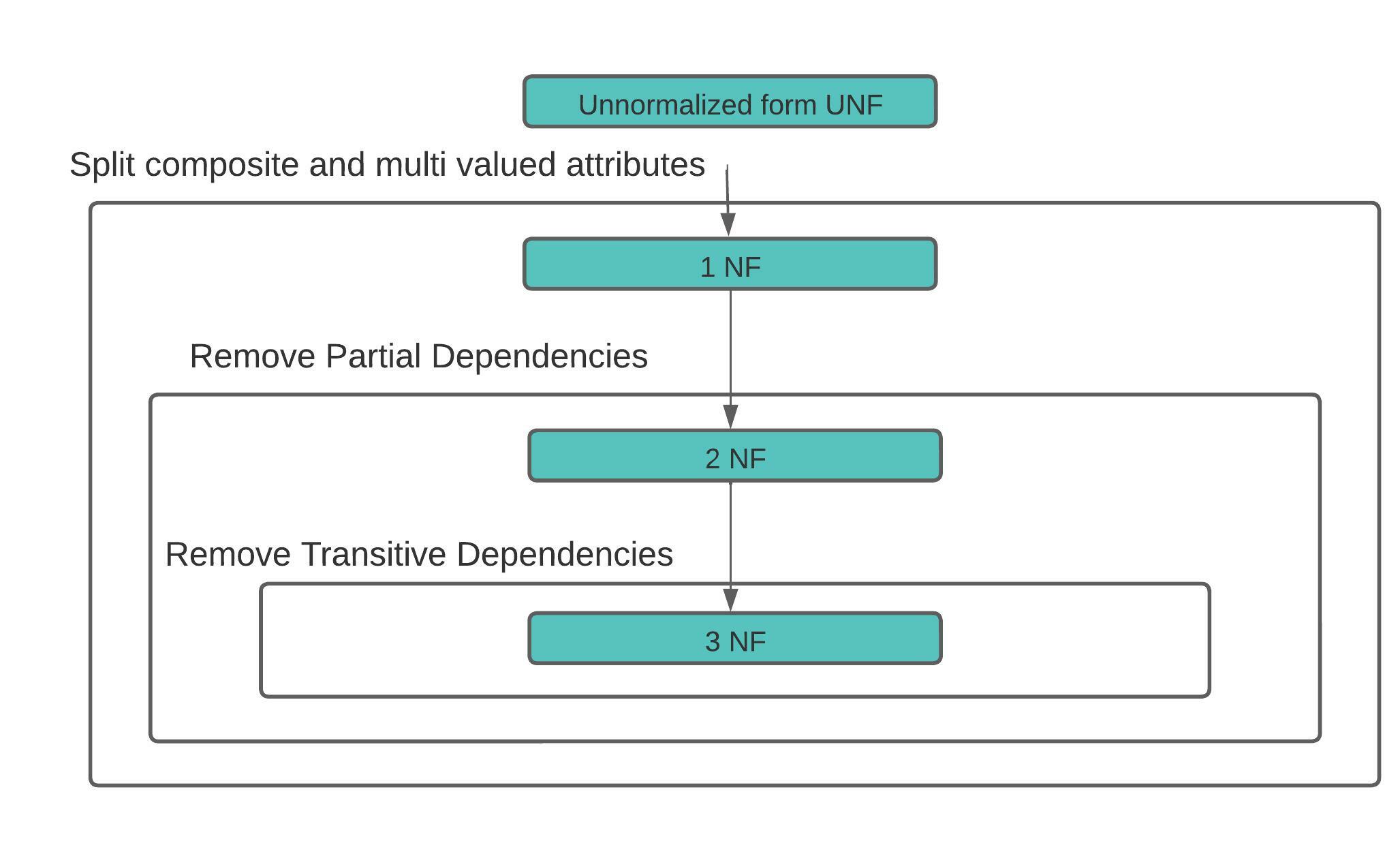
Boyce-Codd Normal Form

Until here, we have seen normal forms up to BCNF. Here are some guidelines to follow while normalizing the database.

Guidelines for Using Normalization in DBMS

* Depending on the business requirements, we can normalize the tables up to the 2nd normal form or the 3rd normal form.
* Prefer tables in 3 NF in applications with extensive data modifications.
* Prefer tables in 2 NF in applications with extensive data retrieval.
* Reason: retrieving data from multiple tables is a costly operation.
* Converting the tables from higher normal form to lower normal form is called “Denormalization”.

**The below picture summarizes how to reach the third normal form from an unnormalized form:**



Any relational database without normalization may lead to problems like large tables, difficulty maintaining the database as it involves searching many records, poor disk space utilization, and inconsistencies. If we fail to eliminate this kind of problem, it would lead to data integrity and redundancy problems. Normalization of a relational database helps to solve these problems. Normalization applies to a series of transformations in terms of normal forms. Any relation in a database must be normalized to get efficient access to the database. Each Normal form eliminates each type of dependency and improves the data integrity.

Advantages of Normalization

Normalization helps a lot with organizing data. Here are some of its advantages:

* It reduces data redundancy: Normalisation assists in removing redundant data from tables, using less storage space, and increasing database effectiveness.
* It improves data consistency: Normalisation guarantees that the data stays organized and consistent, lowering the possibility of data errors and inconsistencies.
* It makes database design simple: Normalization offers rules for arranging tables and data linkages. This facilitates database design and maintenance.
* It handles queries faster: Faster query performance is a result of normalized tables’ generally easier search and data retrieval capabilities.
* It simplifies database maintenance: By dividing a database’s complexity into smaller, more manageable tables, normalization makes it simpler to add, change, and delete data.

Conclusion

This article was aimed at making you understand the normalization process and how to apply it when you design a database system. There is another multi-valued dependency that 4NF and 5NF can eliminate. Try to explore those also.

I hope this article helped you to understand the concept of normalization better. If you have any questions, please let me know in the comments.  I wish you great learning ahead.

**Key Takeaways:**

* Normalization is a technique for organizing the data into multiple related tables to minimize Data Redundancy and Data Inconsistency.
* Insertion, deletion, update, and data redundancy are the various possible anomalies that may occur when building a database.
* There are seven different stages of normalization known as Normal Forms.