**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.

**Following are the various types of Normal forms:**



|  |  |
| --- | --- |
| **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. |
| [4NF](https://www.javatpoint.com/dbms-forth-normal-form) | A relation will be in 4NF if it is in Boyce Codd's normal form and has no multi-valued dependency. |
| [5NF](https://www.javatpoint.com/dbms-fifth-normal-form) | A relation is in 5NF. If it is in 4NF and does not contain any join dependency, joining should be lossless. |

Advantages of Normalization

* Normalization helps to minimize data redundancy.
* Greater overall database organization.
* Data consistency within the database.
* Much more flexible database design.
* Enforces the concept of relational integrity.

Disadvantages of Normalization

* You cannot start building the database before knowing what the user needs.
* The performance degrades when normalizing the relations to higher normal forms, i.e., 4NF, 5NF.
* It is very time-consuming and difficult to normalize relations of a higher degree.
* Careless decomposition may lead to a bad database design, leading to serious problems.

First Normal Form (1NF)

* A relation will be 1NF if it contains an atomic value.
* It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
* First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

**Example:** Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP\_PHONE.

**EMPLOYEE table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385, 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389, 8589830302 | Punjab |

The decomposition of the EMPLOYEE table into 1NF has been shown below:

|  |  |  |  |
| --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_PHONE** | **EMP\_STATE** |
| 14 | John | 7272826385 | UP |
| 14 | John | 9064738238 | UP |
| 20 | Harry | 8574783832 | Bihar |
| 12 | Sam | 7390372389 | Punjab |
| 12 | Sam | 8589830302 | Punjab |

Second Normal Form (2NF)

Advertisement

* In the 2NF, relational must be in 1NF.
* In the second normal form, all non-key attributes are fully functional dependent on the primary key

**Example:** Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

**TEACHER table**

|  |  |  |
| --- | --- | --- |
| **TEACHER\_ID** | **SUBJECT** | **TEACHER\_AGE** |
| 25 | Chemistry | 30 |
| 25 | Biology | 30 |
| 47 | English | 35 |
| 83 | Math | 38 |
| 83 | Computer | 38 |

In the given table, non-prime attribute TEACHER\_AGE is dependent on TEACHER\_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

**TEACHER\_DETAIL table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **TEACHER\_AGE** |
| 25 | 30 |
| 47 | 35 |
| 83 | 38 |

**TEACHER\_SUBJECT table:**

|  |  |
| --- | --- |
| **TEACHER\_ID** | **SUBJECT** |
| 25 | Chemistry |
| 25 | Biology |
| 47 | English |
| 83 | Math |
| 83 | Computer |

**Third Normal Form (3NF)**

* A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
* 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
* If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

**Example:**

**EMPLOYEE\_DETAIL table:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 222 | Harry | 201010 | UP | Noida |
| 333 | Stephan | 02228 | US | Boston |
| 444 | Lan | 60007 | US | Chicago |
| 555 | Katharine | 06389 | UK | Norwich |
| 666 | John | 462007 | MP | Bhopal |

**Super key in the table above:**

* 1. {EMP\_ID}, {EMP\_ID, EMP\_NAME}, {EMP\_ID, EMP\_NAME, EMP\_ZIP}....so on

**Candidate key:** {EMP\_ID}

**Non-prime attributes:** In the given table, all attributes except EMP\_ID are non-prime.

Here, EMP\_STATE & EMP\_CITY dependent on EMP\_ZIP and EMP\_ZIP dependent on EMP\_ID. The non-prime attributes (EMP\_STATE, EMP\_CITY) transitively dependent on super key(EMP\_ID). It violates the rule of third normal form.

That's why we need to move the EMP\_CITY and EMP\_STATE to the new <EMPLOYEE\_ZIP> table, with EMP\_ZIP as a Primary key.

**EMPLOYEE table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_ID** | **EMP\_NAME** | **EMP\_ZIP** |
| 222 | Harry | 201010 |
| 333 | Stephan | 02228 |
| 444 | Lan | 60007 |
| 555 | Katharine | 06389 |
| 666 | John | 462007 |

**EMPLOYEE\_ZIP table:**

|  |  |  |
| --- | --- | --- |
| **EMP\_ZIP** | **EMP\_STATE** | **EMP\_CITY** |
| 201010 | UP | Noida |
| 02228 | US | Boston |
| 60007 | US | Chicago |
| 06389 | UK | Norwich |
| 462007 | MP | Bhopal |

**Partial, Full, and Transitive Dependencies**

Functional Dependency is a key feature of a Database management System. Functional Dependency is used to maintain the relationship between various attributes in a given database.

**What is Functional Dependency?**

Functional dependency states the relationship between two sets of attributes where a value of a set of attributes is dependent on the other set of attributes.

It is a relationship that typically exists between two attributes such that with the help of one attribute we can get the values of another [attribute](https://www.geeksforgeeks.org/understanding-data-attribute-types-qualitative-and-quantitative/). The attribute that is used for finding the values of other attributes is called the primary key attribute.

**Why is Functional Dependency Used?**

* To maintain [Data Integrity](https://www.geeksforgeeks.org/data-integrity-testing-in-software-testing/)
* Easy to maintain
* Efficient Data Storage
* Improved Data Redundancy

**Example**

We have a table with student details such as roll number, name, and city.

| **roll\_no** | **name** | **city** |
| --- | --- | --- |
| 1 | Yash | Delhi |
| 2 | Kartik | Mumbai |
| 3 | Aditya | Delhi |
| 4 | Kartik | Pune |

* Here roll\_no is only unique attribute. So the primary key for the given table will be **roll\_no**. Other attributes such as name and city are dependent on the roll\_no i.e. on the basis of roll\_no we can get student's name and its city.But we can not get roll\_no of student based on it's name or city as it will create ambiguity.
* For example, if we take name as kartik there will be 2 records with the name kartik which will result into ambiguity. Also if we take city as Delhi there will be 2 records i.e. roll\_no 1 and roll\_no 3.
* So here we can say that name and city are functionally dependent on roll\_no.

**Types of Dependencies**

* Partial Dependency
* Full Dependency
* Transitive Dependency

**Partial Dependency**

* If the value of a non-primary attribute can be defined using part of the primary key then it is called a **partial dependency.** Partial dependency occurs when primary key is formed using more than one attribute.This type of key also called as[composite key](https://www.geeksforgeeks.org/composite-key-in-sql/).
* In below given example, the primary key is formed using **roll\_no + sub\_id**which can also be called as composite key.When composite key is present and one of the non-primary attribute can is dependent on part of the primary key instead of whole primary key then it is called as Partial Dependency.

**Example**

Let's take an example, we have a table where we have columns of student roll number, subject ID, sub name, and marks obtained.

**Table**

| **roll\_no** | **sub\_id** | **sub\_name** | **sub\_mark** |
| --- | --- | --- | --- |
| 1 | 121 | Science | 80 |
| 1 | 131 | Math | 65 |
| 2 | 131 | Math | 95 |
| 2 | 141 | English | 75 |

* Here primary key will be **roll\_no+ sub\_id** because multiple roll\_no can have the same sub\_id and the same roll\_no can have multiple sub\_id.In the given example, roll\_no 1 has two sub\_id i.e. 121 and 131 where as sub\_id 131 has two roll\_no 1 and 2.So here primary key will be **roll\_no + sub\_id**.
* But we do have another column of sub\_name and the value of sub\_name can be easily obtained by only sub\_id which is part of the primary key.For example, sub\_id = 131 will have the sub\_name = ‘math’ here we required only partial primary key i.e. sub\_id.
* This type of functional Dependency is known as **Partial Dependency.**

**Full Dependency**

* If all attributes of the primary key are required for the identifying value of a non-primary attribute then it is known as **Full Dependency.**
* When all non-primary attribute are dependent on whole primary key and they cannot be get defined using only partial part of primary key then it is called as **Full Dependency**.
* If the dependency is **non-partial dependency** then it can be called as Full Deppendency also.

**Example**

Let's take an example, we have a table where we have columns of student roll number, subject ID, and marks obtained.

**Table**

| **roll\_no** | **sub\_id** | **marks** |
| --- | --- | --- |
| 1 | 121 | 80 |
| 1 | 131 | 65 |
| 2 | 131 | 95 |
| 2 | 141 | 75 |

* Here the primary key is **roll\_no+ sub\_id**. If we want a mark of any student, we require both roll\_no and sub\_id. We cannot obtain marks based on one attribute from the primary key.
* If we want to know the marks of sub\_id=131 there will be two records and ambiguity will be created. If we take roll\_id=1 there will be two records with the same roll number and ambiguity will be created here.This ambiguity will be solved using full attributes of the primary key i.e. roll\_no + sub\_id. So we required the full attributes of the primary key.
* This type of functional Dependency is known as **Full Dependency**.

**Transitive Dependency**

* If the value of a non-primary attribute can be defined using another non-primary attribute then it is called a transitive dependency**.**
* When any attribute does not require primary key and can easily get value using another non-primary attribute then it is called as **Transitive Dependency**.

**Example**

Let's take an example, we have a table where we have columns of student roll number, name, city where student live, and zip-code of city .

**Table**

| **roll\_no** | **name** | **city** | **zip-code** |
| --- | --- | --- | --- |
| 1 | abc | pune | 411044 |
| 2 | jkl | mumbai | 400001 |
| 3 | uvw | pune | 411044 |
| 4 | xyz | delhi | 110001 |

* Here the primary key is **roll\_no** but we can identify the city using zip-code where city and zip-code both are the primary key
* So here **roll\_no → city** and **city→zip-code** eventually resulting into **roll\_no →zip-code**. so we can find a non-primary attribute using another non-primary attribute.For example, roll-no = 1 has city=pune and city=pune will have zip-code=411044.So wherever city is pune , zip-code will be 411044
* This type of functional Dependency is known as **Transitive Dependency.**