Normalization in DBMS

# Normalization in DBMS is a technique using which you can organize the data in the database tables so that:

* There is less repetition of data,
* A large set of data is structured into a bunch of smaller tables,
* and the tables have a proper relationship between them.

DBMS Normalization is a systematic approach to **decompose (break down) tables** to eliminate data redundancy(repetition) and undesirable characteristics like Insertion anomaly in DBMS, Update anomaly in DBMS, and Delete anomaly in DBMS.

It is a **multi-step process** that puts data into tabular form, removes duplicate data, and set up the relationship between tables.

Why we need Normalization in DBMS?

Normalization is required for,

* Eliminating redundant(useless) data, therefore handling **data integrity**, because if data is repeated it increases the chances of inconsistent data.
* Normalization helps in keeping **data consistent** by storing the data in one table and referencing it everywhere else.
* Storage optimization although that is not an issue these days because Database storage is cheap.
* Breaking down large tables into smaller tables with relationships, so it makes the database structure more scalable and adaptable.
* Ensuring data dependencies make sense i.e. data is logically stored.
* Problems without Normalization in DBMS
* If a table is not properly normalized and has data redundancy(repetition) then it will not only **eat up extra memory space** but will also make it difficult for you to handle and update the data in the database, without losing data.
* Insertion, Updation, and Deletion Anomalies are very frequent if the database is not normalized.
* To understand these anomalies let us take an example of a **Student** table.

A black and white table with white text

AI-generated content may be incorrect.

In the table above, we have *data for four Computer Sci. students*.

As we can see, data for the fields **branch**, **hod**(Head of Department), and **office\_tel** are repeated for the students who are in the same branch in the college, this is **Data Redundancy**.

1. Insertion Anomaly in DBMS

* Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as **NULL**.
* Also, if we have to insert data for 100 students of the same branch, then the branch information will be repeated for all those 100 students.
* These scenarios are nothing but **Insertion anomalies**.
* If you have to repeat the same data in every row of data, it's better to **keep the data separately** and **reference that data** in each row.
* So in the above table, we can keep the branch information separately, and just use the **branch\_id** in the student table, where **branch\_id** can be used to get the branch information.

2. Updation Anomaly in DBMS

* What if Mr. X leaves the college? or Mr. X is no longer the HOD of the computer science department? In that case, all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency.
* This is an Updation anomaly because you need to update all the records in your table just because one piece of information got changed.

3. Deletion Anomaly in DBMS

* In our **Student** table, two different pieces of information are kept together, the **Student information** and the **Branch information**.
* So if only a single student is enrolled in a branch, and that student leaves the college, or for some reason, the entry for the student is deleted, we will lose the branch information too.
* So never in DBMS, we should keep two different entities together, which in the above example is Student and branch,

Primary Key and Non-key attributes

Before we move on to learn different Normal Forms in DBMS, let's first understand what is a primary key and what are non-key attributes.

A diagram of a student

AI-generated content may be incorrect.

As you can see in the table above, the **student\_id** column is a **primary key** because using the **student\_id** value we can uniquely identify each row of data, hence the *remaining columns* then become the **non-key attributes**.

Types of DBMS Normal forms

Normalization rules are divided into the following normal forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Fourth Normal Form
6. Fifth Normal Form

Let's cover all the Database Normal forms one by one with some basic examples to help you understand the DBMS normal forms.

1. First Normal Form (1NF)

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(**atomic**) valued attributes/columns.
2. Values stored in a column should be of the same domain.
3. All the columns in a table should have unique names.
4. And the order in which data is stored should not matter.

Let's see an example.

If we have an **Employee** table in which we store the *employee information* along with the *employee skillset*, the table will look like this:

A screenshot of a computer

AI-generated content may be incorrect.

The above table has 4 columns:

* All the columns have different names.
* All the columns hold values of the same type like **emp\_name** has all the names, **emp\_mobile** has all the contact numbers, etc.
* The order in which we save data doesn't matter
* But the **emp\_skills** column holds *multiple comma-separated values*, while as per the First Normal form, each column should have a single value.

Hence the above table fails to pass the First Normal form.

So how do you fix the above table? There are two ways to do this:

1. Remove the **emp\_skills** column from the **Employee** table and keep it in some other table.
2. Or add multiple rows for the employee and each row is linked with one skill.

1. Create Separate tables for Employee and Employee Skills

So the **Employee** table will look like this,

A screenshot of a computer

AI-generated content may be incorrect.

And the new **Employee\_Skill** table:

A screenshot of a computer

AI-generated content may be incorrect.

2. Add Multiple rows for Multiple skills

You can also *simply add multiple rows* to add multiple skills. This will lead to repetition of the data, but that can be handled as you further Normalize your data using the Second Normal form and the Third Normal form.

A white table with black text

AI-generated content may be incorrect.

2. Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have **Partial Dependency**.

What is Partial Dependency?

When a table has a primary key that is made up of two or more columns, then all the columns(not included in the primary key) in that table should depend on the entire primary key and not on a part of it. If any column(which is not in the primary key) depends on a part of the primary key then we say we have Partial dependency in the table.

Confused? Let's take an example.

If we have two tables Students and Subjects, to store student information and information related to subjects.

**Student** table:

A screenshot of a computer

AI-generated content may be incorrect.

**Subject** Table:

A white box with black text

AI-generated content may be incorrect.

And we have another table **Score** to store the marks scored by students in any subject like this,

A screenshot of a computer

AI-generated content may be incorrect.

Now in the above table, the primary key is **student\_id + subject\_id**, because both these information are required to select any row of data.

But in the **Score** table, we have a column **teacher\_name**, which depends on the subject information or just the **subject\_id**, so we *should not keep* that information in the **Score** table.

The column **teacher\_name** should be in the **Subjects** table. And then the entire system will be Normalized as per the Second Normal Form.

Updated **Subject** table:

A screenshot of a computer

AI-generated content may be incorrect.

Updated **Score** table:

A screenshot of a computer

AI-generated content may be incorrect.

3. Third Normal Form (3NF)

A table is said to be in the Third Normal Form when,

1. It satisfies the First Normal Form and the Second Normal form.
2. And, it doesn't have Transitive Dependency.

What is Transitive Dependency?

In a table we have some column that acts as the primary key and other columns depends on this column. But what if a column that is not the primary key depends on another column that is also not a primary key or part of it? Then we have Transitive dependency in our table.

Let's take an example. We had the **Score** table in the Second Normal Form above. If we have to store some extra information in it, like,

1. **exam\_type**
2. **total\_marks**

To store the type of exam and the total marks in the exam so that we can later calculate the percentage of marks scored by each student.

The **Score** table will look like this,

A white grid with black text

AI-generated content may be incorrect.

* In the table above, the column **exam\_type** depends on both **student\_id** and **subject\_id**, because,
  + a student can be in the CSE branch or the Mechanical branch,
  + and based on that they may have different exam types for different subjects.
  + The CSE students may have both Practical and Theory for Compiler Design,
  + whereas Mechanical branch students may only have Theory exams for Compiler Design.
* But the column **total\_marks** just depends on the **exam\_type** column. And the **exam\_type** column is not a part of the primary key. Because the primary key is **student\_id + subject\_id**, hence we have a Transitive dependency here.

How to Transitive Dependency?

You can create a separate table for **ExamType** and use it in the **Score** table.

New **ExamType** table,

A screenshot of a computer

AI-generated content may be incorrect.

We have created a new table **ExamType** and we have added more related information in it like **duration**(duration of exam in mins.), and now we can use the **exam\_type\_id** in the **Score** table.

4. Boyce-Codd Normal Form (BCNF)

* **Boyce and Codd Normal Form** is a higher version of the Third Normal Form.
* This form deals with a certain type of anomaly that is not handled by 3NF.
* A 3NF table that does not have **multiple overlapping candidate keys** is said to be in BCNF.
* For a table to be in BCNF, the following conditions must be satisfied:
  + R must be in the 3rd Normal Form
  + and, for each functional dependency ( X → Y ), X should be a Super Key.

5. Fourth Normal Form (4NF)

A table is said to be in the Fourth Normal Form when,

1. It is in the Boyce-Codd Normal Form.
2. And, it doesn't have Multi-Valued Dependency.

5. Fifth Normal Form (5NF)

* The fifth normal form is also called the **PJNF** - **Project-Join Normal Form**
* It is the most advanced level of Database Normalization.
* Using Fifth Normal Form you can fix **Join dependency** and reduce data redundancy.
* It also helps in fixing **Update anomalies** in DBMS design.