**Normalization**

[**https://www.studytonight.com/dbms/database-normalization.php**](#_top)

A methodical technique called "DBMS normalization" is used to dissect tables in order to get rid of repetitive or redundant data as well as undesired features including "insertion anomalies," "update anomalies," and "deletion anomalies."   
Data is tabulated, duplicate data is eliminated, and table relationships are established through a multi-step process.

**Why we need Normalization in DBMS?**

Normalization is essential for :

* Data integrity is handled by removing duplicate or useless data since inconsistent data is more likely to occur when data is repeated.
* By putting all of the data in one database and referencing it everywhere, normalization aids in maintaining consistency in the data.
* Although database storage is so inexpensive these days, storage optimization is not a problem.
* Breaking down big tables into more manageable, related smaller tables, which increases the scalability and flexibility of the database structure.
* Ensuring that data is logically stored and that data dependencies make sense.

**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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **rollno** | **name** | branch | hod | office\_tel |
| 401 | Akon | CSE | Mr. X | 53337 |
| 402 | Bkon | CSE | Mr. X | 53337 |
| 403 | Ckon | CSE | Mr. X | 53337 |
| 404 | Dkon | CSE | Mr. X | 53337 |

We have data for four Computer Science students in the table above.   
  
Data redundancy occurs when data for the fields branch, hod (Head of Department), and office\_tel are repeated for students enrolled in the same college branch.

1.**Insertion Anomaly in DBMS**

* Assume that for a fresh admission, student data cannot be entered until and unless the student chooses a branch; otherwise, we will have to set the branch information to NULL.
* Furthermore, the branch information will be repeated for each of the 100 students if data needs to be entered for that same 100 students.
* All these situations are considered oddities related to Insertion.
* It is preferable to keep the data distinct and make reference to it in each row if you must repeat the same information in every data row.
* Therefore, we may keep the branch information separate in the above database and just use the branch\_id to obtain the branch information in the student table.

2. **Updation Anomaly in DBMS**

* How might Mr. X quit the college? or Mr. X isn't the department head of computer science anymore? Then, all of the student records would need to be updated, and data inconsistencies would result if we unintentionally missed any entries.
* Because you must update every entry in your database simply because one piece of information changed, this is an oddity involving updates.

3. **Deletion Anomaly in DBMS**

* The Student information and the Branch information are two distinct sets of data that are combined in our Student database.
* Therefore, we will also lose the branch information if a single student enrolled in a branch quits the college or if the student's entry is removed for whatever reason.
* Therefore, in DBMS, we should never retain two distinct things together, like the student and branch in the example above.

**Primary Key and Non-key attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| Student\_id | Student\_name | Mob\_num | Gender |
| 1 | John | 9854656656 | Male |
| 2 | Ron | 8200696325 | Female |
| 3 | Ron | 7889532306 | Female |

Primary Key Non-key attributes

The student\_id column, as shown in the above table, is a primary key since it allows us to uniquely identify each row of data. The remaining columns, on the other hand, serve as non-key properties.

**Types of DBMS Normal forms**

1. **First Normal Form (1NF)**
2. **Second Normal Form(2NF)**
3. **Third Normal Form(3NF)**
4. **BCNF (3.5 NF/BCNF)**
5. **Fourth Normal Form (4NF)**
6. **Fifth Normal Form (5NF)**

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)**

A table must adhere to the following four conditions in order to be in the First Normal Form:

* It can only include single-valued attributes or columns.
* A column's values ought to belong to the same domain.
* A table's columns ought to have distinct names
* And the order in which the data is kept should be irrelevant.

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:

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_mobile | emp\_skills |
| 1 | John Tick | 9999957773 | Python, JavaScript |
| 2 | Darth Trader | 8888853337 | HTML, CSS, JavaScript |
| 3 | Rony Shark | 7777720008 | Java, Linux, C++ |

The above table has 4 columns:

* Each column has a unique name.
* The values in each column are of the same kind; for example, all of the names are in emp\_name, all of the contact numbers are in emp\_mobile, etc.
* It doesn't matter what order we save our data in.
* However, according to the First Normal form, each column should only have one value; however, the emp skills field contains many comma-separated entries.

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.
3. **Create Separate tables for Employee and Employee Skills**

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

|  |  |  |
| --- | --- | --- |
| emp\_id | emp\_name | emp\_mobile |
| 1 | John Tick | 9999957773 |
| 2 | Darth Trader | 8888853337 |
| 3 | Rony Shark | 7777720008 |

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

|  |  |
| --- | --- |
| emp\_id | emp\_skill |
| 1 | Python |
| 1 | JavaScript |
| 2 | HTML |
| 2 | CSS |
| 2 | JavaScript |
| 3 | Java |
| 3 | Linux |
| 3 | C++ |

**2. Add Multiple rows for Multiple skills**

To add more skills, you may also just add more rows. Repeating the data will result from this, but you can deal with that as you go Use the Second Normal form and the Third Normal form to normalize your data.

|  |  |  |  |
| --- | --- | --- | --- |
| emp\_id | emp\_name | emp\_mobile | emp\_skill |
| 1 | John Tick | 9999957773 | Python |
| 1 | John Tick | 9999957773 | JavaScript |
| 2 | Darth Trader | 8888853337 | HTML |
| 2 | Darth Trader | 8888853337 | CSS |
| 2 | Darth Trader | 8888853337 | JavaScript |
| 3 | Rony Shark | 7777720008 | Java |
| 3 | Rony Shark | 7777720008 | Linux |
| 3 | Rony Shark | 7777720008 | C++ |

**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**.

To further grasp partial reliance and the Second Normal Form, let's look at an example.

**What is Partial Dependency?**

When a table contains a primary key that consists of two or more columns, all of the columns that aren't part of the primary key should rely on the complete primary key rather than just a portion of it. We refer to a partial dependency in the table when any column (that is, one that is not part of the primary key) depends on any portion of the primary key.

Confused? Let's take an example.

To hold information about students and subjects, we can create two tables: Students and Subjects.

**Student** table:

|  |  |  |
| --- | --- | --- |
| student\_id | student\_name | branch |
| 1 | Akon | CSE |
| 2 | Bkon | Mechanical |

**Subject** Table:

|  |  |
| --- | --- |
| subject\_id | subject\_name |
| 1 | C Language |
| 2 | DSA |
| 3 | Operating System |

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

|  |  |  |  |
| --- | --- | --- | --- |
| Student\_id | Subject\_id | marks | Teacher\_name |
| 1 | 1 | 70 | Miss. C |
| 1 | 2 | 82 | Mr. D |
| 2 | 1 | 65 | Mr. Op |

Since both **Student\_id** +**Subject \_id** are necessary to select any row of data in the preceding table, they are the main keys.   
However, we shouldn't retain that data in the **Score** table because it depends on the subject information or only the **Subject\_id** in the **Teacher\_name** column.   
The **Subjects** table should contain the column **Teacher\_nam**e. After that, the Second Normal Form will be applied to the entire system to normalize it.   
Updated **Subject** Table:

|  |  |  |
| --- | --- | --- |
| Subject\_id | subject\_name | teacher\_name |
| 1 | C Language | Miss. C |
| 2 | DSA | Mr. D |
| 3 | Operating System | Mr. Op |

Updated **Score** table:

|  |  |  |
| --- | --- | --- |
| student\_id | subject\_id | marks |
| 1 | 1 | 70 |
| 1 | 2 | 82 |
| 2 | 1 | 65 |

**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?

One column serves as the primary key in a table, and all other columns are dependent upon it. However, what happens if a non-primary key column depends on a column that is either not a primary key at all or only a subset of it? Then, our table contains transitive dependency.

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,

|  |  |  |  |
| --- | --- | --- | --- |
| student\_id | subject\_id | marks | exam\_type |
| 1 | 1 | 70 | Theory |
| 1 | 2 | 82 | Theory |
| 2 | 1 | 42 | Practical |

1.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.

2.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.

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.

**3.Boyce-Codd Normal Form (3.5NF/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.

**4.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.