**Normalization in DBMS**

* Normalization in DBMS is a technique using which we can organize the data in the database tables so that there is less repetition of data.
* In the process of the normalization A large set of data is structured into a bunch of smaller tables and the tables have a proper relationship between them.

**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
7. **FIRST NORMAL FORM :**

Rules for First Normal Form

The first normal form expects you to follow a few simple rules while designing your database, and they are:

Rule 1: Single Valued Attributes

Each column of your table should be single valued which means they should not contain multiple values.

Rule 2: Attribute Domain should not change

This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.

**For example:** If we have a column **dob** to save **date of births** of a set of people, then you cannot or you must not save 'names' of some of them in that column along with 'date of birth' of others in that column. It should hold only **'date of birth'** for all the records/rows.

**Example**:

The following table contains two phone number values for a single attribute.

|  |  |  |
| --- | --- | --- |
| EMP\_ID | STUDENT\_NAME | PHONE\_NUMBER |
| 1  2 | Tanmay  Rohit | 08830307982  08830307982,08788950680 |

So to convert it into 1NF, we decompose the table as the following -

|  |  |  |
| --- | --- | --- |
| EMP\_ID | STUDENT\_NAME | PHONE\_NUMBER |
| 1  2  2 | Tanmay  Rohit  Rohit | 08830307982  08830307982  08788950680 |

Here, we can notice data repetition, but 1NF doesn’t care about it.

1. **SECOND NORMAL FORM :**

For a table to be in the Second Normal Form, it must satisfy two conditions:

1. The table should be in the First Normal Form.
2. There should be no Partial Dependency.

Q. What is Dependency?

Let's take an example of a **Student** table with columns student\_id, name, reg\_no(registration number), branch and address(student's home address).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STUDENT\_ID** | **NAME** | **REG\_NO** | **BRANCH** | **ADDRESS** |

In this table, student\_id is the primary key and will be unique for every row, hence we can use student\_id to fetch any row of data from this table

Even for a case, where student names are same, if we know the student\_id we can easily fetch the correct record.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STUDENT\_ID** | **NAME** | **REG\_NO** | **BRANCH** | **ADDRESS** |
| 10 | Rohan | 07-WY | CSE | Nagpur |
| 11 | Aniket | 08-WY | IT | Pune |

Hence we can say a **Primary Key** for a table is the column or a group of columns(composite key) which can uniquely identify each record in the table.

I can ask from branch name of student with student\_id **10**, and I can get it. Similarly, if I ask for name of student with student\_id **10** or **11**, I will get it. So all I need is student\_id and every other column **depends** on it, or can be fetched using it.

This is **Dependency** and we also call it **Functional Dependency**.

Q. What is Partial Dependency?

Now that we know what dependency is, we are in a better state to understand what partial dependency is.

For a simple table like Student, a single column like student\_id can uniquely identfy all the records in a table.

But this is not true all the time. So now let's extend our example to see if more than 1 column together can act as a primary key.

Let's create another table for **Subject**, which will have subject\_id and subject\_name fields and subject\_id will be the primary key.

|  |  |
| --- | --- |
| **SUBJECT\_ID** | **SUBJECT\_NAME** |
| 1 | Java |
| 2 | C++ |
| 3 | Php |

Now we have a **Student** table with student information and another table **Subject** for storing subject information.

Let's create another table **Score**, to store the **marks** obtained by students in the respective subjects. We will also be saving **name of the teacher** who teaches that subject along with marks.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SCORE\_ID** | **STUDENT\_ID** | **SUBJECT\_ID** | **MARKS** | **TEACHER** |
| 1 | 10 | 1 | 70 | Java Teacher |
| 2 | 10 | 2 | 75 | C++ Teacher |
| 3 | 11 | 1 | 80 | Java Teacher |

In the score table we are saving the **student\_id** to know which student's marks are these and **subject\_id** to know for which subject the marks are for.

Together, student\_id + subject\_id forms a **Candidate Key**(learn about [Database Keys](https://www.studytonight.com/dbms/database-key.php)) for this table, which can be the **Primary key**.

Confused, How this combination can be a primary key?

See, if I ask you to get me marks of student with student\_id 10, can you get it from this table? No, because you don't know for which subject. And if I give you subject\_id, you would not know for which student. Hence we need student\_id + subject\_id to uniquely identify any row.

Q. But where is Partial Dependency?

Now if you look at the **Score** table, we have a column names teacher which is only dependent on the subject, for Java it's Java Teacher and for C++ it's C++ Teacher & so on.

Now as we just discussed that the primary key for this table is a composition of two columns which is student\_id & subject\_id but the teacher's name only depends on subject, hence the subject\_id, and has nothing to do with student\_id.

This is **Partial Dependency**, where an attribute in a table depends on only a part of the primary key and not on the whole key.

How to remove Partial Dependency?

There can be many different solutions for this, but out objective is to remove teacher's name from Score table.

The simplest solution is to remove columns teacher from Score table and add it to the Subject table. Hence, the Subject table will become:

|  |  |  |
| --- | --- | --- |
| **SUBJECT\_ID** | **SUBJECT\_NAME** | **TEACHER** |
| 1 | Java | Java Teacher |
| 2 | C++ | C++ Teacher |
| 3 | Php | Php Teacher |

And our Score table is now in the second normal form, with no partial dependency.

|  |  |  |  |
| --- | --- | --- | --- |
| **SCORE\_ID** | **STUDENT\_ID** | **SUBJECT\_ID** | **MARKS** |
| 1 | 10 | 1 | 70 |
| 2 | 10 | 2 | 75 |
| 3 | 11 | 1 | 80 |

Quick Recap

1. For a table to be in the Second Normal form, it should be in the First Normal form and it should not have Partial Dependency.
2. Partial Dependency exists, when for a composite primary key, any attribute in the table depends only on a part of the primary key and not on the complete primary key.
3. To remove Partial dependency, we can divide the table, remove the attribute which is causing partial dependency, and move it to some other table where it fits in well.

**Example :**

Consider the following table. Its primary key is {StudentId, ProjectId}.

The Functional dependencies given are -

StudentId → StudentName

ProjectId → ProjectName

|  |  |  |  |
| --- | --- | --- | --- |
| STUDENT | PROJECT\_ID | STUDENT\_NAME | PROJECT\_NAME |
| 1  2  3  4 | P2  P1  P7  P3 | ABHINAV  TANMAY  ROHIT  ROHAN | IOT  CLOUD  IOT  CLOUD |

As it represents partial dependency, we decompose the table as follows –

|  |  |  |
| --- | --- | --- |
| STUDENT | PROJECT\_ID | STUDENT\_NAME |
| 1  2  3  4 | P2  P1  P7  P3 | ABHINAV  TANMAY  ROHIT  ROHAN |

|  |  |
| --- | --- |
| PROJECT\_ID | PROJECT\_NAME |
| P2  P1  P7  P3 | IOT  CLOUD  IOT  CLOUD |

1. **THIRD NORMAL FORM :**

In 3NF, the given relation should be 2NF, and no transitivity dependency should exist, i.e., non-prime attributes should not determine non-prime attributes.

**Requirements for Third Normal Form**

For a table to be in the third normal form,

1. It should be in the Second Normal form.
2. And it should not have Transitive Dependency.

Q. What is Transitive Dependency?

With exam\_name and total\_marks added to our Score table, it saves more data now. Primary key for our Score table is a composite key, which means it's made up of two attributes or columns → **student\_id + subject\_id**.

Our new column exam\_name depends on both student and subject. For example, a mechanical engineering student will have Workshop exam but a computer science student won't. And for some subjects you have Prctical exams and for some you don't. So we can say that exam\_name is dependent on both student\_id and subject\_id.

And what about our second new column total\_marks? Does it depend on our Score table's primary key?

Well, the column total\_marks depends on exam\_name as with exam type the total score changes. For example, practicals are of less marks while theory exams are of more marks.

But, exam\_name is just another column in the score table. It is not a primary key or even a part of the primary key, and total\_marks depends on it.

This is **Transitive Dependency**. When a non-prime attribute depends on other non-prime attributes rather than depending upon the prime attributes or primary key.

Advantage of removing Transitive Dependency

The advantage of removing transitive dependency is,

* Amount of data duplication is reduced.
* Data integrity achieved.

Q. How to remove Transitive Dependency?

Again the solution is very simple. Take out the columns exam\_name and total\_marks from Score table and put them in an **Exam** table and use the exam\_id wherever required.

Score Table: In 3rd Normal Form

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **SCORE\_ID** | **STUDENT\_ID** | **SUBJECT\_ID** | **MARKS** | **EXAM\_ID** |

The new Exam table

|  |  |  |
| --- | --- | --- |
| **EXAM\_ID** | **EXAM\_NAME** | **TOTAL\_MARKS** |
| 1 | Workshop | 200 |
| 2 | Mains | 70 |
| 3 | Practicals | 30 |

Advantage of removing Transitive Dependency

The advantage of removing transitive dependency is,

* Amount of data duplication is reduced.
* Data integrity achieved.

1. **Boyce-Codd Normal Form (BCNF):**

Boyce-Codd Normal Form or BCNF is an extension to the [third normal form](https://www.studytonight.com/dbms/third-normal-form.php), and is also known as 3.5 Normal Form.

Rules for BCNF

For a table to satisfy the Boyce-Codd Normal Form, it should satisfy the following two conditions:

1. It should be in the **Third Normal Form**.
2. And, for any dependency A → B, A should be a **super key**.

The second point sounds a bit tricky, right? In simple words, it means, that for a dependency A → B, A cannot be a **non-prime attribute**, if B is a **prime attribute**.

**Example**

Below we have a college enrolment table with columns student\_id, subject and professor.

|  |  |  |
| --- | --- | --- |
| **STUDENT\_ID** | **SUBJECT** | **PROFESSOR** |
| 101 | Java | P.Java |
| 101 | C++ | P.Cpp |
| 102 | Java | P.Java2 |
| 103 | C# | P.Chash |
| 104 | Java | P.Java |

As you can see, we have also added some sample data to the table.

In the table above:

* One student can enrol for multiple subjects. For example, student with **student\_id** 101, has opted for subjects - Java & C++
* For each subject, a professor is assigned to the student.
* And, there can be multiple professors teaching one subject like we have for Java.

Q. What do you think should be the **Primary Key**?

Well, in the table above student\_id, subject together form the primary key, because using student\_id and subject, we can find all the columns of the table.

One more important point to note here is, one professor teaches only one subject, but one subject may have two different professors.

Hence, there is a dependency between subject and professor here, where subject depends on the professor name.

This table satisfies the **1st Normal form** because all the values are atomic, column names are unique and all the values stored in a particular column are of same domain.

This table also satisfies the **2nd Normal Form** as their is no **Partial Dependency**.

And, there is no **Transitive Dependency**, hence the table also satisfies the **3rd Normal Form**.

But this table is not in **Boyce-Codd Normal Form**.

Q. Why this table is not in BCNF?

ANS - In the table above, student\_id, subject form primary key, which means subject column is a **prime attribute**.

But, there is one more dependency, professor → subject.

And while subject is a prime attribute, professor is a **non-prime attribute**, which is not allowed by BCNF.

Q. How to satisfy BCNF?

ANS - To make this relation(table) satisfy BCNF, we will decompose this table into two tables, **student** table and **professor** table.

Below we have the structure for both the tables.

**Student Table**

|  |  |
| --- | --- |
| **STUDENT\_ID** | **P\_ID** |
| 101 | 1 |
| 101 | 2 |
| and so on... | |

and, **PROFESSOR TABLE**

|  |  |  |
| --- | --- | --- |
| **P\_ID** | **PROFESSOR** | **SUBJECT** |
| 1 | P.Java | Java |
| 2 | P.Cpp | C++ |
| and so on... | | |

And now, this relation satisfy Boyce-Codd Normal Form

**Example 2:**

* Given the following table. Its candidate keys are {Student, Teacher} and {Student, Subject}.
* The Functional dependencies given are -
* {Student, Teacher} → Subject
* {Student, Subject} → Teacher
* Teacher → Subject

|  |  |  |
| --- | --- | --- |
| STUDENT | SUBJECT | TEACHER |
| ABHINAV  TANMAY  ROHIT  ROHAN | PHYSICS  ENGLISH  PHYSICS  ENGLISH | SAGAR  SAGAR\_R  SAGAR  SAGAR\_R |

As this table is not in BCNF form.

So we decompose it into the following tables:

|  |  |
| --- | --- |
| TEACHER | SUBJECT |
| SAGAR  SAGAR\_R  SAGAR  SAGAR\_R | PHYSICS  ENGLISH  PHYSICS  ENGLISH |

|  |  |
| --- | --- |
| STUDENT | TEACHER |
| ABHINAV  TANMAY  ROHIT  ROHAN | SAGAR  SAGAR\_R  SAGAR  SAGAR\_R |

Here Teacher is mentioned in both tables to set up a relationship between them.

1. **Fourth Normal Form (4NF):**

For any relation to be in 4NF, it should have no multi-valued dependencies and is in Boyce Codd Normal Form. It simplifies the database by eliminating the non-trivial multi-valued dependencies besides those including the candidate key.

**Rules for 4th Normal Form**

For a table to satisfy the Fourth Normal Form, it should satisfy the following two conditions:

1. It should be in the **Boyce-Codd Normal Form**.
2. And, the table should not have any **Multi-valued Dependency**.

Let's try to understand what multi-valued dependency is in the next section.

Q. What is Multi-valued Dependency?

A table is said to have multi-valued dependency, if the following conditions are true,

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for a relation R(A,B,C), if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

If all these conditions are true for any relation(table), it is said to have multi-valued dependency.

Example

Below we have a college enrolment table with columns s\_id, course and hobby.

|  |  |  |
| --- | --- | --- |
| **S\_ID** | **COURSE** | **HOBBY** |
| 1 | Science | Cricket |
| 1 | Maths | Hockey |
| 2 | C+ | Cricket |
| 2 | Physics | Hockey |

As you can see in the table above, student with s\_id **1** has opted for two courses, **Science** and **Maths**, and has two hobbies, **Cricket** and **Hockey**.

You must be thinking what problem this can lead to, right?

Well the two records for student with s\_id **1**, will give rise to two more records, as shown below, because for one student, two hobbies exists, hence along with both the courses, these hobbies should be specified.

|  |  |  |
| --- | --- | --- |
| **S\_ID** | **COURSE** | **HOBBY** |
| 1 | Science | Cricket |
| 1 | Maths | Hockey |
| 1 | Science | Hockey |
| 1 | Maths | Cricket |

And, in the table above, there is no relationship between the columns course and hobby. They are independent of each other.

So there is multi-value dependency, which leads to un-necessary repetition of data and other anomalies as well.

Q. How to satisfy 4th Normal Form?

To make the above relation satify the 4th normal form, we can decompose the table into 2 tables.

**CourseOpted Table**

|  |  |
| --- | --- |
| **S\_ID** | **COURSE** |
| 1 | Science |
| 1 | Maths |
| 2 | C+ |
| 2 | Physics |

And, **Hobbies Table**,

|  |  |
| --- | --- |
| **S\_ID** | **HOBBY** |
| 1 | Cricket |
| 1 | Hockey |
| 2 | Cricket |
| 2 | Hockey |

Now this relation satisfies the fourth normal form.

A table can also have functional dependency along with multi-valued dependency. In that case, the functionally dependent columns are moved in a separate table and the multi-valued dependent columns are moved to separate tables.

If you design your database carefully, you can easily avoid these issues.

**Example:**

Consider the following table:

|  |  |  |
| --- | --- | --- |
| STUDENT | SUBJECT | STUDENT\_PHONE\_NUMBER |
| ABHINAV  TANMAY  ROHIT  ROHAN | PHYSICS  ENGLISH  DATABASE  MATHEMATICS | 07709191350  08830307982  08830307982  08308383573 |

In the above table, subject and student phone number are two independent entities, showing no relation between subject and phone number. So to convert it in 4NF, we decompose the table as

|  |  |
| --- | --- |
| STUDENT | STUDENT\_PHONE\_NUMBER |
| ABHINAV  TANMAY  ROHIT  ROHAN | 07709191350  08830307982  08830307982  08308383573 |

|  |  |
| --- | --- |
| STUDENT | SUBJECT |
| ABHINAV  TANMAY  ROHIT  ROHAN | PHYSICS  ENGLISH  DATABASE  MATHEMATICS |

Here Student is mentioned in both tables to set up a relationship between them.

1. **FIFTH NORMAL FORM(5NF):**

The fifth normal form (5NF) is also called the Project-Join Normal Form (PJNF). A relation is in 5NF if it is in 4NF and does not contain any join dependencies that could result in data loss during the join operation.

Representing the pinnacle of normalization, 5NF involves decomposing a table into smaller constituent tables. This decomposition serves to eradicate data redundancy and enhance data integrity. The primary objective of achieving 5NF is to maximize the efficiency of data storage and retrieval while minimizing the chances of anomalies or inconsistencies.

**Example for Fifth Normal Form (5NF)**

Consider a relation involving suppliers, parts, and projects:

Initial Table (SupplierPartsProjects)

|  |  |  |
| --- | --- | --- |
| **SUPPLIER** | **PART** | **PROJECT** |
| S1  S1  S1  S2 | P1  P2  P1  P2 | J1  J1  J2  J2 |

Assume the following constraints for our example:

1. Every part supplied for a project is supplied by all suppliers supplying any part for that project.
2. Every part supplied by a supplier is supplied by that supplier for all projects to which that supplier supplies any part.

Given the above constraints, the following join dependencies exist on the table:

* {Supplier, Part} ⟶ SupplierPartsProjects
* {Supplier, Project} ⟶ SupplierPartsProjects
* {Part, Project} ⟶ SupplierPartsProjects

To decompose the relation into 5NF:

Supplier\_Parts: Supplier\_Projects: Parts\_Projects:

|  |  |
| --- | --- |
| **SUPPLIER** | **PROJECT** |
| S1  S1  S2 | J1  J2  J2 |

|  |  |
| --- | --- |
| **SUPPLIER** | **PROJECT** |
| P1  P2  P1 | J1  J1  J2 |

|  |  |
| --- | --- |
| **SUPPLIER** | **PART** |
| S1  S1  S2 | P1  P2  P2 |

Now, these decomposed tables eliminate the redundancy caused by the specific constraints and join dependencies of the original relation. When you take the natural join of these tables, you will get back the original table.

It's worth noting that reaching 5NF can lead to an increased number of tables, which can complicate queries and database operations. Thus, achieving 5NF should be a conscious decision made based on the specific requirements and constraints of a given application.

COURTESY:-

<https://www.naukri.com/code360/library/3nf-third-normal-form>

<https://www.studytonight.com/dbms/create-query.php>

<https://www.almabetter.com/bytes/articles/normalization-in-dbms>

<https://studyglance.in/dbms/display.php?tno=35&topic=Fifth-Normal-Form-(5NF-or-PJNF)-in-DBMS>