Normalization of Database

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anomalies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Problems Without Normalization

If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anomalies are very frequent if 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

In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields `branch`, `hod`(Head of Department) and `office_tel` is repeated for the students who are in the same branch in the college, this is **Data Redundancy**.

Insertion Anomaly

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 of 100 students of same branch, then the branch information will be repeated for all those 100 students.

These scenarios are nothing but **Insertion anomalies**.

Updation Anomaly

What if Mr. X leaves the college? or is no longer the HOD of 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 Updation anomaly.

Deletion Anomaly

In our **Student** table, two different informations are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

Normalization Rule

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

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, does not matter.

If tables in a database are not even in the 1st Normal Form, it is considered as **bad database design**.

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. We will explain this with help of an example later, let's see the other rules for now.

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 you 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.

Rule 3: Unique name for Attributes/Columns

This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data.

If one or more columns have same name, then the DBMS system will be left confused.

Rule 4: Order doesn't matters

This rule says that the order in which you store the data in your table doesn't matter.

Time for an Example

Although all the rules are self explanatory still let's take an example where we will create a table to store student data which will have student's roll no., their name and the name of subjects they have opted for.

Here is our table, with some sample data added to it.

roll_no	name	subject
101	Akon	OS, CN
103	Ckon	Java
102	Bkon	C, C++

Our table already satisfies 3 rules out of the 4 rules, as all our column names are unique, we have stored data in the order we wanted to and we have not inter-mixed different type of data in columns.

But out of the 3 different students in our table, 2 have opted for more than 1 subject. And we have stored the subject names in a single column. But as per the 1st Normal form each column must contain atomic value.

How to solve this Problem?

It's very simple, because all we have to do is break the values into atomic values.

Here is our updated table and it now satisfies the First Normal Form.

roll_no	name	subject
101	Akon	OS
101	Akon	CN
103	Ckon	Java
102	Bkon	C
102	Bkon	C++

By doing so, although a few values are getting repeated but values for the `subject` column are now atomic for each record/row.

Using the First Normal Form, data redundancy increases, as there will be many columns with same data in multiple rows but each row as a whole will be unique.

Second Normal Form (2NF)

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.

What is **Partial Dependency**? Do not worry about it. First let's understand what is **Dependency** in a table?

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	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat

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**.

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) 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.

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.

Third Normal Form (3NF)

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

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.

Here is the **Third Normal Form** tutorial. But we suggest you to first study about the second normal form and then head over to the third normal form.

In our last tutorial, we learned about the second normal form and even normalized our **Score** table into the 2nd Normal Form.

So let's use the same example, where we have 3 tables, **Student**, **Subject** and **Score**.

Student Table

student_id	**name**	**reg_no**	**branch**	**address**

10	Akon	07-WY	CSE	Kerala
11	Akon	08-WY	IT	Gujarat
12	Bkon	09-WY	IT	Rajasthan

Subject Table

subject_id	subject_name	teacher
1	Java	Java Teacher
2	C++	C++ Teacher
3	Php	Php Teacher

Score Table

score_id	student_id	subject_id	marks
1	10	1	70
2	10	2	75
3	11	1	80

In the Score table, we need to store some more information, which is the exam name and total marks, so let's add 2 more columns to the Score table.

score_id	student_id	subject_id	marks	exam_name	total_marks

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.

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.

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.

Boyce and Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

- R must be in 3rd Normal Form
- and, for each functional dependency (X → Y), X should be a super Key.

To learn about BCNF in detail with a very easy to understand example, head to **Boye-Codd Normal Form**tutorial.

Follow the video above for complete explanation of BCNF. Or, if you want, you can even skip the video and jump to the section below for the complete tutorial.

In our last tutorial, we learned about the third normal form and we also learned how to remove **transitive dependency** from a table, we suggest you to follow the last tutorial before this one.

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**.

Time for an 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.

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**.

Why this table is not in BCNF?

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.

How to satisfy BCNF?

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**

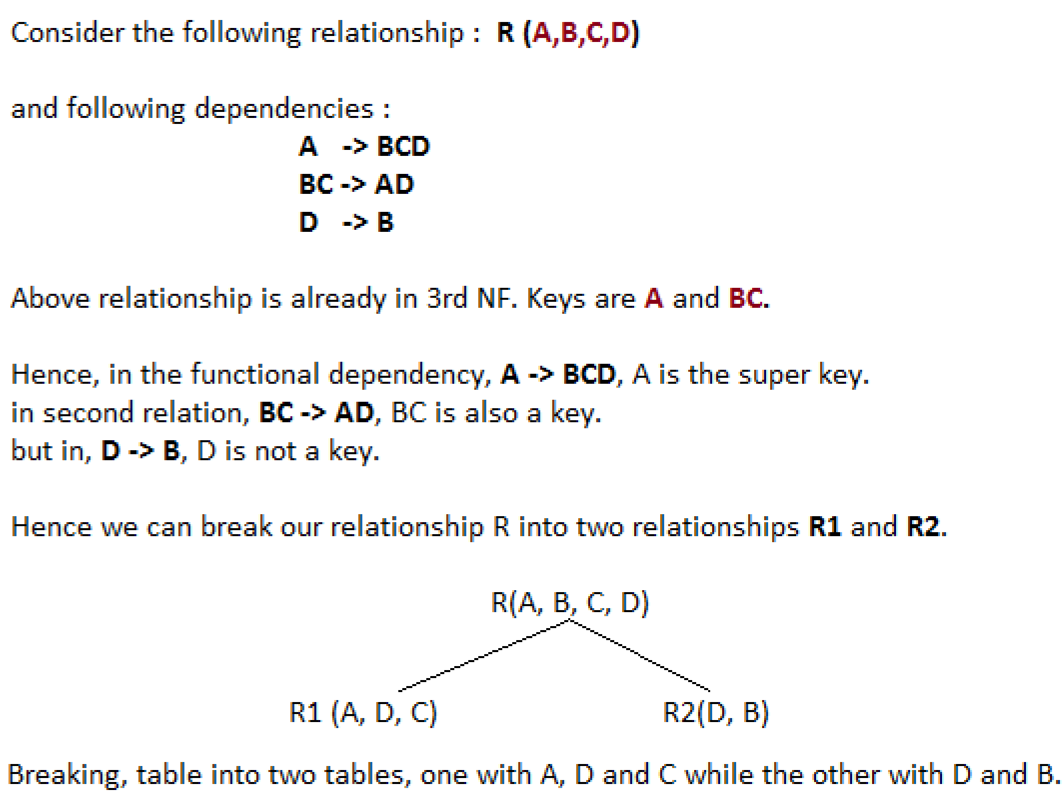
p_id	**professor**	**subject**

1	P.Java	Java
2	P.Cpp	C++
and so on...		

And now, this relation satisfy Boyce-Codd Normal Form. In the next tutorial we will learn about the **Fourth Normal Form**.

A more Generic Explanation

In the picture below, we have tried to explain BCNF in terms of relations.



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.

Here is the **Fourth Normal Form** tutorial. But we suggest you to understand other normal forms before you head over to the fourth normal form.

Follow the video above for complete explanation of 4th Normal Form. Or, if you want, you can even skip the video and jump to the section below for the complete tutorial.

In our last tutorial, we learned about the **boyce-codd normal form**, we suggest you to follow the last tutorial before this one.

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.

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.

Time for an 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	Php	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.

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	Php

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.