UNIT II

NORMALIZATION

Normalization is a process of split the complex data structure into simple data structure. It helps to reduce the data redundancy and increase the data integrity. It also uses to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

CODD'S 12 RULES

Dr Edgar F. Codd, after his extensive research on the Relational Model of database systems, came up with twelve rules of his own, which according to him, a database must obey in order to be regarded as a true relational database. These rules can be applied on any database system that manages stored data using only its relational capabilities. This is a foundation rule, which acts as a base for all the other rules.

Rule 1: Information Rule

The data stored in a database, may it be user data or metadata, must be a value of some table cell. Everything in a database must be stored in a table format.

Rule 2: Guaranteed Access Rule

Every single data element (value) is guaranteed to be accessible logically with a combination of table-name, primary-key (row value), and attribute-name (column value). No other means, such as pointers, can be used to access data.

Rule 3: Systematic Treatment of NULL Values

The NULL values in a database must be given a systematic and uniform treatment. This is a very important rule because a NULL can be interpreted as one the following – data is missing, data is not known, or data is not applicable.

Rule 4: Active Online Catalog

The structure description of the entire database must be stored in an online catalog, known as **data dictionary**, which can be accessed by authorized users. Users can use the same query language to access the catalog which they use to access the database itself.

Rule 5: Comprehensive Data Sub-Language Rule

It states that a relational database management system must have at least one language that can be used to manage all aspects of the database. This language must have a well-defined syntax and be able to support data definition, view definition, data manipulation (both interactively and through programs), integrity constraints, authorization, and transaction boundaries (begin, commit, and rollback).

Rule 6: View Updating Rule

It states that all views of the database that are theoretically updatable must also be updatable by the system. A view is a virtual table that is based on the result of a query, and it provides a way to look at specific data in the database.

In simpler terms, this rule means that if it is theoretically possible to update data through a view, then the database management system must provide a

way to do so. This allows users to update data in the database through views, rather than having to interact directly with the underlying tables

.

Rule 7: High-Level Insert, Update, and Delete Rule

A database must support high-level insertion, updation, and deletion. This must not be limited to a single row, that is, it must also support union, intersection and minus operations to yield sets of data records.

Rule 8: Physical Data Independence

The data stored in a database must be independent of the applications that access the database. Any change in the physical structure of a database must not have any impact on how the data is being accessed by external applications.

Rule 9: Logical Data Independence

The logical data in a database must be independent of its user's view (application). Any change in logical data must not affect the applications using it. For example, if two tables are merged or one is split into two different tables, there should be no impact or change on the user application. This is one of the most difficult rules to apply.

Rule 10: Integrity Independence

A database must be independent of the application that uses it. All its integrity constraints can be independently modified without the need of any change in

the application. This rule makes a database independent of the front-end application and its interface.

Rule 11: Distribution Independence

The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only. This rule has been regarded as the foundation of distributed database systems.

Rule 12: Non-Subversion Rule

It states that if a relational database management system has a low-level language (one that operates on individual records at a time), that low-level language must not be able to subvert or bypass the integrity rules or constraints expressed in the higher-level relational language.

FUNCTIONAL DEPENDENCY

In a **Relational Database Management System (RDBMS)**, a **functional dependency** is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table. The left side of the functional dependency is known as the **determinant** and the right side is known as the **dependent**[1].

For example, consider an employee table with attributes: Emp_Id, Emp_Name, Emp_Address. Here, the Emp_Id attribute can uniquely identify the Emp_Name attribute of the employee table because if we know the Emp_Id, we can tell that employee name associated with it. The functional dependency can be written as: Emp_Id → Emp_Name. We can say that Emp_Name is functionally dependent on Emp_Id

Employee number	Employee Name	Salary	City
1	Dana	50000	San Francisco
2	Francis	38000	London
3	Andrew	25000	Tokyo

In other words, if we have the following table:

We can say that the functional dependencies are: Employee number → Employee Name Employee number → Salary Employee number → City

This means that if we know the value of the Employee number attribute, we can determine the values of the Employee Name, Salary, and City attributes

Multivalued Dependency

Multivalued Dependency occurs when two attributes in a table are independent of each other but both depend on a third attribute. A multivalued dependency consists of at least two attributes that are dependent on a third attribute, that's why it always requires at least three attributes.

Suppose there is a bike manufacturer company that produces two colors (white and black) of each model every year. The table might look like this:

BIKE_MODEL	MANUF_YEAR	COLOR
M2001	2008	White
M2001	2008	Black

BIKE_MODEL	MANUF_YEAR	COLOR
M3001	2013	White
M3001	2013	Black
M4006	2017	White
M4006	2017	Black

In this case, the columns COLOR and MANUF_YEAR are dependent on BIKE_MODEL and independent of each other. These two columns can be called as multivalued dependent on BIKE_MODEL.

The representation of these dependencies is shown below: BIKE_MODEL → → MANUF_YEAR

BIKE_MODEL → → COLOR

This can be read as "BIKE_MODEL multidetermined MANUF_YEAR" and "BIKE_MODEL multidetermined COLOR"

FIRST NORMAL FORM

A database is in first normal form if it satisfies the following conditions:

- Contains only atomic values
- There are no repeating groups

An atomic value is a value that cannot be divided. For example, in the table shown below, the values in the [Color] column in the first row can be divided into "red" and "green", hence [TABLE_PRODUCT] is not in 1NF.

A repeating group means that a table contains two or more columns that are closely related. For example, a table that records data on a book and its author(s) with the following columns: [Book ID], [Author 1], [Author 2], [Author 3] is not in 1NF because [Author 1], [Author 2], and [Author 3] are all repeating the same attribute.

1st Normal Form Example

How do we bring an unnormalized table into first normal form? Consider the following example:

This table is not in first normal form because the [Color] column can contain multiple values. For example, the first row includes values "red" and "green."

To bring this table to first normal form, we split the table into two tables and now we have the resulting tables:

TABLE_PRODUCT_PRICE

Product ID	Price
1	15.99
2	23.99
3	17.50
4	9.99
5	29.99

TABLE_PRODUCT_COLOR

Product ID	Color
1	red
1	green
2	yellow
3	green
4	yellow
4	blue
5	red

Now first normal form is satisfied, as the columns on each table all hold just one value.

SECOND NORMAL FORM

A relation is in second normal form if it is in 1NF and every **non** key attribute is fully functionally dependent on the primary key.

Example

Consider the following example:

TABLE_PURCHASE_DETAIL

Customer ID	Store ID	Purchase Location
1	1	Los Angeles
1	3	San Francisco
2	1	Los Angeles
3	2	New York
4	3	San Francisco

This table has a composite primary key [Customer ID, Store ID]. The non-key attribute is [Purchase Location]. In this case, [Purchase Location] only depends on [Store ID], which is only part of the primary key. Therefore, this table does not satisfy second normal form.

To bring this table to second normal form, we break the table into two tables, and now we have the following:

TABLE_PURCHASE

Customer ID	Store ID
1	1
1	3
2	1
3	2
4	3

TABLE_STORE

Store ID	Purchase Location
1	Los Angeles
2	New York
3	San Francisco

What we have done is to remove the partial functional dependency that we initially had. Now, in the table [TABLE_STORE], the column [Purchase Location] is fully dependent on the primary key of that table, which is [Store ID].

THIRD NORMAL FORM

A database is in third normal form if it satisfies the following conditions:

- It is in second normal form
- There is no transitive functional dependency

By transitive functional dependency, we mean we have the following relationships in the table: A is functionally dependent on B, and B is functionally dependent on C. In this case, C is transitively dependent on A via B.

Example: Consider the following example:

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

In the table able, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type]. Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.

To bring this table to third normal form, we split the table into two as follows:

TABLE_BOOK

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

Now all non-key attributes are fully functional dependent only on the primary key. In [TABLE_BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [TABLE_GENRE], [Genre Type] is only dependent on [Genre ID].

BOYCE-CODD NORMAL FORM (BCNF)

Boyce-Codd Normal Form or BCNF is an extension to the third normal form, 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: It should be in the **Third Normal Form**.

1. 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
102	C++	P.Cpp
103	Java	P.Java2
104	C#	P.Chash
105	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 there 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

S.NO	Course	Hobby
1	Science	Cricket
2	Maths	Hockey
3	Science	Cricket
4	Maths	Hockey

And, **Professor Table**

p_id	**professor**
1	P.Java
2	P.Cpp
and so on...	

And now, this relation satisfies 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.

Consider the following relationship : **R (A,B,C,D)**

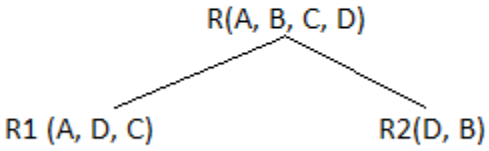
and following dependencies :

A -> BCD
BC -> AD
D -> B

Above relationship is already in 3rd NF. Keys are **A** and **BC.**

Hence, in the functional dependency, **A -> BCD**, A is the super key.
in second relation, **BC -> AD**, BC is also a key.
but in, **D -> B**, D is not a key.

Hence we can break our relationship R into two relationships **R1** and **R2.**



Breaking, table into two tables, one with A, D and C while the other with D and B.

FOURTH NORMAL FORM

Fourth Normal Form comes into picture when **Multi-valued Dependency** occurs in any relation. In this tutorial we will learn about Multi-valued Dependency, how to remove it and how to make any table satisfy the fourth normal form.

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**.

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	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.

Course Opted Table

s_id	**course**
1	Science
1	Maths
2	C#
2	Php

And, **Hobbies Table**,

s_id	**course**
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.
