

DBMS Lec 8 (Normalization) :

eg.

Emp

Emp-id

name

dept.

address

FD: \Rightarrow

$\text{Emp-id} \rightarrow \text{Emp-name}$

$\text{Emp-id} \rightarrow \text{dept}$

① Trivial F.D \vdash $A \rightarrow B$, B is a subset of A.

eg. $\{\text{EMP-ID, Name}\} \rightarrow \text{Emp-ID}$

eg. $A \rightarrow A, A \subseteq A$

$B \rightarrow B, B \subseteq B$

② Non-Trivial F.D \Rightarrow

$\{\text{EMP-ID, name}\} \rightarrow \{\text{Emp-address}\}$

$A \rightarrow B$

Q. $B \subseteq A$? no.

\Rightarrow This is N.T.F.D

$\Rightarrow A \rightarrow B, B \not\subseteq A, A \cap B = \text{NULL}$.

$\text{Emp-id} \rightarrow \text{Emp-dept}$

Normalization in databases is a way to organize data to reduce redundancy and prevent problems like duplicate

or missing information. It uses rules based on functional dependencies to split data into related tables, making the database more efficient and easier to maintain.

What is Normalization?

Normalization is a process used in database design to structure data so that it avoids problems like repeated entries, unnecessary duplication, and inconsistent updates. This is achieved by dividing data into logical tables and defining clear relationships between them.

Understanding Functional Dependency

A functional dependency describes a relationship where one set of data values (attributes) determines another set. For example, knowing a student ID can uniquely determine the student's name and email. Functional dependencies are the foundation for normalization.

Examples of Functional Dependency

Real-life examples, such as using an email ID to identify a department or a username to find an address, help illustrate how certain data points can determine others in a table.

Types of Functional Dependencies

There are different types of dependencies, such as full, partial, and transitive dependencies. Full dependency means an attribute relies on the entire primary key, partial means it depends on part of a composite key, and transitive means it depends indirectly through another attribute.

Problems Without Normalization

Without normalization, databases face issues like data redundancy (repeated data), update anomalies (errors when changing data), and deletion anomalies (loss of useful data when deleting records).

Importance of Normalization

Normalization helps keep databases clean by minimizing unnecessary data and ensuring changes are consistent across the system. It also makes databases easier to manage and scale.

Types of Anomalies Addressed

Normalization addresses three main anomalies: insertion (trouble adding new data), deletion (unintended loss of data), and update (inconsistent data after changes). These are solved by structuring tables properly.

Student

id	name	age	Branch_code	Branch_name	Branch_HOD
1	A	18	1	CS	X
2	B	19	1	CS	X
3	C	18	1	CS	X
4	D	21	2	ECE	Y
5	E	20	2	ECE	Y
6	F	19	3	M.E	Z

① Insertion

⇒ New student.

- Suppose if we made this table of student info above.
- This table has redundant data.
- Three Anomalies will occur →
 - Insertion :
 - If we happen to add a new student named “jiya” and right now her she just took admission and haven’t chosen any course then we would have to put null in fields (Branch_code, Branch_name, Branch_HOD).
 - Deletion:
 - If we happen to remove student with id = 6, then he is the only one in Mech branch and from the branch table, Mech branch will get deleted. On

deletion of particular data, any other data gets deleted too is called as deletion anomaly.

- **Updation:**

- If we happen to update HOD name in Branch_HOD table then we also would have to update HOD name in every entry of table (wherever it is used).

Creating Tables After Normalization

After normalization, data is split into multiple tables, each with a specific purpose, such as one table for students and another for branches. This separation allows for easier updates and fewer errors.

Decomposition and Single Responsibility

The process of breaking down a large table into smaller ones is called decomposition. Each table should have a single responsibility, meaning it stores only one type of information.

First Normal Form (1NF)

1NF requires that each table cell contains only one value and each record is unique. This eliminates

repeating groups and ensures a basic level of organization.


1NF = eg.

emp.

id	name	Phone.
1	A	88
2	B	12,99

↓

id	name	Phone
1	A	88
2	B	12
2	B	99



2NF \rightarrow

$R(\underline{A} \underline{B} C D)$

$\Rightarrow \{A B\} \rightarrow P.K$

$A, B \rightarrow$ prime attributes,

$C, D \rightarrow$ Non-prime.

$\Rightarrow FD \Rightarrow B \rightarrow C \leftarrow$ Partial dependency.

$\approx AB \rightarrow C \checkmark$

$AB \rightarrow D$

2NF Table (Student Project)

<u>Student ID</u>	<u>Proj ID</u>	Student Name	Project name
S89	P09	Arin	Geo
S76	P07	Jacob	Chites
S56	P03	Ava	IoT
S92	P05	Alex	Cloud.

P.K : $\{Student ID, Proj ID\}$

FD : $Student ID \rightarrow Student Name$
 $Project ID \rightarrow Project Name$

- Above is example of Partial dependency
- Student ID and Proj ID are prime attributes (part of the Primary key), they represent PD.
- Solution to above table is:

2NF form
Student

Student ID

S89
S76
S56
S92

Project ID

P09
P07
P03
P05

Student Name

Olivia
Jacob
Ava
alex

Project

Project ID

P09
P07
P03
P05

Project Name

Geo
Chiter
IoT
Cloud

3NF:

- Should be in 2NF.
- Should not have TRANSITIVE DEPENDENCY (A non-prime attribute can't derive another non-prime).

Higher Normal Forms and Dependencies

As normalization progresses, higher normal forms address more complex dependencies like partial and transitive dependencies, further reducing redundancy and improving data integrity.

Practical Example of Normalization

By applying normalization, a database can transform messy, redundant data into organized tables where each piece of information is stored only once and linked logically.

Transitive Dependency and Further Decomposition

Transitive dependency occurs when one attribute depends on another through a third attribute. Removing these by further decomposing tables ensures each dependency is direct and clear.

Many-to-Many Relationships

Some data, like students and professors, have many-to-many relationships, which are handled by creating separate linking tables to keep the data organized and avoid duplication.

Final Thoughts on Dependency and Table Design

Understanding and managing dependencies is key to robust database design. Proper normalization leads to reliable, scalable, and easy-to-maintain databases.