Normalization in DBMS (Database Management System) is the process of organizing data to reduce **redundancy** (repetition of data) and **improve data integrity**. It involves decomposing large tables into smaller, well-structured ones without losing data.

**🌟 Why Normalize?**

* Avoid data redundancy.
* Prevent anomalies during **Insert**, **Update**, and **Delete**.
* Ensure data consistency.

**🔑 Step-by-Step Explanation with Practical Example**

Let's take a practical example of a table before normalization:

**🔴 Unnormalized Table (UNF)**

markdown

CopyEdit

StudentTable

---------------------------------------------------------------

| StudentID | StudentName | Course | Instructor | Marks |

|-----------|-------------|------------|-------------|--------|

| 1 | Raj | DBMS | Dr. Kumar | 85 |

| 1 | Raj | Java | Mr. Sharma | 78 |

| 2 | Priya | DBMS | Dr. Kumar | 90 |

**🟢 Problems in this table:**

* Repeating StudentName for same student.
* Repeating Instructor for same course.
* Insertion/Update anomalies possible.

**✅ Step 1: First Normal Form (1NF)**

**📌 Rule:**

* Atomic values only (no multivalued or repeating groups).

In our case, it already has atomic values (no comma-separated or nested lists), so it's already in **1NF**.

**✅ Step 2: Second Normal Form (2NF)**

**📌 Rules:**

* Must be in 1NF.
* No **Partial Dependency**: No non-prime attribute should depend on a part of the **composite primary key**.

**🔍 Composite Key in UNF: (StudentID, Course)**

Issue: StudentName depends only on StudentID, not on full composite key.

**✅ Split into Two Tables:**

**➤ Student Table**

markdown

CopyEdit

Students

---------------------------

| StudentID | StudentName |

|-----------|-------------|

| 1 | Raj |

| 2 | Priya |

**➤ Enrollment Table**

markdown

CopyEdit

Enrollments

----------------------------------------------------

| StudentID | Course | Instructor | Marks |

|-----------|------------|-------------|-----------|

| 1 | DBMS | Dr. Kumar | 85 |

| 1 | Java | Mr. Sharma | 78 |

| 2 | DBMS | Dr. Kumar | 90 |

**✅ Step 3: Third Normal Form (3NF)**

**📌 Rules:**

* Must be in 2NF.
* No **Transitive Dependency**: Non-prime attribute should not depend on another non-prime attribute.

**🔍 Issue:**

In Enrollments, Instructor depends on Course, not on StudentID.

**✅ Solution: Split Instructor into a new table.**

**➤ Courses Table**

markdown

CopyEdit

Courses

-------------------------------

| Course | Instructor |

|------------|------------------|

| DBMS | Dr. Kumar |

| Java | Mr. Sharma |

**➤ Final Enrollments Table (After 3NF)**

markdown

CopyEdit

Enrollments

---------------------------

| StudentID | Course | Marks |

|-----------|--------|-------|

| 1 | DBMS | 85 |

| 1 | Java | 78 |

| 2 | DBMS | 90 |

**📦 Final Normalized Tables (Up to 3NF)**

**1. Students**

diff

CopyEdit

StudentID | StudentName

----------|-------------

1 | Raj

2 | Priya

**2. Courses**

diff

CopyEdit

Course | Instructor

----------|--------------

DBMS | Dr. Kumar

Java | Mr. Sharma

**3. Enrollments**

diff

CopyEdit

StudentID | Course | Marks

----------|--------|------

1 | DBMS | 85

1 | Java | 78

2 | DBMS | 90

**🧠 Summary of Normal Forms**

| **Normal Form** | **Rule** |
| --- | --- |
| 1NF | Atomic values only |
| 2NF | No partial dependency (on part of composite key) |
| 3NF | No transitive dependency (non-key depends only on key) |
| BCNF | Every determinant is a candidate key |
| 4NF | No multivalued dependency |

If you want, I can show you how to implement this normalized design in SQL with CREATE TABLE and INSERT queries. Would you like that?

**You said:**

yes

**ChatGPT said:**

Great! Let's implement the normalized design in **SQL** using CREATE TABLE and INSERT statements for the **Students**, **Courses**, and **Enrollments** tables.

**✅ 1. Create the Tables**

sql

CopyEdit

-- Students Table

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

StudentName VARCHAR(100)

);

-- Courses Table

CREATE TABLE Courses (

CourseName VARCHAR(100) PRIMARY KEY,

Instructor VARCHAR(100)

);

-- Enrollments Table (Relationship between Students and Courses)

CREATE TABLE Enrollments (

StudentID INT,

CourseName VARCHAR(100),

Marks INT,

PRIMARY KEY (StudentID, CourseName),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (CourseName) REFERENCES Courses(CourseName)

);

**✅ 2. Insert Sample Data**

sql

CopyEdit

-- Insert into Students

INSERT INTO Students (StudentID, StudentName) VALUES

(1, 'Raj'),

(2, 'Priya');

-- Insert into Courses

INSERT INTO Courses (CourseName, Instructor) VALUES

('DBMS', 'Dr. Kumar'),

('Java', 'Mr. Sharma');

-- Insert into Enrollments

INSERT INTO Enrollments (StudentID, CourseName, Marks) VALUES

(1, 'DBMS', 85),

(1, 'Java', 78),

(2, 'DBMS', 90);

**✅ 3. Sample Queries You Can Try**

**🔹 a. Get all students with their enrolled courses and marks:**

sql

CopyEdit

SELECT s.StudentName, e.CourseName, e.Marks

FROM Students s

JOIN Enrollments e ON s.StudentID = e.StudentID;

**🔹 b. Get each course with its instructor:**

sql

CopyEdit

SELECT \* FROM Courses;

**🔹 c. Get all students with course instructors:**

sql

CopyEdit

SELECT s.StudentName, e.CourseName, c.Instructor, e.Marks

FROM Enrollments e

JOIN Students s ON s.StudentID = e.StudentID

JOIN Courses c ON c.CourseName = e.CourseName;

**📘 Normalization Benefits in Practice**

| **🔄 Without Normalization** | **✅ After Normalization** |
| --- | --- |
| Repeated StudentName, Instructor | Stored once in related table |
| Risk of update anomalies | Updates made in a single place |
| Difficult to maintain consistency | Foreign keys enforce relationships |
| No modular structure | Modular, clear separation of concerns |