Date	05/12/2024	Unit	4

Data Normalization

Database Management System (DBMS)

Unit 4:
Data Normalization: Anomalies in relational database design. Decomposition. Functional dependencies. Normalization. First normal form, Second normal form, Third normal form. Boyce-Codd normal form.

What is Data Normalization?

Data Normalization is a systematic process in Relational Database Design that organizes data to minimize redundancy and dependency by dividing large tables into smaller, related tables. It ensures data consistency, reduces duplication, and enhances the efficiency of database operations.

Key Objectives:

1. Eliminate Redundancy: Avoid storing the same data in multiple places.
2. Ensure Data Integrity: Maintain accuracy and consistency across the database.
3. Improve Data Structure: Make it easier to manage and query the database.

Core Concepts:

1. Functional Dependencies: A relationship where one attribute uniquely determines another (e.g., if A → B, then A uniquely determines B).

2. **Anomalies to Avoid:**
   - Insertion Anomaly: Issues with adding data without complete information.
   - Update Anomaly: Inconsistencies during updates due to redundant data.
   - Deletion Anomaly: Loss of valuable data when deleting other data.

Normal Forms (NF):

1. First Normal Form (1NF):
   - Ensures atomicity (no repeating groups or arrays).
   - Each column contains unique values for each row.
2. Second Normal Form (2NF):
   - Builds on 1NF by ensuring all non-key attributes are fully dependent on the primary key.
3. Third Normal Form (3NF):
   - Builds on 2NF by ensuring all attributes are only dependent on the primary key, eliminating transitive dependencies.
4. Boyce-Codd Normal Form (BCNF):
   - A stronger version of 3NF where every determinant must be a candidate key.

Benefits:

- Improved data consistency and integrity.
- Easier maintenance and updates.
- Reduced storage space by eliminating duplicate data.

Anomalies in Relational Database Design

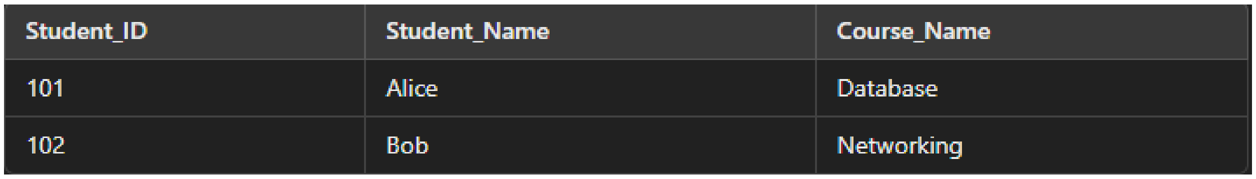
In relational databases, anomalies are problems that occur when data is inserted, updated, or deleted improperly due to a poorly designed schema. These anomalies lead to data inconsistencies, redundancy, and inefficiency.

They are commonly categorized into three types:

1. Insertion Anomaly

Occurs when it's difficult or impossible to add new data without including redundant or incomplete information.

Example: Consider a `Student` table storing information about students and the courses they enroll in:



Student_ID	Student_Name	Course_Name
101	Alice	Database
102	Bob	Networking

Problem:
If a new course needs to be added but no students are enrolled yet, we either:

- Insert incomplete data (e.g., leave `Student_ID` and `Student_Name` empty).
- Leave the course out until a student enrolls, causing data integrity issues.

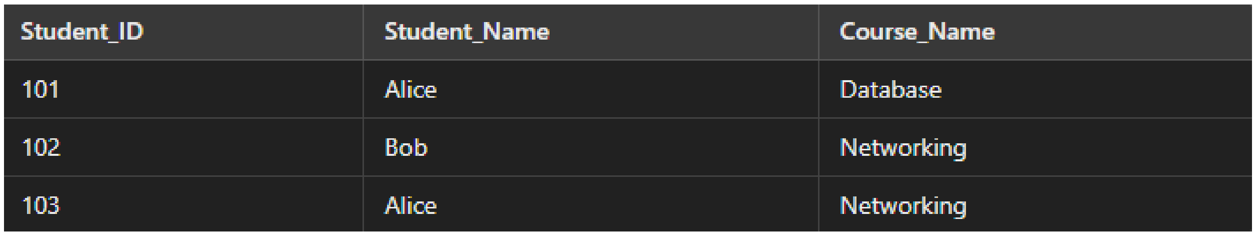
Solution:
Split the data into two tables:

1. `Student(Student_ID, Student_Name)`
2. `Course(Course_Name)`

2. Update Anomaly

Occurs when changes in data require multiple updates, leading to inconsistencies if not all instances are updated.

Example: Using the same table as above:



Student_ID	Student_Name	Course_Name
101	Alice	Database
102	Bob	Networking
103	Alice	Networking

Problem:
If Alice’s name is updated to "Alicia," the change needs to be made in every row where her name appears. If one row is missed, the database becomes inconsistent.

Solution:
Separate the tables:

1. `Student(Student_ID, Student_Name)`
2. `Enrollment(Student_ID, Course_Name)`

3. Deletion Anomaly

Occurs when deleting data unintentionally removes other valuable data.

Example: In the same `Student` table:

Student_ID	Student_Name	Course_Name
101	Alice	Database

Problem:
If we delete the only record of Alice taking the "Database" course, we lose both the student and course information.

Solution:
Separate data into related tables:

1. `Student(Student_ID, Student_Name)`
2. `Course(Course_Name)`
3. `Enrollment(Student_ID, Course_Name)`

Summary of Solutions:

1. **Normalization** helps eliminate these anomalies by organizing data into multiple related tables.
2. Using **Primary Keys** and **Foreign Keys** ensures data integrity.
3. Maintaining proper relationships between tables reduces redundancy and inconsistencies.

What is Decomposition in Database Design?

Decomposition is the process of breaking a larger, complex relation (table) in a relational database into two or more smaller, simpler relations. It is done to eliminate data anomalies (insertion, update, and deletion) and ensure data consistency while preserving the original data.

Decomposition is a key part of **Normalization**, where relations are split to achieve various **Normal Forms (NF)**.

Goals of Decomposition:

1. **Lossless-Join Property**: Ensure that the original relation can be reconstructed by joining the decomposed relations.
2. **Dependency Preservation**: Ensure all functional dependencies are preserved in the decomposed relations.

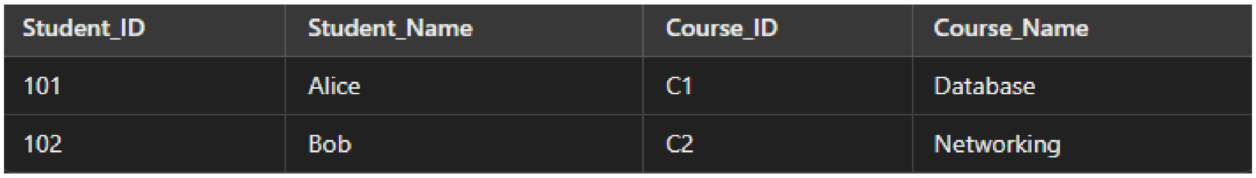
3. **Eliminate Anomalies**: Remove redundancy and anomalies (insertion, update, deletion).

Types of Decomposition:

1. **Lossless (Non-Lossy) Decomposition**:
   - Ensures no data is lost when relations are joined back together.
   - Essential to maintain the integrity of the original data.
2. **Lossy Decomposition**:
   - Some data is lost or cannot be accurately reconstructed after decomposition.
   - This type is undesirable and should be avoided.

Example of Decomposition:

Consider a relation `StudentCourse` with attributes:



Student_ID	Student_Name	Course_ID	Course_Name
101	Alice	C1	Database
102	Bob	C2	Networking

Functional Dependencies (FDs):

1. `Student_ID → Student_Name`
2. `Course_ID → Course_Name`

Issues:

This relation may have update and insertion anomalies. To normalize, we decompose the relation.

Decomposition Steps:

1. Split into two relations:
   - **Student Relation**: `(Student_ID, Student_Name)`
   - **Course Relation**: `(Course_ID, Course_Name)`

   Now we have:

   - `Student(Student_ID, Student_Name)`
   - `Course(Course_ID, Course_Name)`
   - `Enrollment(Student_ID, Course_ID)`

2. Resulting Decomposed Tables:

Student Table:

Student_ID	Student_Name
101	Alice
102	Bob

Course Table:

Course_ID	Course_Name
C1	Database
C2	Networking

Enrollment Table:

Student_ID	Course_ID
101	C1
102	C2

Properties of a Good Decomposition:

1. **Lossless-Join**:
   Rejoining the decomposed tables using natural joins should yield the original relation.

SELECT Student.Student_ID, Student_Name, Course_ID, Course_Name

FROM Student

NATURAL JOIN Enrollment

NATURAL JOIN Course;

Dependency Preservation:

All functional dependencies should still hold in at least one of the decomposed relations.

What are Functional Dependencies (FDs)?

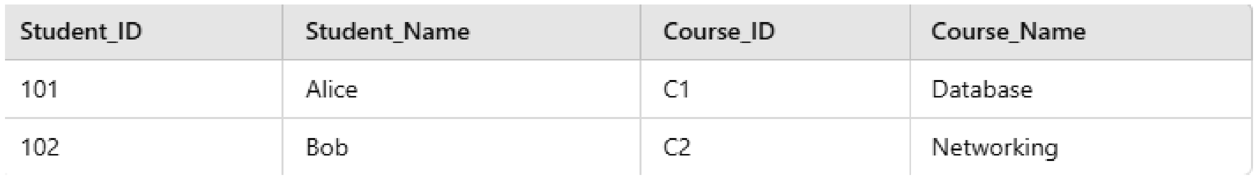
A **Functional Dependency (FD)** is a relationship between attributes in a relational database that defines how one attribute uniquely determines another. It is a fundamental concept in **Relational Database Design** and **Normalization**.

In an FD, if knowing the value of one attribute (or a set of attributes) allows you to determine the value of another attribute, then the second attribute is said to be **functionally dependent** on the first.

Notation:

- **X → Y**
  This means that attribute `X` functionally determines attribute `Y`.
  - `X` is called the **determinant**.
  - `Y` is the **dependent**.

Example of Functional Dependency:



Student_ID	Student_Name	Course_ID	Course_Name
101	Alice	C1	Database
102	Bob	C2	Networking

`Student_ID → Student_Name`:
Knowing `Student_ID` uniquely determines `Student_Name`.

`Course_ID → Course_Name`:
Knowing `Course_ID` uniquely determines `Course_Name`.

Types of Functional Dependencies:

1. **Trivial Functional Dependency**:
   A dependency where the dependent attribute is a subset of the determinant.
   Example:
   `Student_ID, Student_Name → Student_Name`
2. **Non-Trivial Functional Dependency**:
   A dependency where the dependent attribute is not a subset of the determinant.
   Example:
   `Student_ID → Student_Name`

3. **Fully Functional Dependency**:
   A dependency where an attribute is fully dependent on the entire composite key, not just a part of it.
   Example:
   In a relation `Enrollment(Student_ID, Course_ID, Grade)`:
   - `Student_ID, Course_ID → Grade` (Fully dependent)
   - `Student_ID → Grade` (Not valid if it's only partially dependent)
4. **Partial Functional Dependency**:
   A dependency where an attribute is dependent on part of a composite key.
   Example:
   In the relation `Enrollment(Student_ID, Course_ID, Course_Name)`:
   - `Course_ID → Course_Name` (Partial dependency)
5. **Transitive Functional Dependency**:
   If `A → B` and `B → C`, then `A → C` is a transitive dependency.
   Example:
   `Student_ID → Department_ID` and `Department_ID → Department_Name` imply `Student_ID → Department_Name`.

Properties of Functional Dependencies:

1. **Reflexivity**:
   If `Y` is a subset of `X`, then `X → Y`.
2. **Augmentation**:
   If `X → Y`, then `XZ → YZ` (for any attribute set `Z`).
3. **Transitivity**:
   If `X → Y` and `Y → Z`, then `X → Z`.
4. **Union**:
   If `X → Y` and `X → Z`, then `X → YZ`.
5. **Decomposition**:
   If `X → YZ`, then `X → Y` and `X → Z`.

Example of Applying Functional Dependencies:

Relation: Employee(Emp_ID, Emp_Name, Dept_ID, Dept_Name)

Functional Dependencies:

1. `Emp_ID → Emp_Name, Dept_ID`
2. `Dept_ID → Dept_Name`

To eliminate redundancy and anomalies, this can be decomposed into:

1. **Employee(Emp_ID, Emp_Name, Dept_ID)**
2. **Department(Dept_ID, Dept_Name)**

Why Are Functional Dependencies Important?

- **Normalization**: Used to identify keys and ensure tables meet normal forms (1NF, 2NF, 3NF, BCNF).
- **Database Integrity**: Helps maintain data accuracy and consistency.
- **Redundancy Reduction**: Avoids data duplication and ensures efficient storage.

Normalization in Relational Database Design

Normalization is the process of organizing data in a database to reduce redundancy and improve data integrity. It involves decomposing tables into smaller, related tables without losing data and ensuring that the database adheres to certain design rules called **Normal Forms (NF)**.

1. First Normal Form (1NF)

A table is in **1NF** if it meets the following criteria:

1. **Atomicity**: All values in each column must be atomic (indivisible). No multivalued attributes or repeating groups are allowed.
2. **Uniqueness**: Each row must be unique and identifiable by a primary key.

Example (Non-1NF Table):

Student_ID	Student_Name	Courses
101	Alice	Database, SQL
102	Bob	Networking

Problem: The `Courses` column contains multiple values, violating atomicity.

Solution (Convert to 1NF):

Student_ID	Student_Name	Course
101	Alice	Database
101	Alice	SQL
102	Bob	Networking

2. Second Normal Form (2NF)

A table is in **2NF** if:

1. It is in **1NF**.
2. All non-key attributes are **fully functionally dependent** on the entire primary key (no partial dependency).

Example (Non-2NF Table):

Student_ID	Course_ID	Student_Name	Course_Name
101	C1	Alice	Database
102	C2	Bob	Networking

Problem: `Student_Name` depends only on `Student_ID`, and `Course_Name` depends only on `Course_ID`. These are partial dependencies.

Solution (Convert to 2NF):

1.Student Table:

Student_ID	Student_Name
101	Alice
102	Bob

2.Course Table:

Course_ID	Course_Name
C1	Database
C2	Networking

3.Enrollment Table:

Student_ID	Course_ID
101	C1
102	C2

3. Third Normal Form (3NF)

A table is in **3NF** if:

1. It is in **2NF**.
2. There are no **transitive dependencies**, meaning no non-key attribute depends on another non-key attribute.

Example (Non-3NF Table):

Student_ID	Student_Name	Dept_ID	Dept_Name
101	Alice	D1	CS
102	Bob	D2	IT

Problem: `Dept_Name` is transitively dependent on `Student_ID` via `Dept_ID`.

Solution (Convert to 3NF):

1. **Student Table**:

Student_ID	Student_Name	Dept_ID
101	Alice	D1
102	Bob	D2

2. **Department Table**:

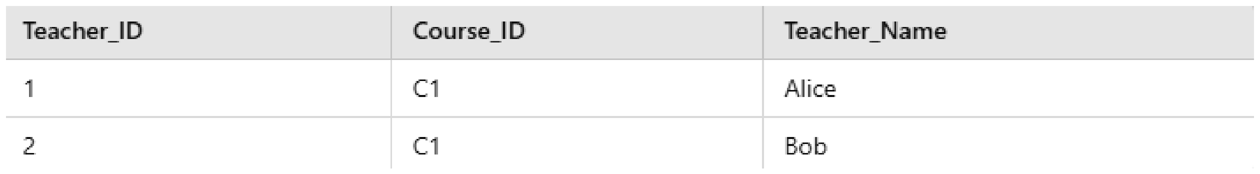
Dept_ID	Dept_Name
D1	CS
D2	IT

4. Boyce-Codd Normal Form (BCNF)

A table is in **BCNF** if:

1. It is in **3NF**.
2. Every determinant (attribute that determines another) is a **candidate key**.

Example (Non-BCNF Table):

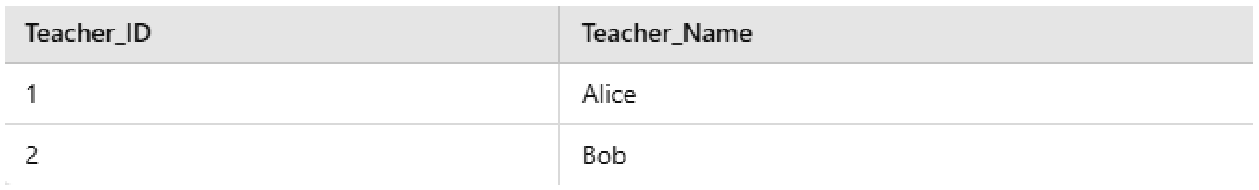


Teacher_ID	Course_ID	Teacher_Name
1	C1	Alice
2	C1	Bob

Problem: `Course_ID → Teacher_ID` is a dependency, but `Course_ID` is not a candidate key.

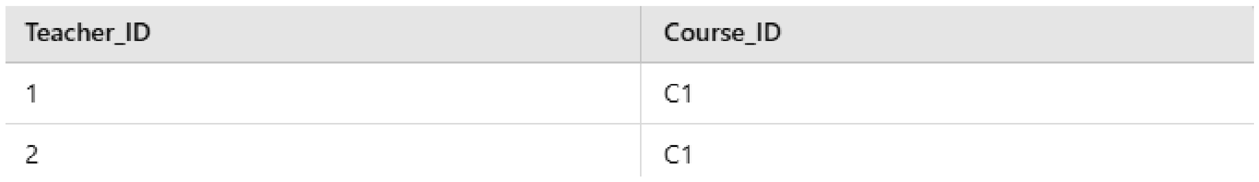
Solution (Convert to BCNF):

1. **Teacher Table**:

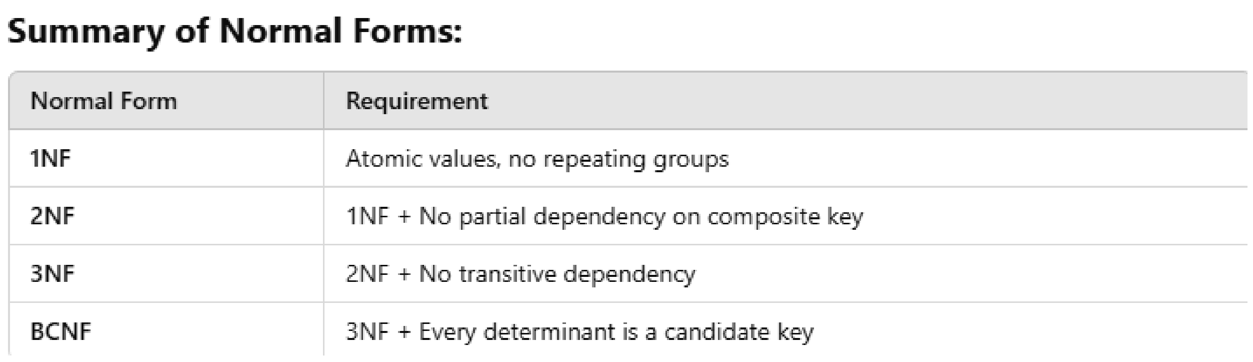


Teacher_ID	Teacher_Name
1	Alice
2	Bob

2. **Course Assignment Table**:



Teacher_ID	Course_ID
1	C1
2	C1



Summary of Normal Forms:

Normal Form	Requirement
1NF	Atomic values, no repeating groups
2NF	1NF + No partial dependency on composite key
3NF	2NF + No transitive dependency
BCNF	3NF + Every determinant is a candidate key

END