Unit 2

Syllabus:

Introduction to RDBMS:-Entity introduction, characteristics, Comparison between DBMS, RDBMS, Generalization and Aggregation Normalization- Functional dependency, types of normalization(1NF,2NF,3NF,BCNF),Data constraint- primary key, foreign key, unique key, null, not null, default key etc.

## Difference between DBMS and RDBMS:

**DBMS (Database Management System)**

A Database Management System (DBMS) is software that is used to define, create, manage, and control access to databases. It allows users to perform various operations on the data, such as insertion, updating, deletion, and retrieval. DBMS provides a systematic and organized way of managing data, ensuring data consistency, integrity, and security.

**RDBMS (Relational Database Management System)**

A Relational Database Management System (RDBMS) is a type of DBMS that is based on the relational model of data. In an RDBMS, data is stored in tables (also known as relations), which consist of rows and columns. RDBMS enforces relationships between tables using keys (primary and foreign keys) and supports normalization to reduce data redundancy. It provides advance features for data integrity, security and concurrent access.

| **Feature** | **DBMS (Database Management System)** | **RDBMS (Relational Database Management System)** |
| --- | --- | --- |
| **Data Storage** | Stores data as files | Stores data in tables |
| **Data Structure** | Hierarchical, network, or object-oriented models | Tabular structure with rows and columns |
| **Relationships** | No relationships between data | Enforces relationships using keys (primary and foreign) |
| **Normalization** | Typically not supported | Supports normalization to reduce redundancy |
| **Data Redundancy** | Higher data redundancy | Lower data redundancy due to normalization |
| **User Support** | Generally single-user | Supports multiple users |
| **Security** | Basic security features | Advanced security features (user authentication, access control) |
| **Data Integrity** | Limited data integrity | Enforces data integrity constraints |
| **Examples** | XML, Windows Registry, dBase | MySQL, PostgreSQL, SQL Server, Oracle |

## What is nornalization:

Normalization in the context of databases is the process of organizing data to minimize redundancy and improve data integrity. This involves dividing a database into two or more tables and defining relationships between the tables. The main goal is to ensure that each piece of data is stored only once, which helps in maintaining data consistency and reducing the chances of anomalies during data operations.

## Type of Normalization in RDBMS:

Sure! Let’s dive deeper into the concepts of 1NF, 2NF, 3NF, and BCNF in database normalization:

### **First Normal Form (1NF)**

**Definition**: A relation is in 1NF if it contains only atomic (indivisible) values and each column contains unique values.

**Key Points**:

* **Atomicity**: Each cell in a table must contain a single value, not a set of values.
* **Uniqueness**: Each column must have a unique name, and the order in which data is stored does not matter.

**Example**: Contact List

**Scenario**: You have a contact list where each contact can have multiple phone numbers.

**Non-1NF Table**:

| **ContactID** | **Name** | **PhoneNumbers** |
| --- | --- | --- |
| 1 | John Doe | 123-4567, 234-5678 |
| 2 | Jane Smith | 345-6789, 456-7890 |

* **Issue**: The PhoneNumbers column contains multiple values, which violates 1NF.
* **1NF Table**:

| **ContactID** | **Name** | **PhoneNumber** |
| --- | --- | --- |
| 1 | John Doe | 123-4567 |
| 1 | John Doe | 234-5678 |
| 2 | Jane Smith | 345-6789 |
| 2 | Jane Smith | 456-7890 |

* **Solution**: Each contact has a single phone number per entry, ensuring atomicity.

### **Second Normal Form (2NF)**

**Definition**: A relation is in 2NF if it is in 1NF and all non-key attributes are fully functionally dependent on the primary key.

**Key Points**:

* **Full Functional Dependency**: Every non-key attribute must depend on the entire primary key, not just part of it.

**Example**: Library System

**Scenario**: You have a library system where each book has a unique ISBN, and all book details depend on that ISBN.

**Non-2NF Table**:

| **ISBN** | **Title** | **Author** | **Publisher** |
| --- | --- | --- | --- |
| 12345 | Book A | Author X | Publisher Y |
| 67890 | Book B | Author Y | Publisher Z |

* **Issue**: If the table had a composite key (e.g., ISBN + Title), partial dependencies could exist. However, in this case, let’s assume the table is already in 1NF but not in 2NF due to partial dependencies.
* **2NF Table**: **Books Table**:

| **ISBN** | **Title** | **Author** |
| --- | --- | --- |
| 12345 | Book A | Author X |
| 67890 | Book B | Author Y |

* **Publishers Table**:

| **ISBN** | **Publisher** |
| --- | --- |
| 12345 | Publisher Y |
| 67890 | Publisher Z |

* **Solution**: All book details depend on the unique ISBN, eliminating partial dependencies.

### **Third Normal Form (3NF)**

**Definition**: A relation is in 3NF if it is in 2NF and all the attributes are functionally dependent only on the primary key.

**Key Points**:

* **Transitive Dependency**: There should be no transitive dependencies, where a non-key attribute depends on another non-key attribute.

**Example**: Employee Database

**Scenario**: You have an employee database where each employee’s department name is stored in a separate table to avoid redundancy.

**Non-3NF Table**:

| **EmployeeID** | **EmployeeName** | **DepartmentID** | **DepartmentName** |
| --- | --- | --- | --- |
| 1 | Alice | 10 | HR |
| 2 | Bob | 20 | IT |

* **Issue**: DepartmentName depends on DepartmentID, which is not the primary key, creating a transitive dependency.
* **3NF Table**:
* **Employees Table**:

| **EmployeeID** | **EmployeeName** | **DepartmentID** |
| --- | --- | --- |
| 1 | Alice | 10 |
| 2 | Bob | 20 |

* **Departments Table**:

| **DepartmentID** | **DepartmentName** |
| --- | --- |
| 10 | HR |
| 20 | IT |

* **Solution**: Each employee’s department name is stored in a separate table, eliminating transitive dependencies.

### **Boyce-Codd Normal Form (BCNF)**

**Definition**: A relation (table) is in BCNF if, and only if, for every functional dependency ( X \rightarrow Y ), ( X ) is a superkey. This means that ( X ) must be a candidate key or a superkey for the table.

**Key Points**:

* **Stricter than 3NF**: BCNF is a stricter version of the Third Normal Form (3NF). While 3NF ensures that non-key attributes are not dependent on other non-key attributes, BCNF ensures that every determinant is a candidate key.
* **Eliminates Redundancy**: BCNF aims to remove all redundancies based on functional dependencies.

### Example to BCNF

Consider a table that stores information about courses, instructors, and rooms:

| **CourseID** | **Instructor** | **Room** |
| --- | --- | --- |
| 101 | Prof. A | 201 |
| 102 | Prof. B | 202 |
| 103 | Prof. A | 201 |
| 104 | Prof. C | 203 |

**Functional Dependencies**:

1. CourseID → Instructor
2. Instructor → Room

In this table:

* CourseID determines Instructor.
* Instructor determines Room.

**Why This Table is Not in BCNF**:

* The functional dependency Instructor → Room violates BCNF because Instructor is not a superkey. A superkey is a set of one or more columns that can uniquely identify a row in a table. Here, Instructor alone cannot uniquely identify a row because multiple courses can be taught by the same instructor in the same room.

**Converting to BCNF**: To convert this table to BCNF, we need to decompose it into two tables where every determinant is a superkey.

**Table 1: Courses**

| **CourseID** | **Instructor** |
| --- | --- |
| 101 | Prof. A |
| 102 | Prof. B |
| 103 | Prof. A |
| 104 | Prof. C |

**Table 2: Instructors**

| **Instructor** | **Room** |
| --- | --- |
| Prof. A | 201 |
| Prof. B | 202 |
| Prof. C | 203 |

Now:

* In **Table 1**, CourseID is a superkey.
* In **Table 2**, Instructor is a superkey.

This decomposition ensures that both tables are in BCNF, eliminating redundancy and ensuring data integrity.

To remember it 🡺

Remembering the different normal forms in database normalization can be challenging, but here are some tips and mnemonics to help you:

### Mnemonic for Normal Forms

A popular mnemonic to remember the key concepts of the first three normal forms is: **“The key, the whole key, and nothing but the key, so help me Codd.”**

* **1NF (The key)**: Ensure that each column contains atomic values and each row is unique.
* **2NF (The whole key)**: Ensure that all non-key attributes are fully dependent on the entire primary key.
* **3NF (And nothing but the key)**: Ensure that all non-key attributes are only dependent on the primary key and not on other non-key attributes.

### Visual Aids

Creating visual aids like diagrams can also help. Draw tables and illustrate the functional dependencies between columns. This can make it easier to see how normalization works in practice.

### Example-Based Learning

Using real-world examples can make the concepts more relatable. For instance:

* **1NF**: Think of a contact list where each contact has a single phone number per entry.
* **2NF**: Imagine a library system where each book has a unique ISBN, and all book details depend on that ISBN.
* **3NF**: Consider an employee database where each employee’s department name is stored in a separate table to avoid redundancy.

### Practice Problems

Regularly practicing normalization problems can reinforce your understanding. Try normalizing different datasets to identify and eliminate anomalies.

### Summary Table

Here’s a quick reference table to summarize the normal forms:

| **Normal Form** | **Description** |
| --- | --- |
| **1NF** | Ensure atomicity of values in columns. |
| **2NF** | Eliminate partial dependencies on composite keys. |
| **3NF** | Remove transitive dependencies. |
| **BCNF** | Ensure every determinant is a candidate key. |