**What is RDBMS**

RDBMS stands for Relational Database Management Systems. It is a program that allows us to create, delete, and update a relational database. A Relational Database is a database system that stores and retrieves data in a tabular format organized in the form of rows and columns.

Relational DBMS owes its foundation to the fact that the values of each table are related to others. It has the capability to handle larger magnitudes of data and simulate queries easily.

Relational Database Management Systems maintains data integrity by simulating the following features:

* **Entity Integrity:** No two records of the database table can be completely duplicate.
* **Referential Integrity:** Only the rows of those tables can be deleted which are not used by other tables. Otherwise, it may lead to [data inconsistency](https://www.geeksforgeeks.org/what-is-data-inconsistency-in-dbms/).
* **User-defined Integrity:** Rules defined by the users based on confidentiality and access.
* **Domain integrity:**The columns of the database tables are enclosed within some structured limits, based on default values, type of data or ranges

**Essential Components of table**

**1. Columns**

Also known as fields or attributes**.**Each type of information present in the **vertical position** is known as columns or fields. They allow us to sort and filter data in a table. In the above-given example, ID, Name, Age, and Course are different columns.

**2. Row**

Also known as a tuple or record. The data of each student in a **horizontal position** is known as row or record. Each record holds the total data for a specific student. First row represents that the student of ID 1 has the name Minal, age 22 and course in Computer Science.

**3. Column Name**

Each column or Field has its unique name. The first column name is ID which stores the ID of each student, second column name is NAME which holds the name of each student for every tuple and so on.

**4. Data Items**

Column values are the information stored in a specific cell within a column for a particular row. Every point where a row and a column cross signifies a distinct piece of data. Every cell in Students table has a data item that represents a particular student attribute, such as ID, NAME, AGE, or COURSE.

**5. Data Types**

Each column in a table is associated with a specific data type that defines the kind of data it can store. Some of the data types are int, varchar, char, date, etc. Data types ensures that the data entered in the column belongs to a particular format and structure to the stored information. For example ID and age might be integer, and Name and course could be strings.

**6. Primary Key**

It is the**unique identifier** for each row in a table. There can only be one primary key in a table, and it can't be null. ID is the primary key for the above table. Each student is uniquely identified by their ID. Now two rows have the same ID. The first column is usually the primary key of the table.

**7. Foreign Key**

A foreign key is a column or a set of columns in a table that **refers to the primary key**of another table. It establishes the relationship between the table. A foreign key can be null and there can be more than one foreign key in the table. For example, in the Department table you might have an emp\_id column as a foreign key. This allows you to link departments to specific employees.

**Intension and Extension in a DataBase**

**Intension (Schema)**

The **intension** refers to the schema of the database, which is the blueprint or structure that defines the database. It includes the definitions of tables, fields (attributes), data types, relationships, constraints, indexes, and other elements that define how data is organized and accessed.

* Provides a framework for data storage and retrieval.
* Ensures data integrity and consistency through constraints and relationships.
* Facilitates understanding and managing the database structure.

**Static Nature of Intension**

The intension, or schema, of a database is **relatively static**. Once defined, it does not change frequently. Modifications to the schema happen only in response to significant changes in business requirements or system upgrades. This stability ensures consistency in how data is organized and accessed.

**Extension (Instance)**

The **extension** refers to the actual data stored in the database at any given time. It consists of the rows (records) of data that populate the tables defined by the schema. Extensions are dynamic, changing frequently due to insertions, deletions, and updates to the data.

* Represents the real-time state of the database.
* Reflects the actual data that users interact with.
* Is critical for day-to-day operations and decision-making based on current data.

**Dynamic Nature of Extensions**

The extension is **highly dynamic** because it changes as users interact with the database. This includes operations such as:

* **Insertions**: Adding new records to tables.
* **Deletions**: Removing records from tables.
* **Updates**: Modifying existing records.

The dynamic nature of extensions means the database state can change over time, reflecting real-world events and transactions.

**Difference between Intension and Extension in a Database**

| **Property** | **Intension** | **Extension** |
| --- | --- | --- |
| **Definition** | Refers to the schema or the structure of a database. It describes the blueprint of the database, including tables, columns, data types, constraints, and relationships. | Refers to the actual data stored in the database at a particular point in time, representing the current state of the database, including records in the tables. |
| **Nature** | Static and does not change frequently. It is predefined and stored in the schema. | Dynamic and changes over time as records are inserted, updated, or deleted. |
| **Purpose** | Provides a framework for how data is stored and managed. It defines the rules and constraints for data integrity and relationships among entities. | Represents the real-world information the database is meant to capture. It is the content users interact with. |
| **Components** | Includes table definitions, column names, data types, constraints (e.g., primary keys, foreign keys), indexes, views, and triggers. | Includes rows (records) in the tables, actual values stored in each field of those rows. |
| **Visibility** | Visible to database designers and developers who define the schema. | Visible to end-users who query and manipulate the data. Users interact with this data through queries and reports. |

**Example of Intension and Extension**

Consider the following example of creating an **Employees** table:

CREATE TABLE Employees (

EmpID INT PRIMARY KEY,

FirstName VARCHAR(50),

LastName VARCHAR(50),

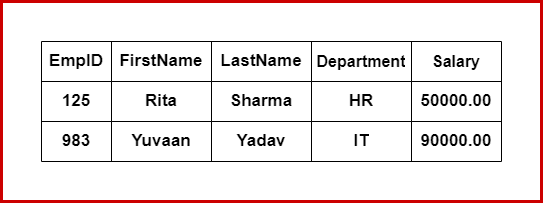
Department VARCHAR(50),

Salary DECIMAL(10, 2)

);

This is the schema definition (intension) for the **Employees** table.

**Extension Example at Time t1**



**Extension Example at Time t2**



**Schema Definition Language (SDL)**

The **Schema Definition Language** (SDL), also known as **Data Definition Language** (DDL), is a subset of SQL used to define and manage the structure of a database. It encompasses various commands to specify how data is stored and how the relationships among data elements are maintained.

**Key Components and Concepts of SDL**

* **Data Types**: Defines the types of data that can be stored. Examples: INT, VARCHAR, DATE, BOOLEAN.
* **Tables**: Basic units of storage, consisting of rows and columns. Each column has a specific data type.
* **Constraints**: Rules to ensure data integrity. Examples: PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK.
* **Indexes**: Improve data retrieval speed. Examples: CREATE INDEX, UNIQUE INDEX.
* **Views**: Virtual tables created from SQL query results.
* **Triggers**: Procedures that are automatically executed in response to certain events on a table or view.

**Common SDL Commands**

| **Command** | **Description** | **Syntax** |
| --- | --- | --- |
| **CREATE** | Creates a new table with specified columns and constraints. | CREATE TABLE table\_name (column1 datatype constraints, column2 datatype constraints, ...); |
| **DROP** | Deletes a table and all its data from the database. | DROP TABLE table\_name; |
| **ALTER** | Modifies an existing table by adding, deleting, or modifying columns and constraints. | ALTER TABLE table\_name ADD column\_name datatype constraints; ALTER TABLE table\_name DROP COLUMN column\_name; ALTER TABLE table\_name MODIFY column\_name datatype constraints; |
| **TRUNCATE** | Removes all rows from a table without deleting the table itself. | TRUNCATE TABLE table\_name; |
| **COMMENT** | Adds comments to the data dictionary. | COMMENT 'comment\_text' ON TABLE table\_name; |
| **RENAME** | Renames an existing object in the database. | RENAME TABLE old\_table\_name TO new\_table\_name |

**Keys**

In a **Database Management System (DBMS)**, keys are essential for defining the structure of a database and maintaining data accuracy. Keys help in uniquely identifying records and retrieving data quickly. Below are the main types of keys used in DBMS:

**Types of Keys**

1. **Candidate Key**

**Definition:** A candidate key is an attribute or set of attributes that can uniquely identify a record in a table.

**Characteristics:**

* There can be multiple candidate keys in a table.
* One of the candidate keys is selected as the primary key.

**Example:** In a table of students, possible candidate keys could be StudentID, Roll No, or Aadhar Card.

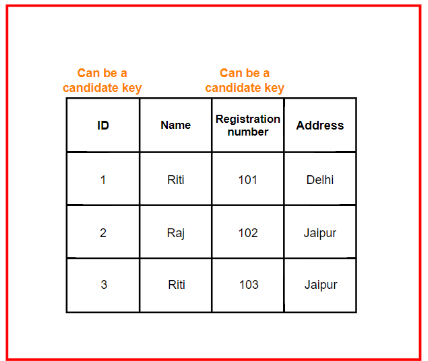
1. **Primary Key**

**Definition:** A primary key is a unique identifier for a record in a table. It ensures that no two records have the same value in this field or combination of fields.

**Characteristics:**

* Primary keys must be unique across the table.
* They cannot contain NULL values.
* There can be only one primary key per table.

**Example:** ID and Registration numbers can be primary keys. However, PAN number cannot be a primary key because some students might not have a PAN card, and names cannot be unique.



1. **Foreign Key**

**Definition:** A foreign key is an attribute in one table that refers to the primary key in another table. It creates a link between two tables.

**Characteristics:**

* Used to establish a link between data in two tables.
* Ensures referential integrity between the tables.

**Example:** A StudentID in the "Enrollments" table can be a foreign key that references the StudentID in the "Students" table.

1. **Super Key**

**Definition:** A super key is a set of one or more attributes that can uniquely identify a record in a table.

**Characteristics:**

* A super key may contain additional attributes that aren't necessary for unique identification.
* Every primary key is a super key, but not all super keys are primary keys.

**Example:** ID and Name together can be a super key, but only the ID can be a primary key since the Name is not unique.

**Advantages of Keys in DBMS**

* **Uniquely Identifying Data:** Keys, especially primary keys, ensure that each record in a table is unique.
* **Maintaining Data Integrity:** Keys help prevent data duplication and ensure the accuracy of the data.
* **Efficient Data Retrieval:** Keys are used to build indexes, making it easier to retrieve data from large datasets.
* **Establishing Relationships:** Foreign keys help create links between tables, representing real-world relationships.
* **Data Security:** Keys can be used to control access to sensitive information, limiting access based on specific keys

**Normalisation and its Types(1NF, 2NF, 3NF)**

Normalization is the process of organizing a database to reduce redundancy and dependency. This ensures that each piece of data is stored in one place. We achieve this by splitting data into smaller tables, which follow specific rules known as "normal forms" (like 1NF, 2NF, and 3NF). These rules help ensure that each table has a unique identifier (primary key) and that other information (non-key attributes) depends on the primary key.

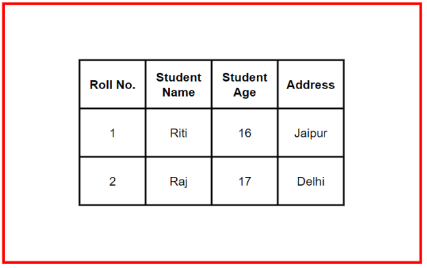
**Why normalize?**

* **Reduces data duplication:** Storing each piece of data only once minimizes inconsistencies and makes data easier to update and manage.
* **Improves data consistency:** Since each data point exists in only one place, updating it in one spot keeps all your information consistent.
* **Minimizes data anomalies:** Normalization reduces reliance on other data, making the database easier to understand and use while avoiding issues like insertion, update, and deletion anomalies.
* **Enhances scalability and performance:** By managing less redundant data, the database becomes faster and easier to maintain.

**First Normal Form (1NF)**

A table is in 1NF if:

* All columns contain atomic (indivisible) values.
* Each column holds values of a single type.
* There are no repeating groups or arrays in columns (only one value per row).



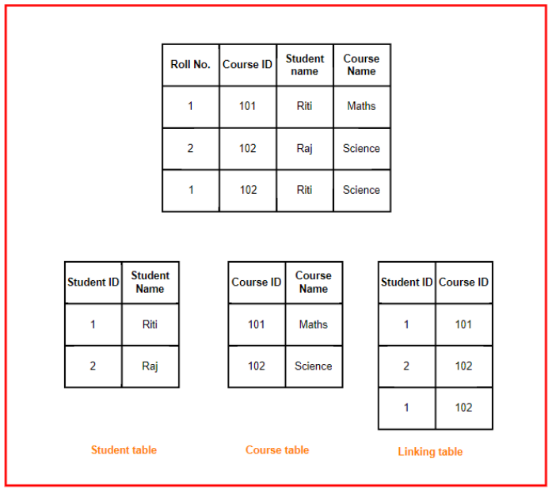
This table follows 1NF as it has no composite or multi-valued attributes.

**Second Normal Form (2NF)**

A table is in 2NF if:

* It is already in 1NF.
* All non-key attributes are fully functionally dependent on the entire primary key.

Partial dependency occurs when a non-prime attribute depends on only part of a composite key, violating 2NF rules. Here's an example:

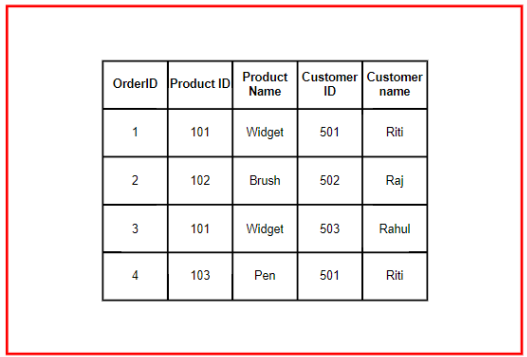


In this case, **StudentName** depends on **StudentID** and **CourseName** depends on **CourseID**, showing partial dependency. To convert this to 2NF, we break it into two tables: one for students and one for courses.

**Third Normal Form (3NF)**

A table is in 3NF if:

* It is in 2NF.
* All attributes are functionally dependent only on the primary key.

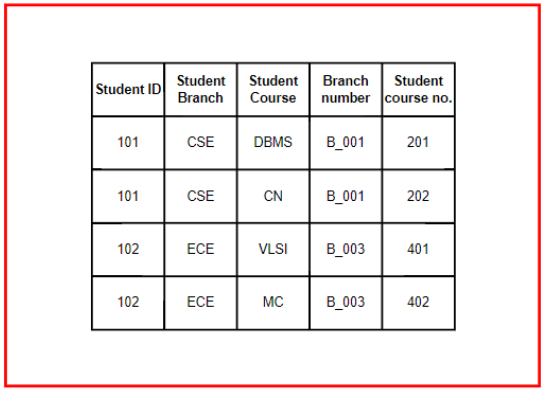


This example has a transitive dependency: **CustomerName** depends on **CustomerID**, which in turn depends on the primary key. To remove this, we separate customers into a new table, achieving 3NF.

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

A table is in BCNF if:

* It is in 3NF.
* For every functional dependency, the left-hand side (X) is a super key.

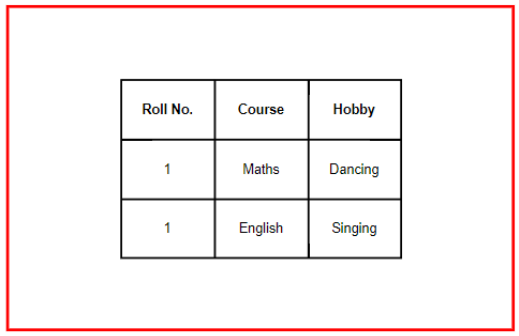


In BCNF, we further decompose tables to remove any dependencies where X is not a super key. This is demonstrated by breaking down tables in the example.

**Fourth Normal Form (4NF)**

A table is in 4NF if:

* It is in BCNF.
* There are no multi-valued dependencies.

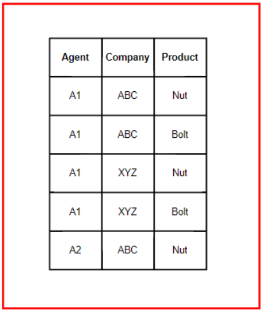


Here, we see a multi-valued dependency between Course and Hobby, which is independent of each other. To achieve 4NF, we decompose the table into separate entities.

**Fifth Normal Form (5NF)**

A table is in 5NF if:

* It is in 4NF.
* It cannot be decomposed into smaller tables without losing data or introducing redundancy.

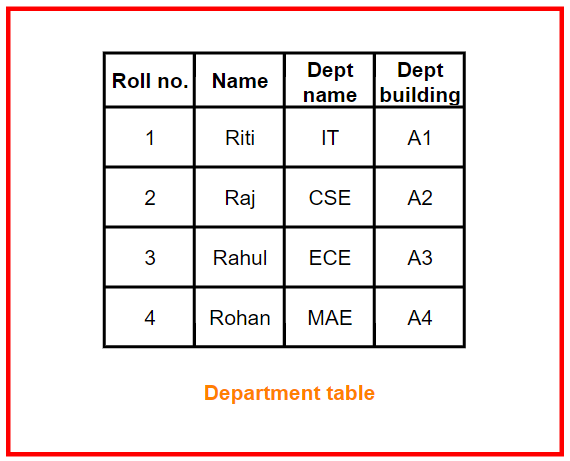


In complex relationships, 5NF ensures no data is lost and no unnecessary redundancy is introduced. The table is broken into smaller ones only when absolutely necessary

**Functional dependency**

In a relational database management system, a functional dependency describes the relationship between two sets of attributes where one set determines the value of another. It is written as 𝑋→𝑌, where 𝑋 (the attribute set on the left) is known as the **Determinant**, and 𝑌 (the attribute set on the right) is known as the **Dependent**.

Functional dependencies help understand how attributes relate to each other in a database, which is crucial for database design and normalization.



**Types of Functional Dependency**

* 1. **Full Functional Dependency**

A functional dependency 𝑋→𝑌 is a **full functional dependency** if removing any attribute from 𝑋 means the dependency no longer holds. In simpler terms, 𝑌 depends entirely on 𝑋.

**Example:** If we have a table Enrollment(StudentID, CourseID, Grade), the dependency (StudentID, CourseID) → Grade is a full functional dependency because neither StudentID nor CourseID alone can determine the Grade.

* 1. **Partial Functional Dependency**

A **partial functional dependency** occurs when only part of a composite primary key determines a non-key attribute.

**Example:** In a table Enrollment(StudentID, CourseID, StudentName), StudentName depends only on StudentID and not the entire composite key (StudentID, CourseID). This is a partial functional dependency.

* 1. **Trivial Functional Dependency**

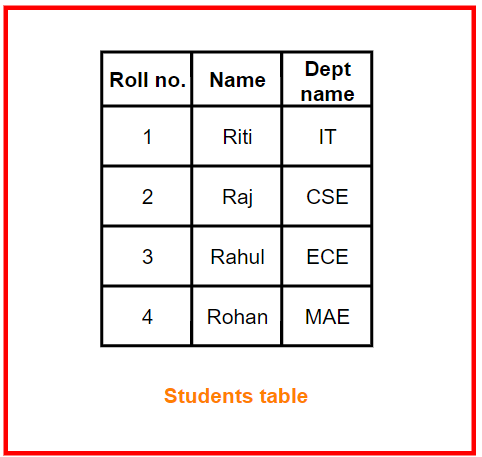
A **trivial functional dependency** happens when the dependent attribute is already part of the determinant attribute set.

**Example:** {Roll no., Name} → {Name} is a trivial functional dependency because Name is already part of the determinant (Roll no., Name).

* 1. **Non-trivial Functional Dependency**

A **non-trivial functional dependency** occurs when the dependent attribute is not part of the determinant attribute set.

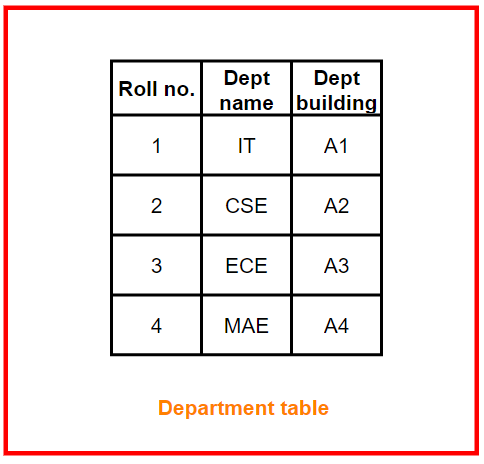
**Example:** {Roll no., Name} → {Dept name} is a non-trivial functional dependency because Dept name is not part of {Roll no., Name}.



* 1. **Transitive Dependency**

A **transitive dependency** happens when there is an indirect relationship between attributes. If A → B and B → C, then A → C is a transitive dependency.

**Example:** If Roll no. → Dept name and Dept name → Dept building, then Roll no. → Dept building is a transitive dependency.



* 1. **Multivalued Dependency**

In a **multivalued dependency**, the dependent attributes are independent of each other. If A → {B, C}, and there is no relationship between B and C, it is called a multivalued dependency.

**Example:** Roll no → {Name, Dept name} is a multivalued functional dependency because Name and Dept name are independent

**Denormalisation**

**Denormalization** is the process of combining normalized tables into fewer tables, adding redundancy, or optimizing a database structure to enhance performance. Unlike normalization, which aims to reduce data redundancy and improve data integrity, denormalization focuses on improving the read performance of a database by reducing the complexity of joins between tables.

**Importance:** Denormalization can be particularly beneficial in large databases where query performance and data retrieval speed are critical. It provides a way to balance the needs of maintaining data integrity with the practical requirements of quick data access and simpler query structures.

**Reasons for Denormalization**

Denormalization is implemented for various reasons, most of which revolve around improving the performance and efficiency of database operations:

**1. Performance Optimization**

Denormalization helps optimize the performance of a database by reducing the time required to perform queries, especially when dealing with large datasets. By reducing the need for multiple joins, data retrieval can be faster.

**Example:** Consider a sales database where data from multiple tables (customers, products, orders) needs to be frequently queried together. Combining these into a single denormalized table can reduce the time required for queries.

**2. Query Simplification**

By storing redundant data, denormalization can simplify complex SQL queries. Instead of joining multiple normalized tables, denormalized tables can store related data together, making the queries simpler and easier to write.

**Example:** A denormalized table storing order information along with customer and product details makes it easier to generate reports without the need for multiple joins.

**3. Reducing Joins**

One of the primary reasons for denormalization is to reduce the number of joins in queries. Joins can be resource-intensive, especially with large tables, so reducing them can enhance overall database performance.

**Example:** Instead of joining a product table with a category table every time product data is queried, denormalization could involve storing the category name directly in the product table.

**4. Enhancing Data Retrieval**

Denormalization can enhance data retrieval times by storing all required data in a single table or by precomputing frequently needed data. This is particularly useful for read-heavy applications where the speed of data retrieval is crucial.

**Example:** In an analytics dashboard that frequently needs to access sales data, a denormalized table that pre-aggregates sales figures can provide faster response times for reporting.

**Techniques for Denormalization**

Several techniques are used to achieve denormalization, allowing database administrators to optimize databases according to specific use cases:

**1. Adding Redundant Data**

One technique for denormalization is to store the same data in multiple places. This reduces the need to fetch related information from other tables, thus saving time and computational resources.

**Example:** Storing a customer’s address directly in the orders table instead of keeping it in a separate customer table.

CREATE TABLE orders (

order\_id INT,

customer\_name VARCHAR(100),

customer\_address VARCHAR(255),

order\_total DECIMAL(10, 2)

);

**2. Creating Summary Tables**

Summary tables are denormalized tables that store precomputed data, such as aggregates or summarized values. This helps speed up queries that would otherwise require expensive calculations.

**Example:** A table storing monthly sales totals for each product instead of calculating these totals on the fly during reporting.

CREATE TABLE monthly\_sales\_summary (

product\_id INT,

month DATE,

total\_sales DECIMAL(10, 2)

);

**3. Storing Derived Values**

Derived values are calculated fields that are stored in a table instead of computing them each time they are needed. This can improve the efficiency of queries.

**Example:** Storing the age of a customer as a column in the database instead of calculating it every time from the date of birth.

ALTER TABLE customers ADD COLUMN age INT;

**Advantages of Denormalization**

* **Faster Query Performance:** Denormalization can significantly improve query performance by reducing the need for complex joins.
* **Improved Readability:** Simplifies the database schema, making it easier for developers to understand and work with the data.
* **Better Reporting:** Summary tables can speed up report generation by providing precomputed data.

**Disadvantages of Denormalization**

* **Increased Data Redundancy:** Denormalization can lead to data duplication, making the database larger and harder to manage.
* **Higher Maintenance:** Updating data becomes more complex as changes need to be reflected in multiple places.
* **Risk of Inconsistency:** With data stored in multiple places, there is a higher risk of data becoming inconsistent across different tables