DBMS

Introduction

A Database Management System (DBMS) is a software system that is designed to manage and organize **data** in a structured manner within a **database.** It allows users to **create, modify, and query a database**, as well as manage the security and access controls for that database.

Key Features of DBMS

1. **Data modeling:** A DBMS provides tools for creating and modifying data models, which define the structure and relationships of the data.
2. **Data storage and retrieval:** A DBMS is responsible for storing and retrieving data from the database and can provide various methods for searching and querying the data.
3. **Concurrency control:** A DBMS provides mechanisms for controlling concurrent access to the database, to ensure that multiple users can access the data without conflicting with each other.
4. **Data integrity and security:** A DBMS provides tools for enforcing data integrity and security constraints, such as constraints on the values of data and access controls that restrict who can access the data.
5. **Backup and recovery:** A DBMS provides mechanisms for backing up and recovering the data in the event of a system failure.

Classification of DBMS

Relational Database Management System (RDBMS)

- Organizes data in tables with rows and columns.
- Establishes relationships between tables using primary and foreign keys.

Non-Relational Database Management System (NoSQL)

- Organizes data in diverse structures like key-value pairs, documents, graphs, or column-based formats.
- Tailored to handle large-scale, high-performance scenarios efficiently.

History of DBMS

- In the **1950s and early 1960s**, data storage and processing were primarily handled using **magnetic tapes and punched card decks.** Data processing tasks, such as payroll, involved sequentially reading data from tapes or card decks, making the process time-consuming and rigid. Accessing data directly was not possible due to the sequential nature of tapes.
- During the early 1960s, **Charles Bachman** developed the **Integrated Data Store (IDS),** one of **the first database management systems** based on the network data model. This significant advancement earned him the **Turing Award.** IDS allowed more complex data relationships and laid the groundwork for future DBMS developments.
- In the **late 1960s, IBM** introduced the **Integrated Management System**, based on the **hierarchical data model**. This period also saw the introduction of hard disks, allowing direct data access and enabling more flexible and efficient data management, marking a technological shift.
- **The 1970s** marked a revolutionary change with **Edgar Codd's** introduction of the **relational database model.** This model simplified data management and querying by organizing data into tables with rows and columns. Its simplicity and efficiency quickly made it the standard for database systems.
- During **the 1980s, IBM developed the Structured Query Language (SQL)** as part of the System R project, which became the standard **language for managing and querying relational databases**. This decade saw the emergence of commercial relational databases like IBM DB2, Oracle, and Ingres, which eventually replaced older network and hierarchical models.
- In the early **1990s**, the focus shifted to **decision support and querying applications.** Tools for analyzing large datasets became increasingly important, and parallel database products were introduced to handle the growing data volumes. Databases also began incorporating object-relational support to enhance functionality.

- The **1990s** witnessed the explosive growth of the World Wide Web, significantly **impacting database deployment.** Databases needed to support high transaction-processing rates, reliability, and 24/7 availability. Web interfaces to databases became essential to meet the demands of web applications.
- In the **2000s**, **XML** and **XQuery emerged** for complex data types and data exchange, while relational databases remained core for large-scale applications. Open-source databases like **PostgreSQL and MySQL** saw significant growth. Specialized databases such as column-stores for data analysis and highly parallel systems were developed. Distributed data-storage systems for large web platforms like Amazon and Google also emerged, along with advancements in streaming data management and data-mining techniques.

Different Types of DBMS

There are **various types of database management systems** based on database structures. We can arrange data in various formats for a variety of use cases. Let's see these types of DBMS one by one:

Centralized DBMS

In a centralized database, a single central database is used to serve data to multiple devices. Each user can access the database after authentication and be able to work with it.

Decentralized DBMS

In the decentralized database, all the data is collectively stored in multiple databases. All these databases are connected with the help of networking. To the end-user, this entire system appears like a single coherent system.

Relational DBMS

The relational database management system is also known as RDBMS. It is one of the types of DBMS which is widely used for commercial applications. It contains tables in which data is stored in the form of rows and columns like

an Excel sheet. Some of the tables possess the relationship among them and the data is retrieved with the help of join operation. This join operation helps us to get data from 2 or more tables with the help of logical queries.

NoSQL DBMS

NoSQL or non-relational databases are the most popular databases due to their high scalability and availability. In this type of database, the data is stored in collections, and it doesn't contain tables like relational databases.

Collection is simply the group of documents in which we have data with similar meanings and similar purposes.

In NoSQL, we can store data in key-value pairs as well as with the help of graphs. It increases productivity by a significant amount and comparatively, it is easy to work with them.

Hierarchical DBMS

In hierarchical databases, data is arranged in a tree-like format where we have a parent-child relationship between nodes. The parent can have many children, but children contain only one parent.

Network DBMS

The network database model has various nodes, and these nodes are connected with each other. These models are complex in nature. This model allows multiple parents for a single child node so we can create more complicated structures with it.

Object-Oriented DBMS

The Object-oriented database management system is one of the types of DBMS, in which data is stored in the objects. These objects are created from the classes. Classes are nothing but the description of an object. It is like object-oriented programming languages.

Some Popular Database Management Systems

There are several popular **Database Management Systems (DBMS)** that cater to different needs and preferences. Some widely used **DBMS** are:

1. MySQL:

- An **open-source relational database management system (RDBMS).**
- Known for its reliability, ease of use, and strong community support.
- Frequently used for web applications and small to medium-sized databases.

2. PostgreSQL:

- An **open-source object-relational database system.**
- Emphasizes extensibility and standards compliance.
- Suitable for complex applications and large-scale databases.

3. Microsoft SQL Server:

- A **relational database management system** developed by **Microsoft**.
- Offers a comprehensive suite of features and tools for enterprise-level applications.
- Commonly used in conjunction with Microsoft's .NET framework.

4. Oracle Database:

- A powerful and widely used **relational database management system.**
- Known for its scalability, security features, and support for complex transactions.
- Popular in large enterprises and critical business applications.

5. MongoDB:

- A leading **NoSQL database management system.**
- Stores data in flexible, JSON-like documents in a schema-less fashion.
- Ideal for handling large amounts of unstructured or semi-structured data.

6. SQLite:

- A self-contained, serverless, and zero-configuration **relational database engine.**
- Lightweight and suitable for embedded systems, mobile applications, and small-scale deployments.

7. Redis:

- An **in-memory data structure store** that is often used as a cache or message broker.
- Provides high-performance data storage and retrieval for key-value pairs. Commonly used in real-time applications as a caching mechanism.

Advantages of DBMS

1. **Security and Reliability:** DBMS ensures secure data storage through authentication and user authorization.
2. **Data Redundancy Reduction:** Normalization techniques help minimize and remove data redundancy.
3. **Multiple Data Views:** Provides different data views tailored for different users' needs.
4. **Backup and Recovery:** Facilitates data backup and recovery to prevent data loss.
5. **Integration with Programming Languages:** Can be integrated with languages like Python and Java to enhance database functionalities.

Disadvantages of DBMS

1. **Complexity:** DBMS systems can be complex to work with and manage.
2. **Cost of Hardware:** Involves significant cost for purchasing necessary hardware for data storage.
3. **Setup Time:** Setting up a DBMS can be time-consuming.
4. **Licensing Costs:** Many commercial DBMS products require paid licenses.
5. **Need for Skilled Staff:** Requires skilled technical staff, adding to the operational costs.

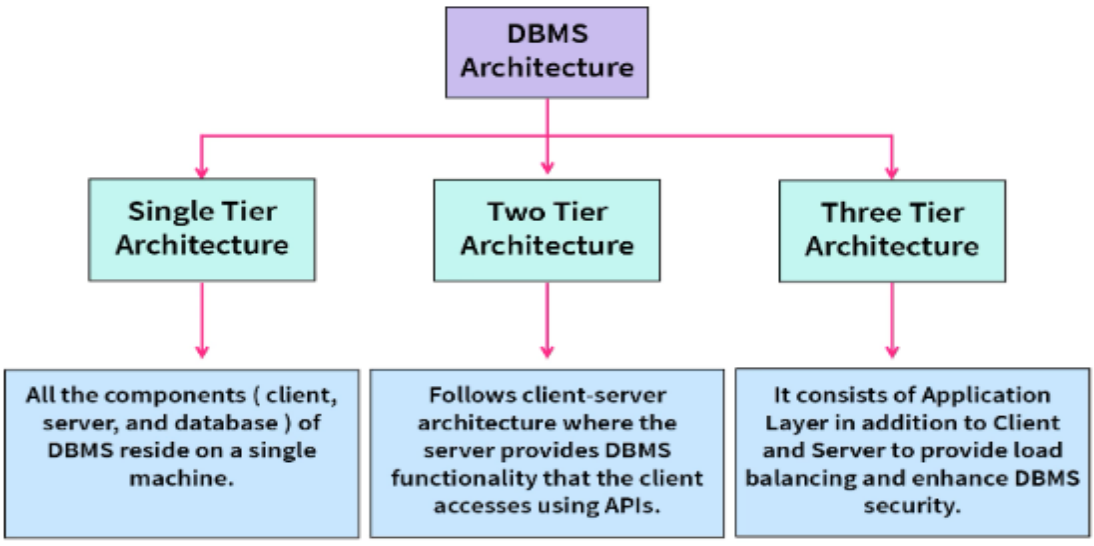
DBMS Architecture Types

Database management systems (DBMS) are organized into **multiple levels of abstraction** to ensure proper functioning. These layers describe both the design and the operations of the DBMS, facilitating a structured approach to database management. A **DBMS** is **not always directly accessible** by users or applications; instead, **various architectures** are employed based on how users connect to the database. These architectures are classified into **tiers**, defining the number of layers in the DBMS structure.

An **n-tier DBMS architecture** divides the entire DBMS into related but independent **layers**. For example, a **one-tier architecture** has a **single layer**, a **two-tier architecture** has **two layers**, and a **three-tier architecture** has **three layers.** As the **number of layers increases**, so does the **level of abstraction**, enhancing both the security and complexity of the DBMS. Importantly, these layers are independent of each other, meaning changes in one layer do not impact others, allowing for **modular** and **flexible** system management.

Now, let's look at the most common DBMS architectures:

- *Single Tier Architecture (One-Tier Architecture)*
- *Two-Tier Architecture*
- *Three-Tier Architecture*



1-Tier Architecture:

Definition: In a 1-tier architecture, the database is directly accessible by the user or application without any intermediary layers.

Explanation: The user interacts directly with the database. All the data, business logic, and presentation logic are handled within a single layer, often on a single machine.

Example: When learning SQL, you set up an SQL server and a database on your local system. This setup allows you to interact directly with the relational database and execute operations without a network connection. This direct interaction on a single machine is an example of 1-Tier DBMS architecture.

2-Tier Architecture:

Definition: A 2-tier architecture consists of two layers: the client (user interface) and the server (database).

Explanation: In this architecture, the client communicates directly with the database server. The business logic and database management are handled on the server side, while the user interface runs on the client side.

Example: When you visit a bank to withdraw cash, the banker enters your withdrawal amount and account details into a system. The client-side application then communicates with the server-side database to check your account balance. This interaction between the client application and the server database is an example of 2-Tier DBMS architecture.

3-Tier Architecture

Definition: A 3-tier architecture includes three layers: the presentation layer (user interface), the application layer (business logic), and the database layer.

Explanation: The user interface interacts with the application server, which in turn communicates with the database server. This separation allows for more

scalable and manageable systems, where the business logic is handled separately from data storage and user interaction.

Example: A web application, such as an online shopping site. The user's browser (presentation layer) interacts with a web server (application layer) that processes request and communicates with a database server (database layer) to fetch and store data.

Database

A **database** is an **organized collection of data** that is stored and **accessed electronically.** Databases are designed to efficiently **store, retrieve, and manage data.** They can be structured in various ways, such as in **tables (relational databases) or as documents (NoSQL databases).** A database typically supports operations like querying, updating, and managing data to ensure its integrity, security, and availability.

When we say a database is **"electronically accessed,"** it means that the data within the database can be **retrieved, updated, managed, and manipulated** using **electronic devices, typically computers.** This access is facilitated through various software tools and interfaces, allowing users and applications to interact with the data without needing to handle the physical storage media directly.

Why Use a Database?

Efficient Data Storage: Databases can store vast amounts of records effectively, ensuring data is organized and easily retrievable.

Quick Data Retrieval: Locating data in a database is fast and straightforward, enhancing productivity and decision-making.

Ease of Data Modification: Adding, updating, or deleting data in a database is simple, allowing for flexible data management.

Advanced Search Capabilities: Techniques like indexing and binary searching make it easy to search for specific data within a database.

Data Sorting and Importing: Databases enable quick and easy sorting of data and seamless import into other applications.

Multi-Access: Multiple users can access and use the same database simultaneously, promoting collaboration and efficiency.

Enhanced Security: Databases offer robust security measures, providing better protection for data compared to physical paper files.

Transaction Management: Databases ensure consistency and accuracy during transactions, maintaining data integrity.

Evolution of Databases

The history of databases spans over 50 years, evolving through several key stages:

Navigational Databases: Early systems like hierarchical databases (tree-like structure) and network databases (flexible relationship model) were the first to manage data.

Relational Databases: Gained popularity in the 1980s, offering more flexibility and efficiency in data management.

Object-Oriented Databases: Emerged in the 1990s, integrating object-oriented programming concepts with database systems.

NoSQL Databases: Developed in response to the need for faster processing of unstructured data due to the expansion of the internet.

Modern Databases: Cloud databases and self-driving databases are now used for faster processing and cloud-based data storage, reflecting the ongoing evolution of database technology.

Components of a Database

Databases are comprised of five key components, each playing a critical role in the DBMS environment:

Hardware: Physical devices like I/O devices, computers, and storage disks that interface between computers and real-world systems. They include data servers used to store database data.

Software: Programs that control and manage the database, including DBMS software, operating systems, network software, and applications for accessing data. These programs integrate with hardware to manage all data transactions.

Data: The raw information stored in the database, which can be texts, numbers, or binary data. It is the primary content that databases manage and process.

Procedures: Rules and guidelines for using the database, including creating and running databases and managing data. Procedures act as manuals for users.

Database Access Language (DAL): Programming languages like SQL used to read, update, and delete data from a database. DAL enables users to create databases, tables, and manipulate data efficiently.

Data Models in DBMS

Data models in DBMS help in understanding the design at conceptual, physical, and logical levels.

They describe how data is stored, accessed, and updated using symbols and text for clarity.

These models provide conceptual tools to represent the description of data, aiding developers in creating a physical database.

Types of Data Models

Hierarchical Model

- **Description:** Organizes data in a tree-like structure with records having a parent-child relationship.
- **History:** Developed by IBM in the 1950s.
- **Example:** Vehicle database classifying vehicles into two-wheelers and four-wheelers.
- **Drawback:** Supports only one-to-many relationships, limiting its modern application.

Network Model

- **Description:** Generalization of the hierarchical model allowing many-to-many relationships.
- **Structure:** Represents data as a graph with object types as nodes and relationships as edges.
- **Example:** College database linking departments to a director.
- **Advantages:** Efficient data access with multiple paths to a node.
- **Drawbacks:** Complex insertion and deletion processes.

Entity-Relationship (ER) Model

- **Description:** Uses ER diagrams to describe the database structure pictorially.
- **Components:**
- **Entity:** Anything with an independent existence (e.g., Car, Employee).
- **Entity Set:** Collection of similar entities (e.g., Set of students).
- **Attributes:** Properties defining entities (e.g., Employee Name).
- **Relationships:** Associations between entities (e.g., Employee working in a Company).
- **Example:** ER diagram showing the relationship between Employee and Company, with attributes for each entity.

Relational Model

Description: Represents the database as a collection of relations (tables) in rows and columns.

Example:

Stu. Id	Name	Branch
101	Naman	CSE
102	Saloni	ECE
103	Rishabh	IT
104	Pulkit	ME

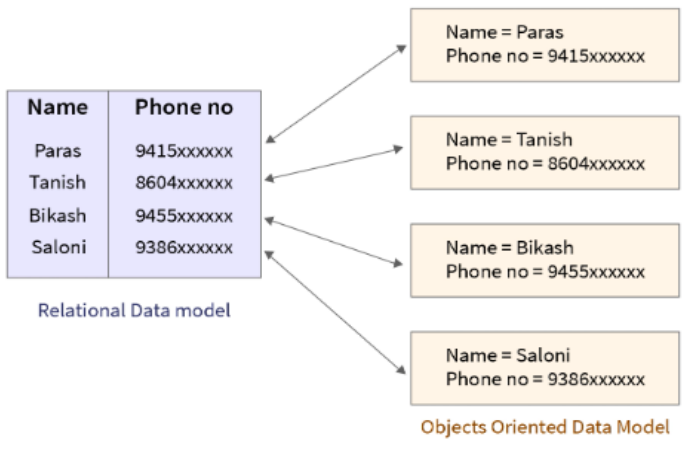
Attributes: Stu. Id, Name, and Branch.

Advantages: Simplifies data organization and access.

Object-Oriented Data Model

- **Description:** Combines object-oriented programming with relational data models.
- **Structure:** Data and their relationships are represented as objects.
- **Example:** Employee and Department objects linked by Department_Id.
- **Advantages:** Easily stores multimedia data (audio, video, images).

Object-Relational Data Model



Description: Integrates object-oriented and relational models.

Structure: Supports objects, classes, and tabular structures.

Advantages: Combines features of both models.

Drawbacks: Complex and difficult to handle.

Relational Model

The **relational model** in DBMS is an abstract model used to organize and manage the data stored in a database. It stores data in **two-dimensional inter-related tables,** also known as **relations** in which each row represents an entity, and each column represents the properties of the entity.

The key concepts of the relational model include:

Relation

Definition: A two-dimensional table used to store a collection of data elements.

Example:

```
Student
--------
Stu_ID | Name   | Branch
-------|--------|-------
101    | Naman  | CSE
102    | Saloni | ECE
103    | Rishabh| IT
104    | Pulkit | ME
```

Tuple

Definition: A row in the relation, representing a single data item or entity.

Example: The tuple (101, Naman, CSE) in the Student relation represents one student.

Attribute/Field

Definition: A column in the relation, representing a property that describes the relation.

Example: In the Student relation, Stu_ID, Name, and Branch are attributes.

Attribute Domain

- **Definition:** A set of predefined atomic values that an attribute can take.
- **Example:** For the Branch attribute in the Student relation, the domain might be {CSE, ECE, IT, ME}.

Degree

- **Definition:** The total number of attributes present in a relation.
- **Example:** The Student relation has a degree of 3 (attributes: Stu_ID, Name, Branch).

Cardinality

- **Definition:** The total number of tuples (rows) present in a relation.
- **Example:** The Student relation has a cardinality of 4 (four rows).

Relational Schema

- **Definition:** The logical blueprint of a relation, describing the design and structure, including table name, attributes, and their types.
- **Example:** STUDENT (Stu_ID INT, Name VARCHAR, Branch VARCHAR)

Relational Instance

- **Definition:** The collection of records present in the relation at a given time.
- **Example:** The current data in the Student table as shown above.

Relation Key

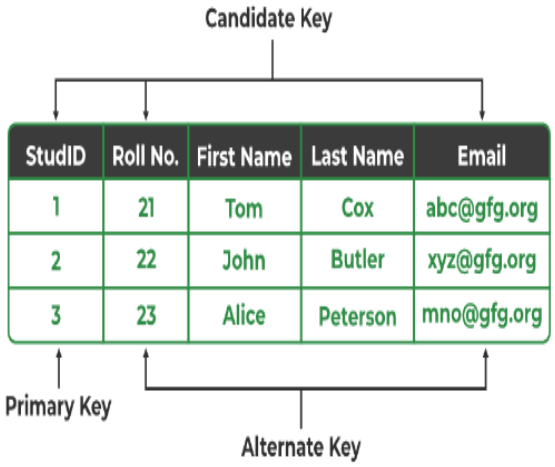
- **Definition:** An attribute or a group of attributes that can be used to uniquely identify an entity in a table or to determine the relationship between two tables.
- Relation keys are of 6 different types:

1. Candidate Key
2. Super Key
3. Composite Key
4. Primary Key
5. Alternate Key
6. Foreign Key

Keys in Relational Model

Keys are widely used to identify the **tuples(rows) uniquely** in the table. We also use keys to **set up relations** amongst various columns and tables of a relational database.

Candidate Key



The minimal set of attributes that **can uniquely identify a tuple** is known as a **candidate key.**

Candidate key is a **super key** with **no repeated data** and can contain **NULL values.**

Every **table** must have **at least a single or more candidate keys** but can have only **one primary key**.

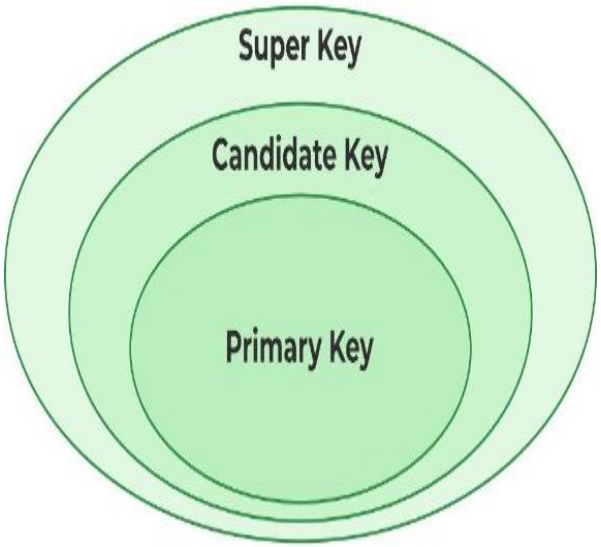
Primary Key

- There can be **more than one candidate key** in relation out of which one can be chosen as the **primary key.**
- Primary key is a **candidate key** with **no repeated data** and cannot contain **NULL values.**
- A **primary key** can be composed of **multiple columns**, and this is known as a **composite primary key.** It is used to uniquely identify records when a single column is insufficient for uniqueness. This helps maintain data integrity and ensures that each record can be uniquely identified by the combination of values from multiple columns.

```sql
CREATE TABLE StudentCourses (
    Student_ID INT,
    Course_ID VARCHAR(10),
    Enrollment_Date DATE,
    PRIMARY KEY (Student_ID, Course_ID)
);
```

A **StudentCourses** table with **primary keys (Student_ID, Course_ID)**

Super Key



The **set of attributes** that can **uniquely identify a tuple** is known as **Super Key.**

Adding zero or more attributes to the **candidate key** generates the **super key.**

A **candidate key is a super key** but vice versa is not true.

Super Key must **contain Unique Values** but can contain **NULL Values.**

Example: STUDENTS(STUD_NO, STUD_MAIL)

Alternate Key

An **alternate key** in a relational database is **any candidate key** that is **not chosen** to be the **primary key**. Since a table can have multiple candidate keys, the ones that are not selected as the primary key are referred to as alternate keys.

Example:

Consider the following `Employee` table:

EmployeeID	SSN	Email	FirstName	LastName
1	123-45-6789	john.doe@example.com	John	Doe

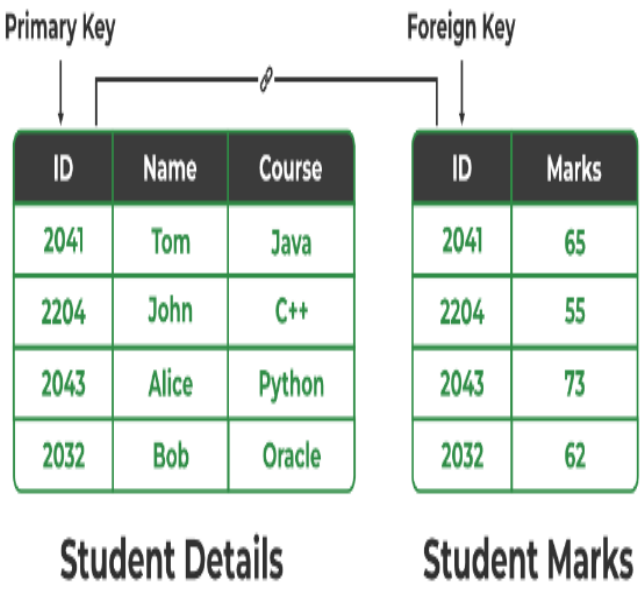
Candidate Keys: EmployeeID, SSN, and Email

Primary Key: EmployeeID

Alternate Keys: The remaining candidate keys, SSN & Email.

Foreign Key

- It is a key that acts as a **primary key** in one table, and it acts as **secondary key** in another table.
- It **combines two or more relations** (tables) at a time.
- They act as a **cross-reference** between the tables.
- **Primary Key: Student Details (ID) = Foreign Key: Student Marks(ID)**



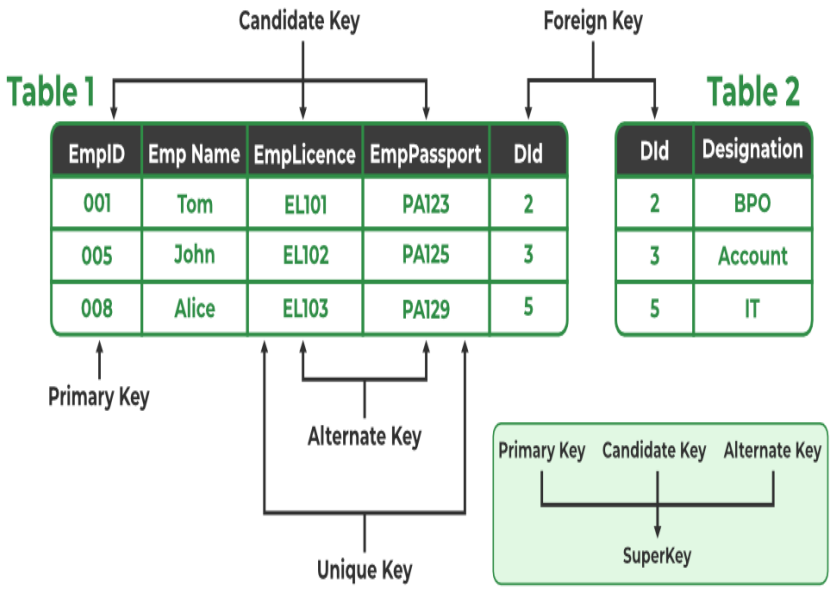
Foreign key can **contain null values** because a **null value** represents a **missing or unknown value,** which means that the specific row does not need to relate to a row in the referenced table.

Foreign key can **contain duplicate values** because the same foreign key value can be used in multiple rows to establish a relationship to the same row in the referenced table.

Composite Key

A **composite key is a primary key** that consists of **two or more columns** used together to uniquely identify a record in a table. This is useful when **a single column is not sufficient to uniquely identify rows.**

Summary



Primary Key (EmpID): Uniquely identifies each row in **Table 1.**

Unique Key (EmpPassport): Uniquely identifies rows in **Table 1** but can differ from the primary key.

Alternate Key (EmpLicence): A candidate key not chosen as the primary key.

Candidate Key (EmplD, EmpLicence): A set of fields that can uniquely identify row in **Table 1**; includes the primary key.

Foreign Key (Dld): Ensures referential integrity by linking **Dld** in **Table 2 to Dld in Table 1.**

Super Key: A set of one or more columns that can uniquely identify rows in a table. All keys in the image are super keys.

Relational Calculus

Relational Calculus is a **declarative query language,** which means it **tells** the system **what data to retrieve,** not how to retrieve it. It is **based on predicate logic** and has **two types:**

1. Tuple Relational Calculus (TRC)

- **TRC** in DBMS **uses a tuple variable (t)** that goes to each row of the table and checks if the **predicate is true or false** for the given row. Depending on the given predicate condition, it **returns the row or part of the row.**
- **Syntax:** {T | condition(T)}
- **Example:** {T | T ∈ Employee AND T.age > 25}
  ➔ Finds all tuples T in Employee where age > 25.

2. Domain Relational Calculus (DRC)

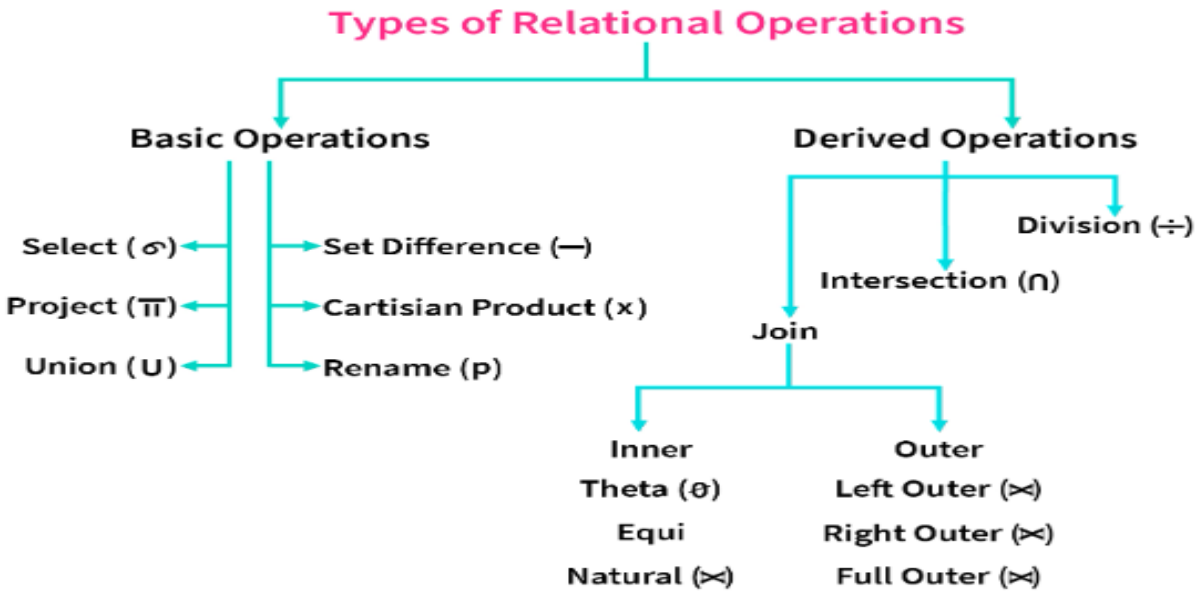
- **DRC** uses **domain Variables to get the column values** required from the database based on the predicate expression or condition.
- **Notation:** {<d1, d2, ..., dn> | condition (d1, d2, ..., dn)}
- **Example:** {<n, a> | ∃e (e ∈ Employee AND e.name = n AND e.age = a AND a > 25)}
  ➔ Finds names (n) and ages (a) of **employees(name, age)** older than 25.

Note: ***In TRC, the entire tuple is considered and returned if it satisfies the given predicate whereas in DRC, only the specified attribute values that satisfy the predicate are returned.***

Relational Algebra

Relational Algebra is a **procedural query language,** which means it tells the system **how to perform operations** to retrieve the desired data.

Relational Algebra came in **1970** and was given by **Edgar F. Codd (Father of DBMS).** It is also known as **Procedural Query Language (PQL)** as in **PQL,** a programmer/user must mention two things, **"What to Do" and "How to Do".**



Basic Operations

1. Select (σ)

- **Select operation** is used to **retrieve tuples(rows)** from the table where the given condition is satisfied. It is a **unary operator** means it requires only one operand.
- **Notation:** σ p(R)
  - ➔ Where **σ** is used to represent **SELECTION**
  - ➔ **R** is used to represent **RELATION**
  - ➔ **p** is the **logic formula**
- **Example:** σ AGE=20 (STUDENT)
  - ➔ **selects the row(s) from STUDENT Relation where "AGE" is 20**

Projection (π)

Purpose: Select specific columns.

Notation: π_{column1, column2, ...}(Relation)

Example: π_{name, age}(Employee)

Selects only the name and age columns from the Employee relation.

Union (∪)

Purpose: Combine two relations with the same attributes.

Notation: Relation1 ∪ Relation2

Example: π_{name}(Employee) ∪ π_{name}(Manager)

Combines names from both Employee and Manager relations.

Set Difference (−)

Purpose: Find rows in one relation but not in another.

Notation: Relation1 − Relation2

Example: π_{name}(Employee) − π_{name}(Manager)

Finds names of employees who are not managers.

Cartesian Product (×)

Purpose: Combine all rows from two relations.

Notation: Relation1 × Relation2

Example: Employee × Department

Combines each row of Employee with each row of Department.

Rename (ρ)

Purpose: Rename a relation or its attributes.

Notation: ρ_{new_relation_name (new_attribute_names)}(Relation)

Example: ρ_{E(emp_id, emp_name)}(Employee)

Renames Employee relation to E with attributes emp_id and emp_name.

Join (⋈)

Purpose: Combine related rows from two relations based on a common attribute.

Notation: Relation1 ⋈_{condition} Relation2

Example: Employee ⋈_{Employee.dept_id = Department.dept_id} Department

Joins Employee and Department relations where dept_id matches.