What is Database?

- A database is an organized collection of data that can be modified, retrieved, or updated.
- The data, stored in the database, is in the row and column format, which is called a **table**.
- Every website, which needs us to sign up, uses a database.
- There is **no internet** without databases.

Types of Databases

- **Object-oriented databases**
  - Object-oriented databases (OODB) are databases that represent data in the form of **objects** and **classes**. In object-oriented terminology, an object is a real-world entity, and a class is a collection of objects. Example - **PostgreSQL**

- **Relational databases**
  - A relational database is a collection of data items with pre-defined **relationships** between them. These items are organized as a set of **tables** with **columns** and **rows**. Example - **MySQL**

- **Distributed databases**
  - A distributed database (DDB) is an integrated collection of databases that is **physically distributed** across sites in a computer network. Example - **Cassandra**

- **Hierarchical databases**
  - A hierarchical database is a data model in which data is stored in the form of records and organized into a **tree-like** structure, or parent-child structure, in which on have many child nodes connected through links. Example - **IBM** Informati System (**IMS**)

SCALER

What is DBMS?

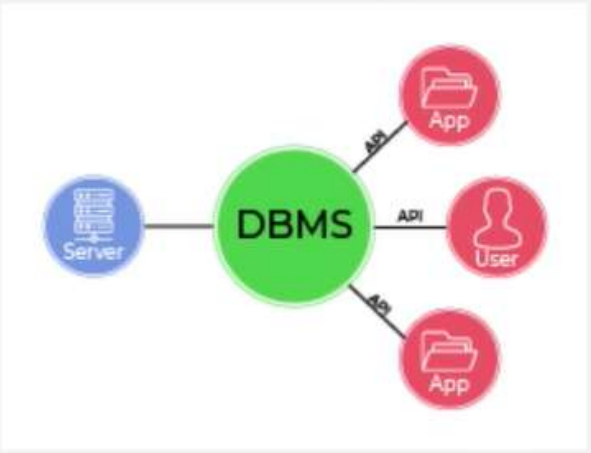
- Database Management System (DBMS) is a **software** for storing and retrieving users' data while considering appropriate security measures.

- It consists of a group of programs which **manipulate** the database.

- The DBMS accepts the request for data from an application and instructs the operating system to provide the specific data.

- In large systems, a DBMS helps users and other third-party software to store and retrieve data.

- DBMS allows users to create their own databases as per their requirement. It provides an interface between the data and the software application.

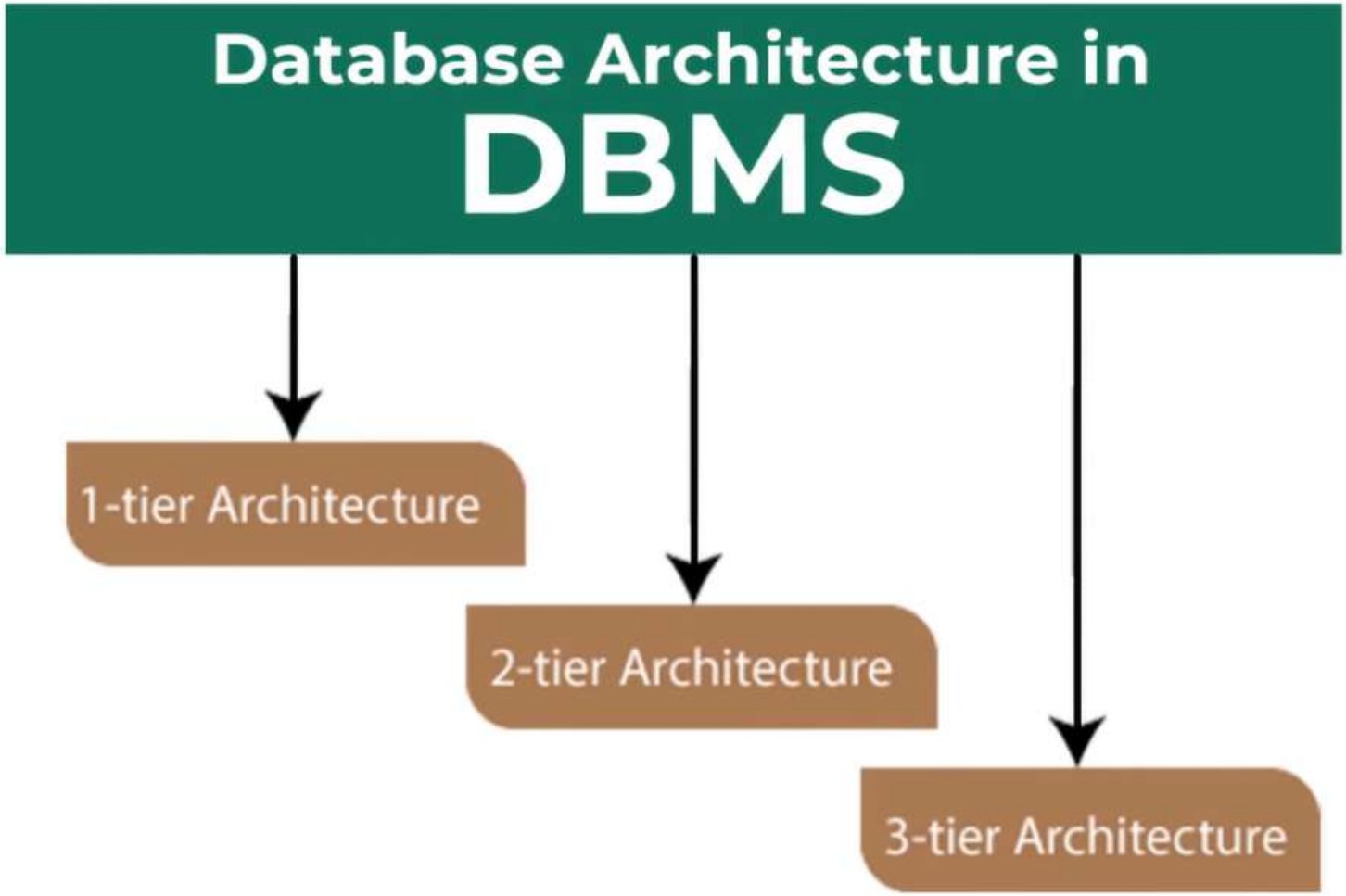


Characteristics of DBMS

- Real-world entity
- Relational databases
- Structured query language
- Isolation of data and application
- ACID properties
- Multiuser and concurrent access
- Transactional processing
- Less redundancy and consistency
- Data security and integrity

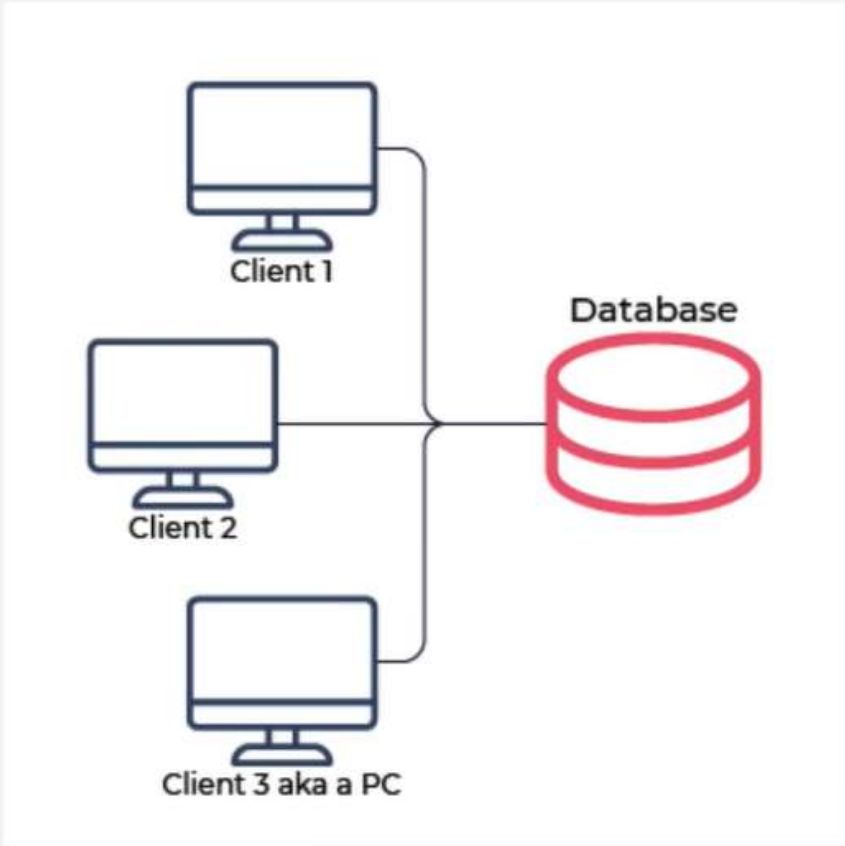
Applications Of DBMS

Sector	Use Of DBMS
Banking	For customer information, account activities, payments, deposits, loans, etc.
Airlines	For reservations and schedule information.
Universities	For student information, course registrations, colleges and grades.
Telecommunication	It helps to keep call records, monthly bills, maintaining balances, etc.
Finance	For storing information about stock, sales, and purchases of financial instruments like stocks and bonds.
Sales	Use for storing customer, product & sales information.
Manufacturing	It is used for the management of supply chain and for tracking production of items. Inventories status in warehouses.
HR Management	For information about employees, salaries, payroll, dec generation of paychecks, etc.

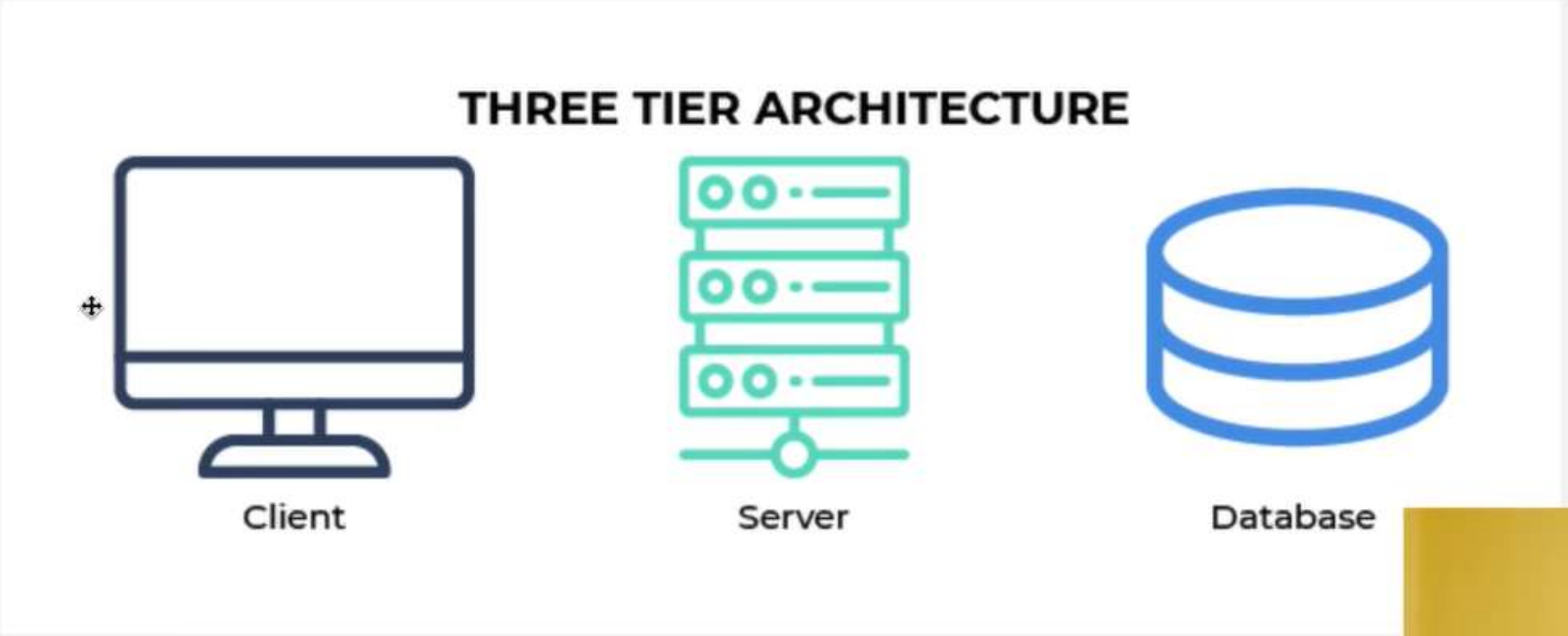

Database Architecture in
DBMS
1-tier Architecture
2-tier Architecture
3-tier Architecture

- **1-Tier Architecture :** 1 Tier Architecture in DBMS is the simplest architecture of Database in which the client, server, and Database all reside on the same machine. A simple one tier architecture example would be anytime you install a Database in your system and access it to practice SQL queries. But such architecture is rarely used in production.

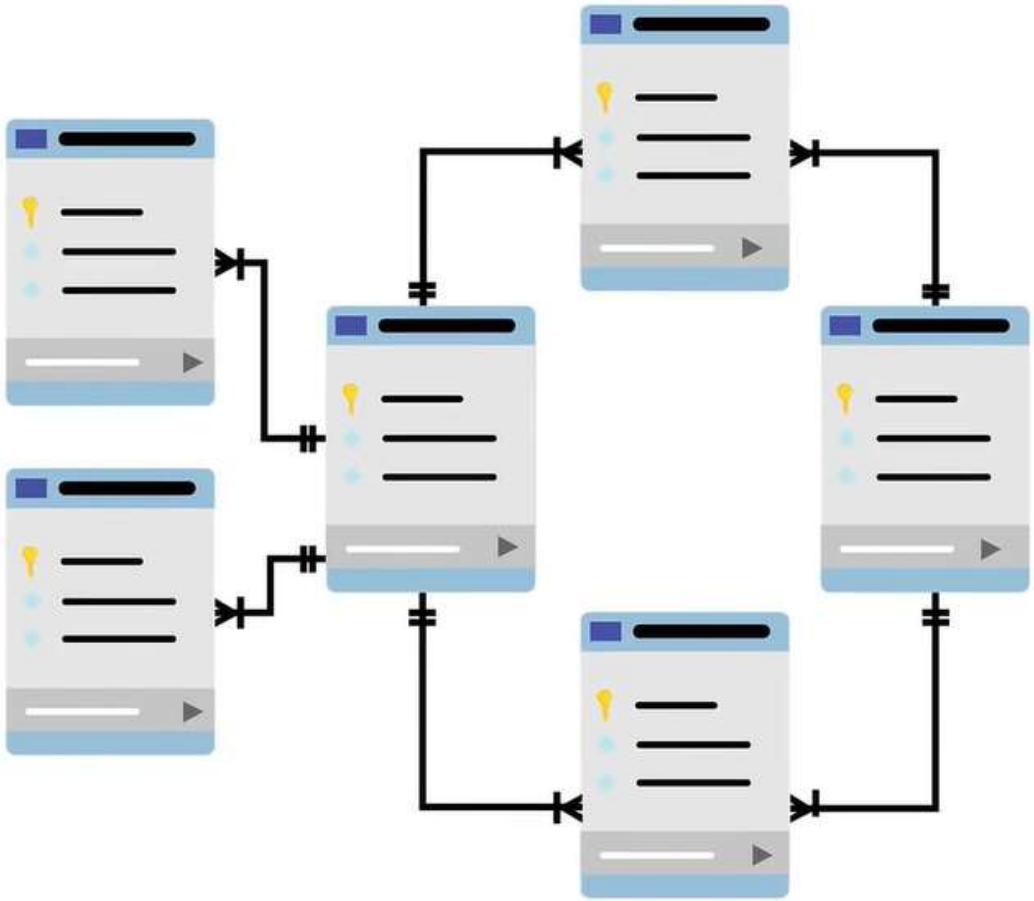
- **2-Tier Architecture :** A 2 Tier Architecture in DBMS is a Database architecture where the presentation layer runs on a client (PC, Mobile, Tablet, etc.), and data is stored on a server called the second tier. Two tier architecture provides added security to the DBMS as it is not exposed to the end-user directly. It also provides direct and faster communication.



- **3-Tier Architecture :** A 3 Tier Architecture in DBMS is the most popular client server architecture in DBMS in which the development and maintenance of functional processes, logic, data access, data storage, and user interface is done independently as separate modules. Three Tier architecture contains a presentation layer, an application layer, and a database server.



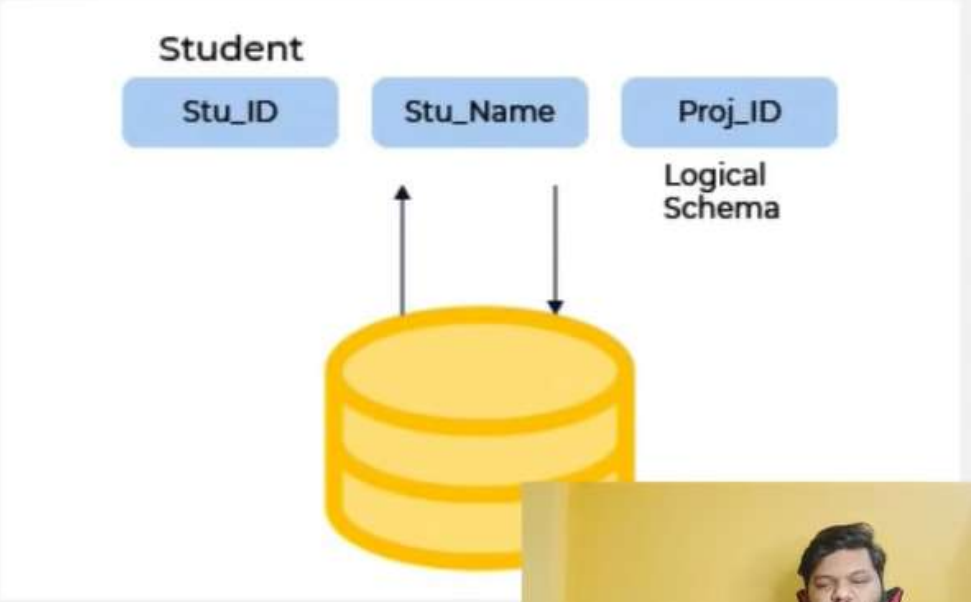
DBMS Schemas



A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.

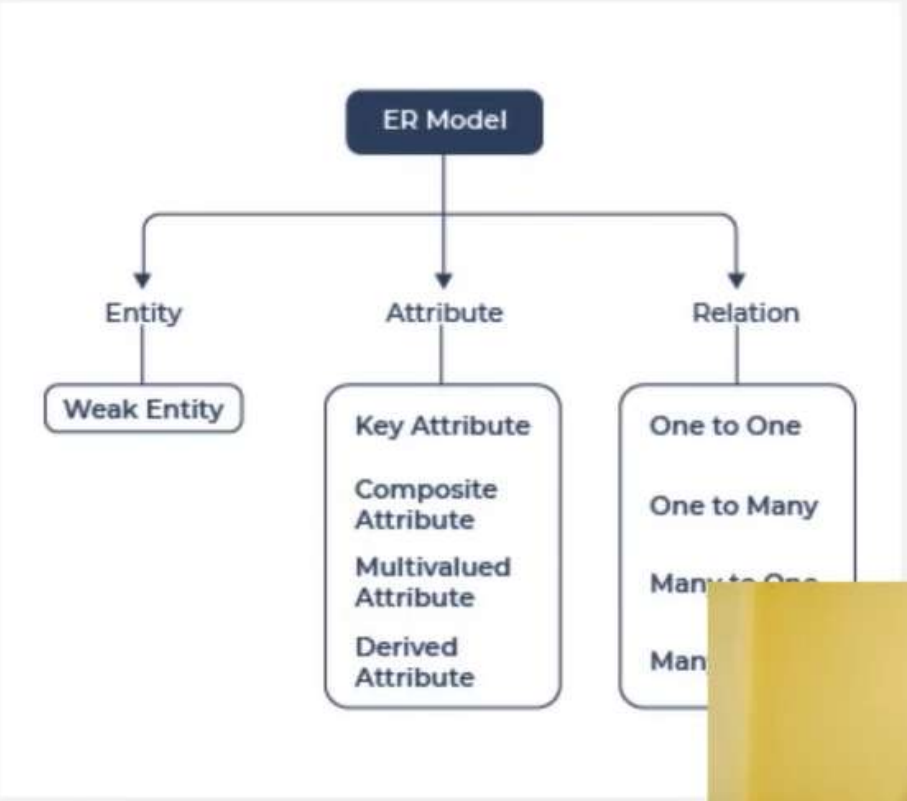
A database schema can be divided broadly into two categories −

- **Physical Database Schema** − This schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
- **Logical Database Schema** − This schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

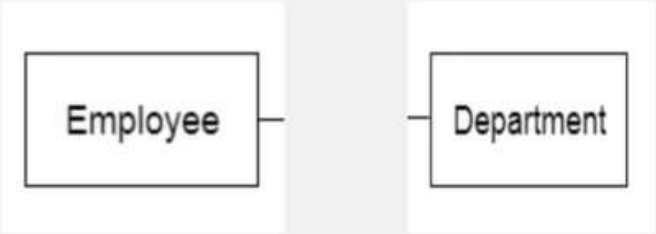


ER Model

- ER model stands for an **Entity-Relationship** model. It is a high-level data model.
- This model is used to define the data elements and relationship for a specified system.
- It develops a **conceptual** design for the database. It also develops a very simple and easy to design view of data.
- In ER modeling, the database structure is portrayed as a diagram called an **entity-relationship diagram**.

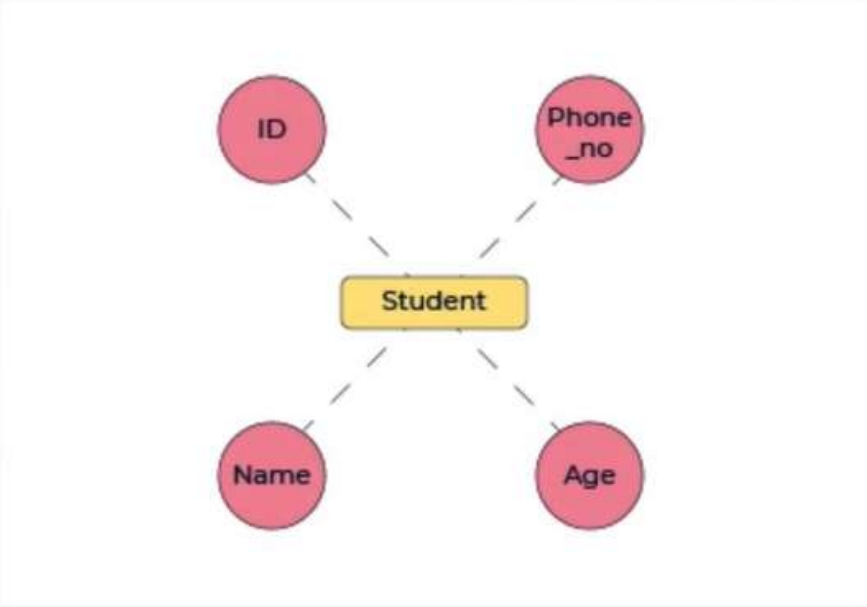


- **Entity :**
  - An entity may be any object, class, person or place.
  - In the ER diagram, an entity can be represented as **rectangles**.
  - Consider an organization as an example- manager, product, employee, department etc. can be taken as an entity.

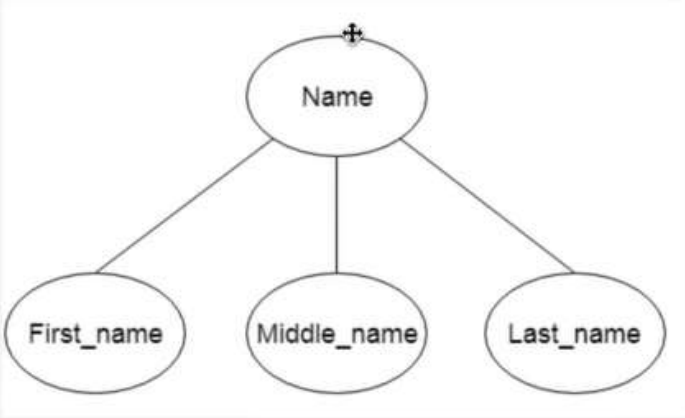


- **Weak Entity :**
  - An entity that depends on **another entity** called a weak entity. The weak entity doesn't contain any key attribute of its own. The weak entity is represented by a **double rectangle**.

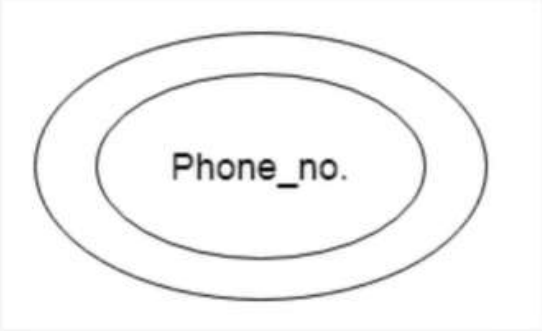
- **Attribute :** The attribute is used to describe the property of an entity. Eclipse is used to represent an attribute. **For example**, id, age, contact number, name, etc. can be attributes of a student.
    - **Key Attribute:** The key attribute is used to represent the **main** characteristics of an entity. It represents a primary key. The key attribute is represented by an eclipse with the **text underlined**.



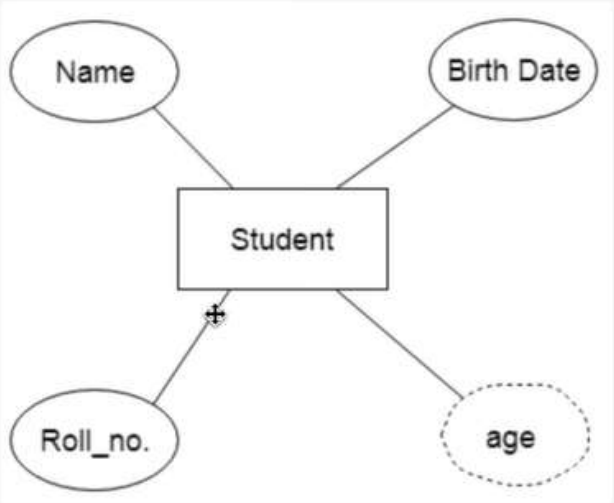
- **Composite Attribute:** An attribute that composed of many other attributes is known as a composite attribute.



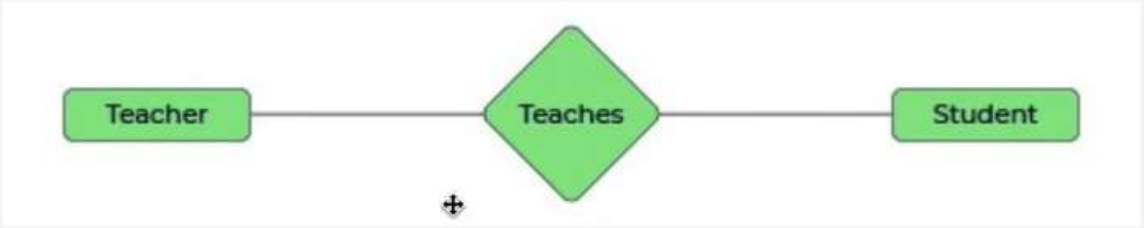
- **Multivalued Attribute:** An attribute can have more than one value. These attributes are known as a multivalued attribute.



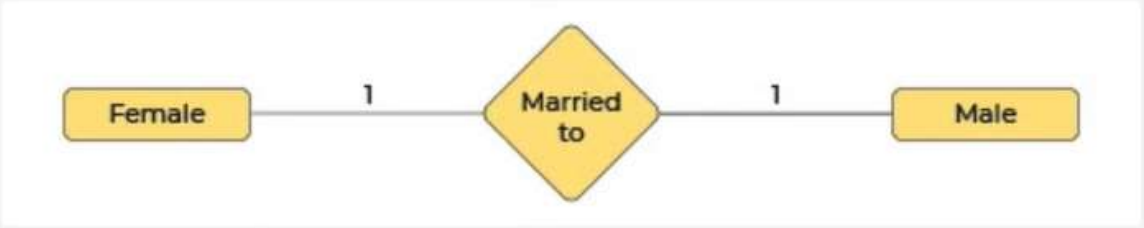
- **Derived Attribute:** An attribute that can be derived from other attribute is known as a derived attribute. It can be represented by a dashed eclipse.



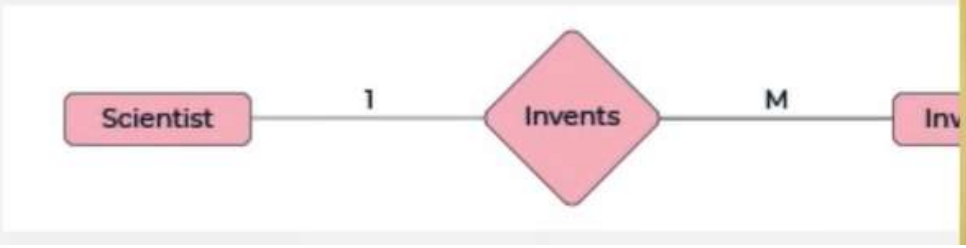
- **Relationship:** A relationship is used to describe the relation between entities. **Diamond** symbol is used to represent the relationship.



  - **One-to-One Relationship:** When only one instance of an entity is associated with the relationship, then it is known as one to one relationship.

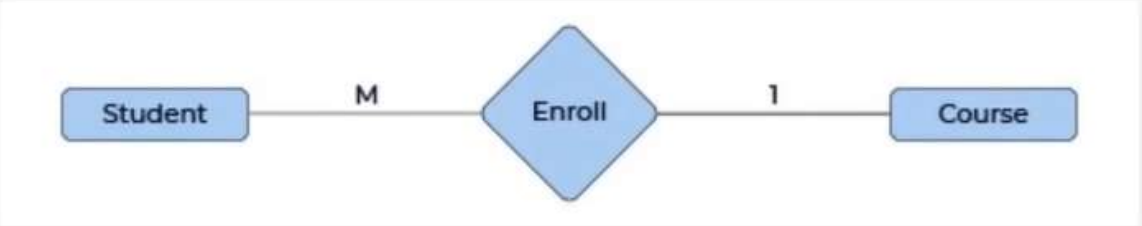


  - **One-to-Many Relationship:** When only one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then this is known as a one-to-many relationship.

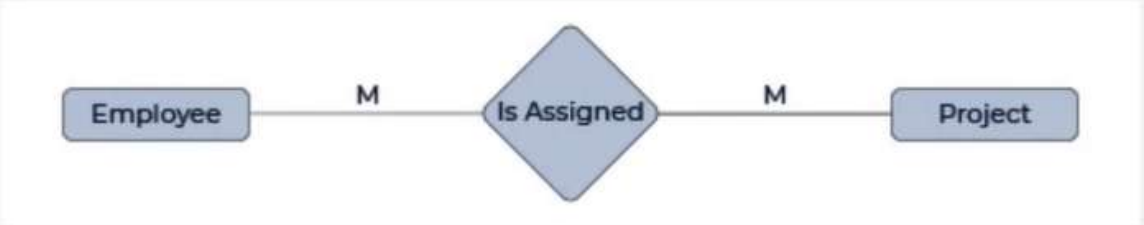


SCALER

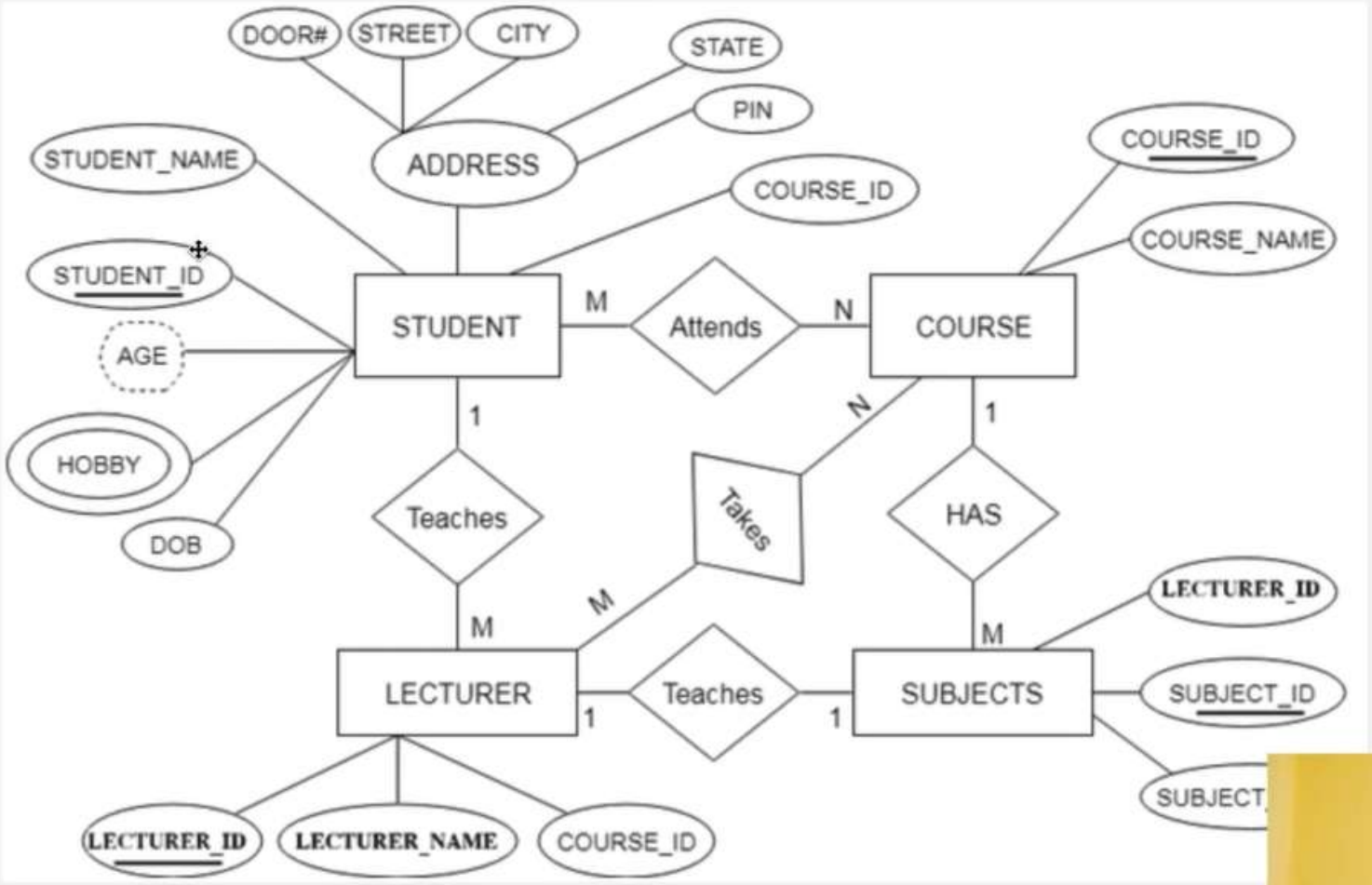
- **Many-to-One Relationship:** When more than one instance of the entity on the left, and only one instance of an entity on the right associates with the relationship then it is known as a many-to-one relationship.



- **Many-to-Many Relationship:** When more than one instance of the entity on the left, and more than one instance of an entity on the right associates with the relationship then it is known as a many-to-many relationship.

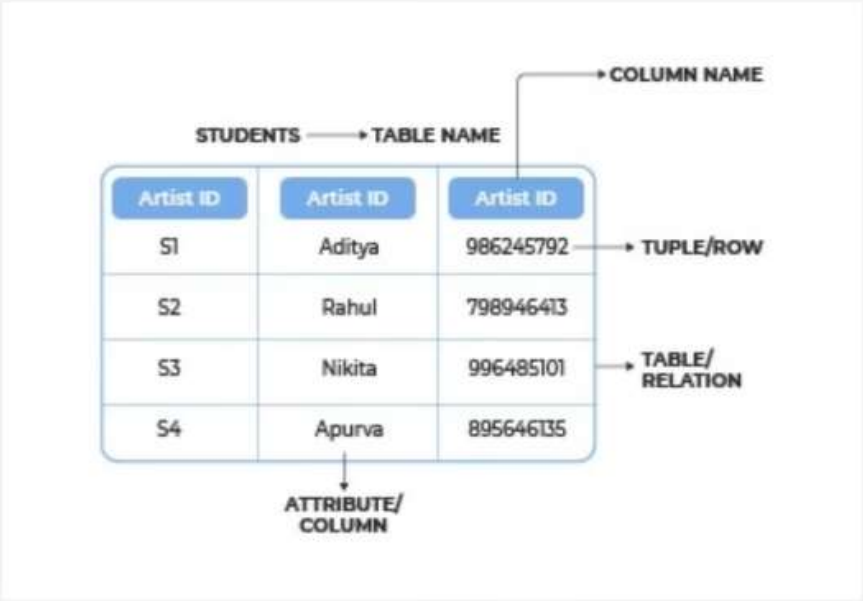


- **ER Diagram Sample Case Study**



What is **RDBMS?**

- A **relational** database is a type of database that stores and provides access to data poi that are related to one another.
- Relational databases are based on the **relational model**, an intuitive, straightforward way of representing data in tables.
- In a relational database, each **row** in the **table** is a record with a **unique ID** called the key.
- The columns of the table hold attributes of the data, and each record usually has a value for each attribute, making it easy to establish the relationships among data points.
- The major DBMS like SQL, **My-SQL**, ORACLE are all based on the principles of relational DBMS.



Advantages of RDBMS :

- ***Easy to manage:*** Each table can be independently manipulated without affecting others.
- ***Security:*** It is more secure consisting of multiple levels of security. Access of data shared can be limited.
- ***Flexible:*** Databases can easily be extended to incorporate more records, thus providing greater scalability. Also, facilitates easy application of SQL queries.
- ***Users:*** RDBMS supports client-side architecture storing multiple users together.
- Facilitates storage and retrieval of large amount of data.
- ***Easy Data Handling:***
  - Data fetching is faster because of relational architecture.
  - Data redundancy or duplicity is avoided due to **keys**, **indexes**, and **normalization** principles.
  - Data consistency is ensured because RDBMS is based on **ACID properties** for data transactions(Atomicity Consistency Isolation Durability).
- ***Fault Tolerance:*** Replication of databases provides simultaneous access and helps the system recover in case of disasters, such as power failures or sudden shutd

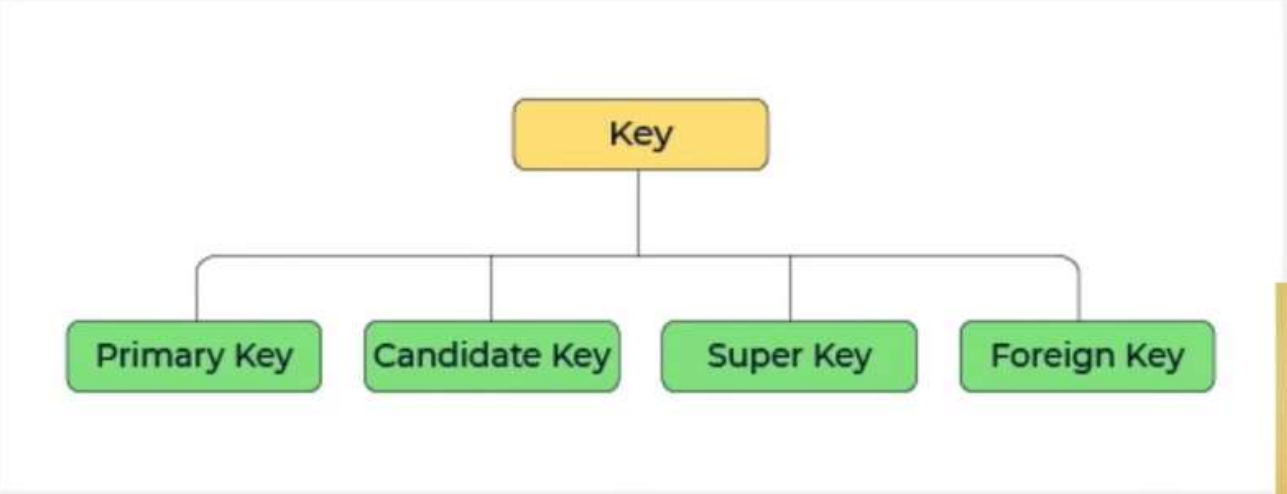
Concept Of Keys in **RDBMS?**

Why we need a Key?

Here are some reasons for using sql key in the DBMS system.

- Keys help you to **identify** any row of data in a table. In a real-world application, a table could contain thousands of records. Moreover, the records could be duplicated. Keys in RDBMS ensure that you can **uniquely** identify a table record despite these challenges.
- Allows you to establish a **relationship** between and identify the relation between tables
- Help you to enforce identity and **integrity** in the relationship.

Types of Keys?

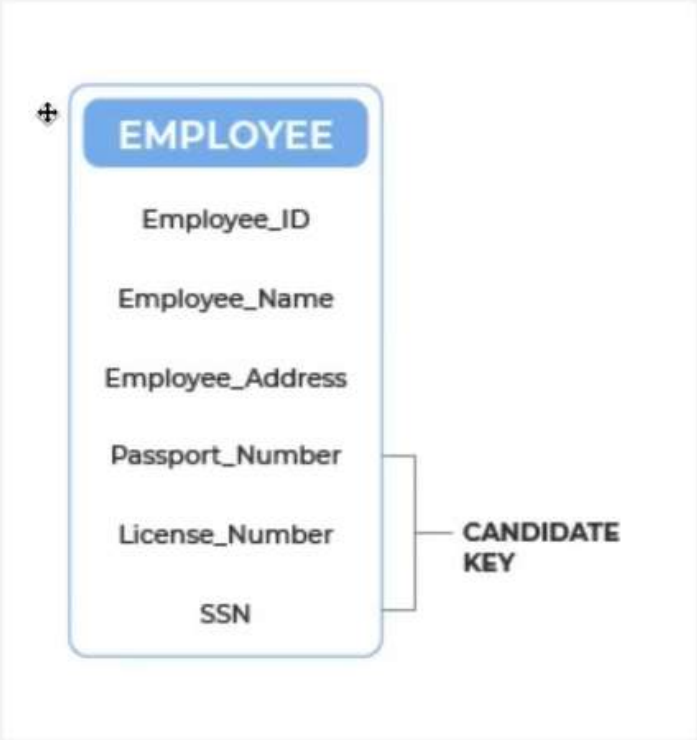


Primary Key :

- PRIMARY KEY in DBMS is a column or group of columns in a table that **uniquely** identify every row in that table.
- The Primary Key can't be a **duplicate** meaning the same value can't appear more than once in the table.
- A table cannot have more than one primary key.

Candidate Key :

- A candidate key is an attribute or set of an attribute which can uniquely identify a tuple.
- The **remaining attributes** except for primary key are considered as a candidate key.
- The candidate keys are as strong as the primary key.
- **For example:** In the EMPLOYEE table, id is best suited for the primary key. Rest of the attributes like SSN, Passport_Number, and License_Number, etc. are considered as a candidate key.

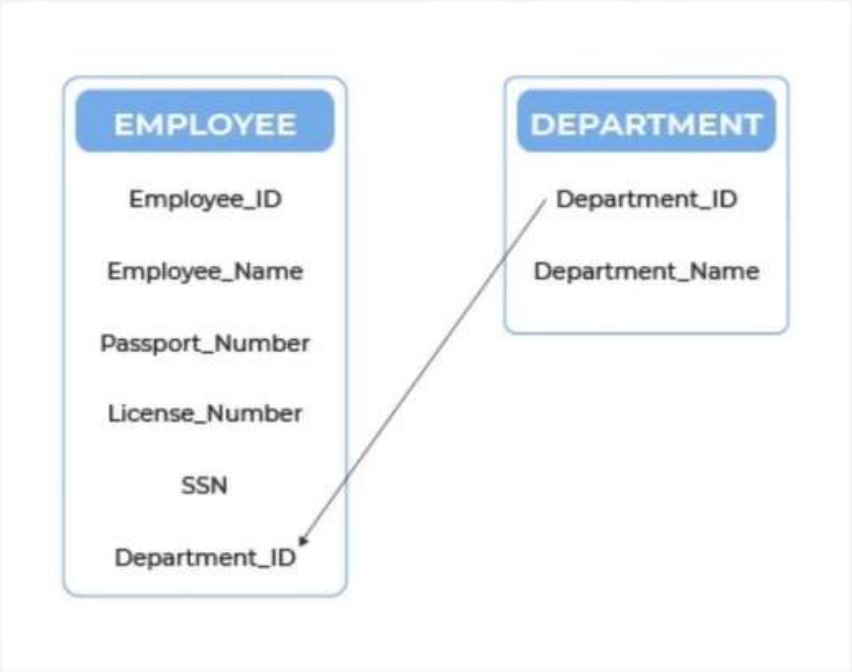


Super Key :

- Super key is a set of an attribute which can uniquely identify a tuple.
- Super key is a **superset** of a candidate key.
- **For example:** In the above EMPLOYEE table, for(EMPLOEE_ID, EMPLOYEE_NAME) the name of two employees can be the same, but their EMPLYEE_ID can't be the same. Hence, this combination can also be a key.
- The super key would be EMPLOYEE-ID, (EMPLOYEE_ID, EMPLOYEE-NAME), etc.

Foreign Key :

- Foreign keys are the column of the table which is used to point to the **primary key of another table**.
- In a company, every employee works in a specific department, and employee and department are two different entities. So we can't store the information of the department in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id as a new attribute in the EMPLOYEE table.
- Now in the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



Transactions

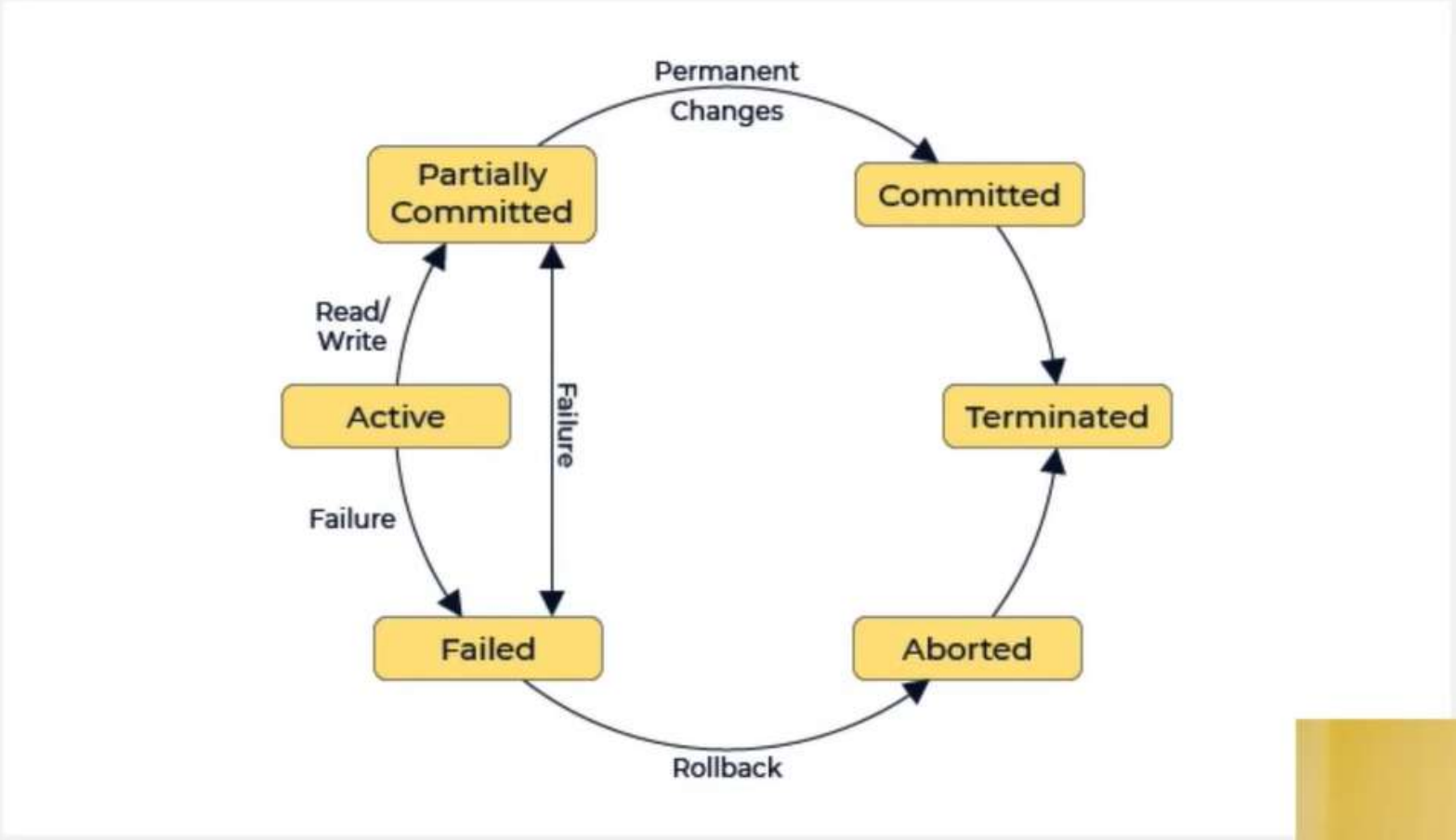
Transactions : A transaction is a **single logical unit** of work which accesses and possibly modifies the contents of a database. **For example -** You are transferring money from your bank account to your friend's account, the set of operations would be like this:

Simple Transaction Example

- Read your account balance
- Deduct the amount from your balance
- Write the remaining balance to your account
- Read your friend's account balance
- Add the amount to his account balance
- Write the new updated balance to his account

This whole set of operations can be called a **transaction**. Although I have shown you read, write and update operations in the above example but the transaction can have operations like **read, write, insert, update, delete.**

Transaction States



Transaction States:

- **Active State** - As we have discussed in the DBMS transaction introduction that a transaction is a sequence of operations. If a transaction is in execution then it is said to be in active state. It doesn't matter which step is in execution, until unless the transaction is executing, it remains in active state.

- **Failed State** -If a transaction is executing and a failure occurs, either a hardware failure or a software failure then the transaction goes into failed state from the active state.

- **Partially Committed State** - As we can see in the above diagram that a transaction goes into "partially committed" state from the active state when there are read and write operations present in the transaction. A transaction contains number of read and write operations. Once the whole transaction is successfully executed, the transaction goes into partially committed state where we have all the read and write operations performed on the main memory (local memory) instead of the actual database.

  The reason why we have this state is because a transaction can fail during execution so if we are making the changes in the actual database instead of local memory, database may be left in an inconsistent state in case of any failure. This state helps us to rollback the changes database in case of a failure during execution.

- **Committed State** - If a transaction completes the execution successfully then all the changes made in the local memory during partially committed state are permanently stored in the database. You can also see in the above diagram that a transaction goes from partially committed state to committed state when everything is successful.

- **Aborted State** - As we have seen above, if a transaction fails during execution then the transaction goes into a failed state. The changes made into the local memory (or buffer) are rolled back to the previous consistent state and the transaction goes into aborted state from the failed state. Refer the diagram to see the interaction between failed and aborted state.

ACID Properties

- **Atomicity** : By this, we mean that either the entire transaction takes place at once or doesn't happen at all. There is no midway i.e. transactions do not occur partially. Each transaction is considered as one unit and either runs to completion or is not executed at all. It involves the following two operations.
  - **Abort**: If a transaction aborts, changes made to database are not visible.
  - **Commit**: If a transaction commits, changes made are visible.

  Atomicity is also known as the **'All or nothing rule'**.

Consider the following transaction T consisting of T1 and T2: Transfer of 100 from account X to account Y.

Before: X : 500	Y: 200
Transaction T	
T1	T2
Read (X) X: = X – 100 Write (X)	Read (Y) Y: = Y + 100 Write (Y)
After: X : 400	Y : 300

If the transaction fails after completion of T1 but before completion of T2.(say, after writ
write(Y)), then amount has been deducted from X but not added to Y. This results in an i
database state. Therefore, the transaction must be executed in entirety in order to ensu
database state.

SCALER

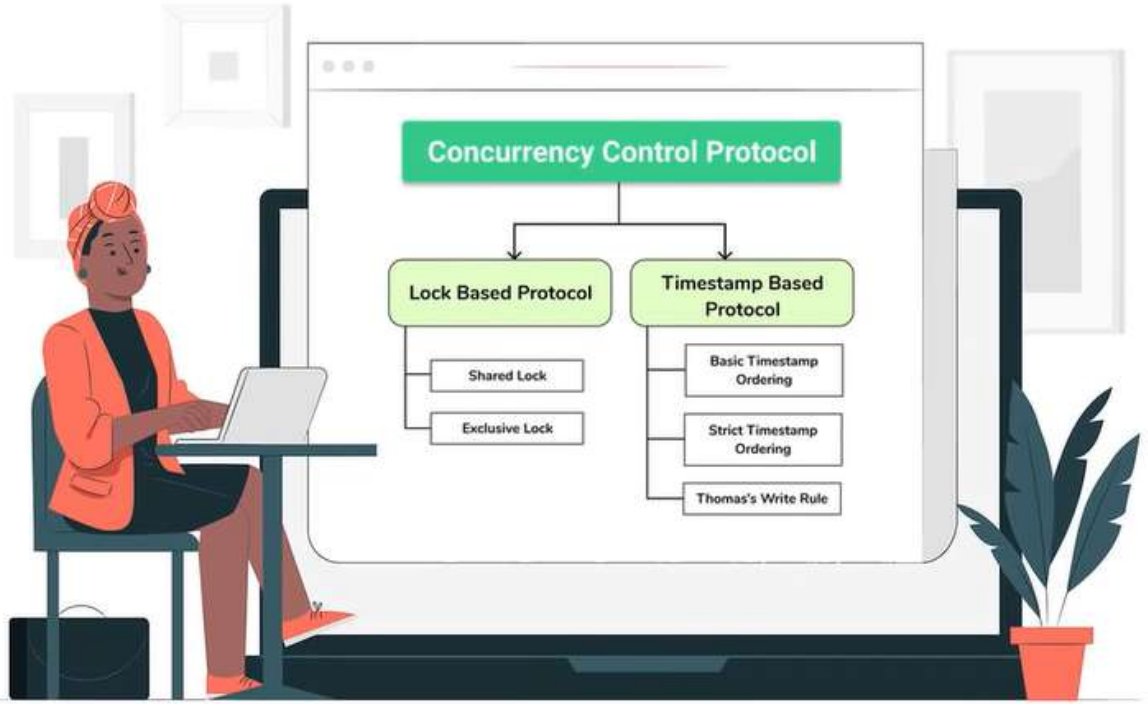
- **Consistency** : This means that integrity constraints must be maintained so that the database is consistent before and after the transaction. It refers to the correctness of a database. Referring to the example above,
  - The total amount before and after the transaction must be maintained.
  - Total before T occurs = 500 + 200 = 700.
  - Total after T occurs = 400 + 300 = 700.

  Therefore, database is consistent. Inconsistency occurs in case T1 completes but T2 fails. As a result T is incomplete.

- **Isolation** : This property ensures that multiple transactions can occur concurrently without leading to the inconsistency of database state.
  - Transactions occur independently without interference.
  - Changes occurring in a particular transaction will not be visible to any other transaction until that particular change in that transaction is written to memory or has been committed.
  - This property ensures that the execution of transactions concurrently will result in a state that is equivalent to a state achieved these were executed serially in so

- **Durability** :
  - This property ensures that once the transaction has completed execution, the updates and modifications to the database are stored in and written to disk and they persist even if a system failure occurs. T
  - these updates now become permanent and are stored in non-volatile memory. The effects of the transaction, thus, are never lost.

The **ACID** properties, in totality, provide a mechanism to ensure **correctness** and **consistency** of a database in a way such that each transaction is a group of operations that acts a single unit, produces consistent results, acts in isolation from other operations and updates that it makes are durably stored.



Concurrency

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories −

- Lock based protocols
- Time stamp based protocols

Lock Based Protocols : Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds −

- **Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.

- **Shared/exclusive** − This type of locking mechanism differentiates the locks based on their uses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

Timestamp-based Protocols:

- The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.
- Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.
- Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.
- In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last ‘read and write’ operation was performed on the data item.

Indexing

Indexing is a data structure technique to **efficiently retrieve** records from the database files based on some attributes on which the indexing has been done. Indexing in database systems is similar to what we see in books.

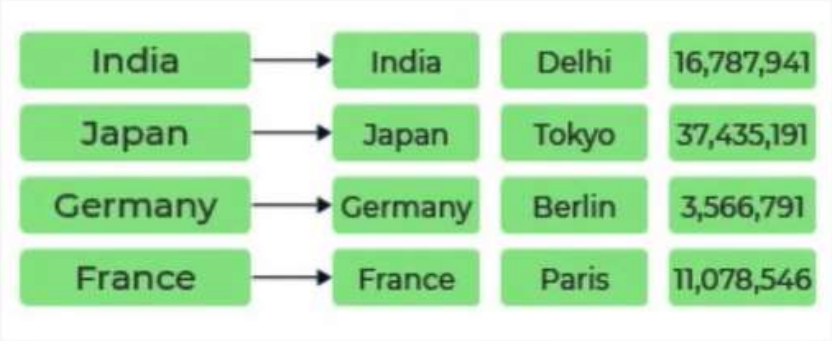
Indexing is defined based on its indexing attributes. Indexing can be of the following types −

- **Primary Index** − Primary index is defined on an ordered data file. The data file is ordered on a key field. The key field is generally the primary key of the relation.
- **Secondary Index** − Secondary index may be generated from a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values.
- **Clustering Index** − Clustering index is defined on an ordered data file. The data file is ordered on a non-key field.

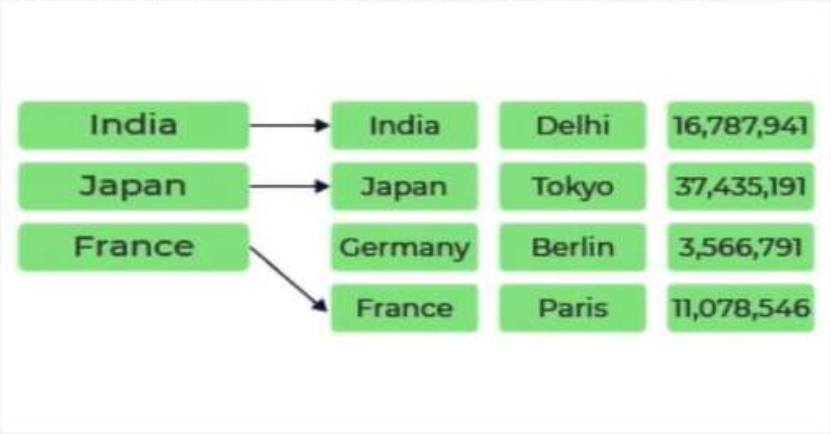
Primary Indexing is of two types −

- **Dense Index**
- **Sparse Index**

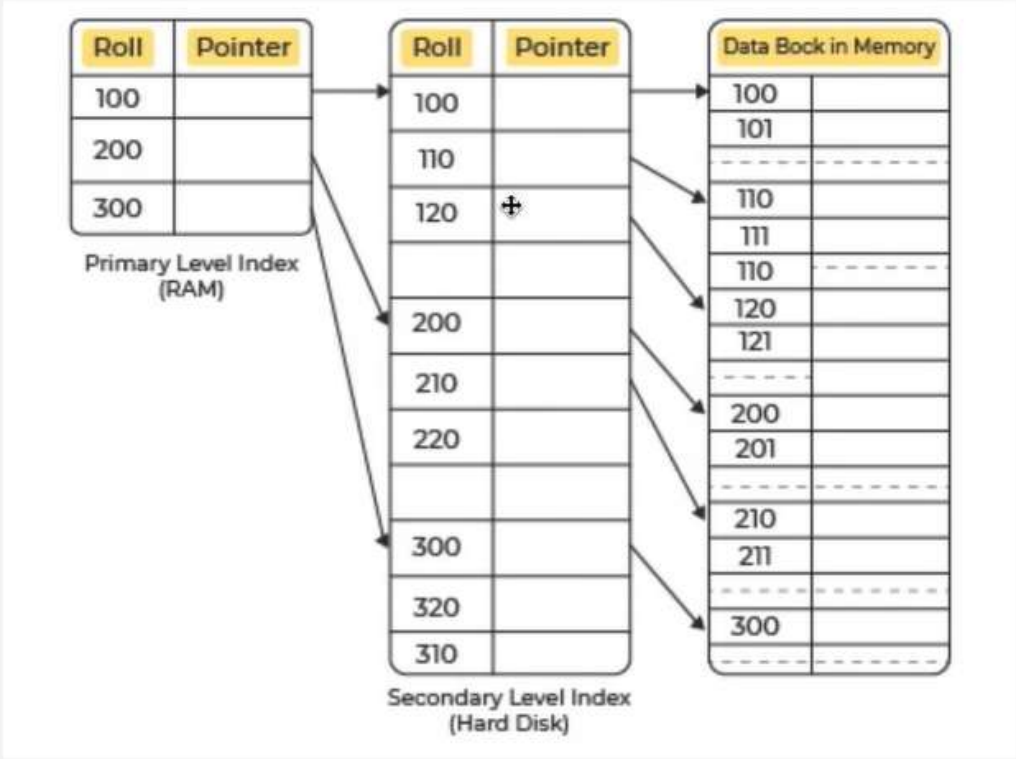
Dense Index – In dense index, there is an index record for every search key value in the database. This makes searching faster but requires more space to store index records itself. Index records contain search key value and a pointer to the actual record on the disk.



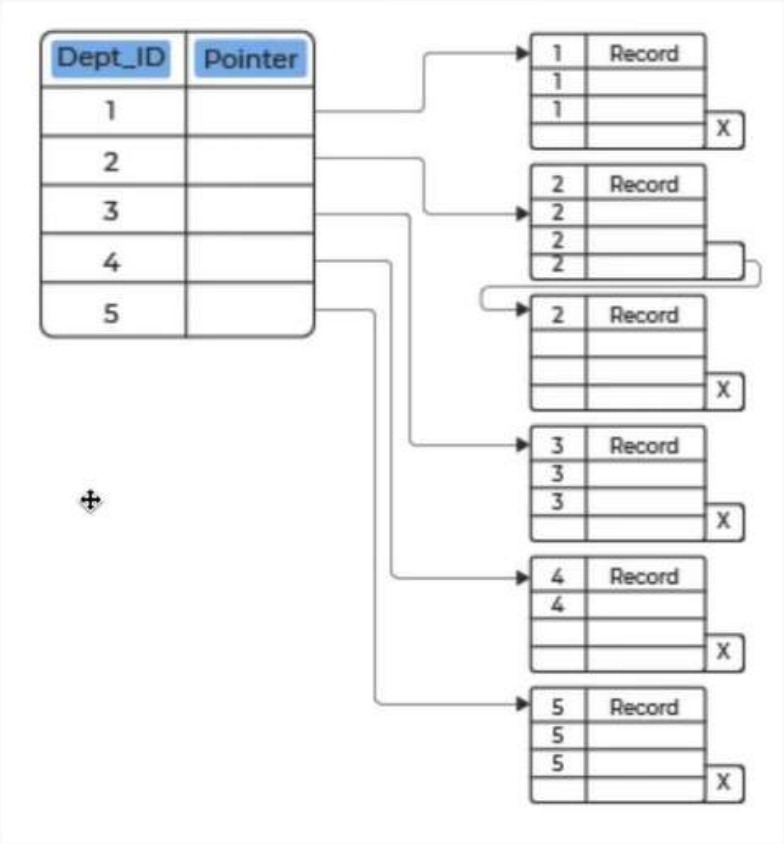
Sparse Index – In sparse index, index records are not created for every search key. An index record here contains a search key and an actual pointer to the data on the disk. To search a record, we first proceed by index record and reach at the actual location of the data. If the data we are looking for is not where we directly reach by following the index, then the system starts sequential search until the desired data is found.



- **Secondary Index** :
  - In secondary indexing, to reduce the size of mapping, another level of indexing is introduced.
  - In this method, the huge range for the columns is selected initially so that the mapping size of the first level becomes small. Then each range is further divided into smaller ranges.
  - The mapping of the first level is stored in the primary memory, so that address fetch is faster. The mapping of the second level and actual data are stored in the secondary memory (hard disk).



- **Clustering Index** :
  - Sometimes the Index is created on non-primary key columns which might not be unique for each record.
  - In such a situation, you can group two or more columns to get the unique values and create an index which is called clustered Index. This also helps you to identify the record faster.

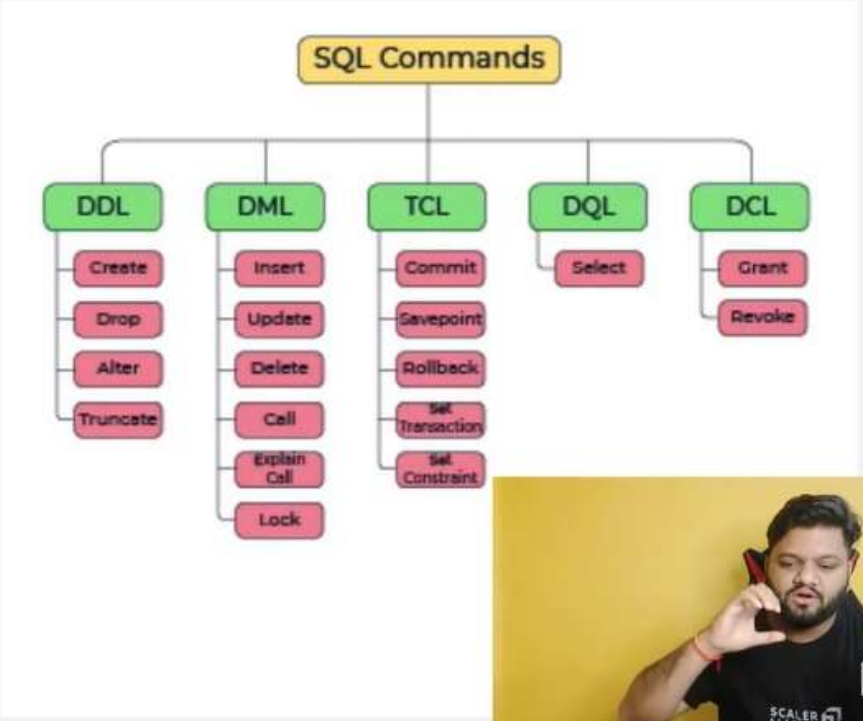


SQL

- SQL stands for **Structured Query Language**. It is used for storing and managing data in relational database management system (RDMS).
- It is a standard language for Relational Database System. It enables a user to create, read, update and delete relational databases and tables.
- All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language.
- SQL allows users to query the database in a number of ways, using English-like statements.

These SQL commands are mainly categorized into four categories as:

➔ DDL – Data Definition Language
➔ DQl – Data Query Language
➔ DML – Data Manipulation Language
➔ DCL – Data Control Language



Things we can do in SQL queries:

- Where Clause
- Order By
- Group By
- Aggregations Functions i.e; Sum, Min, Max
- Case-When
- Joins
- Union

JOINS In SQL

Different Types of SQL JOINs:

Here are the different types of the JOINs in SQL:

- **INNER JOIN:** Returns records that have matching values in both tables
- **LEFT JOIN:** Returns all records from the left table, and the matched records from the right table
- **RIGHT JOIN:** Returns all records from the right table, and the matched records from the left table
- **FULL JOIN:** Returns all records when there is a match in either left or right table