



# LET'S LEARN DATABASE MANAGEMENT SYSTEM (DBMS)

*-By Riti Kumari*

# TOPICS TO BE COVERED

1

DBMS Introduction

2

DBMS Architecture

3

Data Abstraction

4

Types of data model

5

ER Model

6

Relational Model

7

Type of keys

8

Normalisation

9

Denormalization

10

Transactions & Concurrency control

11

Indexing(B ,B+ trees)

12

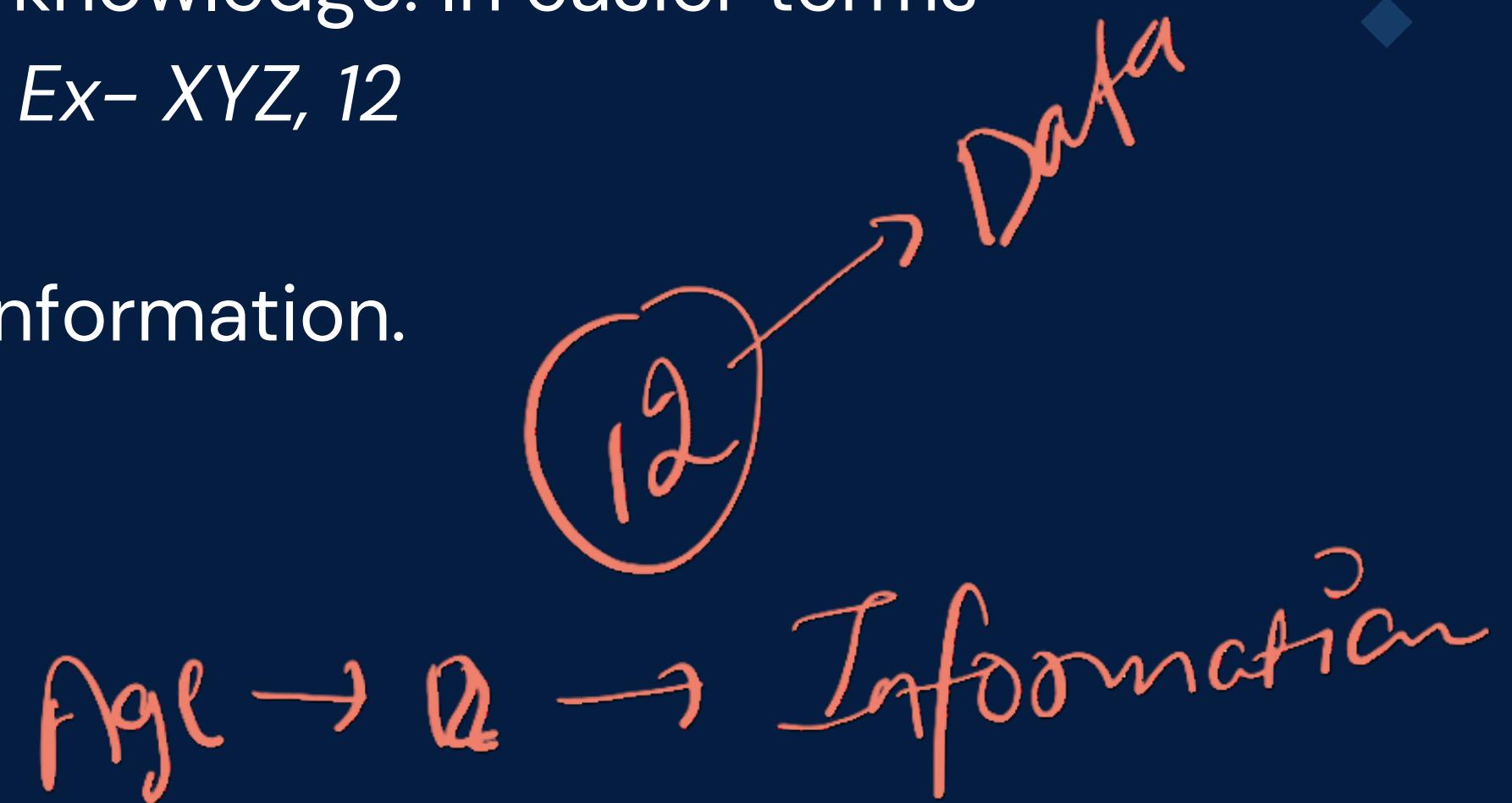
SQL

# DATA & INFORMATION



**Data** : Data refers to a collection of raw facts or figures that can be processed to derive meaning or knowledge. In easier terms we can say any fact that can be stored. Ex- XYZ, 12

**Information** : Processed data is called information.  
Ex- Your name, temperature, etc.



# DATABASE

Collection of interrelated data is called a database.

- It can be stored in the form of table
- It can be any size

Multimedia database



College database



# EXAMPLE

ID	Name	subject
1	Rahul	PHE
2	Raj	ECO
3	Riti	IT

ID	Name	Place
1	Rahul	DELHI
2	Raj	KOLKATA
3	Riti	MUMBAI

Collection of related data

if I combine these  
it will give  
me a related  
data.

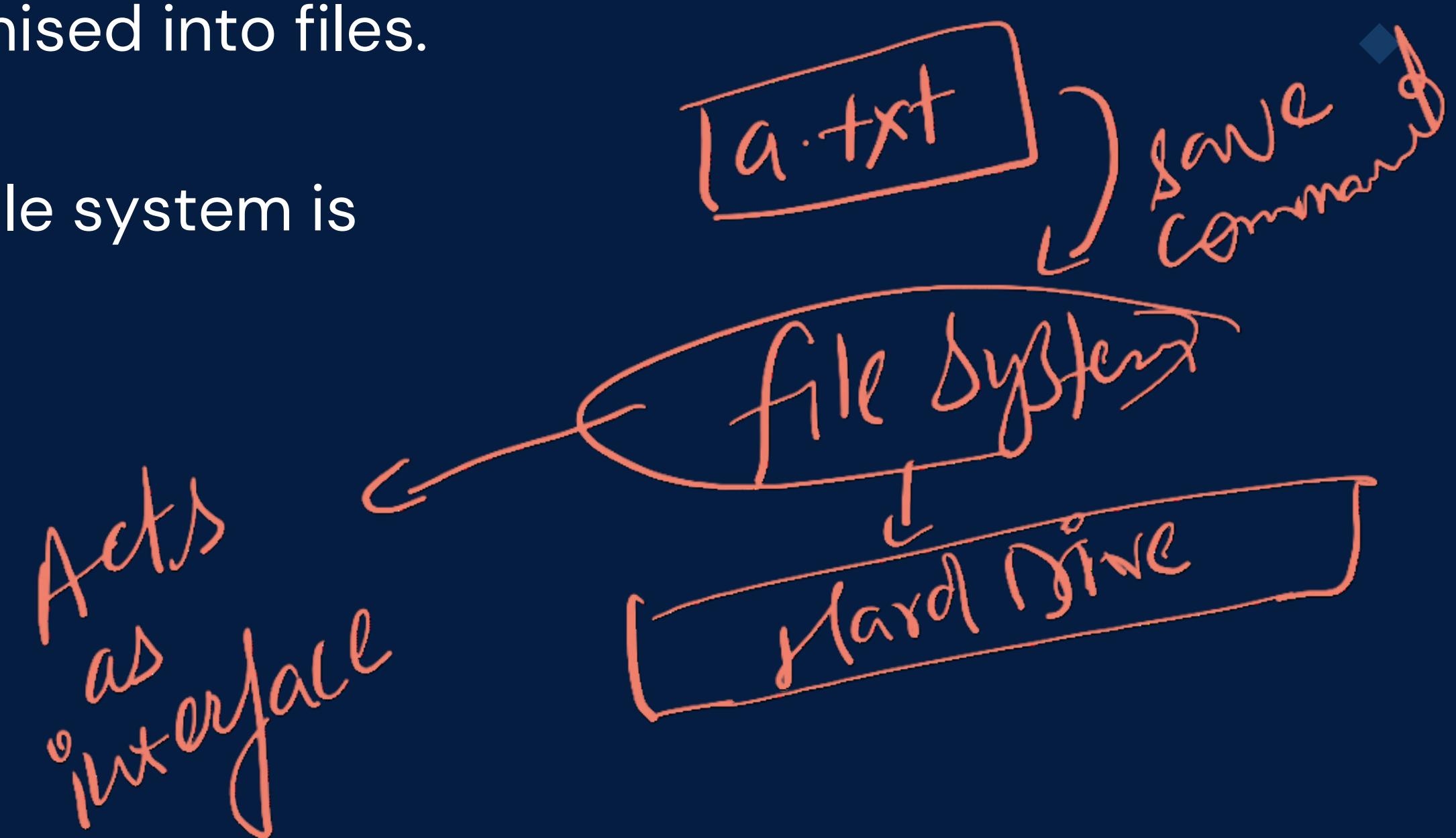
# FILE SYSTEM

An operating system's approach for organising and storing data on storage units like hard drives is called a file system.

In a file system, data is organised into files.

The major disadvantage of file system is

- Data redundancy
- Poor Memory utilisation
- Data inconsistency
- Data security



# DATABASE MANAGEMENT SYSTEM

The acronym DBMS stands for "Database Management System."

Users can access databases, save data, retrieve it, update it, and manage it safely and effectively with the use of a software program or combination of programs.

The presence of rules and regulations in the management system is crucial as they are necessary to uphold and maintain the database effectively.

# APPLICATION OF DBMS

- Schools and Colleges



- Banks



- Airlines



# APPLICATION OF DBMS

- Schools and Colleges - DBMS is used to create and maintain a student information system that stores student records, including personal details, academic performance, attendance, and extracurricular activities.
- Banks - DBMS is used to maintain a centralised and secure database of customer information, including personal details, account numbers, contact information, and transaction history.

# TYPES OF DATABASES

1

Relational Databases (RDBMS)

2

NoSQL Databases

3

Object-Oriented Databases

4

In-Memory Databases

5

Time-Series Databases

6

Spatial Databases

7

Multimedia Databases

8

Columnar Databases

9

XML Databases

10

NewSQL Databases

11

Blockchain Databases

SQL

→ Geometric Space

# TYPES OF DATABASES

- Relational Databases (RDBMS)- These databases structure data into organized tables that have predefined connections between them. Data manipulation and querying are performed using SQL (Structured Query Language). Well-known instances encompass MySQL, PostgreSQL, Oracle Database, and Microsoft SQL Server.

→ NoSQL

- NoSQL Databases- NoSQL databases are created to handle data that doesn't fit neatly into the strict setup of traditional relational databases. Ex- MongoDB(Document Oriented DB)

→ Key value pairs

# TYPES OF DATABASES

- Object-Oriented Databases- These databases hold objects (data and actions) utilized in object-oriented programming. They work well for applications with intricate data designs, like scientific simulations or multimedia software.
- In-Memory Databases- In these databases, data is kept in the primary memory (RAM) rather than on a disk, leading to quicker data retrieval. They're employed in applications that demand instant data processing and top-notch performance.

welds  
objects  
stored in  
primary  
memory

# NEED OF DBMS

DBMS plays a vital role for businesses, institutions, and organizations of all scales in effectively managing their data, ensuring data accuracy and security, and supporting essential decision-making processes.

It serves as the core of contemporary information systems, facilitating efficient data management and serving as a basis for a wide range of applications and services.

Wide range of applications

# ADVANTAGE OF DBMS

- **Data Security**- DBMS implements security mechanisms that regulate access to sensitive information, safeguarding it from unauthorized access and potential data breaches.
- **Data Redundancy and Inconsistency**- DBMS removes data redundancy, minimizing storage needs and ensuring consistency through the maintenance of a unified version of the data.
- **Data Integrity** - DBMS guarantees data integrity by enforcing rules and constraints that prohibit the entry of incorrect or inconsistent data into the database.

→ Achieves  
Lat  
Consistency  
→ Also  
constraints

# ADVANTAGE OF DBMS

- **Data Scalability**- DBMS can handle large datasets and scale to accommodate increasing amounts of data as an organization grows.
- **Data Abstraction**- DBMS offers data abstraction, allowing users and applications to interact with the database without needing to understand its underlying complexities.

Scalable

Data  
Abstraction

# DISADVANTAGE OF DBMS

- Cost- Acquiring, deploying, and sustaining DBMS software can incur significant costs. Furthermore, the hardware essential for the proficient operation of a DBMS can also lead to substantial expenses.
- Scale Projects- When dealing with modest applications and minimal data storage requirements, adopting a comprehensive DBMS could introduce avoidable intricacies and additional burdens. In these instances, more streamlined data storage alternatives could be better suited.

High cost & maintenance

Not for small projects

# DISADVANTAGE OF DBMS

- Vendor Lock-In- Once you've chosen a specific DBMS, it can be challenging to switch to a different one due to differences in data formats, query languages, and other technical aspects. This can lead to vendor lock-in, where you are dependent on a particular vendor's technology and pricing.



switching  
is  
challenging!

# DATA ABSTRACTION

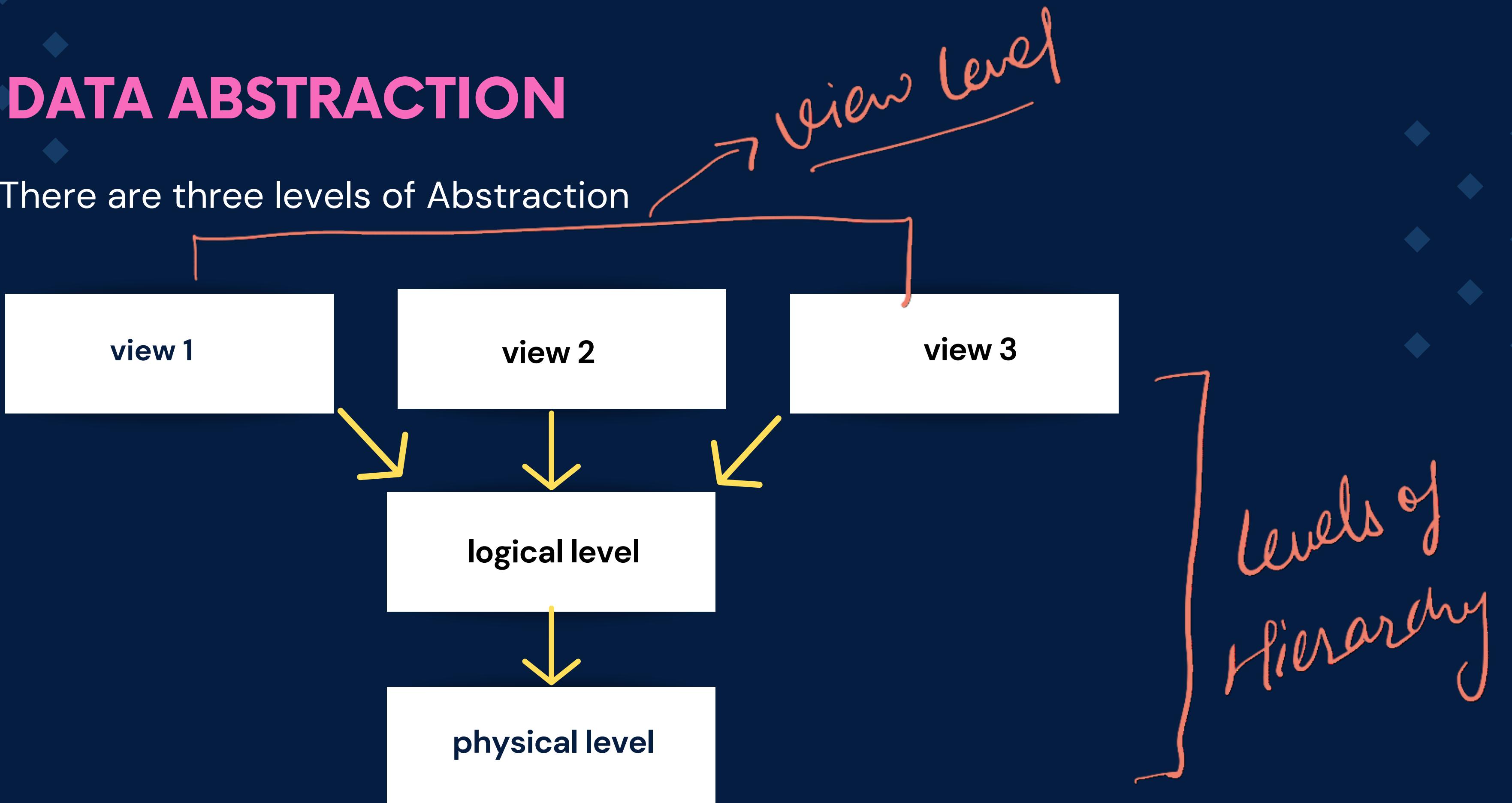
Database systems are built with complex ways of organizing data. To make it easier for people to use the database, the creators hide the complicated stuff that users don't need to worry about. This hiding of unnecessary things from users is called data abstraction.

Users don't need  
to know unnecessary  
details.

Data  
hiding  
is implemented  
in DBMS.

# DATA ABSTRACTION

There are three levels of Abstraction



# TYPES OF LEVEL

- Physical level- This is the lowest level of data abstraction. It describes how data is actually stored in database. You can get the complex data structure details at this level.
- Logical level- This is the middle level of 3-level data abstraction architecture. It describes what data is stored in database.
- View level- Highest level of data abstraction. This level describes the user interaction with database system.

→ New data stored  
→ what is stored.

## Student DB



Memory  
SSD Hardisk  
Physical level  
(we can know how  
level details)

Create database,  
table, modify, etc.

↑  
logical level  
(what data is stored)



need  
JIO  
Marks

need Fee  
details, etc.

Account  
team has  
different  
view in that  
table.

Views

→ We will be able to  
see only that

# SCHEMA & INSTANCE

Let us first learn about some basic concepts:

**Schema-** A schema is a logical container or structure that organizes and defines the structure of a database.

It defines how data is organized, what data types are used, what constraints are applied, and the relationships between different pieces of data. A schema acts as a blueprint for the database, ensuring data integrity, consistency, and efficient data retrieval.

• Data types  
• Constraints  
• Relationships  
etc.

# SCHEMA & INSTANCE

Types of Schema :

1. Physical Schema- A physical schema defines how data is stored on the underlying hardware, including details such as storage format, file organization, indexing methods, and data placement.

- 
- attributes  
of  
physical  
level.
- where data is stored
  - Indexing Methods.
  - Storage Format.

# SCHEMA & INSTANCE

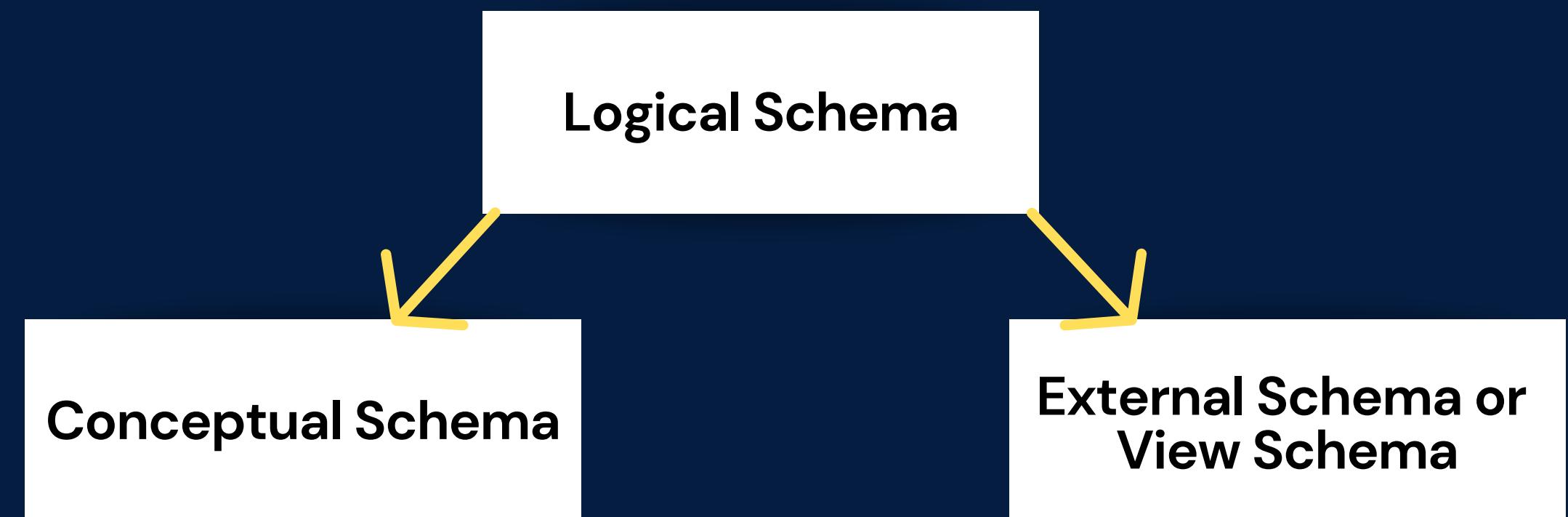
Characteristics of Physical Schema :

- Its primary focus lies in enhancing the storage and retrieval of data to boost performance.
- Modifications made to the physical schema demand meticulous planning and can potentially affect the overall performance of the database.
- Example: Deciding to use clustered indexes on specific columns for faster retrieval.

# SCHEMA & INSTANCE

Types of Schema :

2. Logical Schema- A logical schema defines the database's structure from a logical or conceptual perspective, without considering how the data is physically stored.

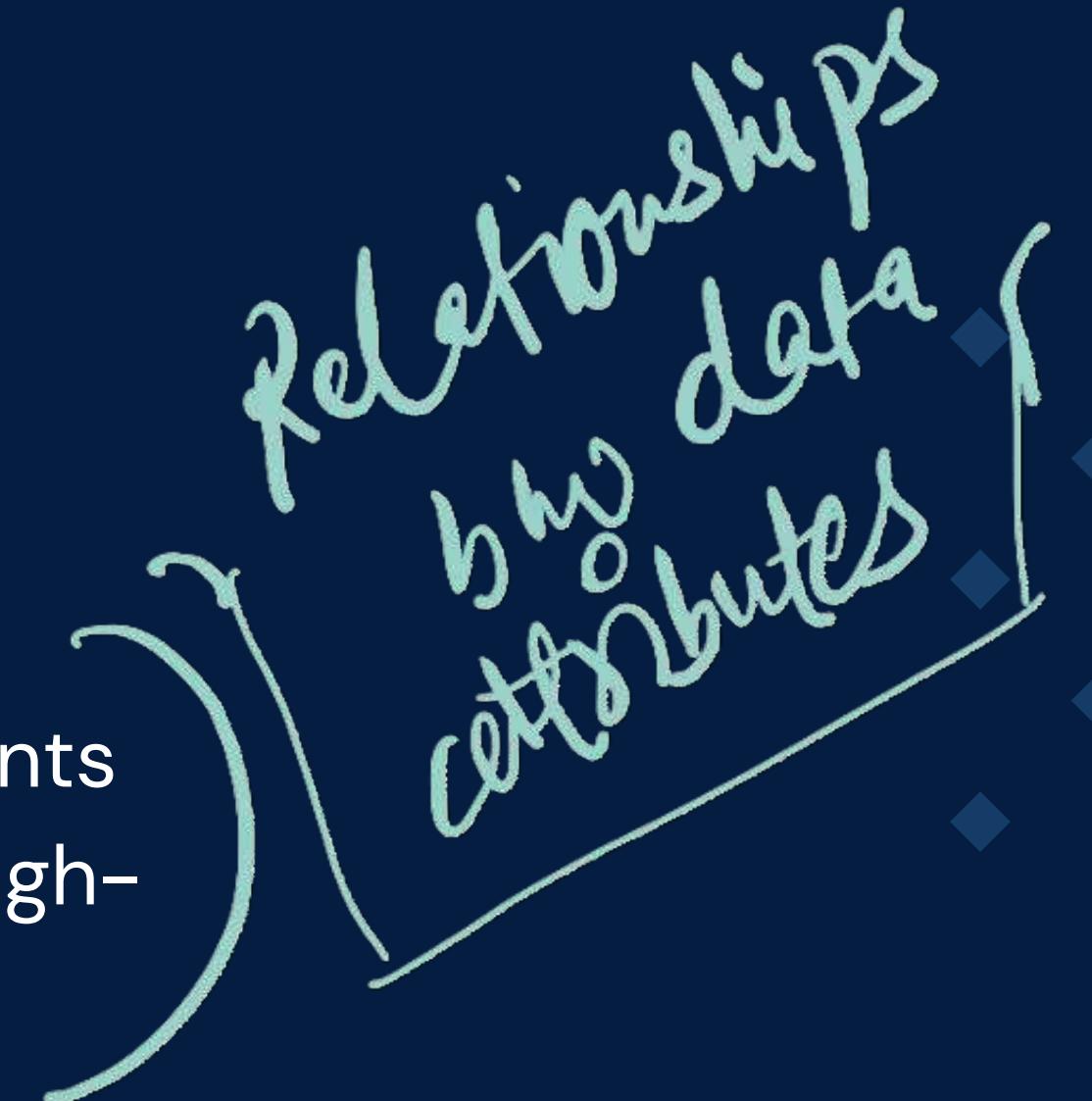


what Data  
is Stored.

# SCHEMA & INSTANCE

Types of Logical Schema :

- **Conceptual Schema:** The conceptual schema represents the overall view of the entire database. It defines the high-level structure and relationships between all data elements.
- **External/View Schema:** An external schema defines the user-specific views of the database. It focuses on the portions of the database that are relevant to specific user roles or applications.



# SCHEMA & INSTANCE

Characteristics of Logical Schema :

- It delineates how data is structured into tables, the interconnections between these tables, and the restrictions placed on the data.
- Logical schemas prioritize data modeling and database design over considerations related to hardware or storage specifics.
- Example: Defining tables, specifying primary and foreign keys, and creating views for data access.

# SCHEMA & INSTANCE

Instance - The information residing within a database at a specific point in time is referred to as the database's "instance."

Within a given database schema, the declarations of variables within its tables pertain to that specific database.

The term "instance" in this context denotes the current values of these variables at a particular moment in time for that database.

Info at a  
particular  
time in  
Database  
is  
called  
Instance

E-commerce  
website

↳ Sales Team

↳ Product Team

See diff.  
views or  
tables

View Schema is  
defined Acc. to this

Instances modify themselves from  
time to time as user modifies something  
or etc.

---

# DBMS ARCHITECTURE

Database Management System architecture, refers to the structural framework and organization of a database management system. It defines how the various components of the system work together to store, manage, and retrieve data efficiently.

*Refines the complexity of  
Application .*

# DBMS ARCHITECTURE

Types of DBMS ARCHITECTURE :

There are several types of DBMS Architecture.

Choice of architecture depends on factors such as the type of database (e.g., relational, NoSQL) and the specific needs of an application.

- **1-Tier Architecture**
- **2-Tier Architecture**
- **3-Tier Architecture**



Complexity ↑  
↑ Needs.

# DBMS ARCHITECTURE

**1-Tier Architecture** - In 1 tier architecture the entire database application, including the user interface, application logic, and data storage, resides on a single machine or computer.

ex- An illustration of a straightforward single-tier architecture can be seen when you install a database on your system and use it to practice SQL queries.



# DBMS ARCHITECTURE

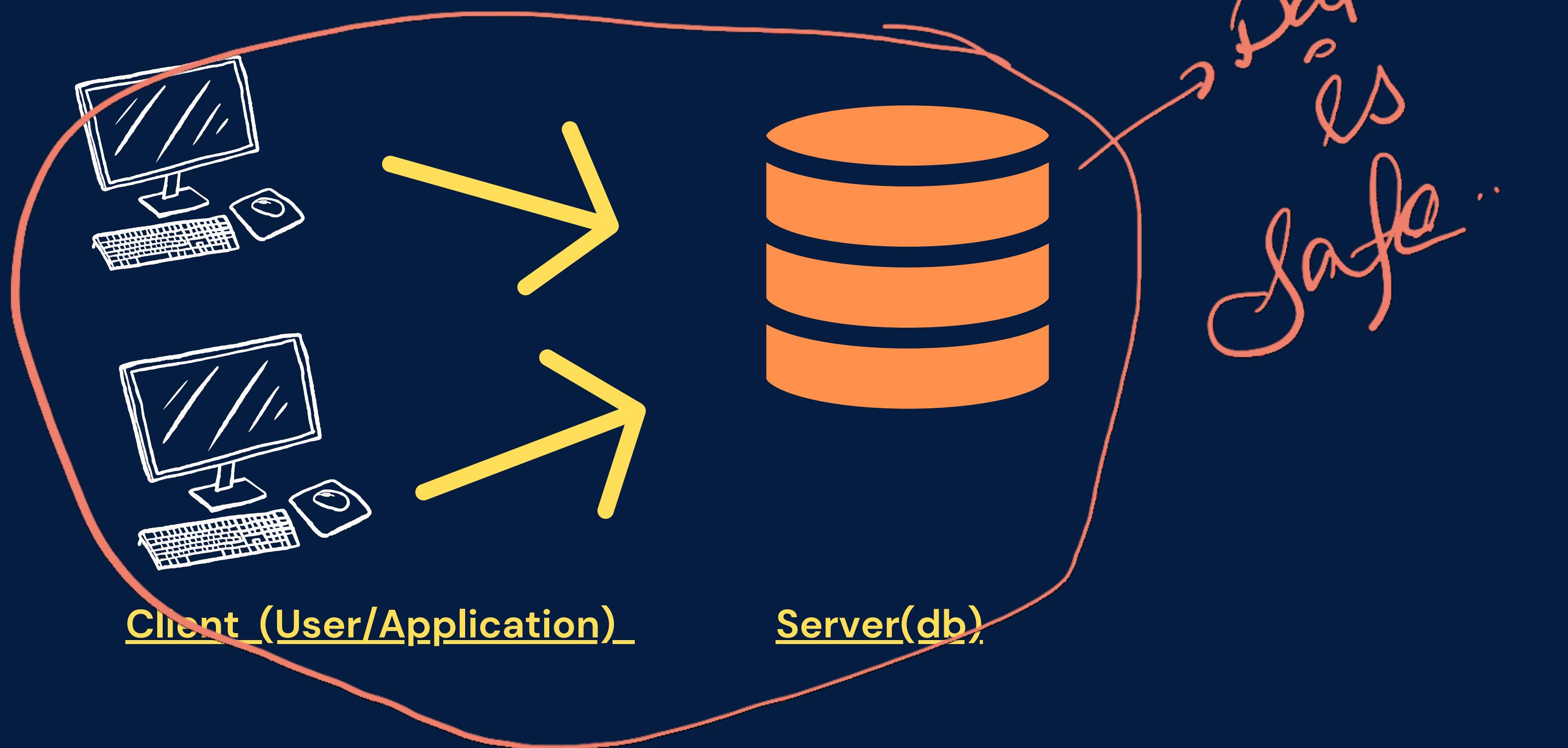
**2-Tier Architecture** – In 2 Tier Architecture the presentation layer runs on a client (PC, Mobile, Tablet, etc.), and data is stored on a server.

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.

---

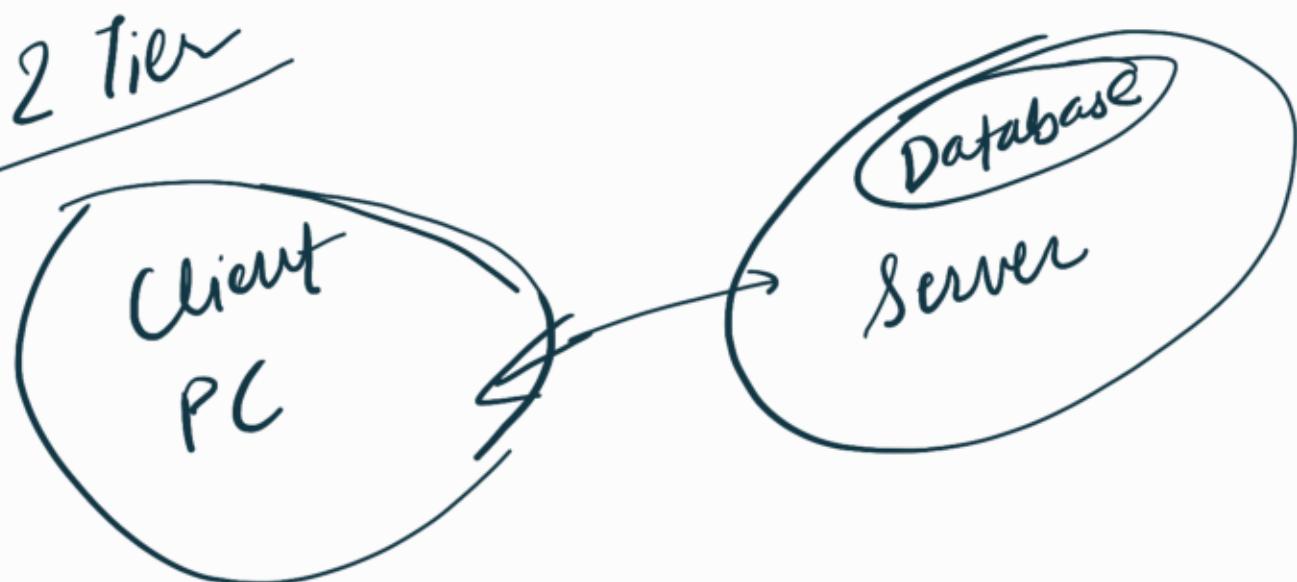
# DBMS ARCHITECTURE

## 2-Tier Architecture



1 Tier  
Single machine is containing Everything → for ex we practising SQL queries in our laptop.

2 Tier



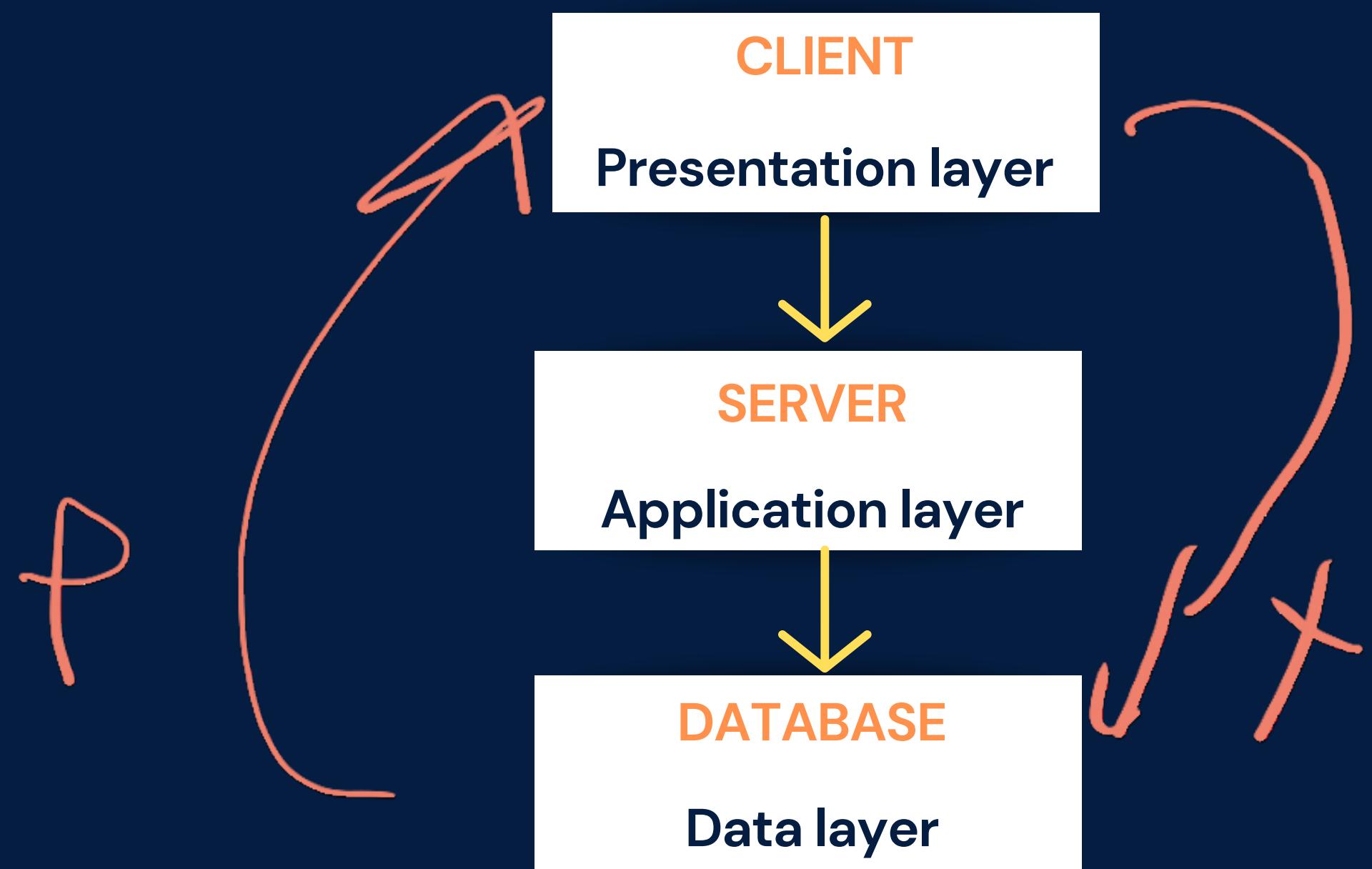
# DBMS ARCHITECTURE

- **3-Tier Architecture** – It separates the application into three logically distinct layers presentation, application, and data layer
- **Presentation layer**– It handles the user interface.  
ex- your PC, Tablet, Mobile, etc
- **Application layer** – It manages business logic  
ex- server
- **Data layer**– It manages data storage and processing.  
ex- Database Server

) UI  
) Backend  
) Database

# DBMS ARCHITECTURE

- 3-Tier Architecture



# DBMS ARCHITECTURE

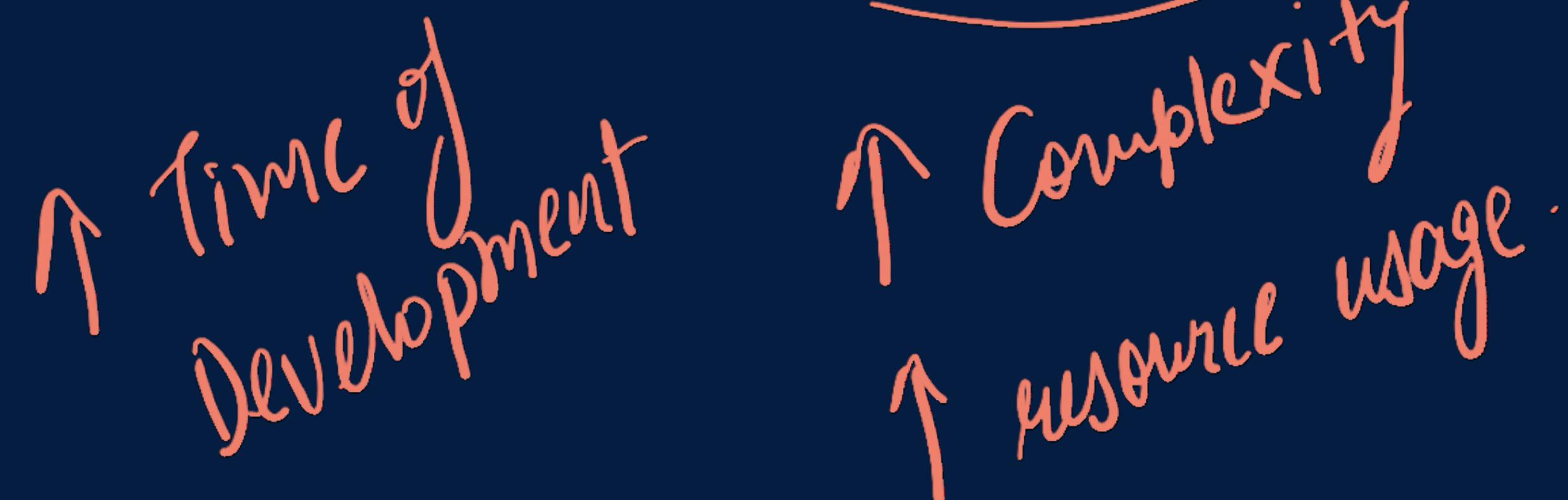
## Advantages of 3-tier-architecture

- Scalability: Easily adjust each tier to handle changing user demands.
- Modularity and Maintainability: Simplify maintenance by separating responsibilities.
- Security: Protect sensitive data with an additional layer.
- Performance: Optimize presentation and application tiers for better performance.

# DBMS ARCHITECTURE

## Disadvantages of 3-tier-architecture

- The disadvantages of 3-Tier Architecture include increased complexity, potential latency issues, longer development time, resource overhead, and the possibility of bottlenecks.

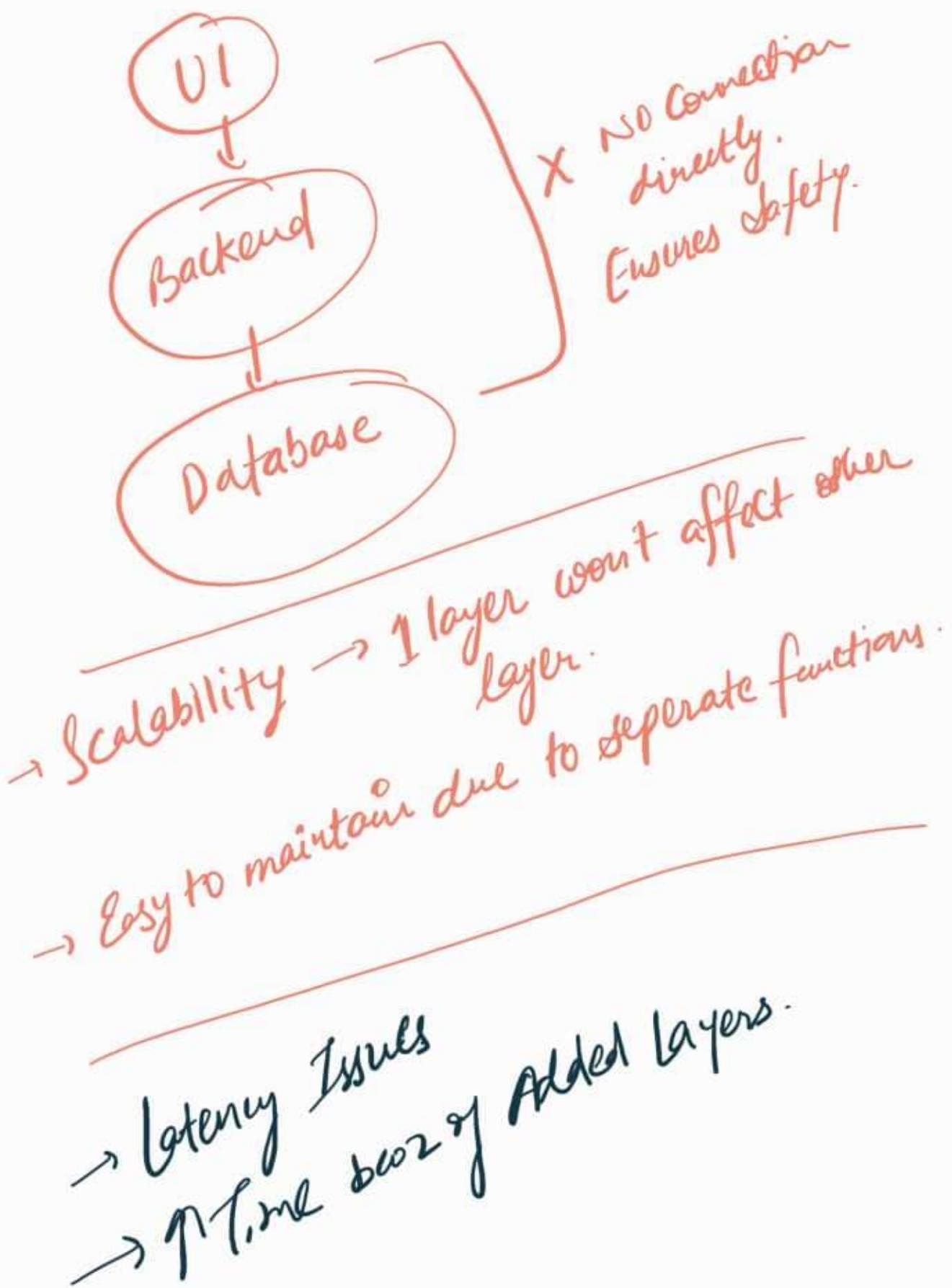


↑ Time of Development

↑ Complexity

↑ Resource usage

A hand-drawn style annotation in red ink is present on the right side of the slide. It consists of three separate lines of text: '↑ Time of Development' with an upward arrow, '↑ Complexity' with an upward arrow, and '↑ Resource usage' with an upward arrow. A large, roughly circular bracket is drawn above the word 'Complexity', enclosing all three lines of text.



# DATA MODEL

A data model within a Database Management System (DBMS) serves as an abstract representation of how data gets structured and organized within a database.

It outlines the logical arrangement of data and the connections between various data components.

Data models play a crucial role in comprehending and shaping databases, acting as a vital link between real-world entities and the actual storage of data within the database.

logical Arrangement  
of Data.

# DATA MODEL

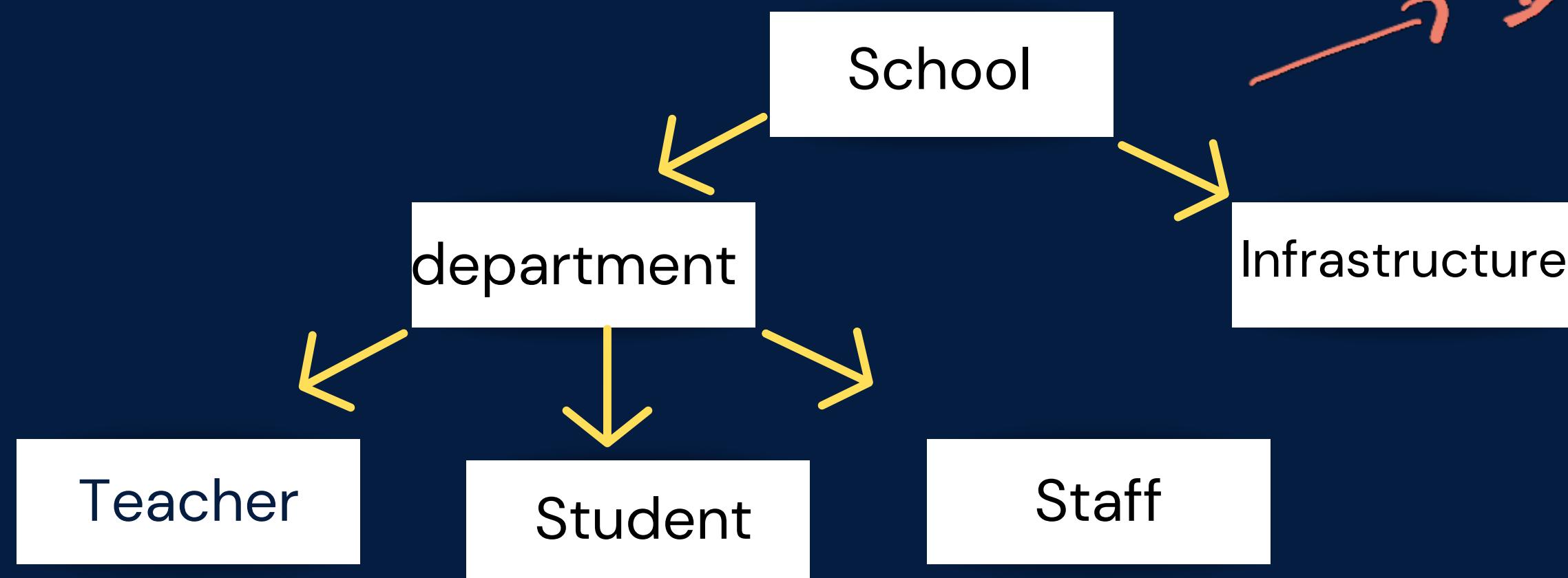
## Types of Data Model

- Hierarchical Data Model
- Network Data Model
- Relational Data Model
- Entity-Relationship Model (ER Model)
- Object-Oriented Data Model
- NoSQL Data Models



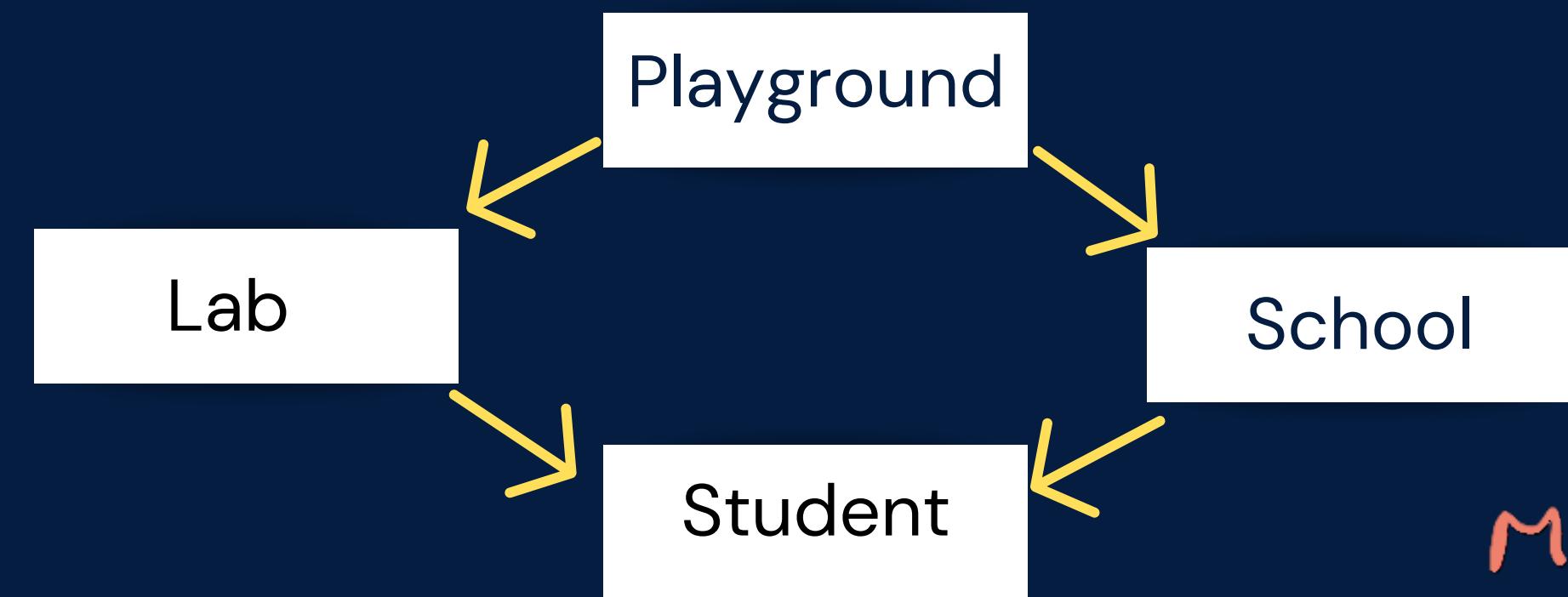
# TYPES OF DATA MODEL

- **Hierarchical Data Model:** This model portrays data in a manner resembling a tree structure, where each record maintains a parent-child relationship. Its primary application lies in older database systems.



# TYPES OF DATA MODEL

- **Network Data Model:** This model shares similarities with the hierarchical approach, permitting records to hold multiple parent-child relationships. It adopts a structure akin to a graph, offering more flexibility compared to the hierarchical model.



like multiple inheritance

multiple parent child relationship

# TYPES OF DATA MODEL

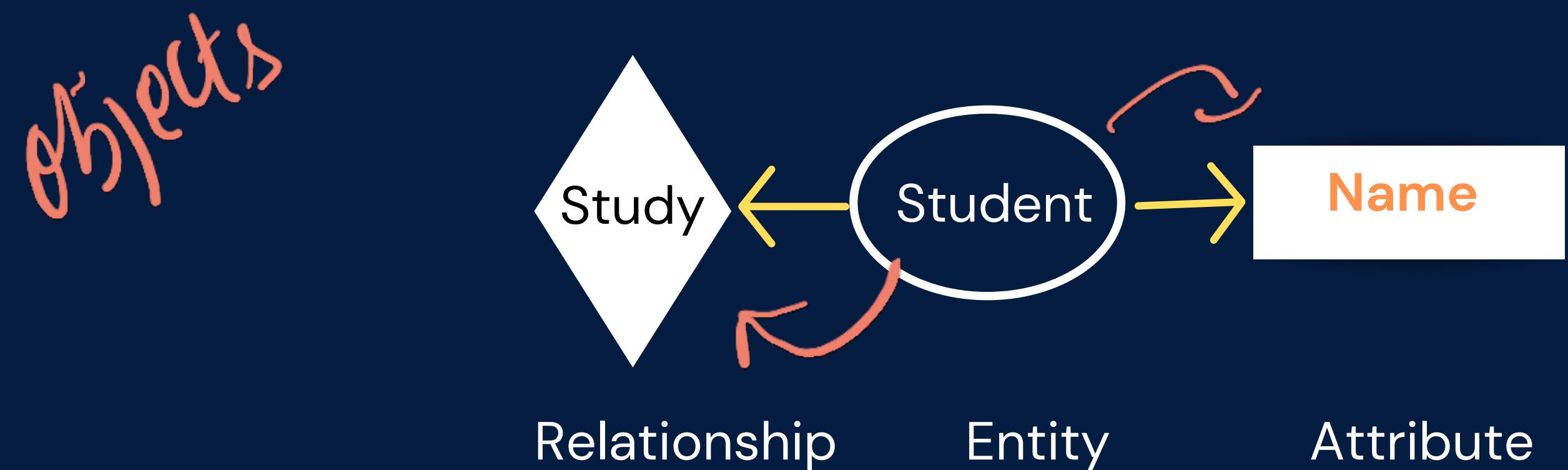
- **Relational Data Model:** Organizing data into tables (known as relations) consisting of rows and columns characterizes the relational model. It stands as the most prevalent data model, rooted in the principles of set theory, and relies on Structured Query Language (SQL) for data manipulation.

The diagram shows a relational database table with three columns: ID, Name, and Place. The table has three rows with data: (1, Rahul, DELHI), (2, Raj, KOLKATA), and (3, Riti, MUMBAI). Handwritten annotations in red ink explain the components: 'Primary key' points to the first column; 'Row/tuple' points to the second row; 'Column/Attribute' points to the 'Name' column; and 'cowsShy' is written vertically along the right side of the table.

ID	Name	Place
1	Rahul	DELHI
2	Raj	KOLKATA
3	Riti	MUMBAI

# TYPES OF DATA MODEL

- Entity-Relationship Model (ER Model): Utilized for crafting relational databases, the ER model represents data through entities (objects), attributes (entity properties), and relationships connecting these entities.



# TYPES OF DATA MODEL

- **Object-Oriented Data Model:** Extending the principles of object-oriented programming into the database domain, this model depicts data as objects complete with attributes and methods, fostering support for inheritance and encapsulation.
- **NoSQL Data Models:** NoSQL databases encompass a diverse array of data models, such as document-oriented (e.g., MongoDB), key-value (e.g., Redis), column-family (e.g., Cassandra), and graph (e.g., Neo4j). These models are designed to offer scalability and flexibility when handling extensive volumes of unstructured or semi-structured data.

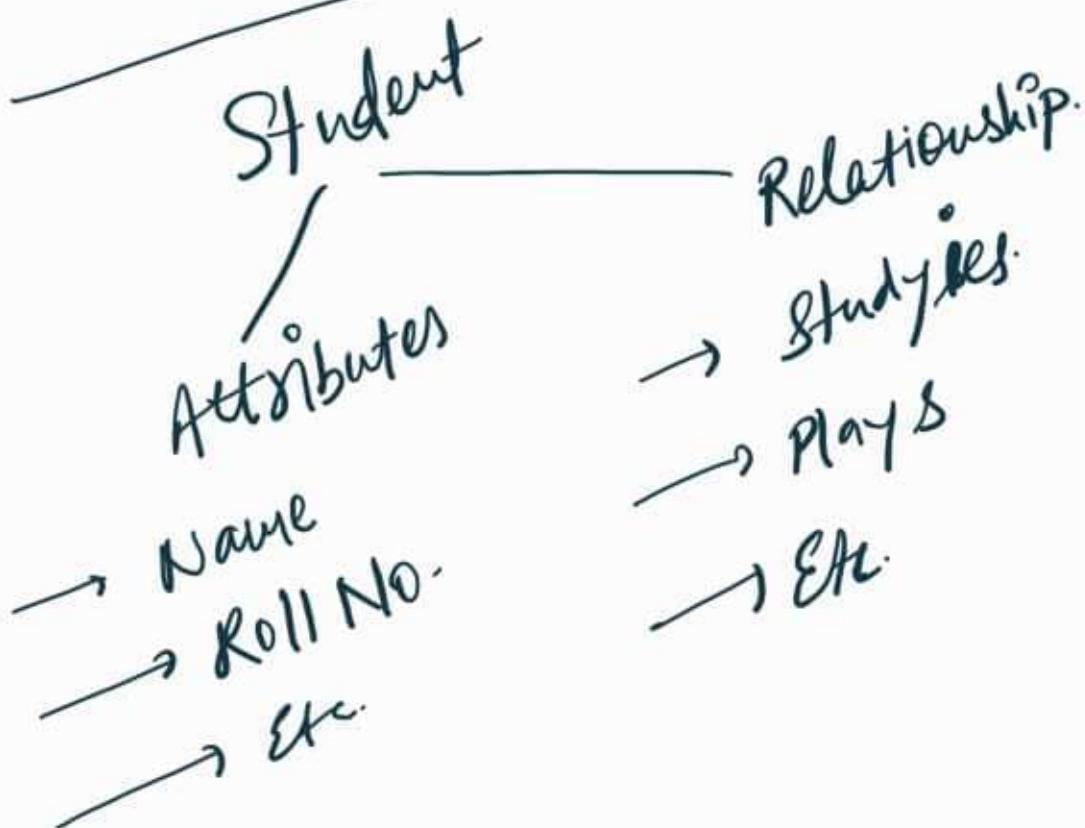
OODPs

key  
value  
pairs  
exist.

Diff. b/w Data model & Schema

Data Model → High level view conceptual framework

Schema → Actual Implementation of that Data.



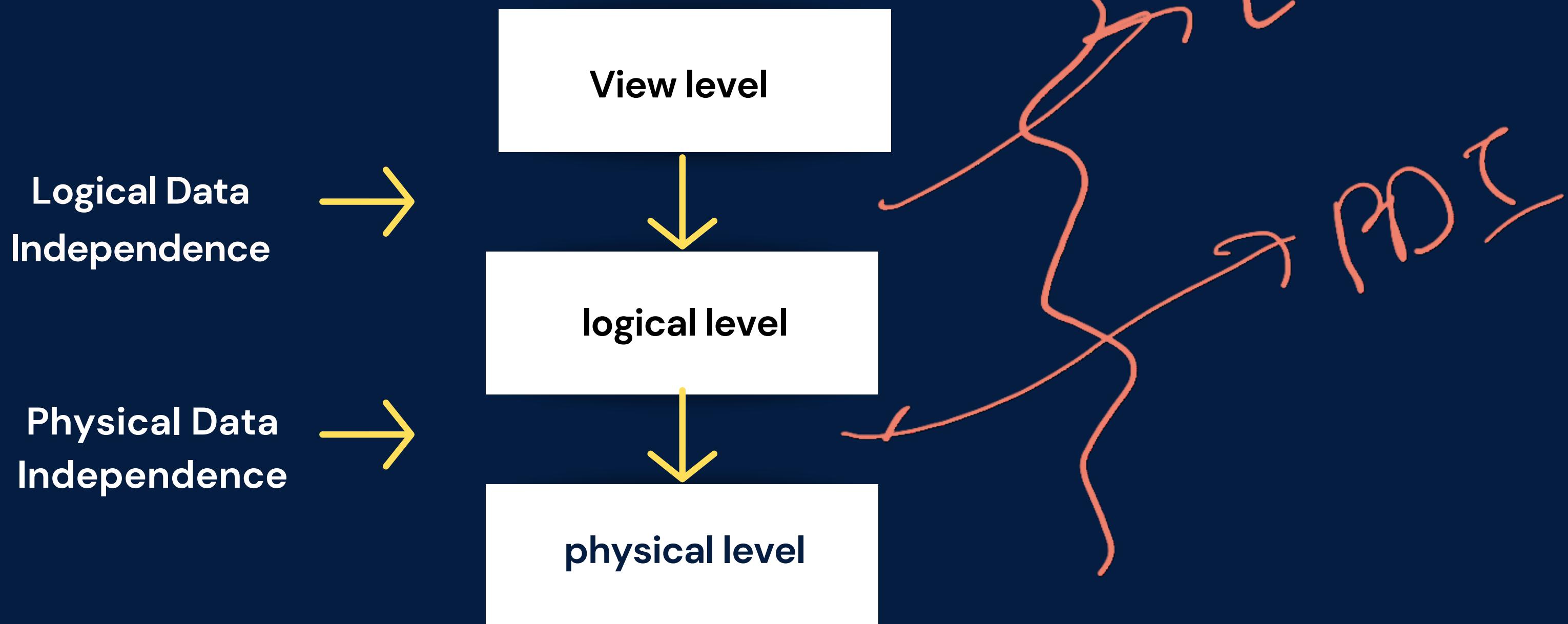
# DATA INDEPENDENCE

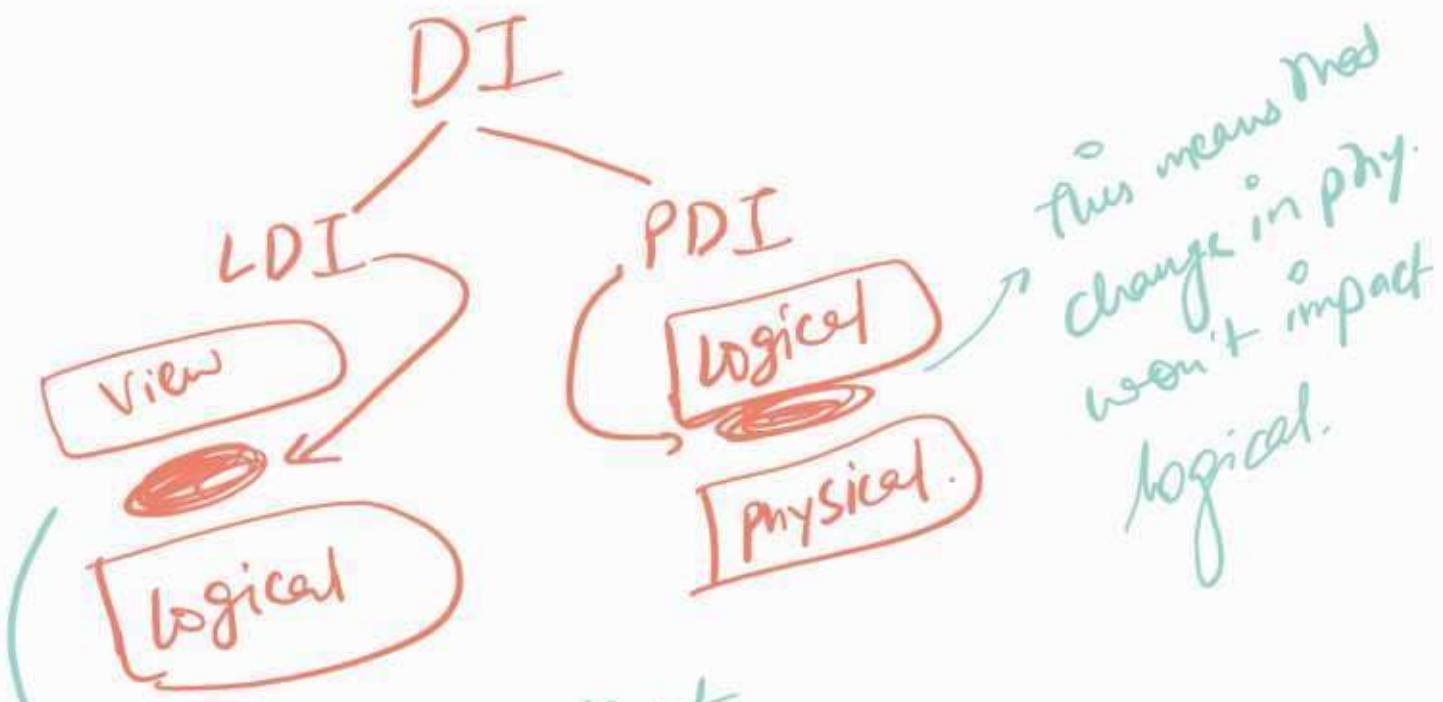
Data independence is a fundamental concept within database design and management, emphasizing the distinction between the logical and physical dimensions of data storage and administration in a database management system (DBMS). This principle yields various benefits, such as enhanced flexibility, heightened security, and simplified maintenance.

- Making user independent of Data.
- Basically it means one level doesn't change another level's data.

# DATA INDEPENDENCE

There are three levels of Abstraction





this means mod  
change in phy  
won't impact  
logical.

→ This means that  
change in logical level  
won't impact view level.  
If we have 2 users changes made by U1  
won't be visible to U2. → that is  
LDI.

# ESSENTIAL COMPONENTS OF TABLES

**Row/Tuple** - Rows, also known as records or tuples, represent individual entries or instances of data within the table.

**Cardinality** - No of rows in a table

**Column/Attribute** - Columns represent the attributes of the data being stored and are named to describe the information they hold (e.g., "ID," "Name," "Age").

**Degree** - No of Columns in a ta in a table

# ESSENTIAL COMPONENTS OF TABLES

Rows/  
Tuples

ID	Name	Place
1	Rahul	DELHI
2	Raj	KOLKATA
3	Riti	MUMBAI

Primary Key

Columns/  
Attributes

# ESSENTIAL COMPONENTS OF TABLES

**Constraints** - Constraints define rules or conditions that must be satisfied by the data in the table.

Common constraints include uniqueness, nullability, default values, etc.

- Unique constraint: Ensures values in a column are unique across the table.
- Not null constraint: Ensures a column cannot have a null value.
- Check constraint: Enforces a condition to be true for each row.
- Default constraint: Provides a default value for a column if no value is specified.

**Keys** - A primary key is a unique identifier for each record in the table. It ensures that each row can be uniquely identified and accessed within the table.

A foreign key is a field in a table that refers to the primary key of another table. It establishes relationships between tables.

Row / tuple  
Column / attribute  
Cardinality  
Degree

Constraints

- Unique
- not Null
- check
- default

---

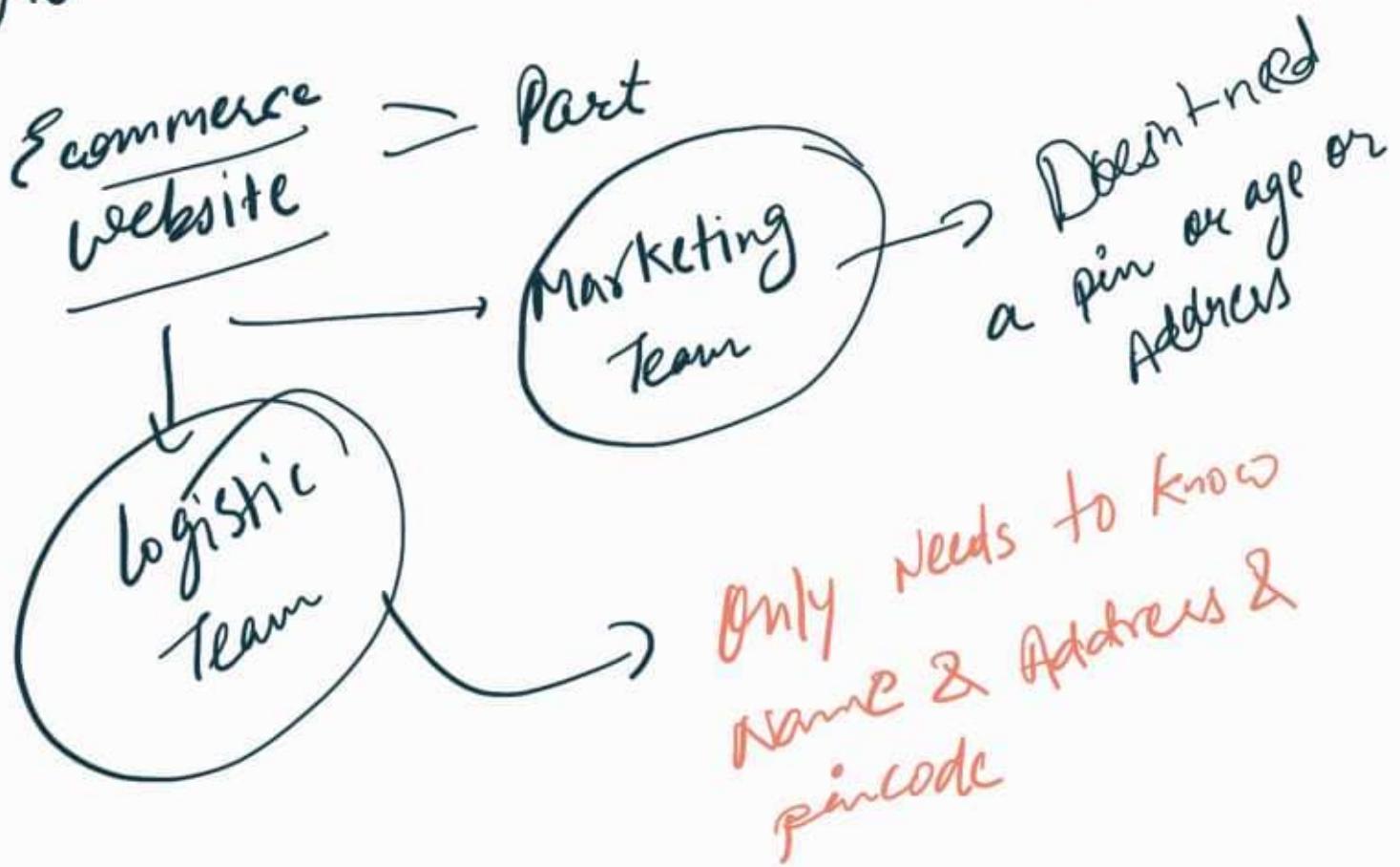
# VIEWS IN DBMS

View is a virtual table that is derived from one or more underlying tables. This means that it doesn't physically store data but rather provides a logical representation of data.

Customer DB

ID.	NAME	phn	Address	Pin	Age	Order
1	Raj	456	blr	123	18	A
2	Ravi	123	delhi	124	21	B
3	Ram	789	hyd	345	22	C

Views → Virtual Tables → Part of a Table



---

So views will only show you the necessary/ required data only.

# KEYS IN DBMS

Keys in DBMS make sure of data integrity, uniqueness, and the quick retrieval of information. Key is a attribute in table

Types of keys :

- Candidate Key
- Primary Key
- Foreign Key
- Super Key

*Defines uniqueness*

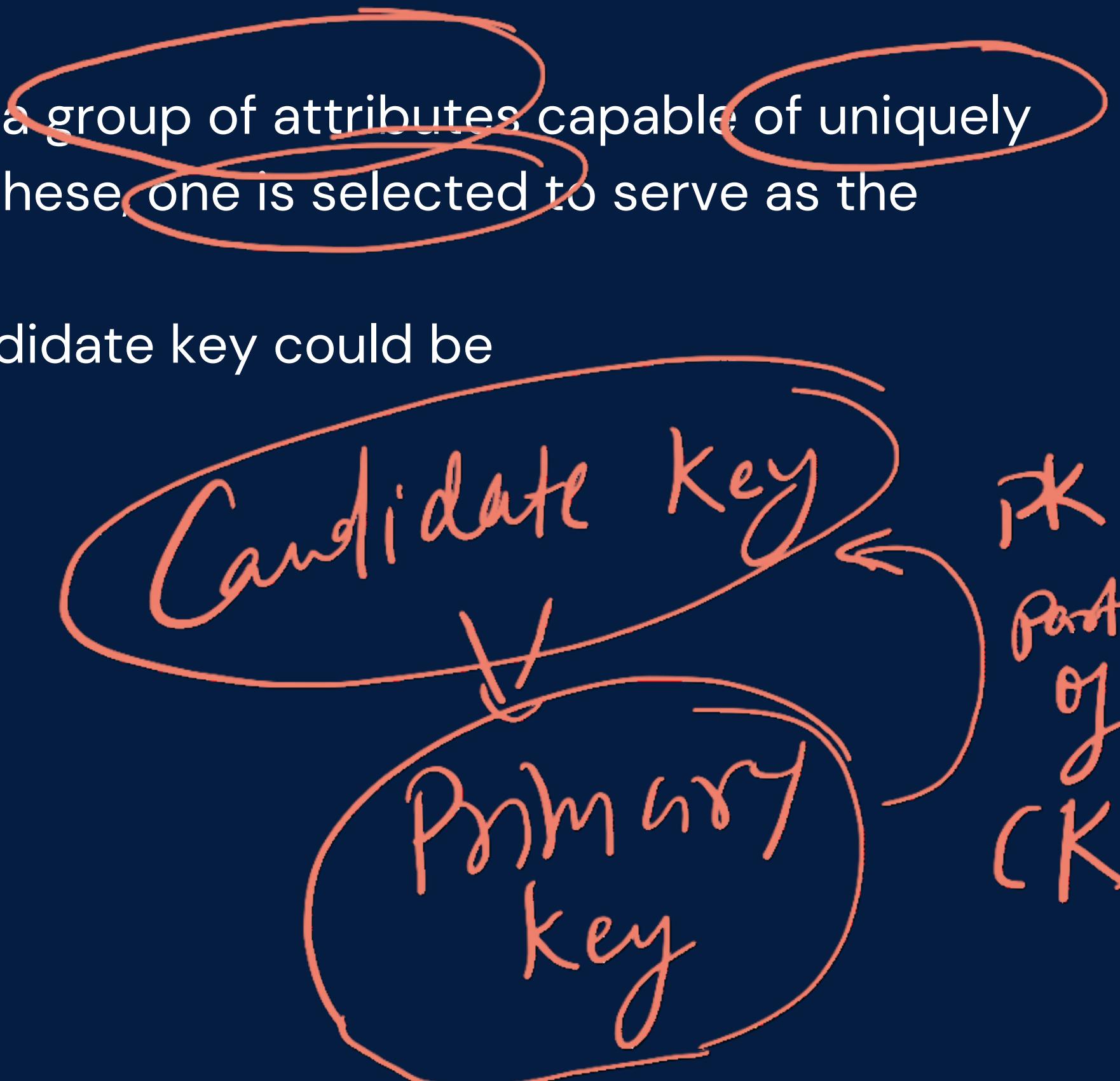
# KEYS IN DBMS

**Candidate Key** : A candidate key refers to a group of attributes capable of uniquely identifying a record within a table. Among these, one is selected to serve as the primary key.

Ex- For student possible attributes for candidate key could be

**Student<ID, Roll no , Aadhar Card>**

Age	Name	Hometown
20	Rahul	KOLKATA
21	Raj	KOLKATA
20	Riti	DELHI



# KEYS IN DBMS

Primary Key: A primary key is a key which uniquely identifies each record in a table. It ensures that each tuple or record can be uniquely identified within the table.

It is always Unique+ Not null

ID	Name	Hometown
123	Rahul	KOLKATA
245	Raj	KOLKATA
434	Riti	DELHI

$\{PK \rightarrow \text{unique}$   
 $\text{not Null}$

# KEYS IN DBMS

**Foreign Key**: A foreign key is a field in a table that refers to the primary key in another table. It establishes a relationship between two tables.

**Student**  
(Base/referenced table)

Roll no	Name	Hometown
1	Rahul	KOLKATA
2	Raj	KOLKATA
3	Riti	DELHI

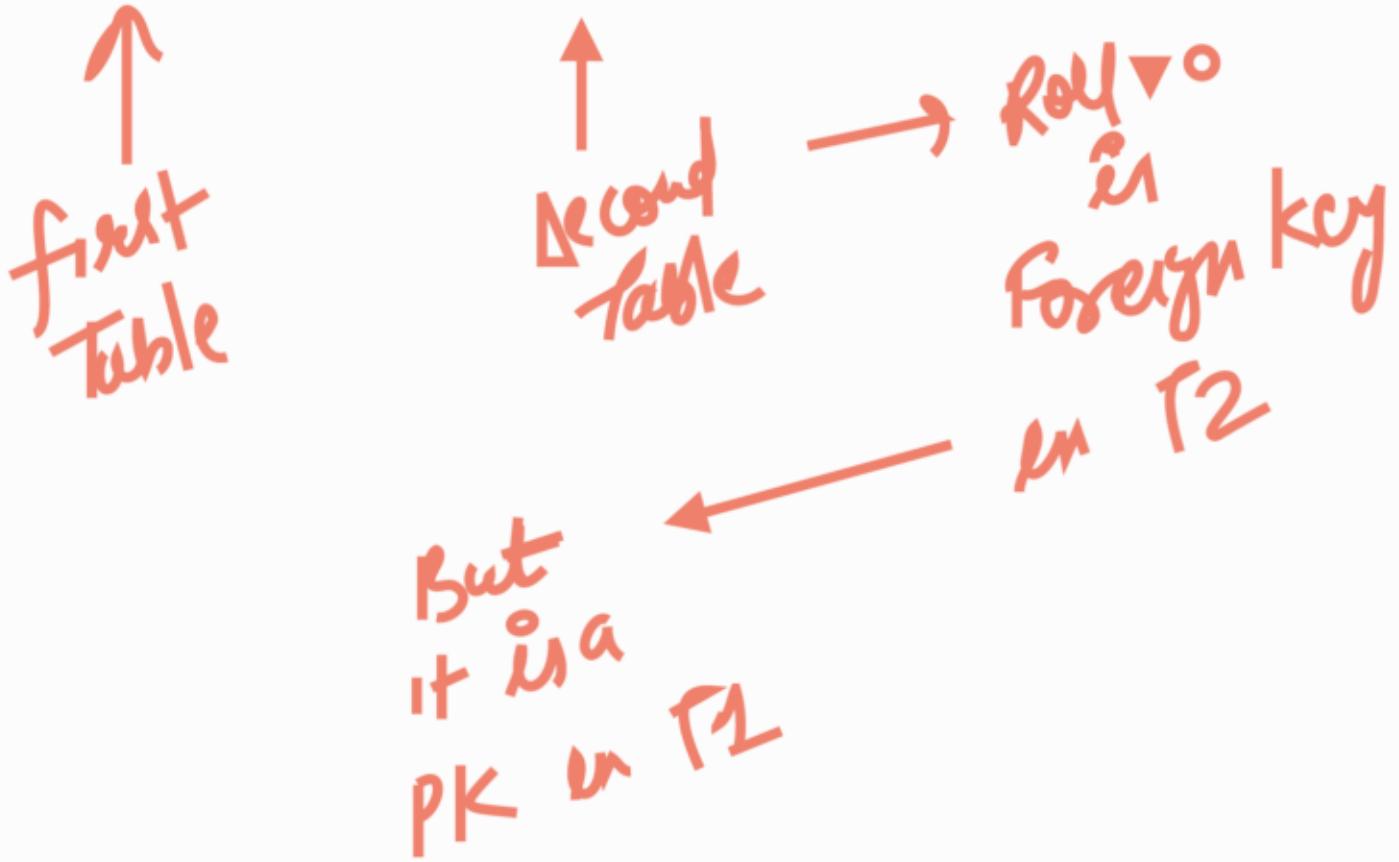
Primary key

**Subject**  
(referencing table)

Roll no	Name	subject
1	Rahul	Maths
2	Raj	SST
3	Riti	Science

Foreign key

*Referential  
Integrity*



# KEYS IN DBMS

Foreign Key: A foreign key is a field in a table that refers to the primary key in another table. It establishes a relationship between two tables.

**Student**  
(Base/referenced table)

Roll no	Name	Hometown
1	Rahul	KOLKATA
2	Raj	KOLKATA
3	Riti	DELHI

Primary key

Foreign key

**Subject**  
(referencing table)

Roll no	Name	subject
1	Rahul	Maths
2	Raj	SST
3	Riti	Science

Roll no is  
that key  
PK for 1<sup>st</sup>  
FK for 2<sup>nd</sup>

# KEYS IN DBMS

Referenced table - Table having primary key (pk)

Referencing table- Table having foreign key(fk)

**Student**  
(Base/referenced table)

Roll no	Name	Hometown
1	Rahul	KOLKATA
2	Raj	KOLKATA
3	Riti	DELHI

↓  
Primary key

**Subject**  
(referencing table)

Roll no	subject id	subject
1	s1	Maths
2	s2	SST
3	s3	Science

↓  
Foreign key

Base table  
↓  
Referenced table.

Other table  
↓  
Referencing table.

# KEYS IN DBMS

## Referential Integrity in Foreign key.

Referential integrity is an important concept in foreign key. We always say foreign key maintains referential integrity.

Referential integrity ensures that the relationships between tables remain accurate, consistent, and meaningful within a relational database.

Changes in  
1<sup>st</sup> Table and  
2<sup>nd</sup> are  
consistent  
& Accurate.

Integrity means consistent / same  
everywhere.

# KEYS IN DBMS

## Referential Integrity in Foreign key

Now consider there are two tables one is referencing and other is referenced table .

Lets see how some operations like insert, update and delete works here.

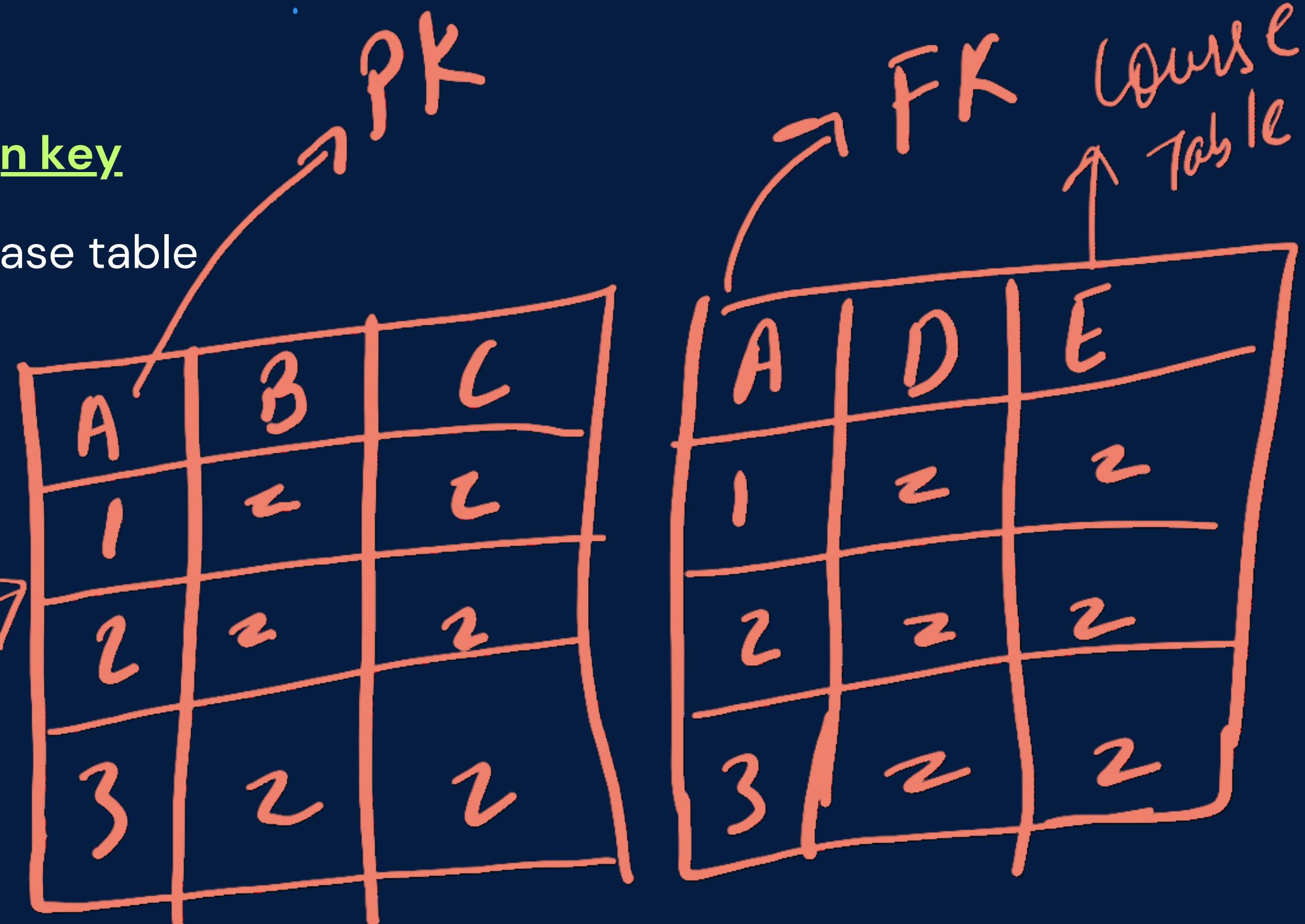
# KEYS IN DBMS

## Referential Integrity in Foreign key.

- Insertion in Referenced/base table

No violation

Insertion  
doesn't  
affect Referencing Table.



# KEYS IN DBMS

## Referential Integrity in Foreign key.

- Deletion in Referenced/base table

May cause violation if the corresponding data is present in referencing table.

if same data is present in both  
table & if it is getting deleted  
from Base T then → Violation

# KEYS IN DBMS

## Referential Integrity in Foreign key.

for this we use cascading  
so that we can maintain  
referential integrity.

If a record in referenced table is deleted or updated , the corresponding records in the referencing table should be deleted or updated to maintain the integrity of the relationship.

We using action like "CASCADE DELETE" for the same. Also we can set null for the values deleted.

Cascade delete is used for managing dt operation on  
Base Table .

# KEYS IN DBMS

## Referential Integrity in Foreign key.

- Updation in Referenced/base table

May cause violation if the corresponding data is present in referencing table. We can use action like "CASCADE UPDATE".

→ Used to maintain  
R. I. when updating  
base table.

# KEYS IN DBMS

## Referential Integrity in Foreign key.

- Insertion in Referencing table

May cause violation

if we insert something in  
subject table with new roll  
no which doesn't exist in  
student table will cause  
violation.

But if any student  
with existing roll is added in subject table  
it won't cause violation.

Student

Roll	Name	Ph NO
1	A	00
2	B	01
3	C	2L

subject

Slid	sName	Roll
01	Ph	1
02	Che	2
03	Bio	3

That's why **May** is used.

# KEYS IN DBMS

## Referential Integrity in Foreign key.

- Deletion in Referencing table

No violation

→ Because student can leave the subject if he doesn't want it but it won't cause any violation in Base Table.

# KEYS IN DBMS

## Referential Integrity in Foreign key.

- Updation in Referencing table

No issues until we are updating foreign key attribute

Violation would be caused on updating

Till the point we  
are not updating the  
FK attribute it

doesn't cause violation

Otherwise it can  
if it doesn't find the entry  
in Base Table.

CK set of all attributes which can uniquely identify the table.

PK one of the CK.



# SUPER KEY IN DBMS

It is a set of one or more attributes (columns) that can uniquely identify a tuple (a row) in a relation (table).

Superset of any candidate key.

A super key becomes a candidate key if it is minimal (i.e. no proper subset of it can uniquely identify a tuple).

SK  $\rightarrow$  Minimal.

↳ consists atleast  $1 CK$  + Extra Attributes



$\rightarrow$  PK is always  
in 1<sup>st</sup> Normal Form

Employee  $\rightarrow \{E_1, E_2, E_3, \dots, E_n\}$

$2^{n-1}$

CK =  $\{E_1\}$

SK = ?

$\hookrightarrow E_1, E_2, E_3, \dots$  etc.



it is mandatory



Student =  $\{sName, sId, sPhoneNo, sAdhar\}$

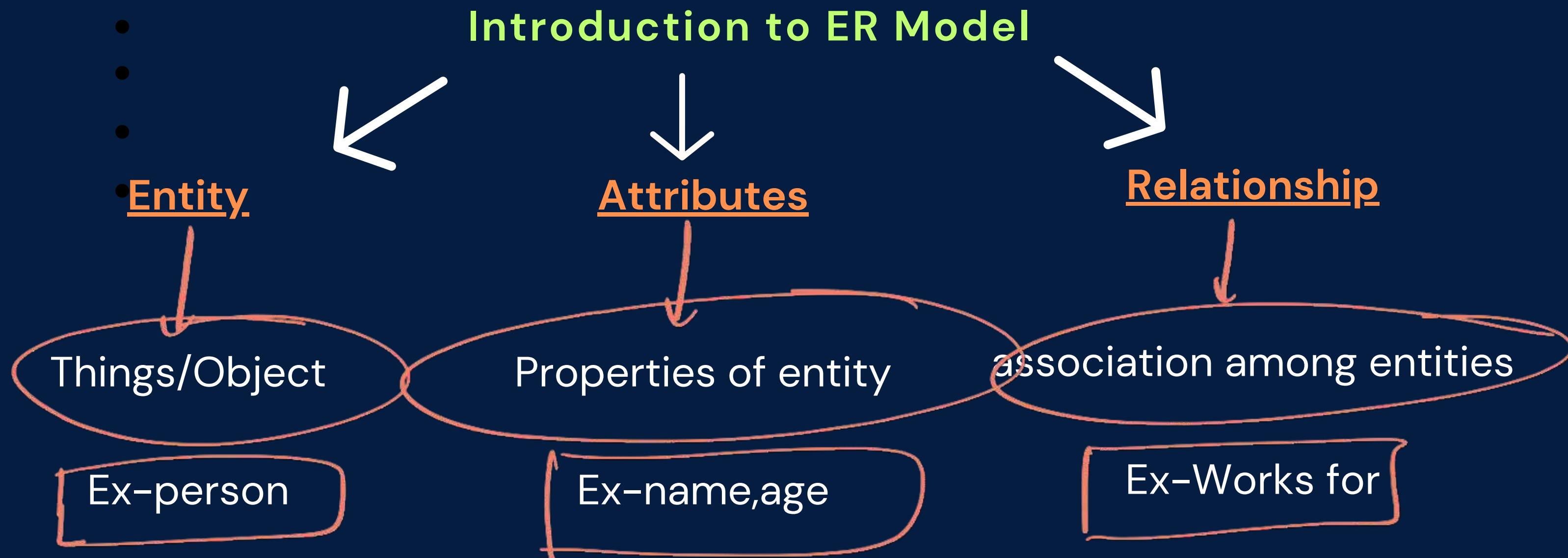
PK = Unique + Not Null  $\Rightarrow sId$

CK =  $\{sId, sPhoneNo, sAdhar\}$

SK =  $\{sId, sName\}$

$\uparrow$        $\downarrow$   
CK + Extra Attribute.

# ER MODEL IN DBMS

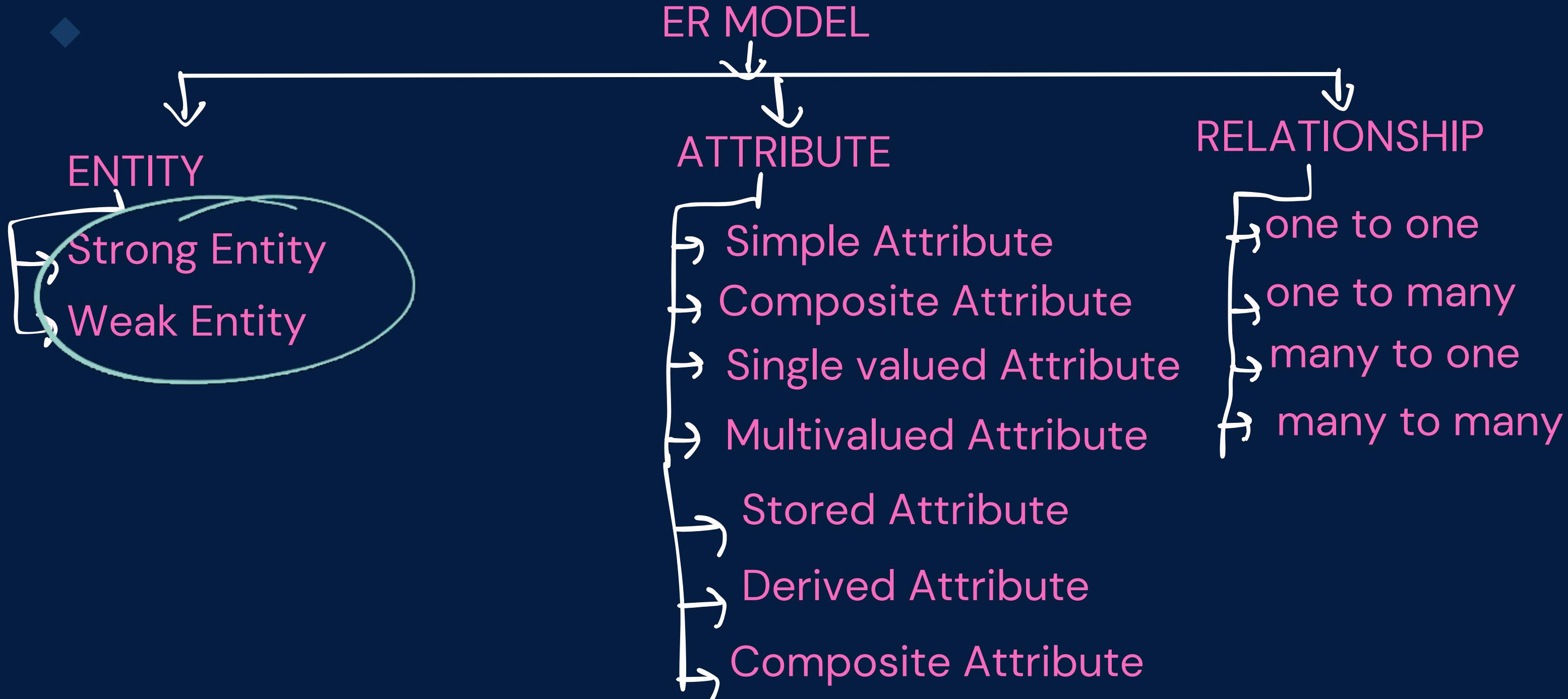


ER Diagram Basically provides a visual representation of database Architecture

## ER MODEL IN DBMS

- The Entity-Relationship (ER) model stands as a prevalent conceptual modeling approach within the realm of database design.
- Its primary role is to offer a visual representation of a database's architecture by illustrating the entities, their respective attributes, and the interconnections between them.
- In the process of database design, the ER model holds significant importance, aiding in the development of an efficient and systematically structured database schema.

# ER MODEL IN DBMS

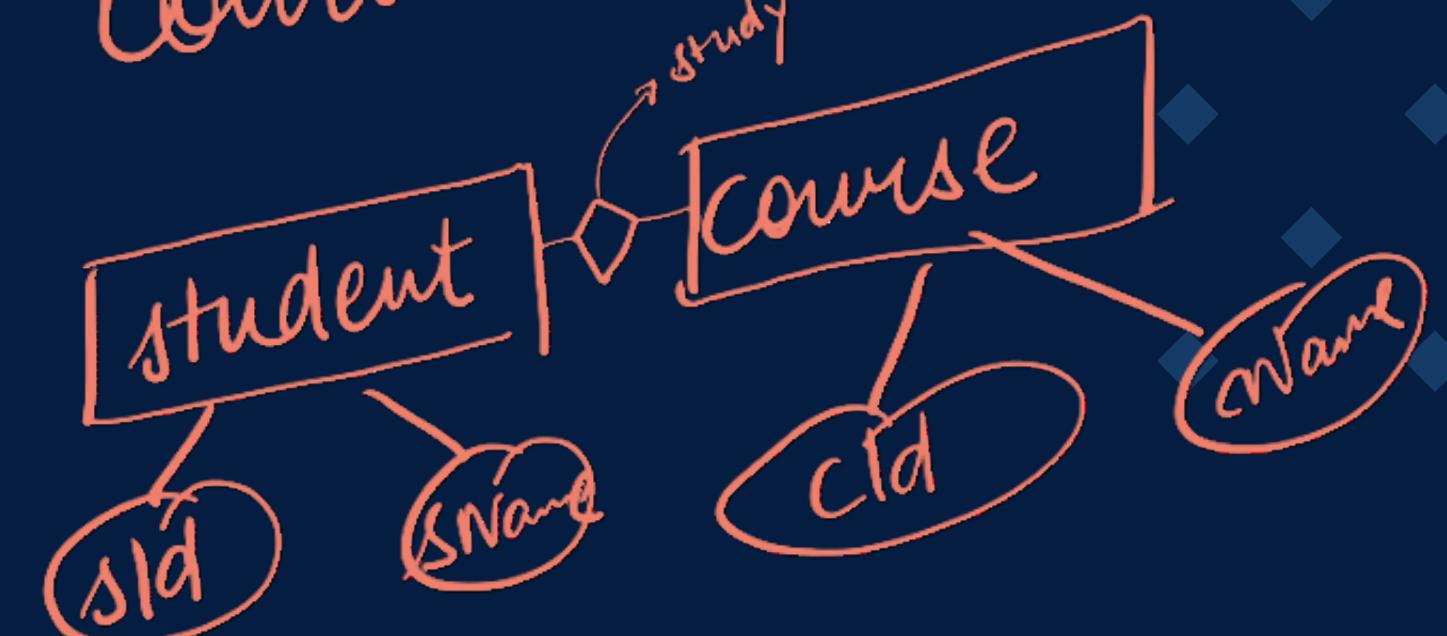


# ER MODEL IN DBMS

## Symbols used in ER Model

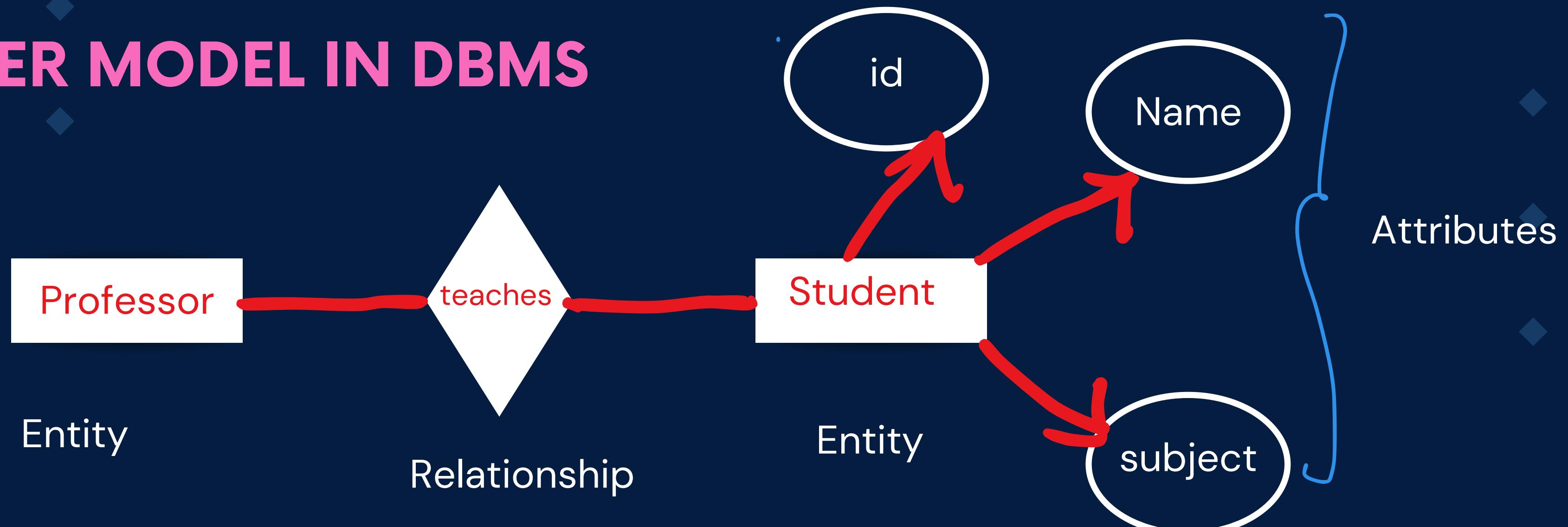
Figures	Symbols	For what
Rectangle		Entity
Ellipse		Attribute
Diamond		Relationship
Line		Attribute to entity relationship
Double ellipse		Multivalued attributes
Double rectangle		Weak Entity

Student (sId, sName)  
Course (cId, cName)



Study is  
Relationship  
btw them

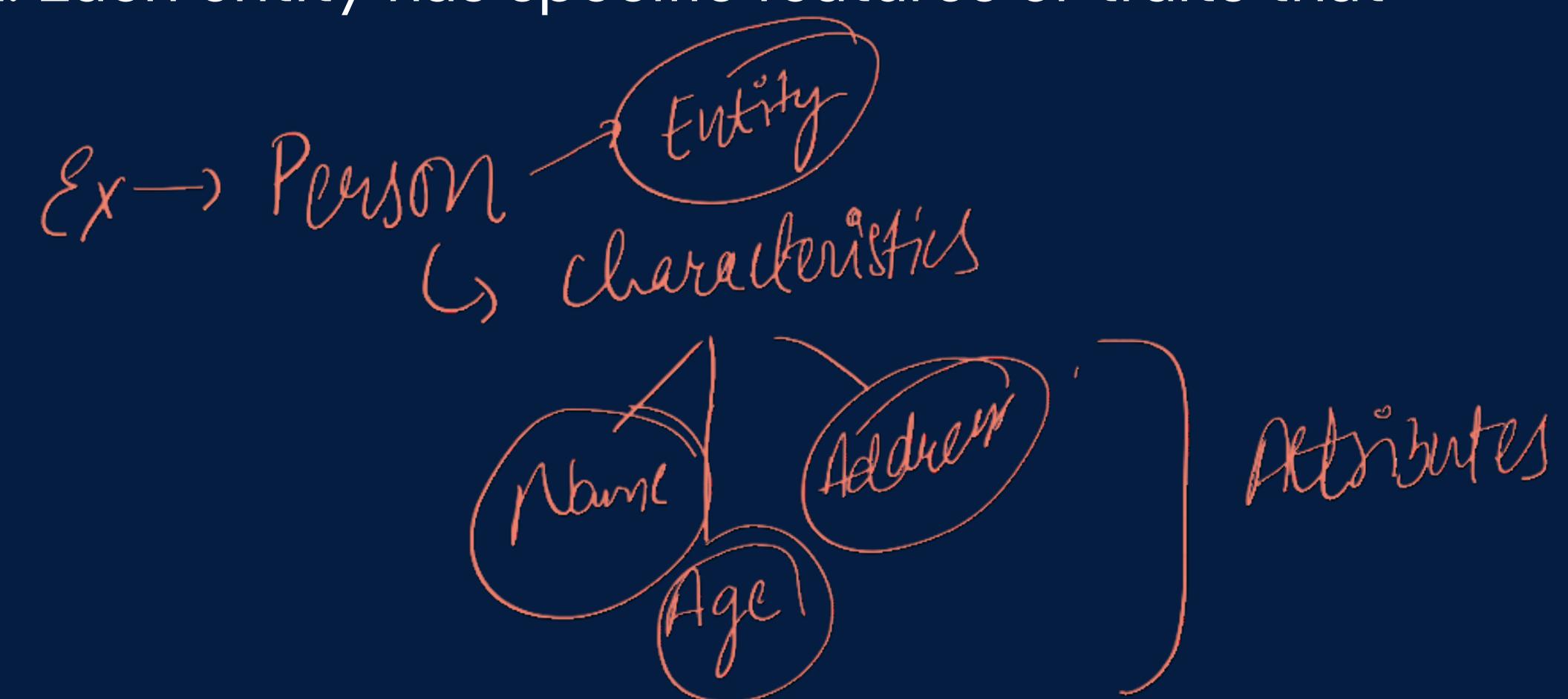
# ER MODEL IN DBMS



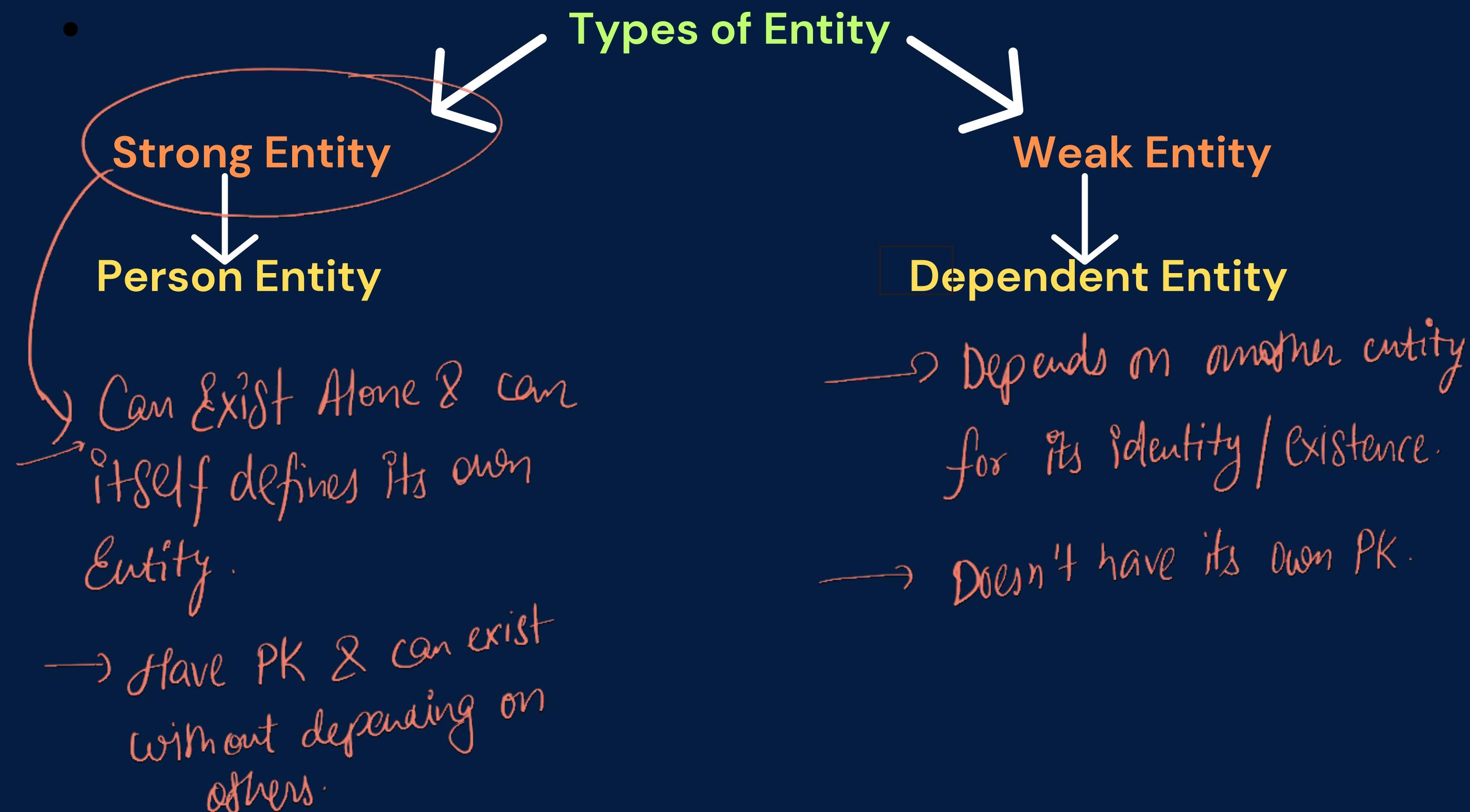
# ER MODEL IN DBMS

- Entity

An entity is something from the real world, like a person, place, event, or idea. Each entity has specific features or traits that describe it.



# ER MODEL IN DBMS



# ER MODEL IN DBMS

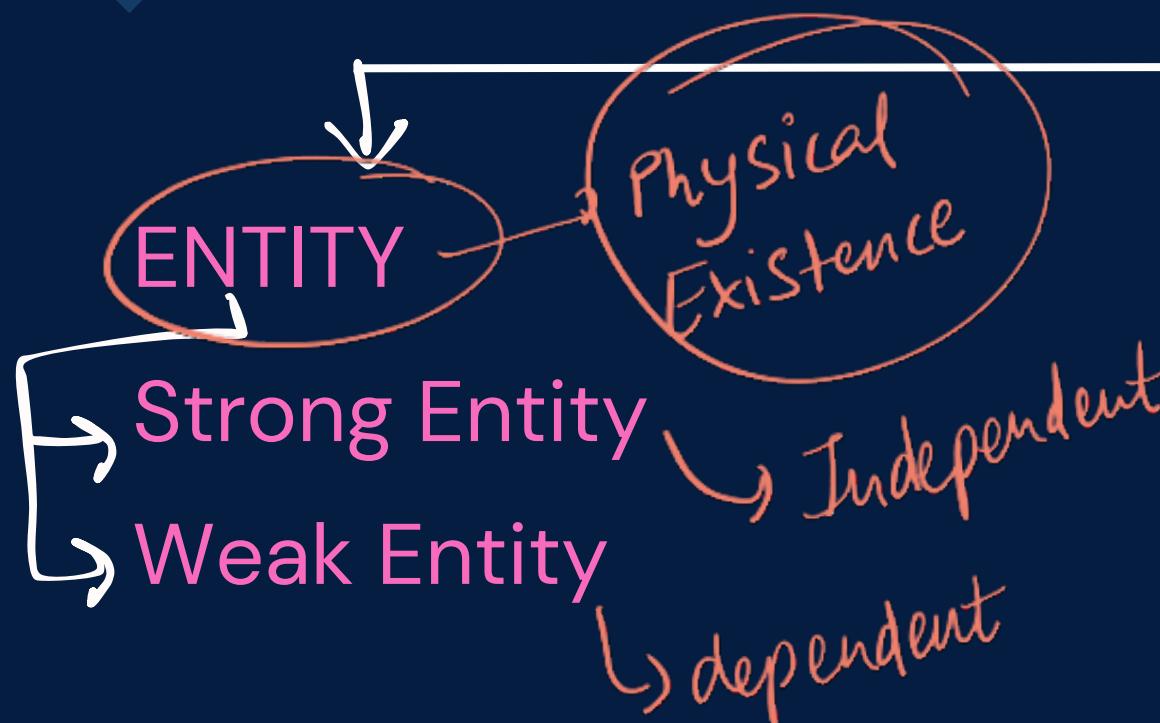
## Types of Entity

**Strong Entity:** A strong entity is an entity that has its own unique identifier (primary key) and is not dependent on any other entity for its existence within the database. Strong entities stand alone and have their own set of attributes.

Ex-Person

**Weak Entity:** A weak entity is an entity that doesn't have a primary key of its own. It relies on a related strong entity (known as the "owner" entity) for its identity. The weak entity's existence is defined by being related to the owner entity.  
ex- dependent

# ER MODEL IN DBMS



## ER MODEL

### ATTRIBUTE

- Simple Attribute
- Composite Attribute
- Single valued Attribute
- Multivalued Attribute
- Stored Attribute
- Derived Attribute
- Composite Attribute

### RELATIONSHIP

- one to one
- one to many
- many to one
- many to many

# ER MODEL IN DBMS

Attribute

→ Properties of an entity.

Attributes represent properties or characteristics of an entity or relationship.

They provide information about the entities and relationships in the database.

# ER MODEL IN DBMS

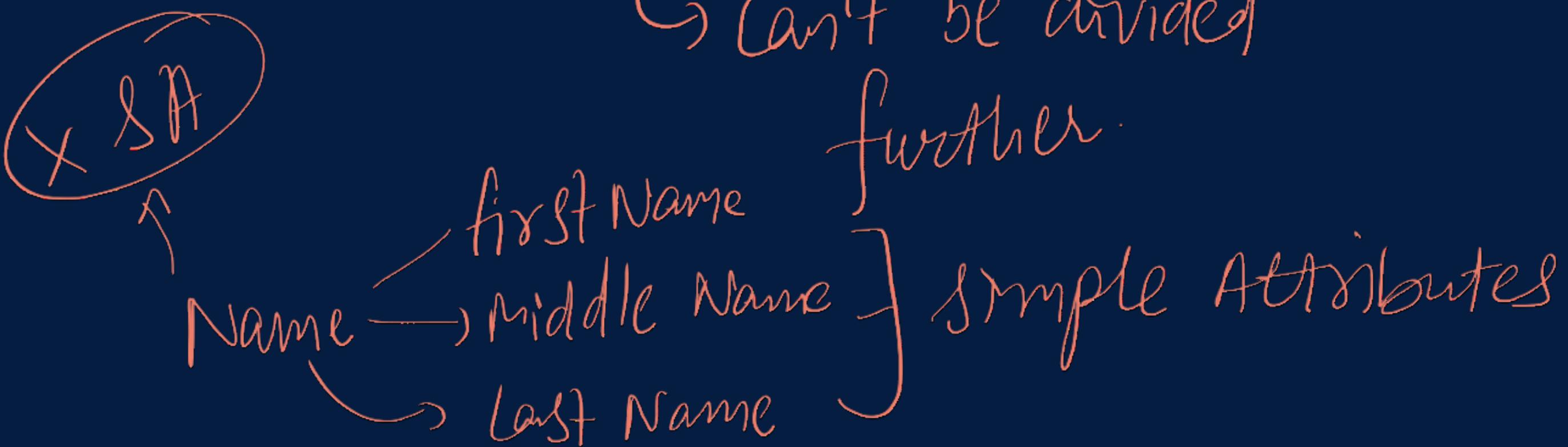
## Types of Attributes

### Simple Attribute

A simple attribute is atomic and cannot be divided any further.

Ex- First Name

→ Can't be divided  
further.



# ER MODEL IN DBMS

## Types of Attributes

### Composite Attribute

A composite attribute is made up of several smaller parts, where each part represents a piece of the whole attribute. In simpler terms it is composed of attributes which can be divided further.

Ex- Name( First Name, lastName)

Composite  
Attribute

# ER MODEL IN DBMS

## Types of Attributes

### Single Valued Attribute

A single-value attribute is an attribute that holds a single value  
for each entity

Ex- Age ↪ Contains only 1 value for each entity.

Age will always have 1 value ⇒ Each Entry.

# ER MODEL IN DBMS

## Types of Attributes

### Multivalued Attribute

A multi-valued attribute in a database is an attribute that can hold multiple values for a single entity.

Ex- Address (permanent, residential) → Multiple Values Exist

⇒ Phone NO  
⇒ Marks, etc.

# ER MODEL IN DBMS

## Types of Attributes

### Stored Attribute

Attribute that is stored as a part of a database record.

Ex- Date of birth

which we can  
store in DB

# ER MODEL IN DBMS

## Types of Attributes

### Derived Attribute

A derived attribute is derived from other attributes within the database.

Ex- Age derived from dob

which can be  
derived from other

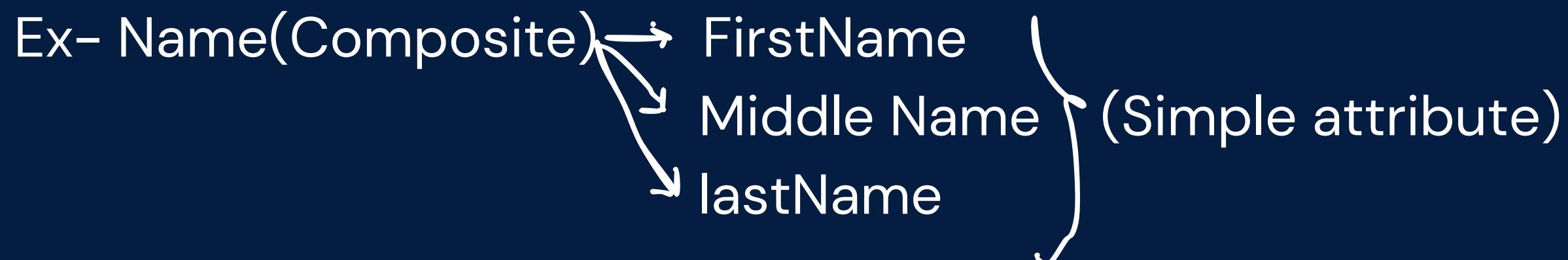
CTC  $\rightarrow$  Basic + Bonus, etc.

# ER MODEL IN DBMS

## Types of Attributes

### Complex Attribute

A complex attribute is an attribute that is made up of multiple smaller attributes



# ER MODEL IN DBMS

## Relationship in ER Model

**Relationship in ER MODEL** is the connection between entities (tables) based on related data.

# ER MODEL IN DBMS

## Types of Relationship

Strong Relationship

↓  
Dependant on  
each other, can't  
exist alone.

Like Salary Table can't exist without employees.

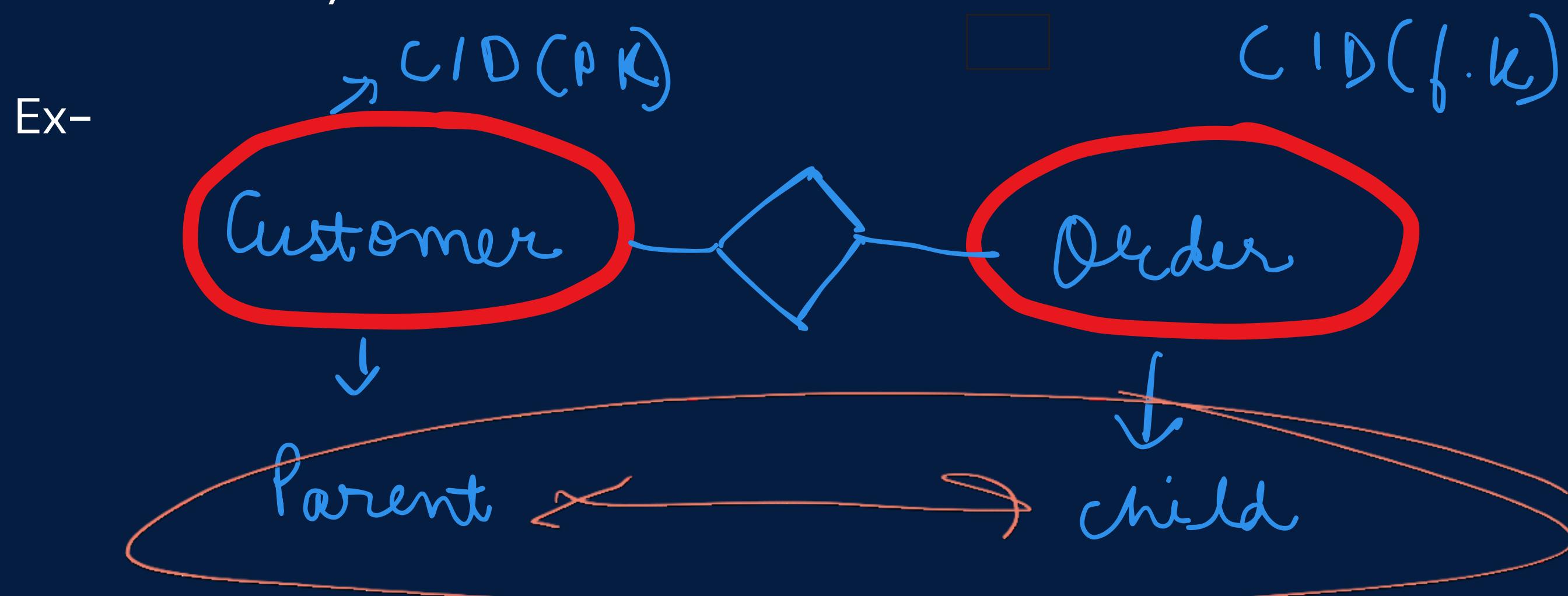
Weak Relationship

Can stay independently  
like  
→ Student Table & Staff Table.

# ER MODEL IN DBMS

## Strong Relationship

A strong relationship exists when two entities are highly dependent on each other, and one entity cannot exist without the other.

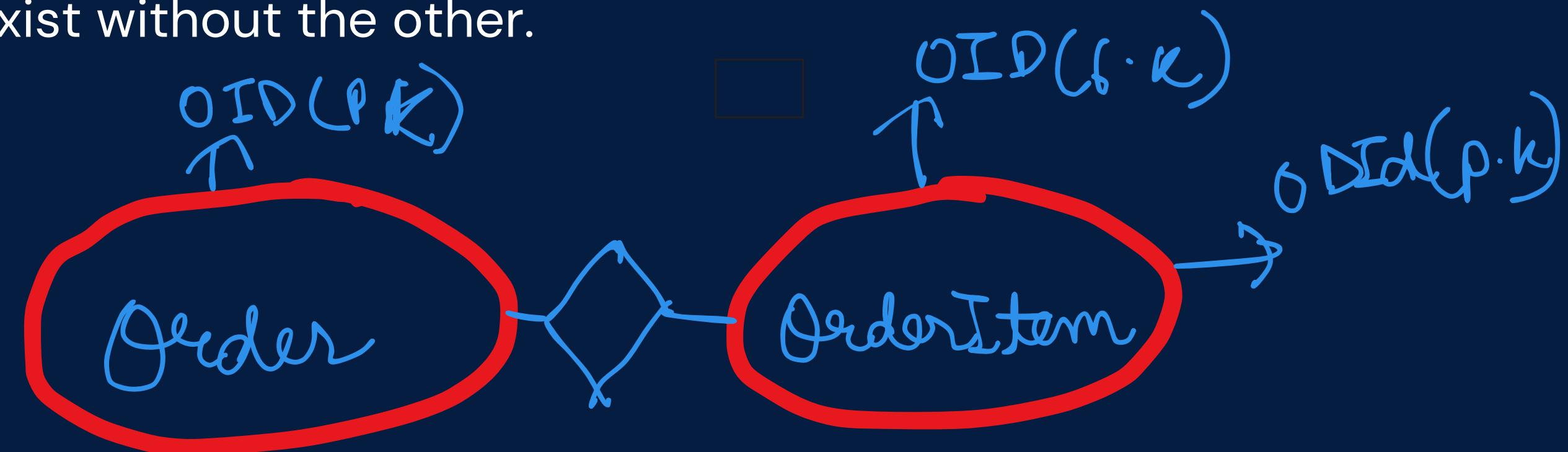


# ER MODEL IN DBMS

## Weak Relationship

A weak relationship, on the other hand, exists when two entities are related, but one entity can exist without the other.

Ex-



# ER MODEL IN DBMS

## Degree in DBMS

A degree in dbms refers to the number of attributes / columns that a relation/table has.

# ER MODEL IN DBMS

## Types of Degree

Degree	Name	Definition
1	Unary Degree	A relation with a single attribute
2	Binary Degree	A relation with two attributes
3	Ternary Degree	A relation with three attributes
n	n-ary Degree	A relation with more than three attributes n>3

student (sId)  
student (sId, sName)  
student (sId, sName, sAge).

# ER MODEL IN DBMS

**Null value :** In databases, a null value can occur for various reasons

① Not Needed Information: Sometimes, some details are asked, but they don't apply to everyone. For instance, asking for a "Spouse Name" from someone who isn't married.

② Don't Know the Answer: Every now and then, we're asked a question, but we don't have an answer yet.

Forgot to Fill In: Like when you're filling out a form, and you accidentally miss putting in some important information.

Jaha lge k  
val empty ho  
skh h vha

NULL

daal

done ka

# ER MODEL IN DBMS

## Types of relationship in dbms (Based on degree)

There are 4 types of relationship:

- one to one (1-1)
- one to many (1-N)
- many to one (N-1)
- many to many (N-N)

Associated

w. of  
attributes

is the

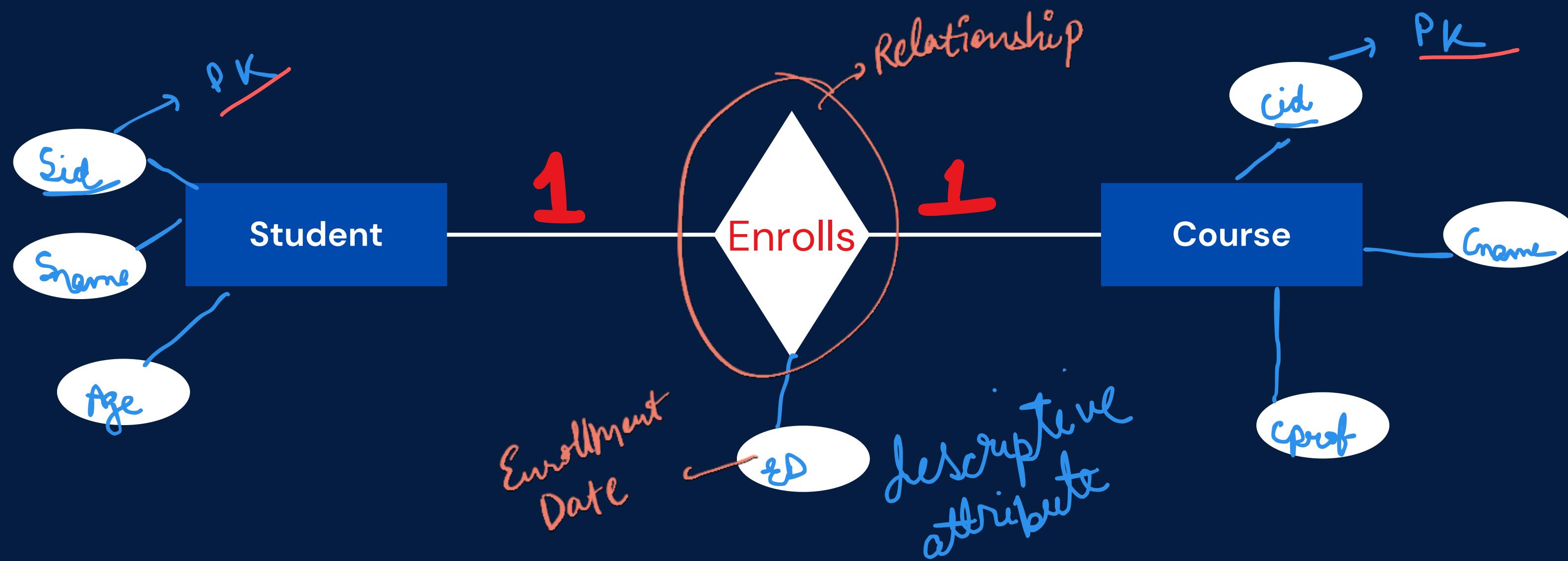
Criteria for differentiation

# ER MODEL IN DBMS

## Types of Relationship(Cardinality)

### 1 to 1 Relationship(1:1)

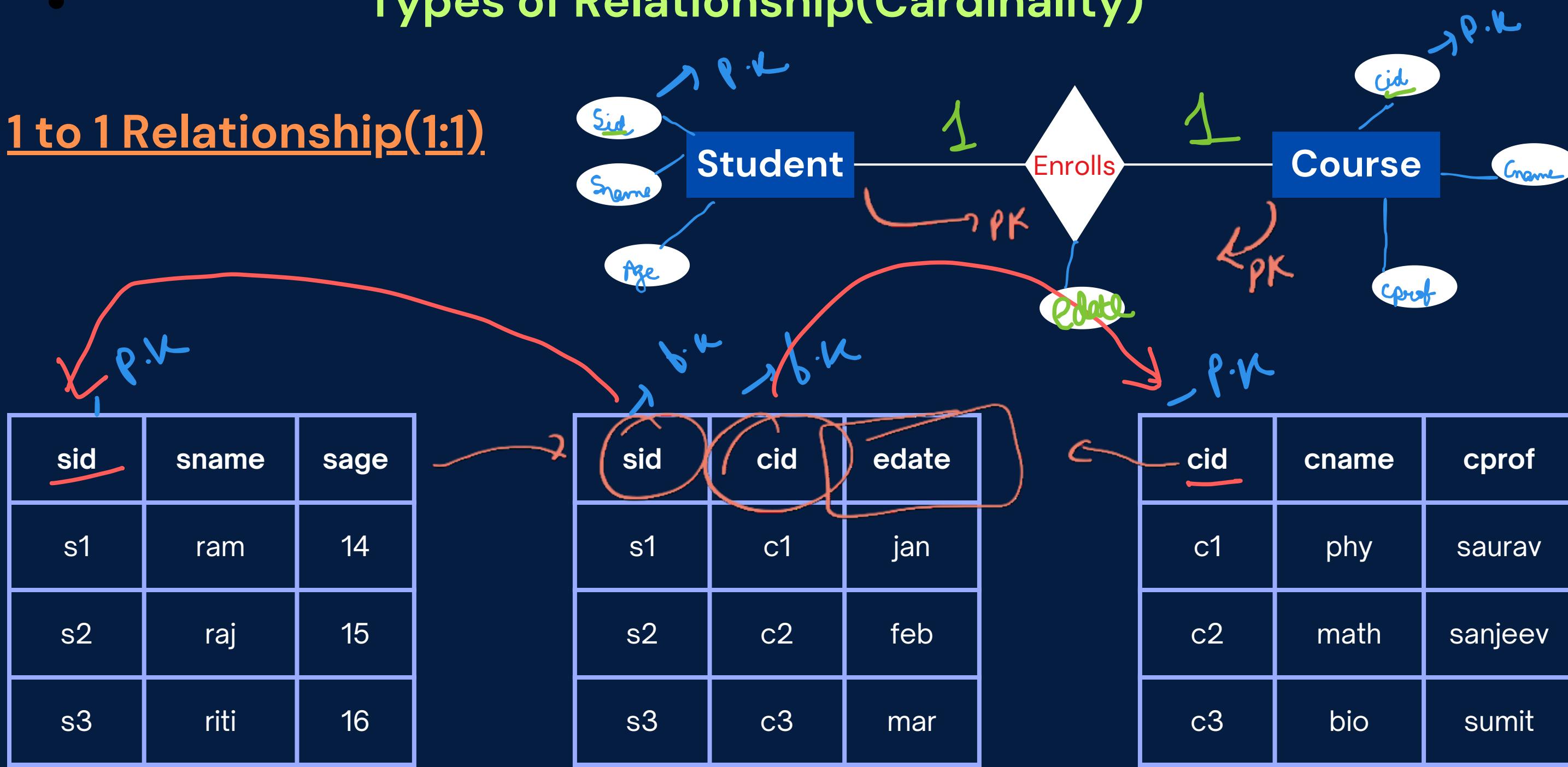
Each row in one table is associated with one and only one row in the other table, and vice versa.



# ER MODEL IN DBMS

## Types of Relationship(Cardinality)

### 1 to 1 Relationship(1:1).



→ Establishing Relationship blw tables.

# ER MODEL IN DBMS

sid	sname	sage
s1	ram	14
s2	raj	15
s3	riti	16

+

Combined

sid	cid	edate
s1	c1	jan
s2	c2	feb
s3	c3	mar

cid	cname	cprof
c1	phy	saurav
c2	math	sanjeev
c3	bio	sumit

assume (P.h)

PK

FK

Reducing Table

sid	sname	sage	cid	edate
s1	ram	14	c1	jan
s2	raj	15	c2	feb
s3	riti	16	c3	mar

PK

cid	cname	cprof
c1	phy	saurav
c2	math	sanjeev
c3	bio	sumit

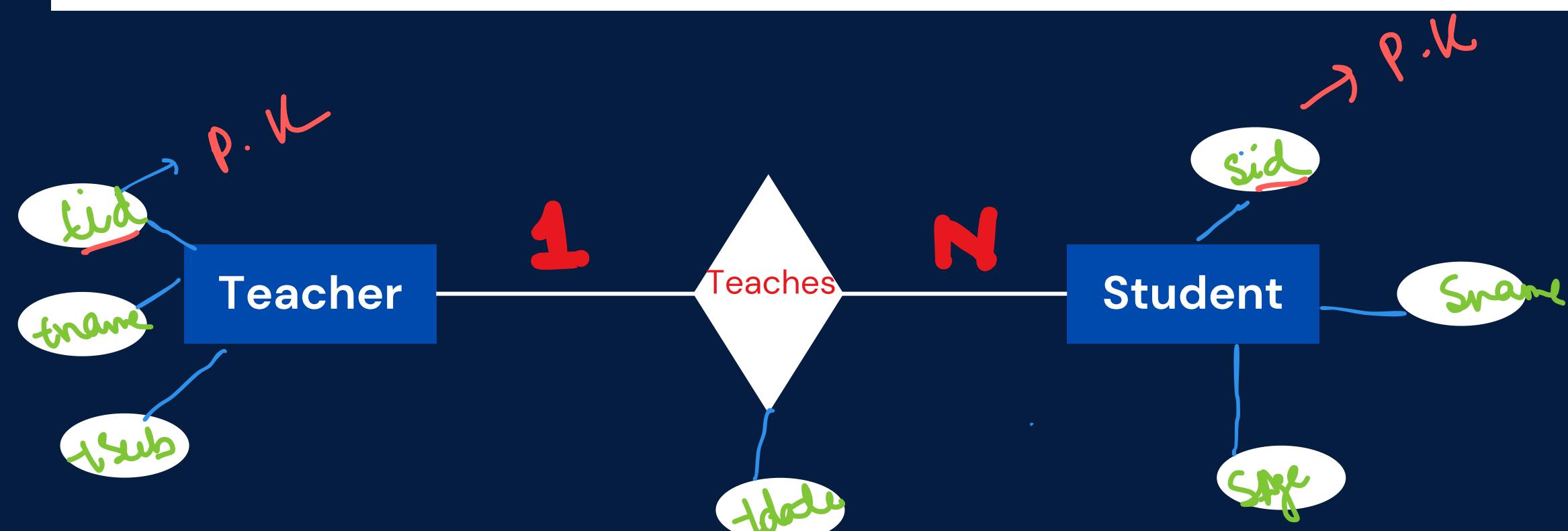
# ER MODEL IN DBMS

## Types of Relationship

### 1 to Many Relationship(1:N).



A single record in one table may have connections to multiple records in another table, while each record in the second table is linked to only one record in the first table.



# ER MODEL IN DBMS

## Types of Relationship

### 1 to Many Relationship(1:N).

<u>sid</u>	sname	sage
s1	ram	14
s2	raj	15
s3	riti	16

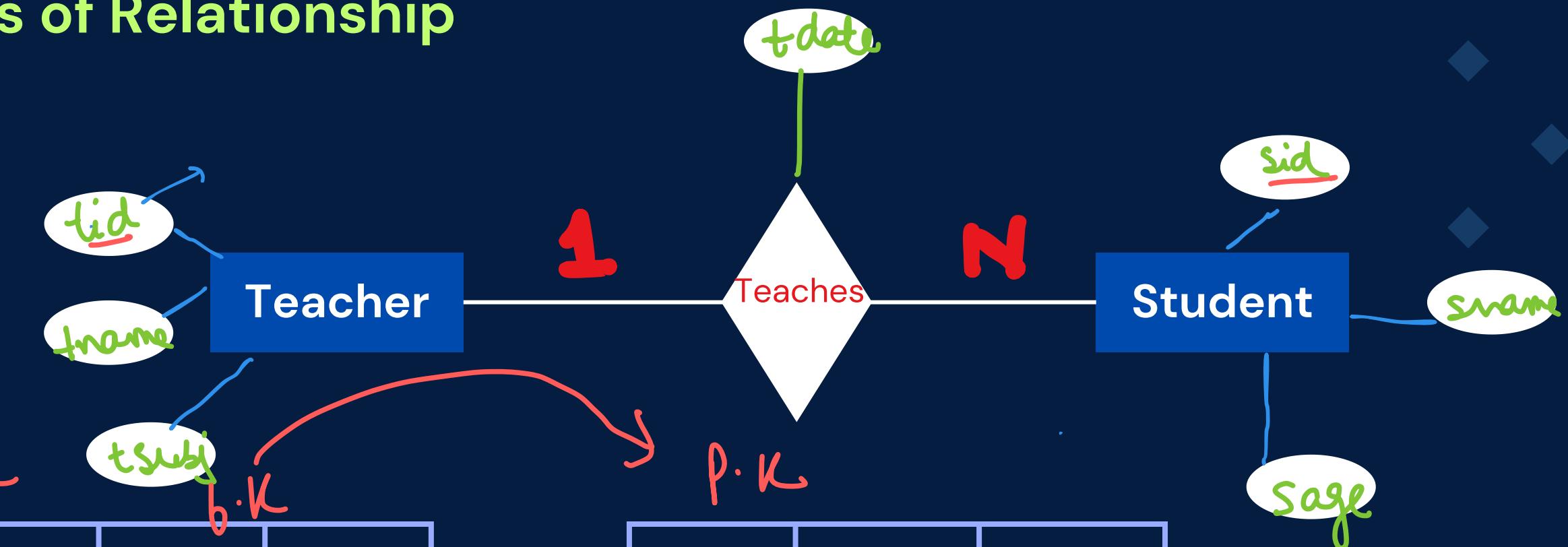
teacher

sid	tid	tdate
s1	c1	jan
s2	c2	feb
s3	c3	mar

teaches

<u>tid</u>	tsubj	tname
c1	phy	saurav
c2	math	sanjeev
c3	bio	sumit

Subj



# ER MODEL IN DBMS

sid	sname	sage
s1	ram	14
s2	raj	15
s3	riti	16

PK

FK

(P-K)

PK

sid	tid	tdate
s1	c1	jan
s2	c2	feb
s2	c3	mar

PK

+

Combines

tid	tname	tsub
c1	phy	saurav
c2	math	sanjeev
c3	bio	sumit

PK

sid	sname	sage
s1	ram	14
s2	raj	15
s3	riti	16

PK

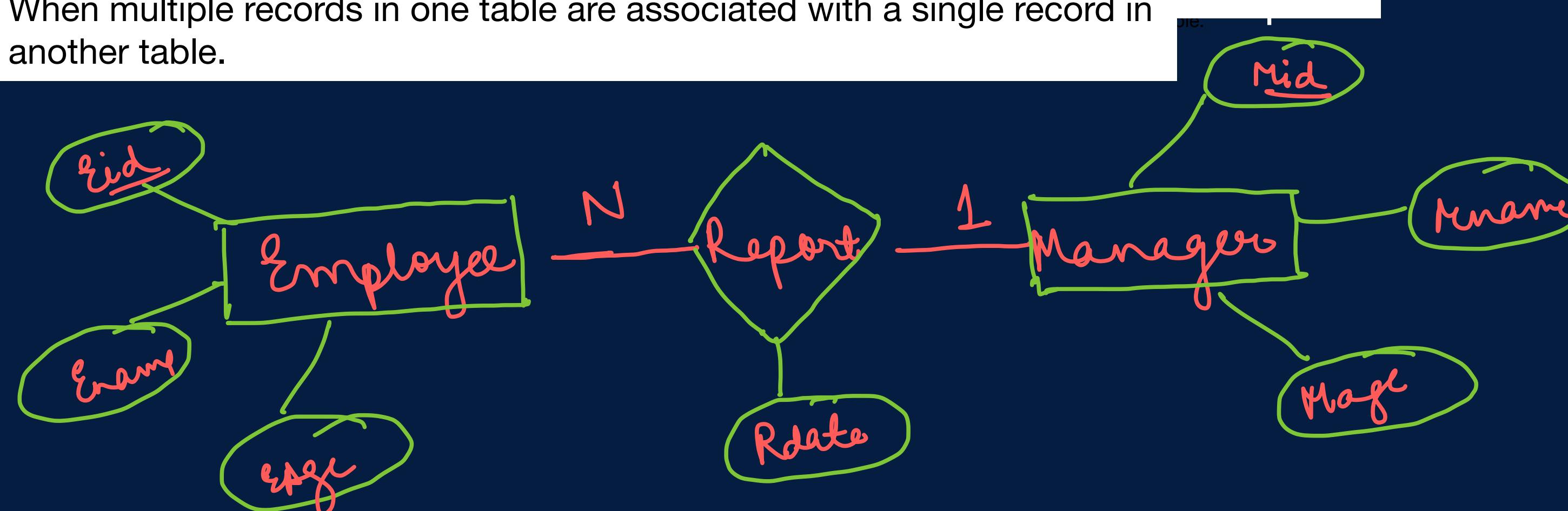
tid	tsub	tname	sid
c1	phy	saurav	s1
c2	math	sanjeev	s2
c3	bio	sumit	s3

# ER MODEL IN DBMS

## Types of Relationship

### Many to 1 Relationship(N:1).

When multiple records in one table are associated with a single record in another table.



# ER MODEL IN DBMS

## Types of Relationship

### Many to 1 Relationship(N:1).

1. Primary Key → the one at the many side(Eid)
2. Reduction → combine the many + relationship table.  
(Emp + Report)

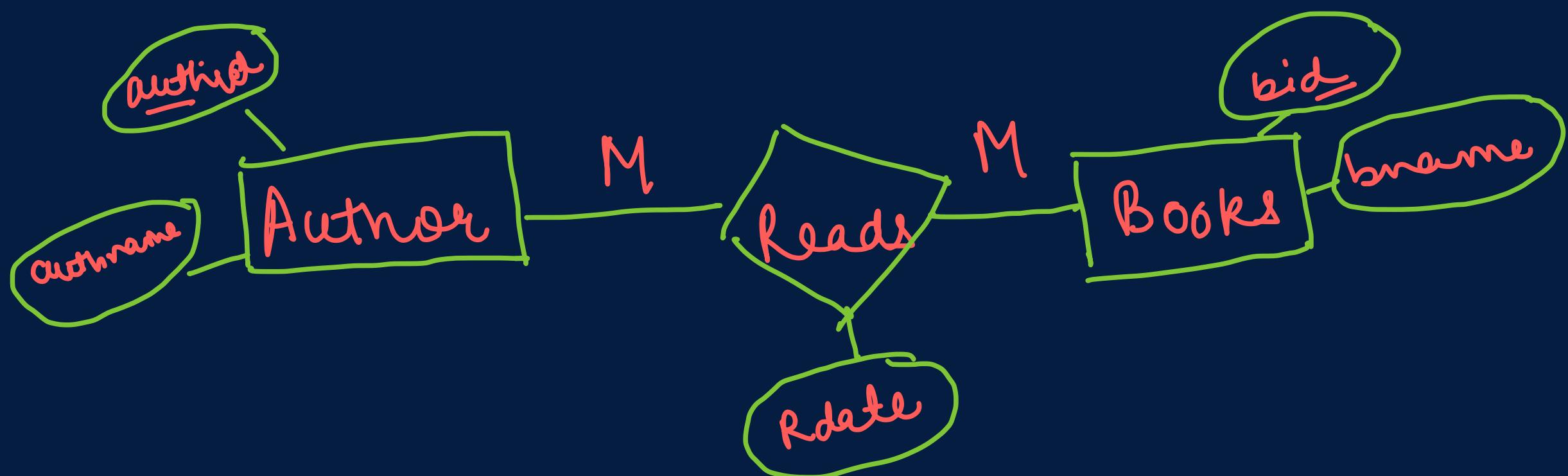
Always Combine  
Many + Relationship  
Tables

# ER MODEL IN DBMS

## Types of Relationship

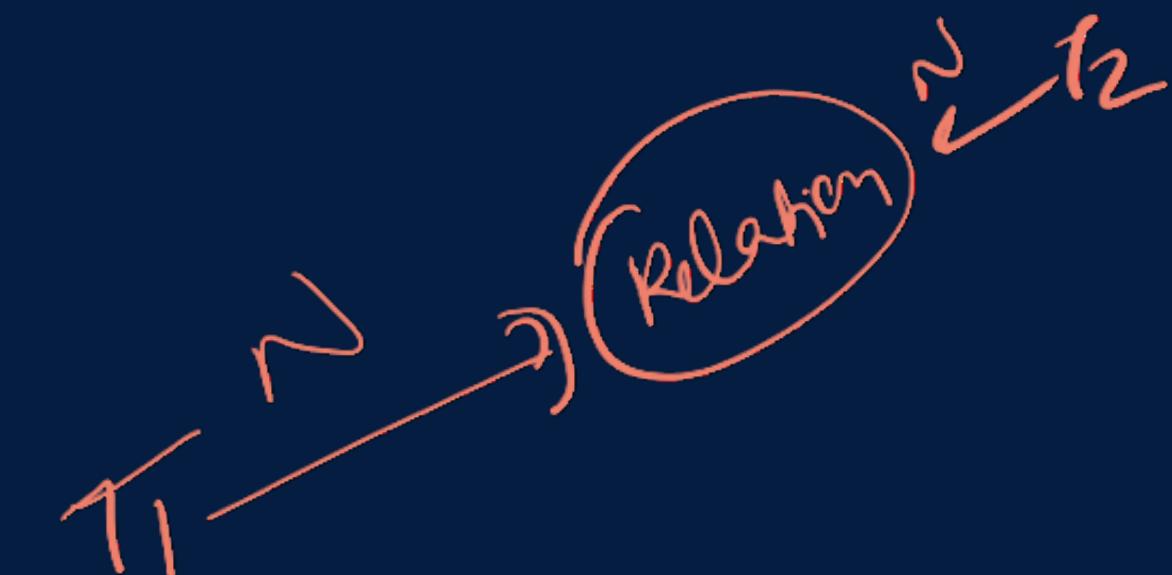
### Many to many Relationship(N:N)

When multiple records in one table can be associated with multiple records in another table



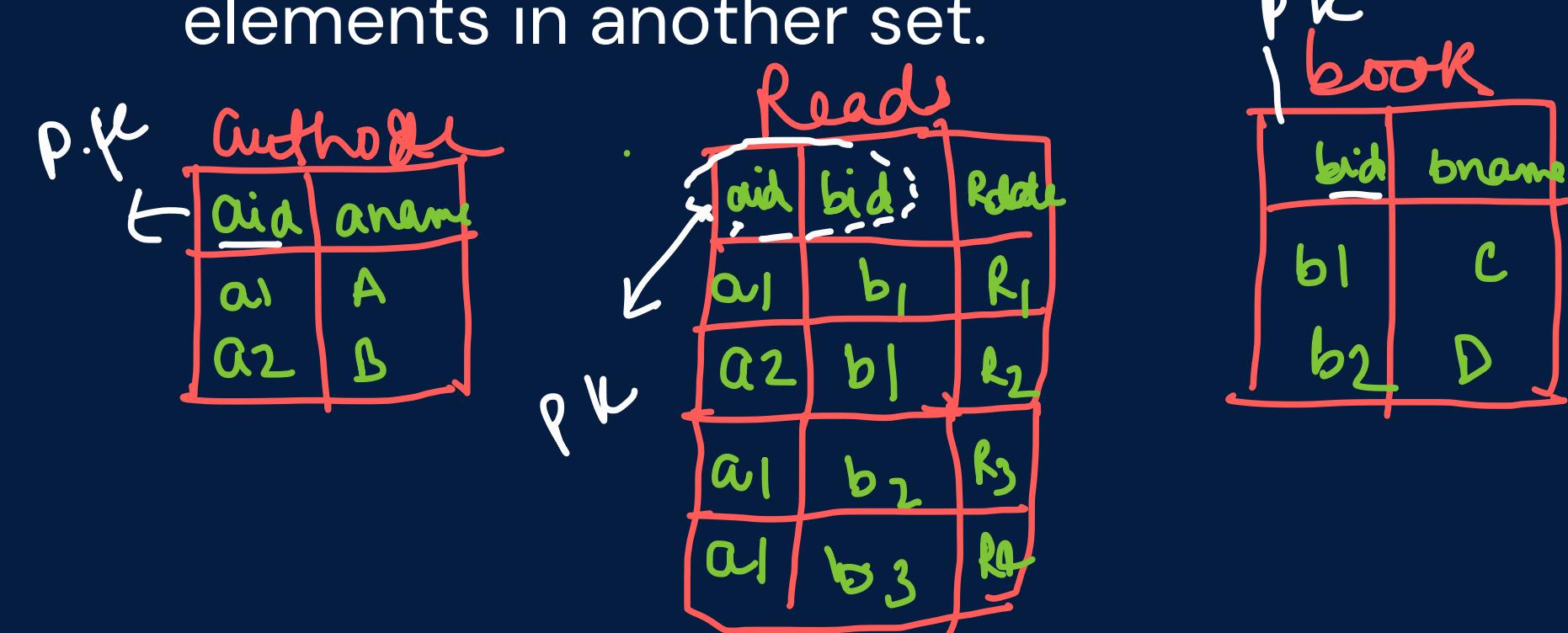
# ER MODEL IN DBMS

## Types of Relationship



### Many to many Relationship(N:N)

In this relationship multiple elements from one set are related to multiple elements in another set.



# ER MODEL IN DBMS

## Participation Constraints

Participation Constraints in an ER model define whether every entity in one group must be connected with at least one entity in another group or if the connection is optional.

# ER MODEL IN DBMS

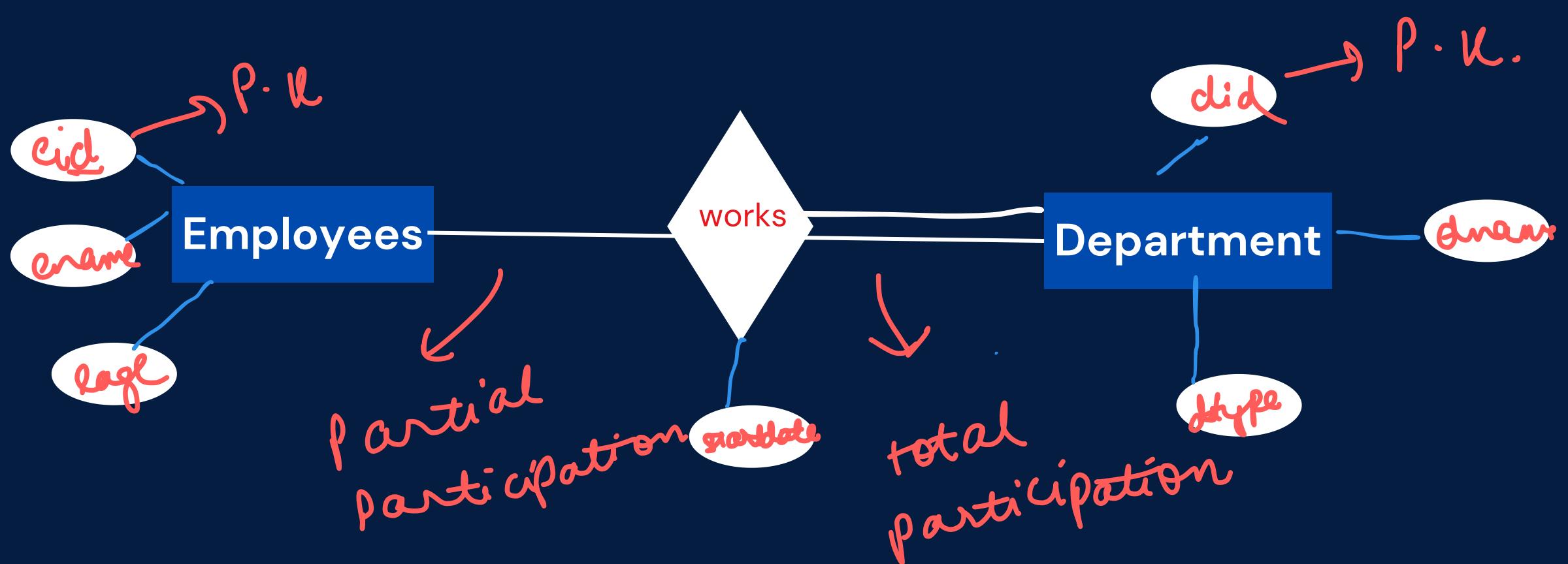
if one entity is connected  
to atleast one entity in related  
entity set

## Types of Participation Constraints

### Total Participation(Mandatory)



In a total participation constraint, each entity in a participation set must be associated with at least one entity in the related entity set.



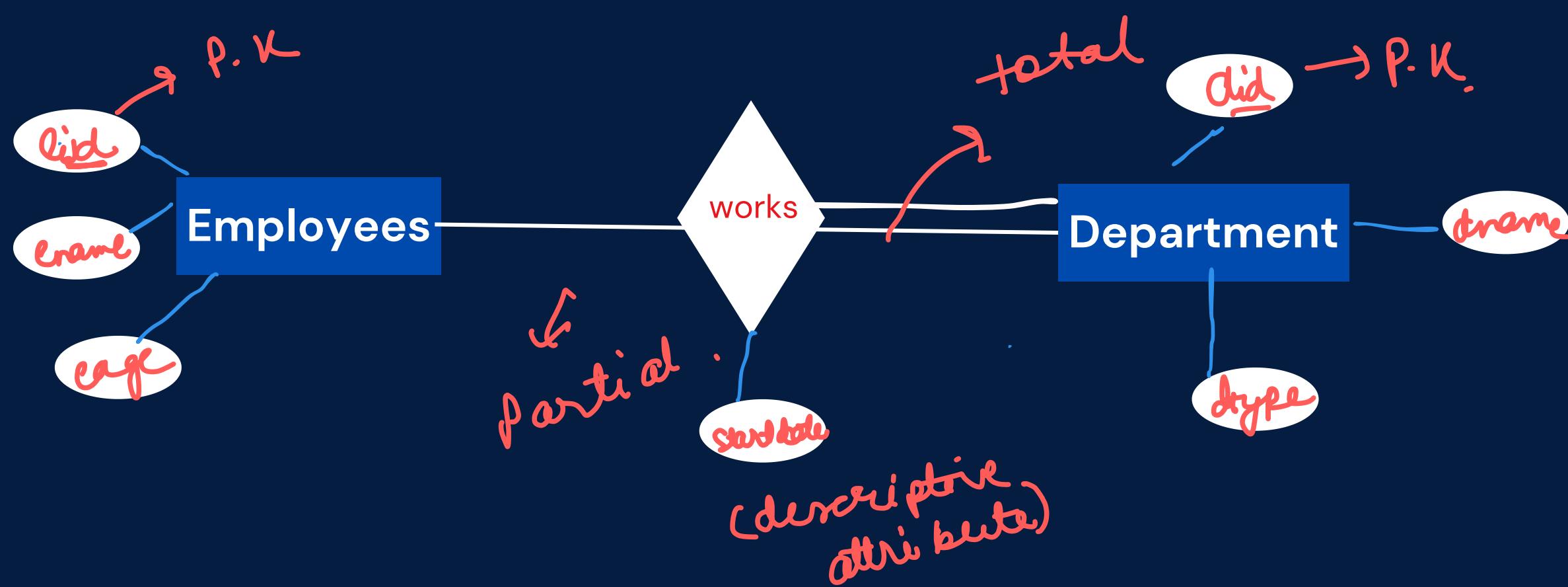
Each department  
must have a  
employee . So  
total participation

# ER MODEL IN DBMS

## Types of Participation Constraints

### Partial Participation(Optional)

In a partial participation constraint, entities in the participating entity set may or may not be associated with entities in the related entity set.



# ER MODEL IN DBMS

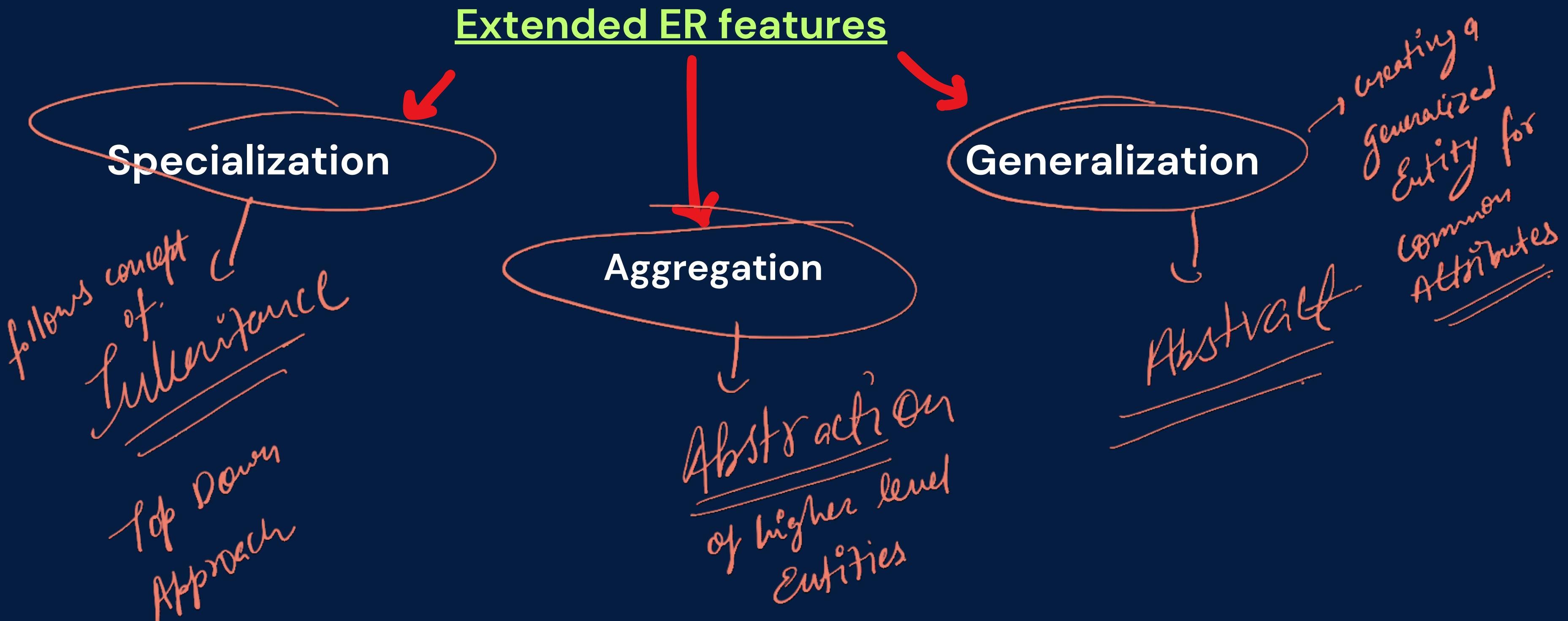
## Extended ER features

Why do we need?

We design ER model for relationship betwn entities

In real-world the data may exhibit some hierarchical relationships, and the EER model provides mechanisms to represent these relationships accurately which helps in code reusability, ensuring data integrity and consistency and lower the complexity.

# ER MODEL IN DBMS



EER → Extended Entity Relationship

# ER MODEL IN DBMS

## Extended ER features

### Specialization

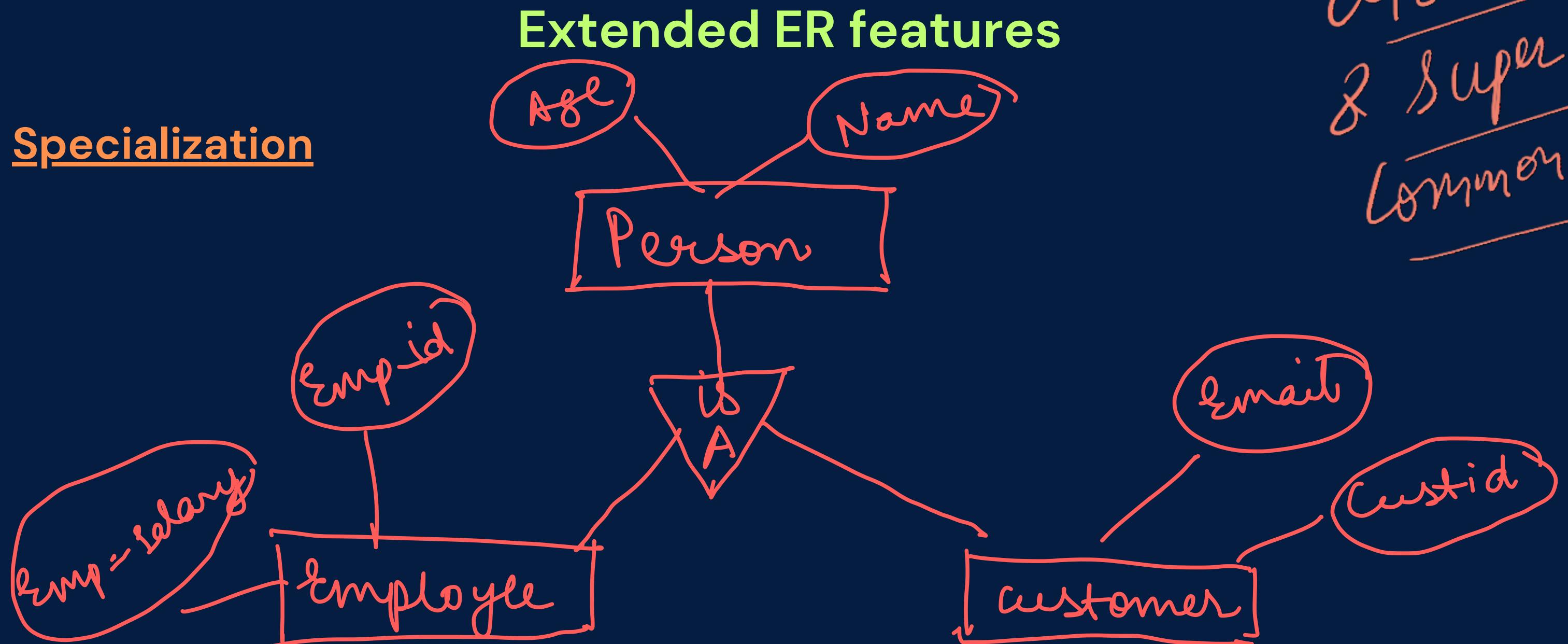
Specialization in the ER model is like categorizing entities based on common features.

A "Supertype" groups entities with shared attributes and relationships, while "Subtypes" have their own unique attributes and relationships. It's a way to organize data efficiently. It is a **Top-Down approach.**

We have **is-a** relationship between superclass and subclass.

# ER MODEL IN DBMS

## Specialization



Sub class have  
distinct attributes  
of super class has  
common attributes.

# ER MODEL IN DBMS

## Extended ER features

### Generalization

Generalization is like finding things that are alike and putting them into a big group to represent what they have in common. It helps make things simpler and organized.

It is a **Bottom-Up approach.**

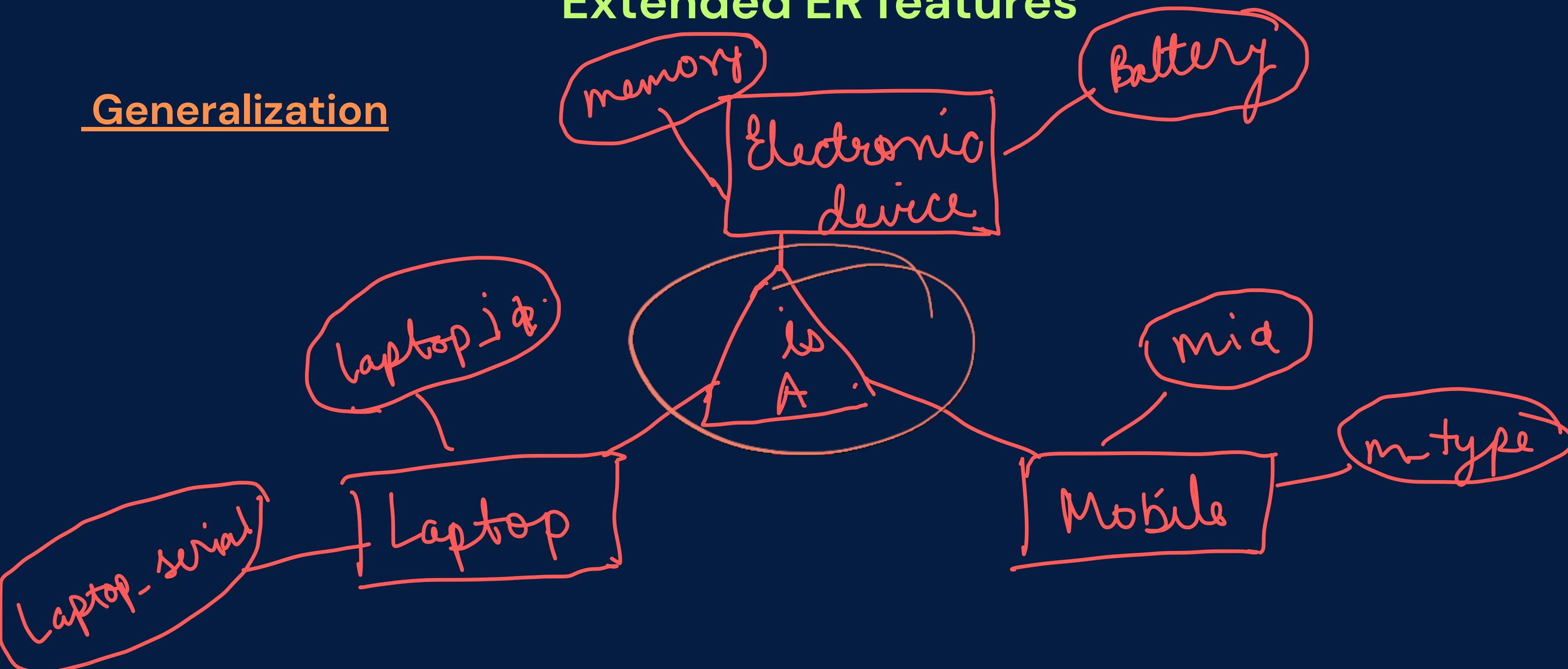
*Bottom Up Approach.*

We have **is-a** relationship between subclass and superclass.

# ER MODEL IN DBMS

## Generalization

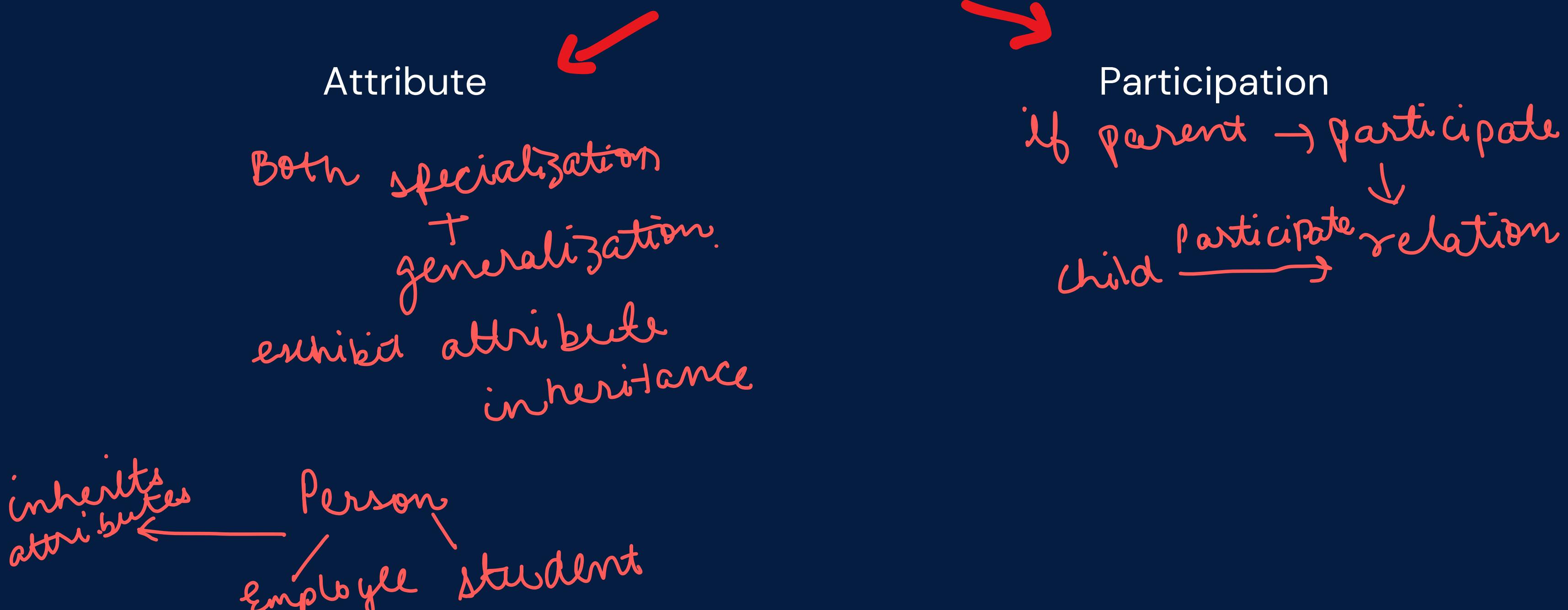
### Extended ER features



# ER MODEL IN DBMS

## Extended ER features

### Inheritance



# ER MODEL IN DBMS

## Extended ER features

### Aggregation

Aggregation is like stacking things on top of each other to create a structure.

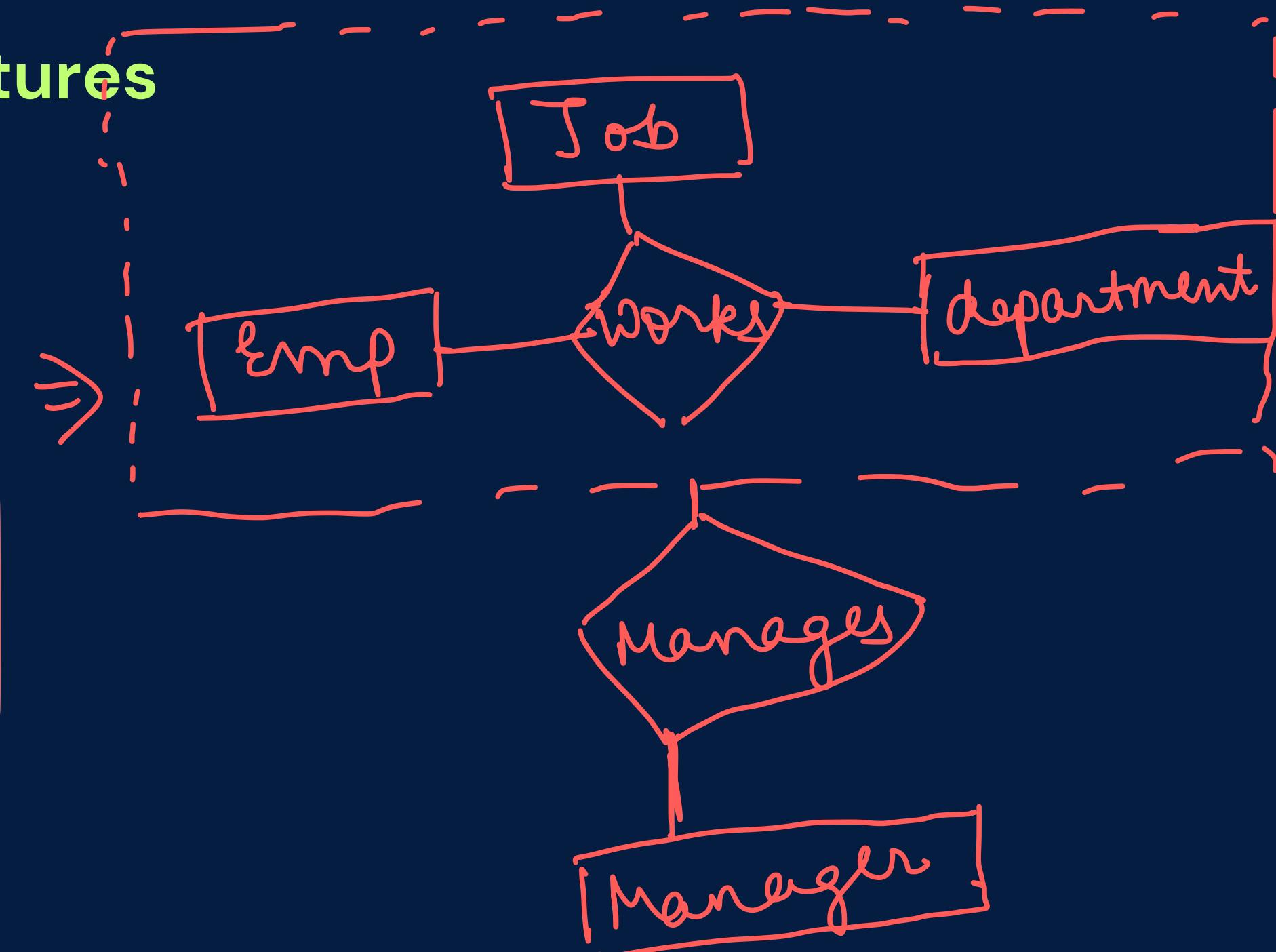
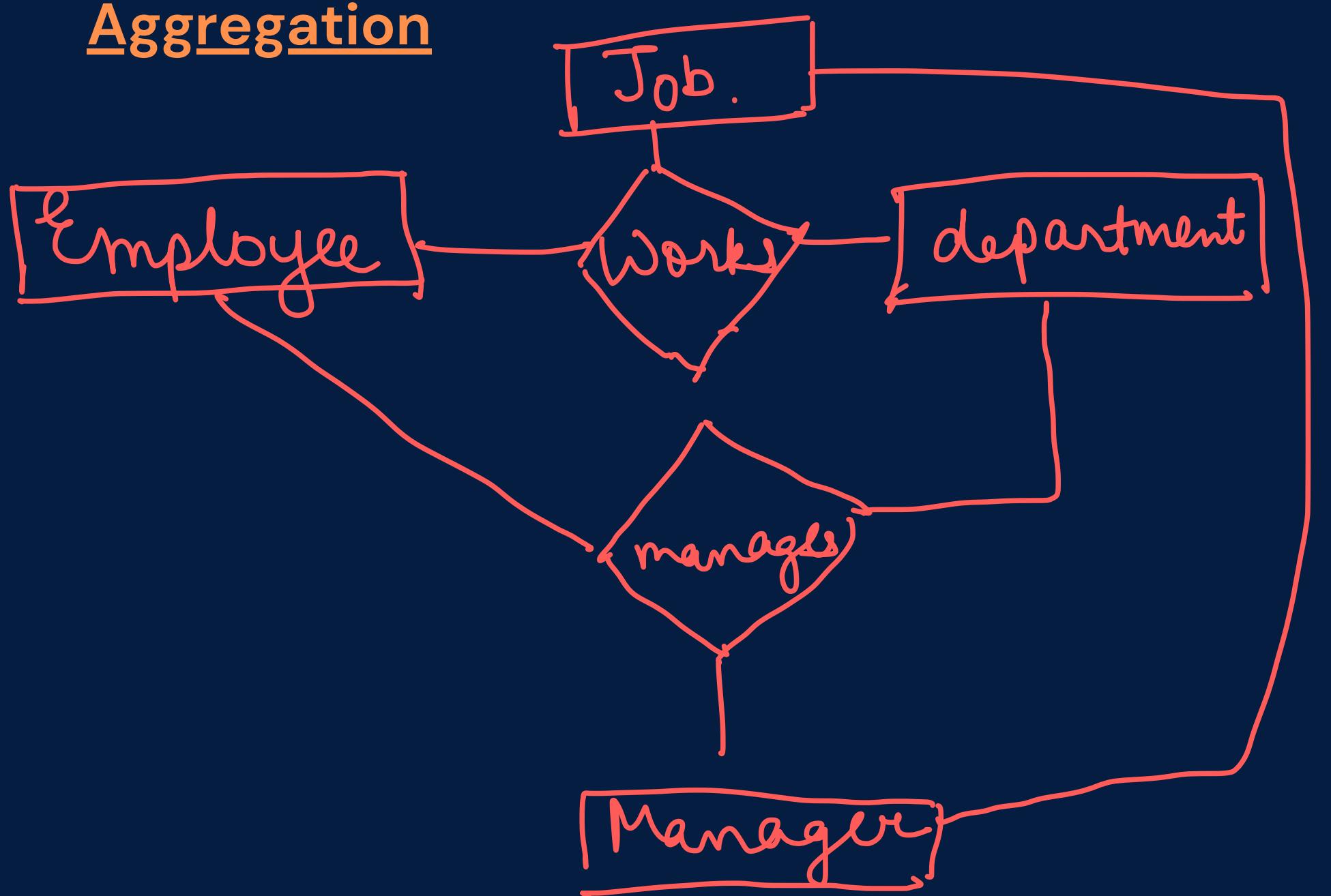
It is used to create a hierarchical structure in data modeling, showing how a higher-level entity is composed of lower-level entities.

**Abstraction** is employed to view relationships from a more general perspective, focusing on a higher-level entity.

# ER MODEL IN DBMS

Extended ER features

Aggregation



# ER MODEL IN DBMS

## ER Model of Instagram

Lets start with what is instagram?

Instagram is a social media platform that allows users to share photos and videos.

# ER MODEL IN DBMS

## ER Model of Instagram

Now what all things we can do on instagram?

- Create our profile
- Add profile picture and details
- Connect with friends
- Upload a post
- Like and comment on post
- Share stories
- and much more

# ER MODEL IN DBMS

## ER Model of Instagram

Lets start with all the steps needs to draw an ER diagram.

### Step-1: Recognize entities sets

Entities

- userProfile
- userFriends
- userPost
- userLogin



UserLikes

# ER MODEL IN DBMS

## ER Model of Instagram

### Step-2 : Specify entity characteristics/attributes

#### Attributes

##### 1. userProfile (user ID, username, email, profile pic)

user ID- primary key ✓

user Name- composite attribute

email - single valued attribute

profile pic - single valued attribute

dob- stored attribute → stored in DB

age- derived attribute → Derived.



# ER MODEL IN DBMS

## ER Model of Instagram

### Step-2 : Specify entity characteristics/attributes

#### Attributes

2. userFriends (followerID, followerName, userID)

followerID- primary key ✓

followerName - single valued attribute

userID - single valued attribute

↳ FK

# ER MODEL IN DBMS

## ER Model of Instagram

### Step-2 : Specify entity characteristics/attributes

#### Attributes

3. userPost (post ID, caption, image, video, likesCount, timestamp)

post ID- primary key ✓

caption - single valued attribute

image - multi valued attribute

video - multi valued attribute

likesCount - single valued attribute

timestamp - single valued attribute

# ER MODEL IN DBMS

## ER Model of Instagram

### Step-2 : Specify entity characteristics/attributes

#### Attributes

4. userLogin (login ID,loginUserName,loginPassword)

login ID- primary key ✓

loginUserName - single valued attribute

loginPassword - multi valued attribute

# ER MODEL IN DBMS

## ER Model of Instagram

### Step-2 : Specify entity characteristics/attributes

#### Attributes

4. userLikes (postID, userID)

postID- primary key ✓

userID - single valued attribute

LFK

# ER MODEL IN DBMS

## ER Model of Instagram

**Step-3: Discover connections/relationships(also constraints like mapping/participation)**

1. userProfile have userFriends (n:n) *n : n*

2. userProfile have userPost (1:n) userPost will always be associated to a userProfile  
therefore total participation

3. userProfile has userLogin (1:1)

4. userProfile has userLikes (1:n) userLikes will always be associated to a userProfile  
therefore total participation

# ER MODEL IN DBMS

## ER Model of Instagram

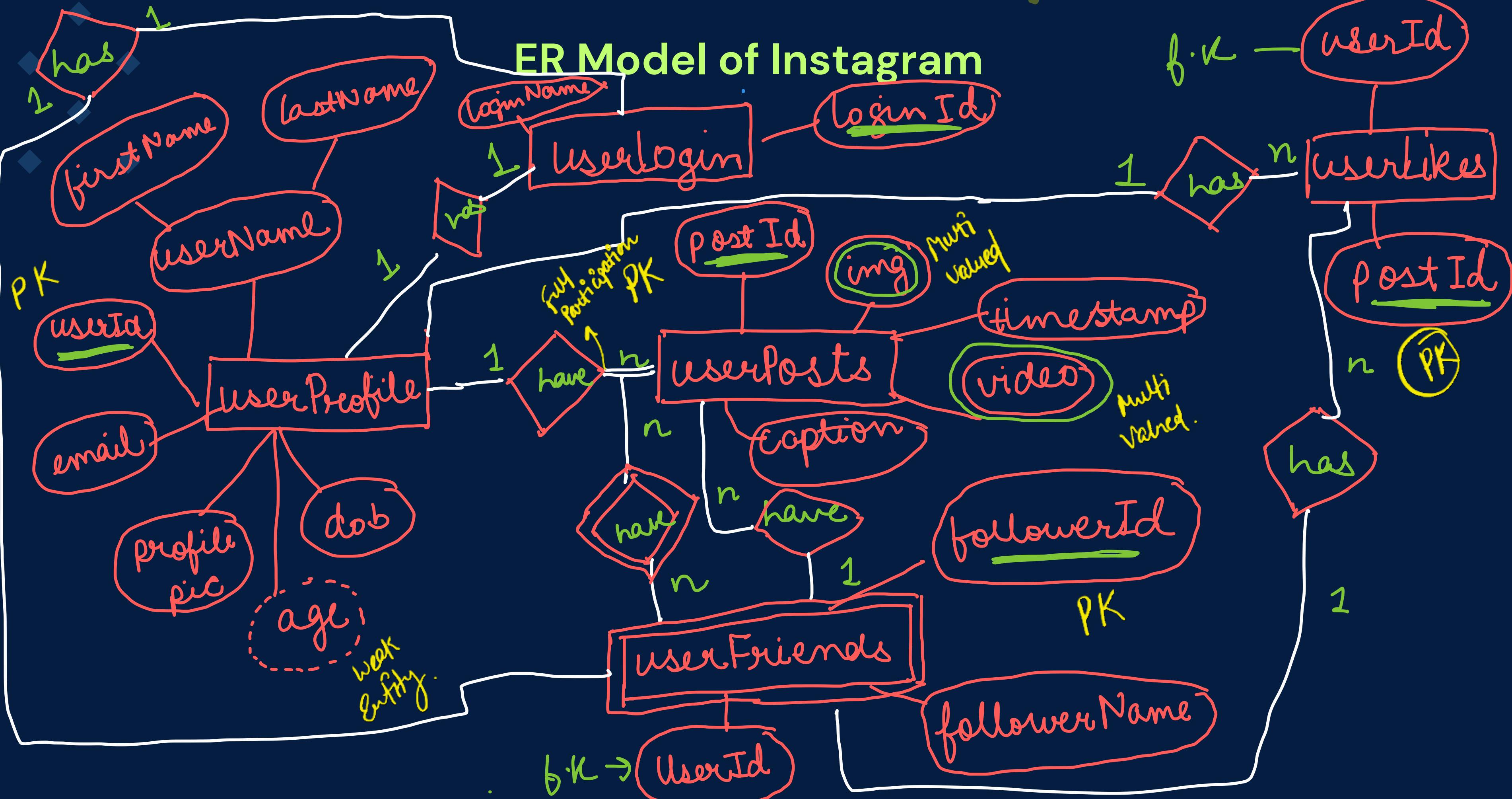
**Step-3: Discover connections/relationships(also constraints like mapping/participation)**

5. userFriends have userPost (1:n) userPost will always be associated to a userProfile  
therefore total participation

6. userFriends has userLogin (1:1)

7. userFriends has userLikes (1:n) userLikes will always be associated to a userProfile  
therefore total participation

# ER Model of Instagram



# RELATIONAL MODEL

It is a way of organizing data in tables.

Some terms used in relational model

1. **Table** – Relation
2. **Row** – Tuple
3. **Column** – Attribute
4. **Record** – Each row in a table
5. **Domain** – The type of value an attribute can hold
6. **Degree** – No. of columns in a relation
7. **Cardinality** – No of tuples

type of  
value

# RELATIONAL MODEL

Relational model is all about:

- Data being organized into tables
- Establishing Relationships between tables using Foreign key
- Maintaining data Integrity
- A flexible and efficient way to store (SQL) and retrieve data

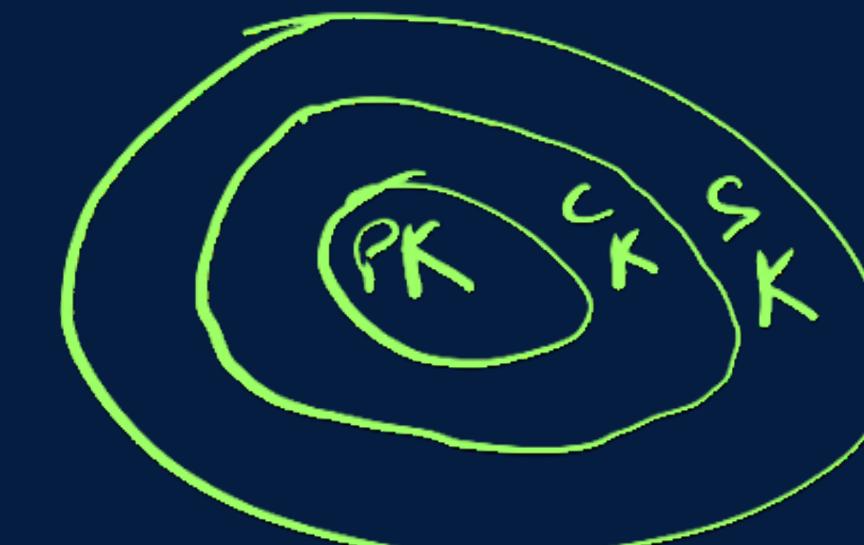
→ Consistency

Relational  
Data

# RELATIONAL MODEL

In relational model we take care of different things like:

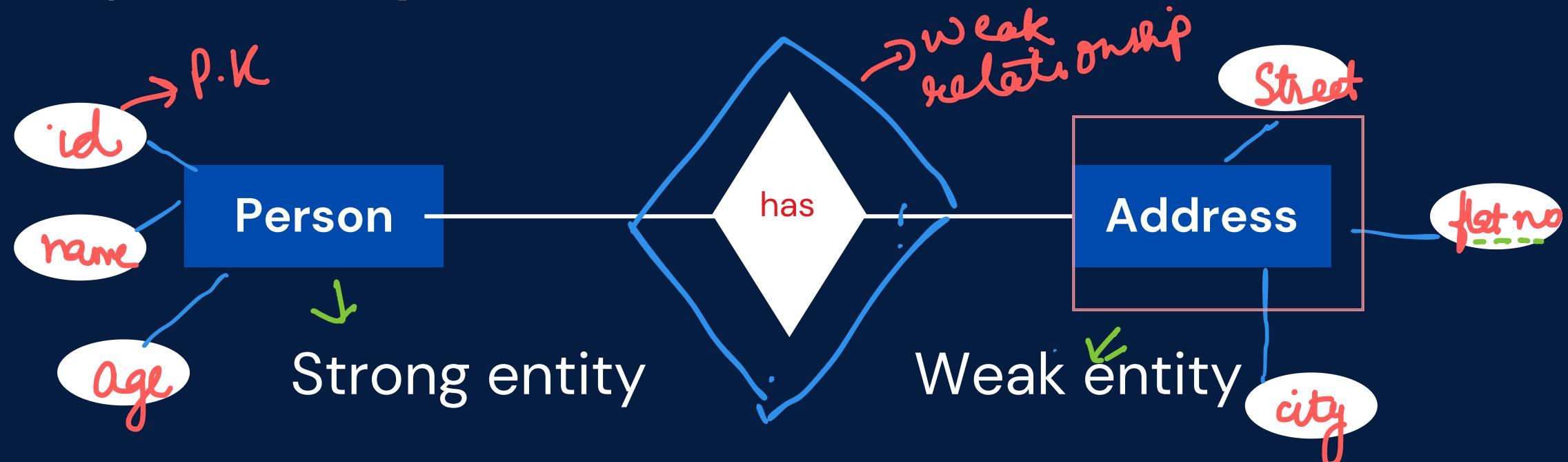
1. Maintaining integrity constraints like domain, entity, referential integrity.
2. The values to be atomic i.e can't be divided further.
3. Each row must be unique, here keys comes into picture i.e candidate, super, primary etc



# CONVERT AN ER MODEL TO RELATIONAL MODEL

Converting an Entity-Relationship (ER) model to a relational model involves several steps:

Step 1: **Identify the entities** – List down all the entities like strong and weak.



1. Person (id, name , age) -> id (p.k)
2. Address (id , flatno, street, city)->id+flatno (p.k) , id (f.k)

# CONVERT AN ER MODEL TO RELATIONAL MODEL

Step 2: **Identify the attributes** - For each entity, identify its attributes which becomes a column in the table.

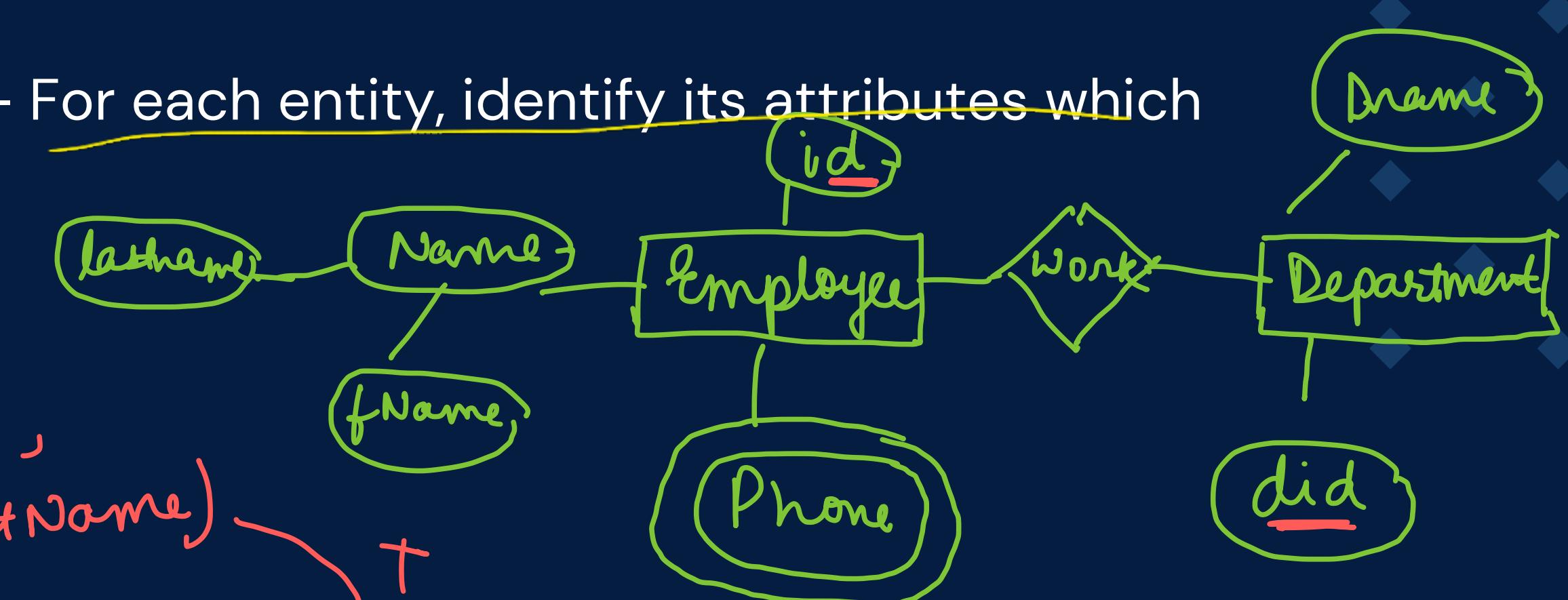
Multivalued attribute

1.  $\text{Employee} \rightarrow (\text{id(P.K.)}, \text{Efname, ElastName})$

Composite attribute

1.  $\text{Employee} \rightarrow (\text{id(P.K.)}, \text{PhoneNb})$

new + table



# CONVERT AN ER MODEL TO RELATIONAL MODEL

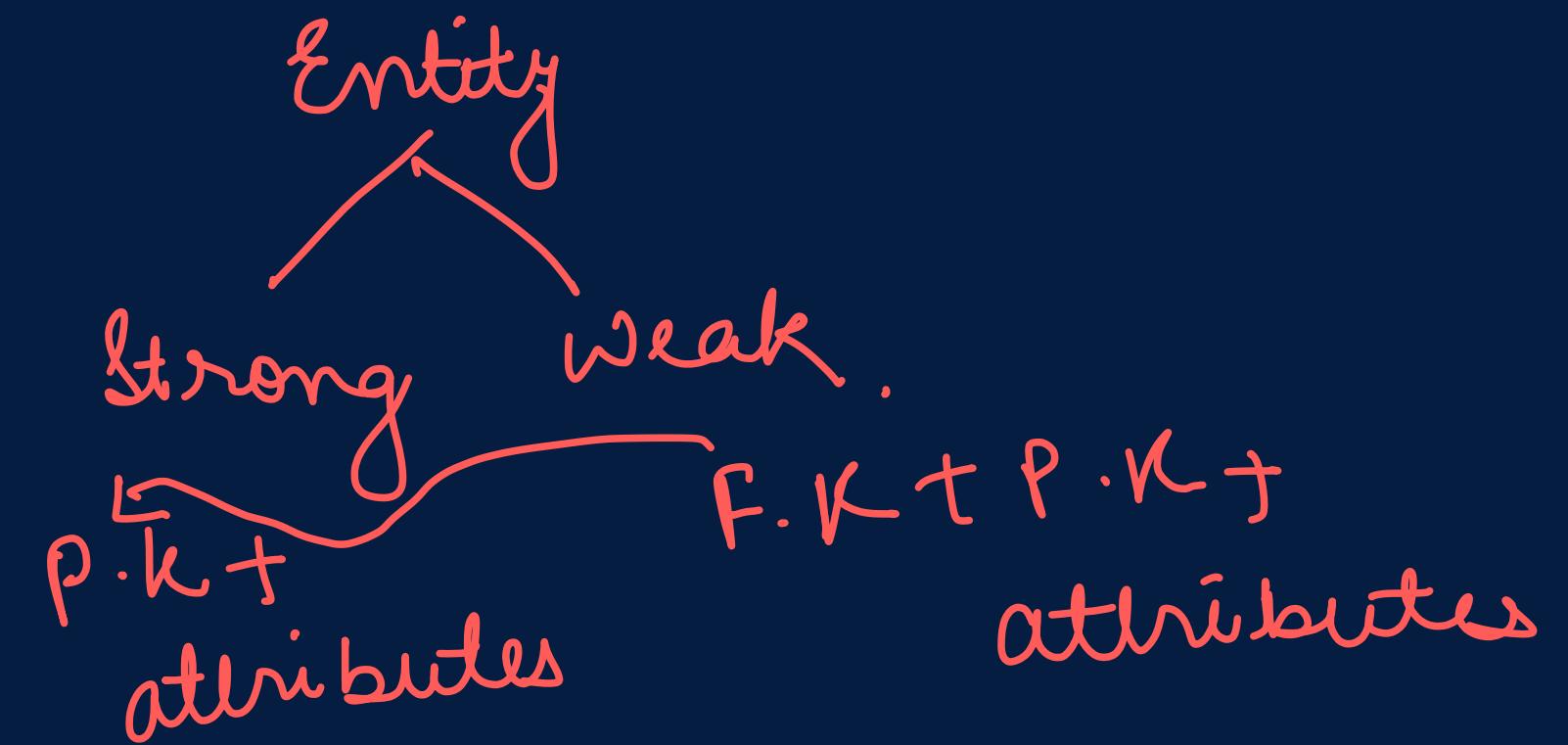
Step 3: **Key selection** - Choose the primary key for each table, for some it can be in form of composite key (Weak entity)

Step 4: **If entities have relationship break it down and the reduce the tables if possible.**

- 
- 1. 1-1 Relationship : 2 tables , P.K can lie on any side
  - 2. 1-Many Relationship : 2 tables , P.K can lie on many side
  - 3. Many -1 relationship : 2 tables , P.K can lie on many side
  - 4. Many-Many relationship : 3 tables , P.K lie in the relation table having pk from both the table acting as fk

# CONVERT AN ER MODEL TO RELATIONAL MODEL

Step 3: Key selection - Choose the primary key for each table, for some it can be in form of composite key (Weak entity)



# LET'S START WITH SQL :)

## RDBMS

**What is RDBMS ?**

**Database** : Collection of data is called as database.

**DBMS**: A software application to manage our data.



# LET'S START WITH SQL :)

RDBMS

What is RDBMS ?

Relational(Use tables to store data)

Relational  
Basically refers  
to Tables

Non-relational(Data is not stored in tables)

MySQL

Oracle

MariaDB,

} RDBMS

MongoDb

} NDBMS

NOSQL

Database

# LET'S START WITH SQL :)

## RDBMS

### What is RDBMS ?

These databases structure data into organized tables that have predefined connections between them.

Data manipulation and querying are performed using SQL (Structured Query Language).



# LET'S START WITH DBMS :).

## Normalisation and its types

### Normalisation

Normalization is a process in which we organize data to reduce redundancy(duplicacy) and improve data consistency. It involves dividing a database into two or more tables

Reducing data Redundancy.

### What is **data redundancy** and **consistency** and why its important

When there is same set of data repeated each and every time it results in duplicacy of data (either in row or column)

Data Redundancy

Now row level duplicacy can be remove by using primary key for unique values.

(let's say  
we have 2 values in  
a table where data is  
repeating we can use Normalization  
to create separate tables  
for them)

# LET'S START WITH DBMS :).

## Normalisation and its types

Now when we have same data for some set of columns , it leads to different anomalies (inconsistencies or errors that occur when manipulating or querying data in a database)

- 1.Insertion Anomaly
- 2.Updation Anomaly
- 3.Deletion Anomaly

Duplicacy in data  
Cause Anomalies

size ↑

Also it also increases the size of database with the same data.

Efficiency ↓

# LET'S START WITH DBMS :).

## Normalisation and its types

### Insertion Anomaly:

It occurs when it is difficult to insert data into the database due to the absence of other required data.

Consider you want to add a new department but there is no employee in that dept yet.

Here let's say A dept. "A" doesn't exist at that time Insertion Anomaly occurs.

Employee					
id	name	age	department	Manager	salary
1	Rahul	25	'IT'	Raj	1500
2	Afsara	26	'HR'	Avinash	1000
3	Abhimanyu	27	'IT'	Raj	1500
4	Aditya	25	'HR'	Avinash	1000
5	Raj	24	'HR'	Avinash	1000

# LET'S START WITH DBMS :).

## Normalisation and its types

### Deletion Anomaly:

It occurs when deleting data removes other valuable data.

Consider if you delete all the record in the table, you will loose the track of dept, their manager and salaries.

Employee					
id	name	age	department	Manager	salary
1	Rahul	25	'IT'	Raj	1500
2	Afsara	26	'HR'	Avinash	1000
3	Abhimanyu	27	'IT'	Raj	1500
4	Aditya	25	'HR'	Avinash	1000
5	Raj	24	'HR'	Avinash	1000

Here if we want to del Employee info but if we remove their data we also remove other data which is valuable like salary, manager.

# LET'S START WITH DBMS :).

## Normalisation and its types

### Updation Anomaly:

It occurs when changes to data require multiple updates

Consider you want to change the salary for people working in HR department, you need to update it at 3 place .

### Employee

id	name	age	department	Manager	salary
1	Rahul	25	'IT'	Raj	1500
2	Afsara	26	'HR'	Avinash	1000
3	Abhimanyu	27	'IT'	Raj	1500
4	Aditya	25	'HR'	Avinash	1000
5	Raj	24	'HR'	Avinash	1000

# LET'S START WITH DBMS :).

## Normalisation and its types

How normalisation helps here?

Divide into  
separate  
tables.

Using normalisation we can divide the employee table in two tables

- 1.Employee
- 2.Department

Employee			
id	name	age	department
1	Rahul	25	IT
2	Afsara	26	HR
3	Abhimanyu	27	IT
4	Aditya	25	HR
5	Raj	24	HR

Department		
department	Manager	salary
IT	Raj	1500
HR	Avinash	1000

Data  
Redundancy ↓  
Not removed  
entire but  
reduces it.

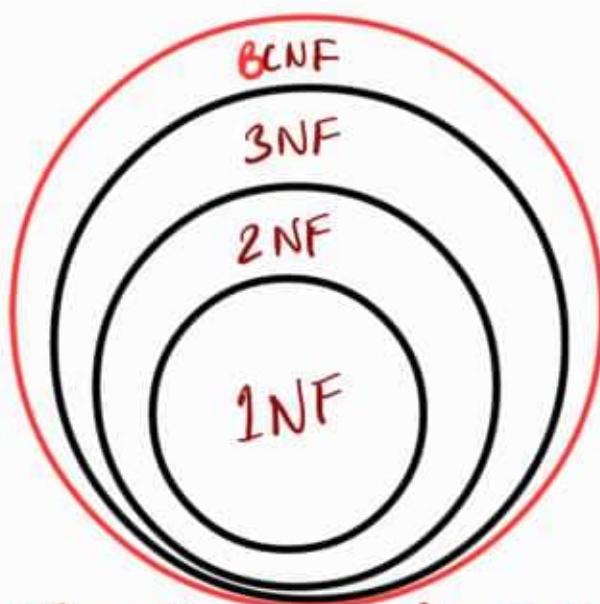
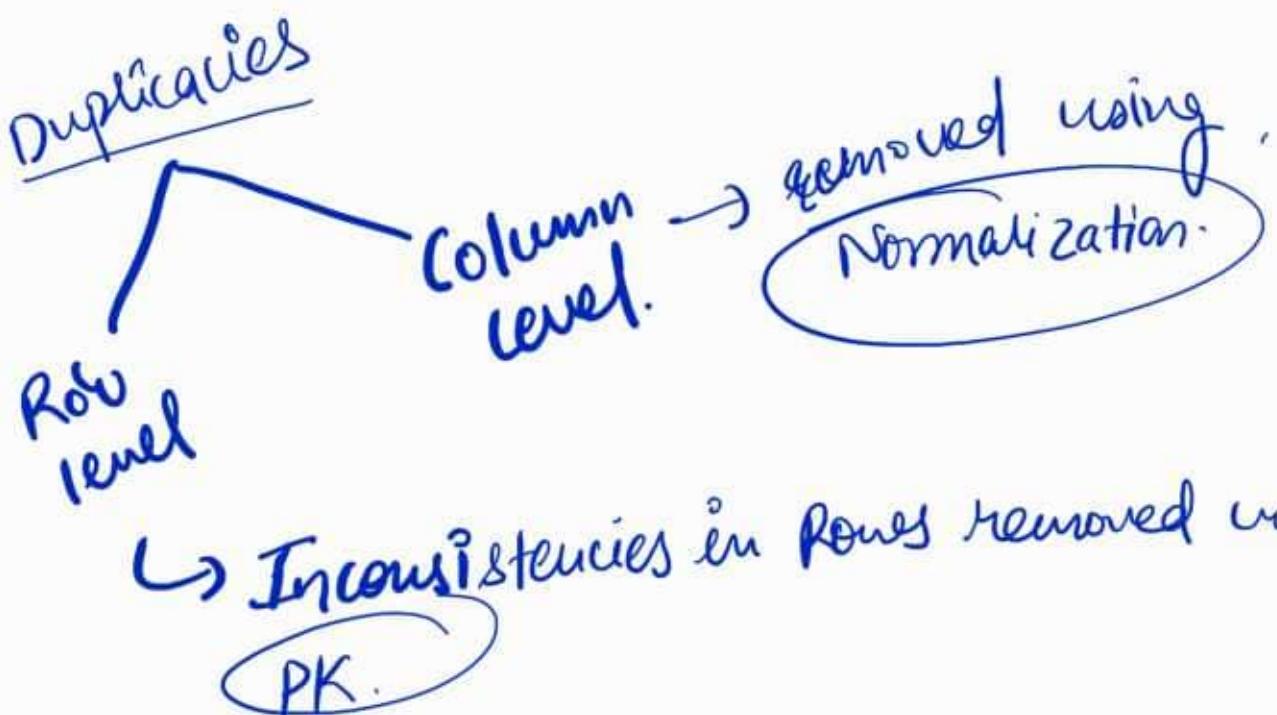
# LET'S START WITH DBMS :).

## Normalisation and its types

### Types of Normalisation

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form(BCNF)

Different  
Normalizatinal  
types.



A Relation will always be in atleast 1NF

## 1NF

- ⇒ Removes repeating groups from the table
- ⇒ Create a separate table for each set of related data.
- Identify each set of related data with a PK.

## 2NF

- ⇒ Should be in 1<sup>st</sup> NF
- ⇒ Should not contain Partial Dependency.

## 3NF

- ⇒ Should be in 2NF
- ⇒ No Transitive dependency for non-prime attributes.

## 3(CNF)

- ⇒ Should be in 3NF
- ⇒ Every FD  $A \rightarrow B$ , A should be a k of that table.

## Tips & Tricks

2NF (when is it not)

$\Rightarrow x \rightarrow y$  is a FD,  $x$  is a subset of CK  
 $y$  is a NP Attribute.

3NF (when is it)

$x \rightarrow y$  is a FD,  $x$  is SK or CK or  $y$  is  
a prime attribute.

$\Rightarrow$  if All Attributes are Prime  $\rightarrow R$  is in 3NF

BCNF (when is it)

$x \rightarrow y$ ,  $x$  is SK or CK

# LET'S START WITH DBMS :).

## Denormalization

This is the opposite of normalization. It involves intentionally introducing some redundancy into a well-normalized database schema to improve query performance.

Consider if you wish to find the salary of Rahul so first you have to make a query in employee table to find department of Rahul and then in department table to find the salary of Rahul

Employee			
id	name	age	department
1	Rahul	25	IT
2	Afsara	26	HR
3	Abhimanyu	27	IT
4	Aditya	25	HR
5	Raj	24	HR

Department		
department	Manager	salary
IT	Raj	1500
HR	Avinash	1000

Employee					
id	name	age	department	Manager	salary
1	Rahul	25	'IT'	Raj	1500
2	Afsara	26	'HR'	Avinash	1000
3	Abhimanyu	27	'IT'	Raj	1500
4	Aditya	25	'HR'	Avinash	1000
5	Raj	24	'HR'	Avinash	1000

Normalization  
K9 Ultra  $\Rightarrow$

Here it increases query performance.  
But introduces redundancy.

# LET'S START WITH DBMS :).

## Denormalization

### Benefits

- Faster Queries: It can reduce the need for complex joins between tables during queries which can eventually improve the speed of retrieving frequently accessed data.
- Simpler Queries: It can simplify queries by allowing them to be executed on a single table instead of requiring joins across multiple tables.

### Disadvantages

- Increased Data Redundancy
- Less Data Consistency
- Denormalization can make the database schema less flexible for future changes. like adding/modifying new data elements

# LET'S START WITH DBMS :).

## Functional Dependency

Functional dependency describes the relationship between attributes in a relation.

A FD is a constraint between two sets of attributes in a relation from a database

For a Relation(table) R, if there are two attributes X and Y then

**FD : X(determinant)  $\rightarrow$  Y(dependent)**

Attribute Y is functionally dependent on attribute X.

It means how  
1 attribute is dependent  
on other.

dependent ←  
X determines  
↳ determinant

R	
X	Y

# LET'S START WITH DBMS :).

## Functional Dependency

If  $x=1$ , we can find the value of  $y$ .

F.D :  $X \rightarrow Y$  ( $X, Y$  is a subset of  $R$ )

What is subset?

EmplD <i>PK</i>	EmpFirstName	EmpLastName
1	Riti	Kumari
2	Rahul	Kumar
3	Suraj	Singh

Let  $A = \{1, 2, 3\}$

Let  $B = \{1, 2, 3, 4, 5\}$

$A$  is a subset of  $B$  because every element of  $A$  is also an element of  $B$ .

let's say we have 2 Tuples  $t_1$  &  $t_2$  and  $t_1 \cdot x = t_2 \cdot x$  it doesn't mean they are functionally dependent till the point  $t_1 \cdot y \neq t_2 \cdot y$  & all of their attributes.

FD:  $\text{EmplD} \rightarrow \text{EmpFirstName}$  (EmpFirstName is functionally dependent on EmplD)

$\text{EmplD} \rightarrow \text{EmpLastName}$

# LET'S START WITH DBMS :).

## Functional Dependency

Properties of Functional Dependencies:

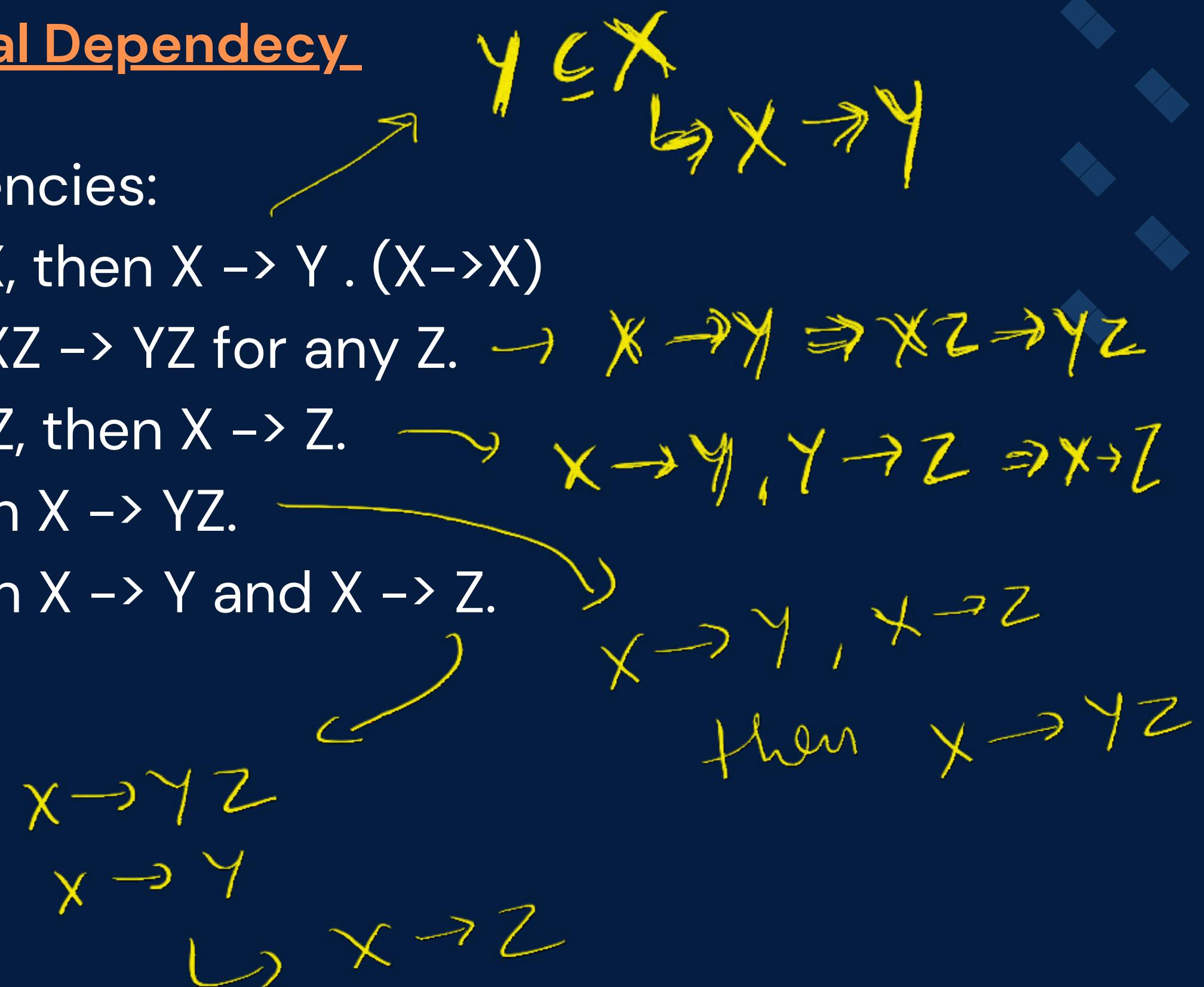
1. Reflexivity: If  $Y$  is a subset of  $X$ , then  $X \rightarrow Y$ . ( $X \rightarrow X$ )
2. Augmentation: If  $X \rightarrow Y$ , then  $XZ \rightarrow YZ$  for any  $Z$ .  $\rightarrow X \rightarrow Y \Rightarrow XZ \rightarrow YZ$
3. Transitivity: If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$ .  $\rightarrow X \rightarrow Y, Y \rightarrow Z \Rightarrow X \rightarrow Z$
4. Union: If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$ .
5. Decomposition: If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$ .

Here the decomposition can only happen

on the dependant side not on determinant

side.  $X \rightarrow YZ$  &  $X \rightarrow Y \Rightarrow X \rightarrow Z$

But  $XY \rightarrow Z$  &  $X \rightarrow Z \not\Rightarrow Y \rightarrow Z$



# LET'S START WITH DBMS :).

## Functional Dependency

Types of Functional Dependency

- 1.Trivial dependency
- 2.Non-trivial dependency

EmpID	EmpFirstName	EmpLastName
1	Riti	Kumari
2	Rahul	Kumar
3	Suraj	Singh

# LET'S START WITH DBMS :).

## Functional Dependency

### Trivial dependency

A functional dependency  $X \rightarrow Y$  is trivial if  $Y$  is a subset of  $X$

We can also say it as  $X \rightarrow X$ .

$\{\text{EmplID}, \text{EmpFirstName}\} \rightarrow \{\text{EmplID}\}$

is trivial because  $\{\text{EmplID}\}$  is a subset of  $\{\text{EmplID}, \text{EmpFirstName}\}$ .

$$X \cap Y = Y$$

$X \cap Y = Y$

Let's say  $X \rightarrow Y$  here  
it is trivial bcoz  $X \subseteq Y$

Imp.

$$Y \subseteq X$$

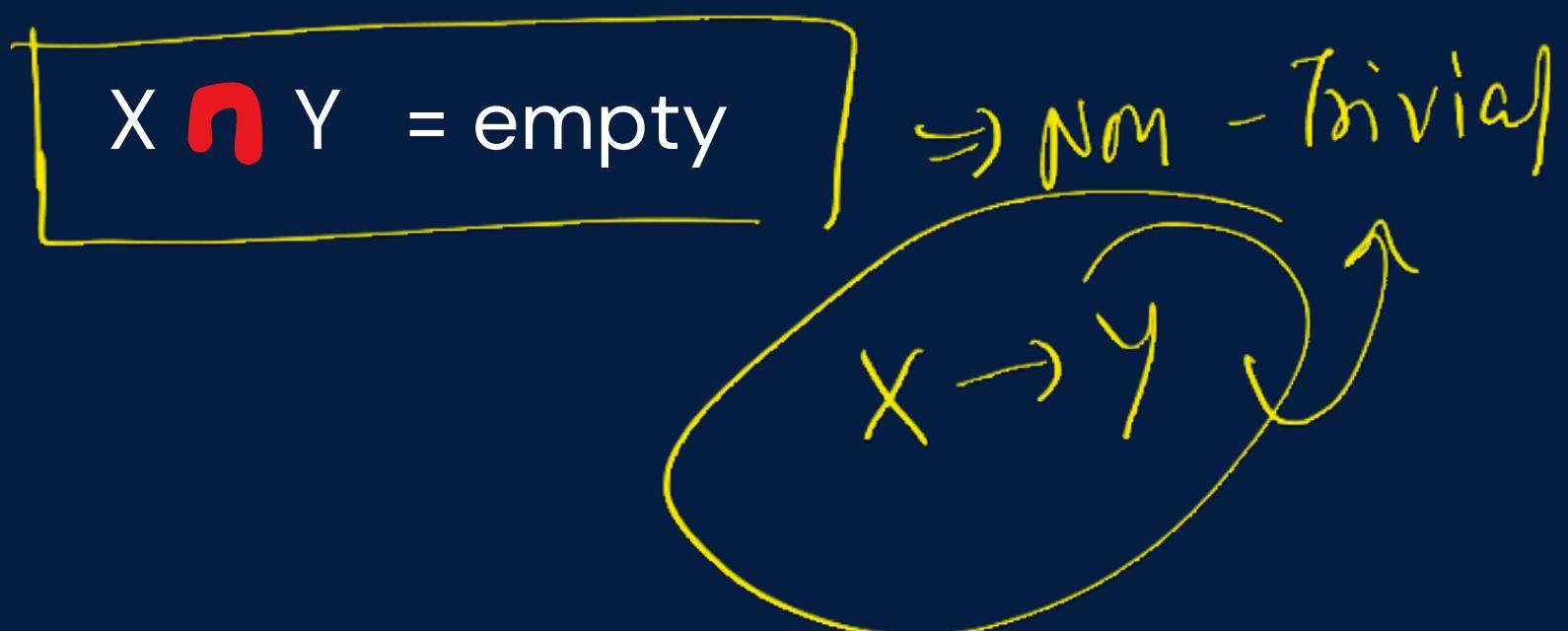
EmplID	EmpFirstName	EmpLastNmae
1	Riti	Kumari
2	Rahul	Kumar
3	Suraj	Singh

# LET'S START WITH DBMS :).

## Functional Dependency

### Non-Trivial dependency

A functional dependency  $X \rightarrow Y$  is non-trivial if  $Y$  is not a subset of  $X$  i.e  $X \neq Y$ .  
 $\{EmpID\} \rightarrow \{EmpFirstName\}$  is trivial because  $\{EmpFirstName\}$  is not a subset of  $\{EmpID\}$ .



EmplD	EmpFirstName	EmpLastNmae
1	Riti	Kumari
2	Rahul	Kumar
3	Suraj	Singh

# LET'S START WITH DBMS :).

↑  
It provides us a set of all attributes  
X can determine  
if  $X \rightarrow A, X \rightarrow B$  then  
 $X^+ = \{A, B\}$

Attribute closure helps us for identifying candidate keys, checking for functional dependencies, and in normalisation.

$X^+$  where  $x$  is an attribute or set of attribute which have all the attributes in a relation which can determine  $X$ .

Ques : Consider we have a relation R with attributes A,B,C,D,E and FD are  
 $A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$

1. Now according to the rule of Reflexivity all the attributes can determine theirself.

$A \rightarrow A, B \rightarrow B, C \rightarrow C, D \rightarrow D, E \rightarrow E$

# LET'S START WITH DBMS :).

## Attribute closure/closure set

Ques : Consider we have a relation R with attributes A,B,C,D,E and FD are

$A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$

1. Now according to the rule of Reflexivity all the attributes can determine theirself.

$A \rightarrow A, B \rightarrow B, C \rightarrow C, D \rightarrow D, E \rightarrow E$

2. Now according to the rule of transitivity if A determines B and B determines C, then A can also determine C and same for all other attributes.

$A \rightarrow C, A \rightarrow D, A \rightarrow E$

$B \rightarrow D, B \rightarrow E$

$C \rightarrow E$

# LET'S START WITH DBMS :).

## Attribute closure

Ques : Consider we have a relation R with attributes A,B,C,D,E and FD are

$A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$

3. Now according to the rule of UNION if as the determinant is same we can combine dependent

For A

$A \rightarrow B, A \rightarrow C, A \rightarrow D, A \rightarrow E, A \rightarrow A$

$A \rightarrow ABCDE$

For C

$C \rightarrow D, C \rightarrow E, C \rightarrow C$

$C \rightarrow DEC$

For E

$E \rightarrow E$

For B

$B \rightarrow C, B \rightarrow D, B \rightarrow E, B \rightarrow B$

$B \rightarrow CDEB$

For D

$D \rightarrow E, D \rightarrow D$

$D \rightarrow ED$

# LET'S START WITH DBMS :).

## Attribute closure

Ques : Consider we have a relation R with attributes A,B,C,D,E and FD are

$A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow E$

4. Now lets find the closure set of attributes

A- {A,B,C,D,E}

B- {B,C,D,E}

C-{C,D,E}

D- {D,E}

E- {E}

AB - {A,B,C,D,E}

## Closure

A, B, C, D, E

$$\begin{array}{l} A \rightarrow B \\ B \rightarrow C \\ C \rightarrow D \\ D \rightarrow E \end{array}$$

$$\begin{array}{l} A \rightarrow A \\ B \rightarrow B \\ C \rightarrow C \\ D \rightarrow D \\ E \rightarrow E \end{array}$$

we need to  
find closures

$A^+, B^+, C^+, D^+, E^+$

$$A^+ = \{A, B, C, D, E\}$$

$$\because B \rightarrow C \Rightarrow A \rightarrow C$$

$$A \rightarrow B$$

$$\therefore C \rightarrow D \Rightarrow A \rightarrow D$$

$$A \rightarrow C$$

$$\therefore D \rightarrow E \Rightarrow A \rightarrow E$$

$$A \rightarrow D$$

for B

$$B \rightarrow C$$

$$B \rightarrow B$$

$$B^+ = \{B, C, D, E\}$$

$$B \rightarrow D$$

$$C \rightarrow D \Rightarrow$$

$$B \rightarrow C$$

$$E$$

Similarly for E

$$A^+ = \{A, B, C, D, E\}$$

$$B^+ = \{B, C, D, E\}$$

$$C^+ = \{C, D, E\}$$

$$D^+ = \{D, E\}$$

$$E^+ = \{E\}$$

Closure  
for  
 $A, B, C, D, E$

# LET'S START WITH DBMS :).

## Attribute closure

5. Now lets find candidate key, super key , prime and non-prime attributes

**Super key** : Set of attributes whose closure contains all the attributes given in a relation  
Super set of any candidate key is super key. A key(combination of all possible attributes) which can uniquely identify two tuples.

How to find Super Key?

- Identify All Attributes
- 2. Analyze the functional dependencies and find closure.
- Generate Power Set : If A has n attributes, the power set will have  $2^n$  subsets.
- Check for Super Key Property : For each subset in the power set, check if it can uniquely identify each tuple in the relation.

$A \rightarrow n$  attributes  
Power set  $\rightarrow 2^n$  subsets

# LET'S START WITH DBMS :).

## Attribute closure

Q. Find all the superkeys for the relation R(A, B, C) and FD : A → B, B → C.

### 1. Identify All Attributes:

$$A = \{A, B, C\}$$

A determines ABC  
∴ A is sk

### 2. Analyze the functional dependencies and find closure.

From A → B and B → C, we can infer A → C

$$A^+ = \{A, B, C\}$$

# LET'S START WITH DBMS :).

## Attribute closure

$B^+ = \{B, C\}$  because  $B \rightarrow C$ , but  $B$  does not determine  $A$ .

$C^+ = \{C\}$

Closure of  $A$  gives or determines all the attributes in the table so we can say its super key.

### 3. How to find all the super keys?

Find the power subset

The power set of  $\{A, B, C\}$  is  $2^{A, B, C} = 8$

$\{\}, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}$

All super keys

# LET'S START WITH DBMS :).

## Attribute closure

A power set is the set of all subsets of a given set, including the empty set and the set itself. If you have a set X, the power set of X is denoted as  $2^X$

Example:

Let's take a simple set  $X=\{a,b\}$

The power set of X would include the following subsets:

1. The empty set:
2. The single-element subsets:  $\{a\}$  &  $\{b\}$
3. The full set itself:  $\{a,b\}$

So, the power set  $P(X)$  would be:  $P(X)=\{\ ,\{a\},\{b\},\{a,b\}\}$

# LET'S START WITH DBMS :).

## Attribute closure

### 4. Verify each subset to see if it is a super key

1. {}: Not a super key.
2. {A}: Super key (as its closure determines all the attributes in relation).
3. {B}: Not a super key (does not determine A).
4. {C}: Not a super key (does not determine A or B).
5. {A, B}: Super key (A is already a super key, so adding B still keeps it a super key).
6. {A, C}: Super key (A is already a super key, so adding C still keeps it a super key).
7. {B, C}: Not a super key (B determines C, but does not determine A).
8. {A, B, C}: Super key (A is already a super key, so adding B and C keeps it a super key).

# LET'S START WITH DBMS :).

## Attribute closure

5. Super keys are : {A}, {A, B}, {A, C}, {A, B, C}

We can also say to find max number of super keys we can use the formula

where

$$2^{n-k}$$

k- candidate key with k attributes ( $k < n$ ) in a relation

n- total no of attributes in a relation

When 1 C.K is there =  $2^{n-1}$

When 2 C.K is there =  $2^{n-1}$ (for 1st ck)+ $2^{n-1}$ (for 2nd ck)- $2^{n-2}$ (for combination of both)

Here  $A \rightarrow B, A \rightarrow C$   
 we have power set =  $\{\emptyset, \{A\}, \{B\}, \{C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A, B, C\}\}$

Here  $SK = \{\emptyset, \{A\}, \{B\}, \{C\}, \{AB\}, \{AC\}, \{BC\}, \{ABC\}\}$

But we always take minimal for CK  $\Rightarrow \textcircled{A}$

$$\max SK = 2^{n-k} = 2^{n-1} = \textcircled{4} \Rightarrow \text{total } SK$$

# LET'S START WITH DBMS :).

## Attribute closure

5. Now lets find candidate key, super key , prime and non-prime attributes

**Candidate key** : A superkey whose proper subset is not a super key

A set A is considered a proper subset of set B

- All elements of A are also elements of B
- Set B has at least one element that is NOT in A.

$$\begin{cases} A = \{1, 2\} \\ B = \{1, 2, 3\} \end{cases}$$

Let's say we have a set  
 $\text{SK} \Rightarrow AB \Rightarrow$  proper subsets  
 $\{A\}$   
 $\{B\}$   
 $\{\emptyset\}$   
Now if all these have a closure which is not a SK then it is a CK.

**Prime Attribute**: An attribute that is part of any candidate key.

**Non-Prime Attribute**: An attribute that is not part of any candidate key.



# LET'S START WITH DBMS :).

## Attribute closure

How to find the number of candidate keys?

1. Take the closure of the entire attribute set
2. Discard the dependents if their determinants are present
3. After discarding the final combination you get is a candidate key if its subset doesn't have a super key and mention them in prime attribute
4. Now check all the functional dependencies and see if in the right hand side is there a attribute which has a determinant present in the prime attribute, if yes mention that as well in the prime attribute and replace the dependent attribute with determinant and check if its candidate key.
5. If no prime attribute is available in teh right hand side of the functional dependency we can say there are no more candidate key and the one key we made after discarding dependent and checking for super key, that is the only candidate key.

$$R = \{A, B, C\}$$

$$JK = ABC^+$$

$$PA = \{A, B\}$$

$$CK = AB$$

$$\text{Now } FD \Rightarrow A \rightarrow C$$

note C is not  
a PA and it  
is in the CK so it  
is OK. If we  
had any FD  $C \rightarrow B$   
Then we had  
to remove B from CK

# LET'S START WITH DBMS :).

## Attribute closure

Q. Find no of candidate and super key for the given relation R (A,B,C,D) and functional dependency  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow D$  ,  $B \rightarrow A$

- 1.Lets find the minimal superkeys
- 2.We will get the value of k and n ,where k- candidate key with k attributes ( $k < n$ ) in a relation , n- total no of attributes in a relation
3. Use this to get the super keys in the relation  $2^{n-k}$

# LET'S START WITH DBMS :).

Now we know that  
we have ABCD as JK attributes  
find its closure

Now FD  $\Rightarrow$   $A \rightarrow B$   $B \rightarrow C$   $C \rightarrow D$   $A \rightarrow C$   $B \rightarrow D$

$\overbrace{ABCD}$   
 $\overbrace{ABC}$   
 $\overbrace{ACD}$   
 $\overbrace{AD}$   
 $A$

How to find the number of candidate keys?

1. Take the closure of the entire attribute set
2. Discard the dependents if their determinants are present
3. After discarding the final combination you get is a candidate key if its subset doesn't have a super key and mention them in prime attribute
4. Now check all the functional dependencies and see if in the right hand side is there a attribute which has a determinant present in the prime attribute, if yes mention that as well in the prime attribute and replace the dependent attribute with determinant and check if its candidate key.
5. If no prime attribute is available in teh right hand side of the functional dependency we can say there are no more candidate key and the one key we made after discarding dependent and checking for super key, that is the only candidate key.

$(A) \Rightarrow \Phi^+$   $\cup \{\Phi\}^+ \Rightarrow X$   
it is  
not a  
JK.  
power  
set -  
Candidate

Key ↳ Because proper subset  
doesn't give a JK.

# LET'S START WITH DBMS :).

PA = {A, Y}

Now moving on 4th step we check if A is the dependent in any of the FD → It will be replaced.

in PA set and A will be replaced.

PA = {A, B, Y} → CK → {B} → SKX

Now B is also a CK. ↴

Now B is not present in right part anywhere. So stop here Only.

JK =  $2^{n-k}$  ⇒ for A →  $2^{4-1} = 2^3$

⇒ for B →  $2^{4-1} = 2^3$

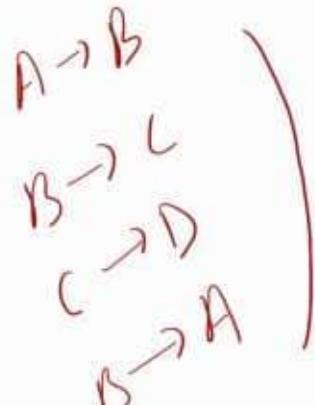
⇒ 16 SK.

But

AB are CK we will take  $2^{4-2} = 4 \text{ SK}$

How to find the number of candidate keys?

1. Take the closure of the entire attribute set
2. Discard the dependents if their determinants are present
3. After discarding the final combination you get is a candidate key if its subset doesn't have a super key and mention them in prime attribute
4. Now check all the functional dependencies and see if in the right hand side is there a attribute which has a determinant present in the prime attribute, if yes mention that as well in the prime attribute and replace the dependent attribute with determinant and check if its candidate key.
5. If no prime attribute is available in teh right hand side of the functional dependency we can say there are no more candidate key and the one key we made after discarding dependent and checking for super key, that is the only candidate key.



$A^+ \rightarrow \{AB\}$   $\rightarrow 8K$   
 $B^+ \rightarrow \{BCD\}$   
 $C^+ \rightarrow \{CD\}$   
 $D^+ \rightarrow \{D\}$

↓  
 Total <sup>now</sup> =  $16 - 4 = 128K$

powerset  
 of  
 $ABCD = 2^4 \Rightarrow 16$

# LET'S START WITH DBMS :).

## Normalisation and its types

### First Normal Form (1NF)

First normal form is the first step in the normalisation process which helps us to reduce data redundancy. Every table should have atomic values i.e there shouldn't be any multivalued attributes

It ensures the following set of rules is followed in a table:

- Atomicity(Attributes should have single value)
- Uniqueness of rows (Each row should be uniquely identifiable)

No multivalued  
Attributes

1NF

# LET'S START WITH DBMS :).

## Normalisation and its types

### First Normal Form (1NF)

- Atomicity: Each column contains only indivisible (atomic) values, meaning each attribute holds a single value.

ID	PersonName	Order
1	Raj	Muffin,Sugar
2	Riti	Muffin
3	Rahul	Sugar,Egg

Here, Order is a multivalued attribute(having more than one value)

# LET'S START WITH DBMS :).

## Normalisation and its types

### First Normal Form (1NF)

How to achieve atomicity?

1. Repeat the values in id and PersonName column twice to store single value of multivalued attribute order

ID	PersonName	Order
1	Raj	Muffin
1	Raj	Sugar
2	Riti	Muffin
3	Rahul	Sugar
3	Rahul	Egg

PK - Order+ ID

→ Increasing  
Data Redundancy.

ID	PersonName	Order
1	Raj	Muffin,Sugar
2	Riti	Muffin
3	Rahul	Sugar,Egg

# LET'S START WITH DBMS :).

## Normalisation and its types

### First Normal Form (1NF)

How to achieve atomicity?

2. Make new columns for each multivalue present.

ID	PersonName	Order1	Order2
1	Raj	Muffin	Sugar
2	Riti	Muffin	Null
3	Rahul	Sugar	Egg

PK - ID

ID	PersonName	Order
1	Raj	Muffin,Sugar
2	Riti	Muffin
3	Rahul	Sugar,Egg

→ New Columns  
Space ↑  
Efficiency ↕

# LET'S START WITH DBMS :).

## Normalisation and its types

### First Normal Form (1NF)

How to achieve atomicity?

3. Divide the table into student(base) and order(referencing) table based on the multivalued attribute order.

pk	
ID	PersonName
1	Raj
2	Riti
3	Rahul

fk	
ID	Order
1	Muffin
1	Sugar
2	Muffin
3	Sugar
3	Egg

ID	PersonName	Order
1	Raj	Muffin,Sugar
2	Riti	Muffin
3	Rahul	Sugar,Egg

## 3 Ways to Make a Table Atomic

- ① Repeat Rows
- ② Create New Columns
- ③ Separate Table into 2 tables

# LET'S START WITH DBMS :).

## Normalisation and its types

### Second Normal Form (2NF)

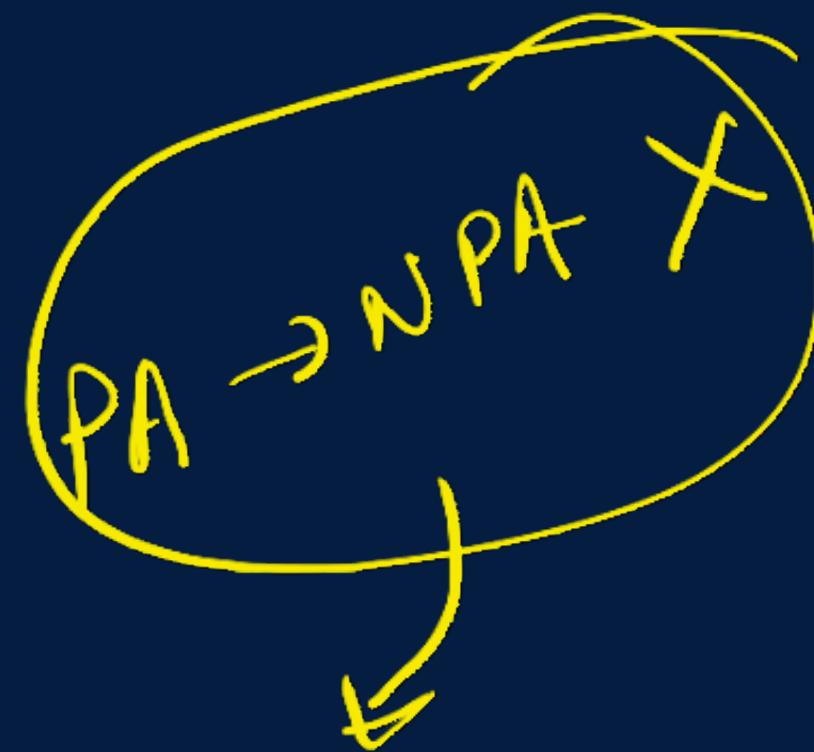
A relation is in 2NF if it satisfies the following conditions:

1. It is in First Normal Form (1NF).
2. It has no partial dependency, which means no non-prime attribute is dependent on a part of any candidate key.

When partial dependency is there in a table?

(LHS is a proper subset of Candidate key AND RHS is a non-prime attribute)

non-prime : an attribute that is not part of any candidate key



# LET'S START WITH DBMS :).

## Normalisation and its types

### Second Normal Form (2NF)

Candidate key : CustomerId+OrderId

Prime attribute :{CustomerId,OrderId}

Non-prime attribute : {OrderName}

CustomerId	OrderId	OrderName
1	1	Muffin
2	1	Muffin
1	2	Sugar
4	2	Sugar

In this relation OrderName is dependent on OrderId only, according to OrderId we provide the OrderName

OrderName is determined by only OrderId.

Here  $QA \rightarrow NPA$   
 $OrderId \rightarrow OrderName$

# LET'S START WITH DBMS :).

## Normalisation and its types

**2NF**

CustomerId	OrderId	OrderName
1	1	Muffin
2	1	Muffin
1	2	Sugar
4	2	Sugar

CustomerId	OrderId
1	1
2	1
1	2
4	2

OrderId	OrderName
1	Muffin
1	Muffin
2	Sugar
2	Sugar

# LET'S START WITH DBMS :).

## Normalisation and its types

### Second Normal Form (2NF)

Consider there is a relation R(A,B,C,D) with FD : AB→C, AB→D, B→C. Find if this is in 2NF?

1. Identify the Candidate Key

A+={A}

B+={B,C}

C+={C}

D+={D}

AB+={A,B,C,D}

More CK : A $\beta$

So, AB is a candidate key here.

# LET'S START WITH DBMS :).

## Normalisation and its types

### Second Normal Form (2NF)

2. Check for Partial Dependencies

1. LHS is a proper subset of Candidate key AND
2. RHS is a non-prime attribute

FD:  $AB \rightarrow C$ ,  $AB \rightarrow D$ ,  $B \rightarrow C$

CK : AB

prime attribute : {A,B} non-prime : {C,D}

- a.  $AB \rightarrow C$  (fully dependent, AB is not a proper subset of candidate key)
- b.  $AB \rightarrow D$  (fully dependent, AB is not a proper subset of candidate key)
- c.  $B \rightarrow C$  (partial dependency as B is a proper subset of CK and C is non-prime)

Not in 2NF.



Partial Dependency  
Means  
LHS is proper  
subset of CK  
& RHS is NPA

# LET'S START WITH DBMS :).

## Normalisation and its types

### Third Normal Form (3NF)

1. It is in Second Normal Form (2NF).
2. It has no transitive dependency, which means no non-prime attribute is transitively dependent on a candidate key.

Transitive dependent: no non-prime attribute should be dependent on another non-prime attribute

$$NPA \rightarrow NPA X$$

For any functional dependency  $X \rightarrow Y$ , one of the following conditions must be true to be in 3rd normal form.

$X$  is a superkey or candidate key(LHS)

$Y$  is a prime attribute (i.e., part of some candidate key). (RHS)

$$CK \rightarrow PA$$

$$SCK \rightarrow PA$$

# LET'S START WITH DBMS :).

## Normalisation and its types

### Third Normal Form (3NF)

Consider there is a relation R(A,B,C,D) with FD : AB→C, C→D. Find if this is in 3NF?

1. Identify the Candidate Key

$$A^+ = \{A\}$$

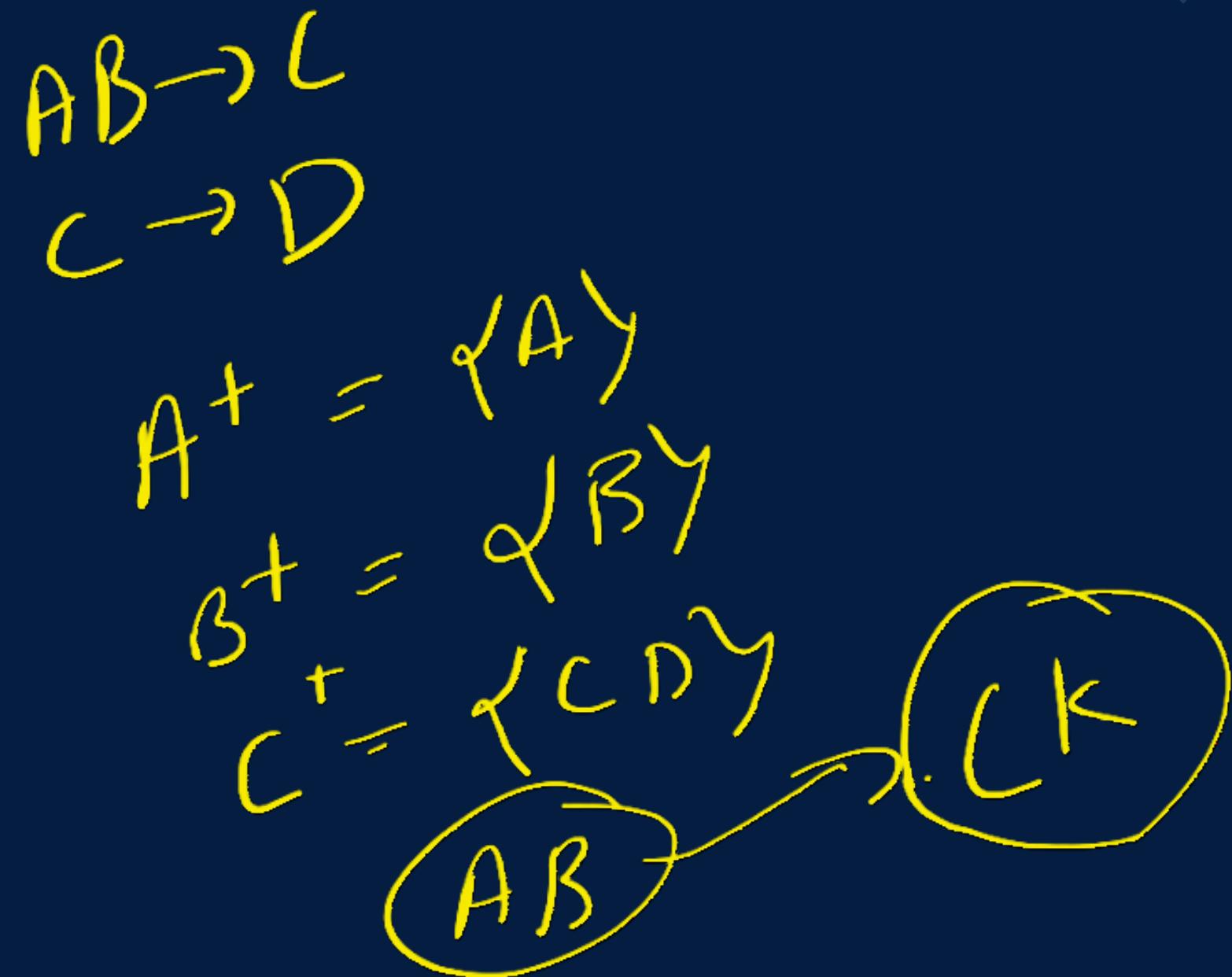
$$B^+ = \{B\}$$

$$C^+ = \{C, D\}$$

$$D^+ = \{D\}$$

$$AB^+ = \{A, B, C, D\}$$

So, AB is a candidate key here.



# LET'S START WITH DBMS :).

## Normalisation and its types

### Third Normal Form (3NF)

2. Check for transitive dependency

→ no non-prime attribute should be dependent on another non-prime attribute

FD: AB→C, C→D

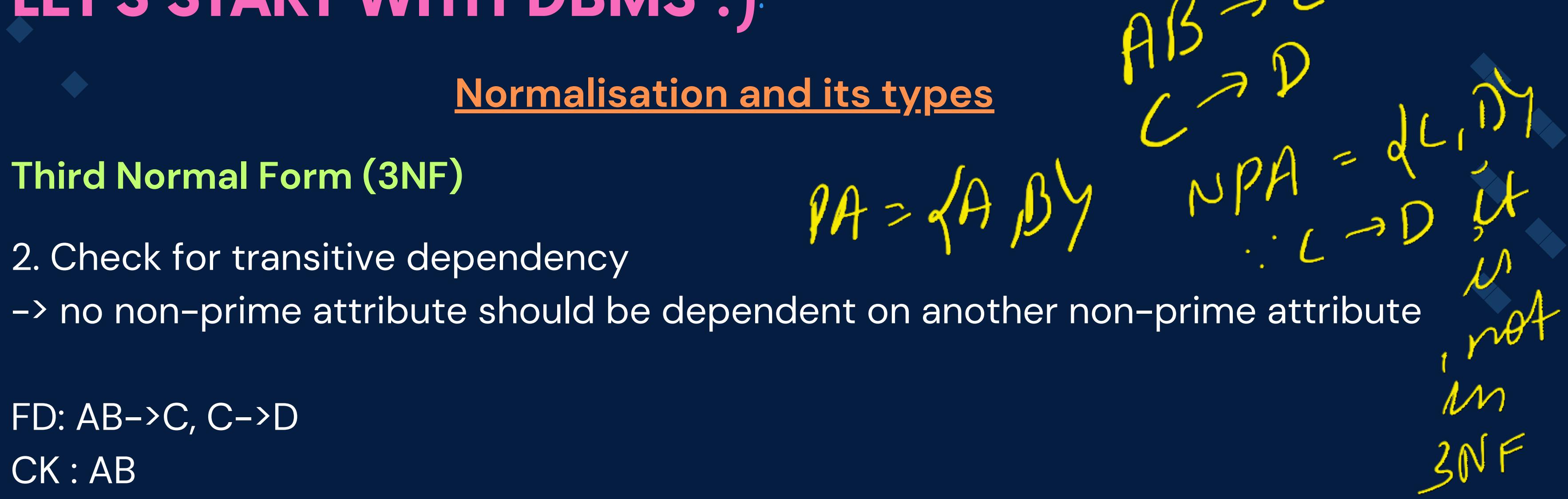
CK : AB

prime attribute : {A,B} non-prime : {C,D}

a. AB→C (no transitive as AB is a C.K)

b. C→D (transitive as C is not a superkey or candidate key or D is a prime attribute)

Not in 3NF.



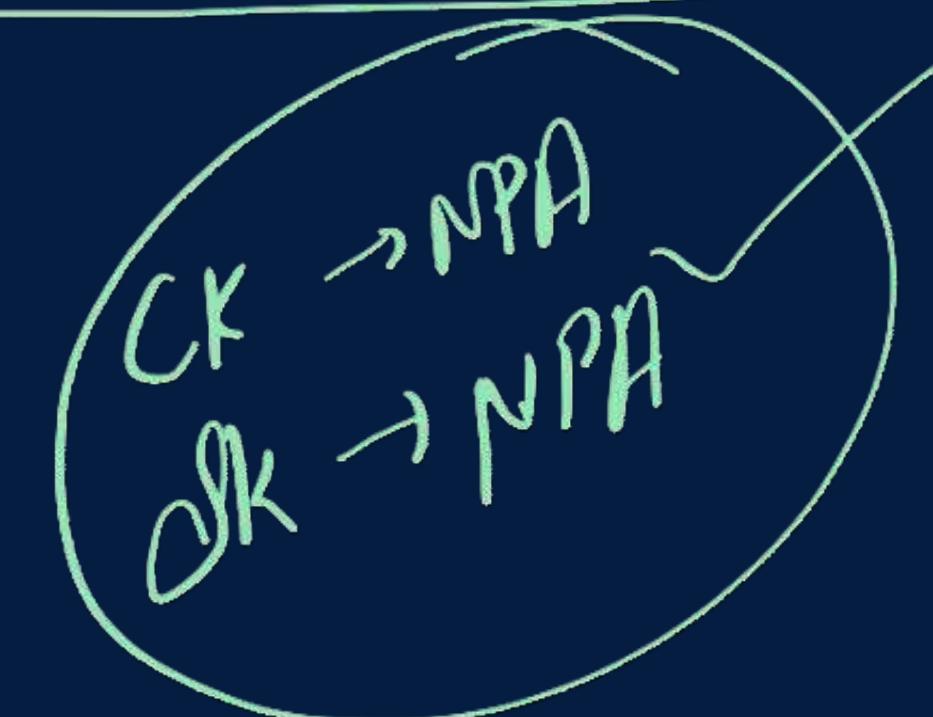
# LET'S START WITH DBMS :).

## Normalisation and its types

### BCNF(Boyce Codd Normal Form)

A relation is in BCNF if it satisfies the following conditions:

1. It is in Third Normal Form (3NF).
2. For a given FD  $X \rightarrow Y$  should always have CK or SK, and should only determine non-prime attributes



# LET'S START WITH DBMS :).

## Normalisation and its types

### BCNF(Boyce Codd Normal Form)

Consider there is a relation R(A,B,C,D) with FD : AB->C, AB->D. Find if this is in BCNF?

1. Identify the Candidate Key

$$A^+ = \{A\}$$

$$B^+ = \{B\}$$

$$C^+ = \{C\}$$

$$D^+ = \{D\}$$

$$AB^+ = \{A, B, C, D\}$$

So, AB is a candidate key here.

$$A^+ = \{A\}$$

$$B^+ = \{B\}$$

$$C^+ = \{C\}$$

$$D^+ = \{D\}$$

$$AB^+ = \{ABCD\}$$

$$AB \rightarrow CK$$

# LET'S START WITH DBMS :).

## Normalisation and its types

BCNF(Boyce Codd Normal Form)

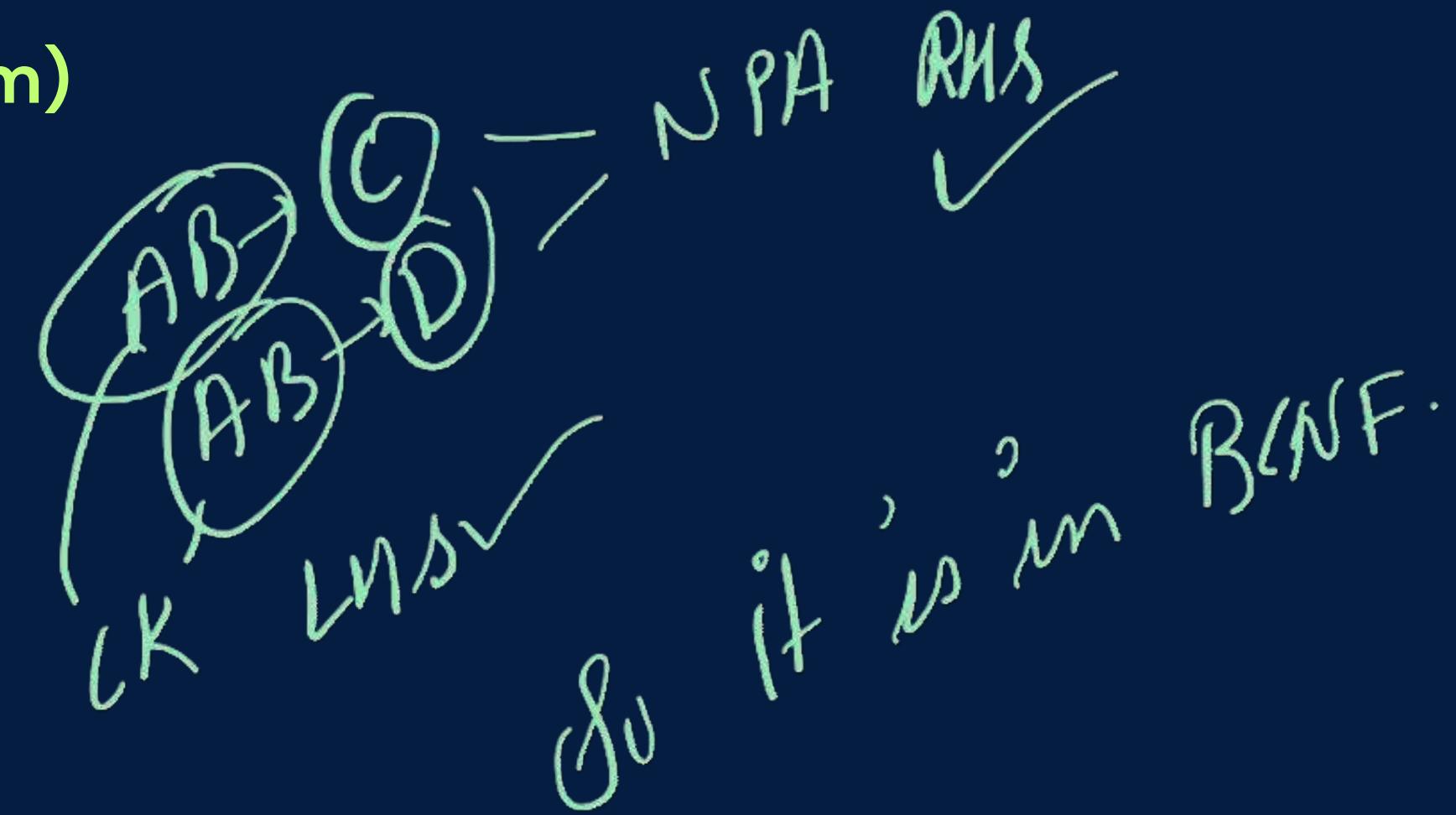
2. Check for C.K or S.K in LHS

FD:  $AB \rightarrow C$ ,  $AB \rightarrow D$

CK : AB

- a.  $AB \rightarrow C$  (LHS i.e AB is a CK)
- b.  $AB \rightarrow D$  (LHS i.e AB is a CK)

It is in BCNF.



# LET'S START WITH DBMS :).

## Lossy and Lossless decomposition

### Lossy Decomposition

In normalisation we generally break/decompose the table into 2 or more tables.

Consider there is a relation R,

**Lossy decomposition** occurs when a relation R is decomposed or broken into two or more relations, but data is lost, and the original relation R can't be reconstructed by joining these decomposed relations.

There are 2 rules for a decomposition to be lossy

1. Some data from the original relation R is lost after decomposition
2. Join of the decomposed relations( $R_1, R_2..R_n$ ) is not equal to the original relation

R

It means at the time of decomposition data is lost & can't be constructed back the same Relation.

# LET'S START WITH DBMS :).

## Lossy and Lossless decomposition

### Lossy decomposition

There is a relation R(A,B,C)

A	B	C
1	2	3
4	2	6

R

Step 1: Lets decompose the relation based on any attribute and keep that attribute as common, for now lets use B as common attribute

Decomposed relations: R1(A,B) and R2(B,C)

A	B
1	2
4	2

R1

B	C
2	3
2	6

R2

# LET'S START WITH DBMS :).

## Lossy and Lossless decomposition

### Lossy decomposition

Step 2: Lets perform a natural join between R1 and R2

When we do R1 natural join R2 we won't get the original relation back(lossy)

A	B	C
1	2	3
1	2	6
4	2	3
4	2	6

We can see some additional tuples that were not in the original relation R (lossy decomposition)

Lossy  
decomposition

R1 natural join R2

A	B	C
1	2	3
4	5	6

R

# LET'S START WITH DBMS :).

## Lossy and Lossless decomposition

### Lossy decomposition

How to ensure a decomposition is lossless

1. Divide or decompose the table on basis of CK or SK present in the relation  
so that there is no duplicacy
2. For a decomposition to be lossless
  - a.  $R_1 \cup R_2 = R$
  - b.  $R_1 \cap R_2 = \text{common attribute}$
3. To ensure that a decomposition is lossless, a common approach is to use the dependency preservation property

Avoid  
duplicacy -

# LET'S START WITH DBMS :).

## Lossy and Lossless decomposition

### Lossless Decomposition

In normalisation we generally break/decompose the table into 2 or more tables.

Consider there is a relation R,

**Lossless decomposition** ensures that when a relation R is decomposed/breaked into two or more relations, no data is lost, and the original relation R can be again reconstructed by joining these decomposed relations.

There are 2 rules for a decomposition to be lossless

1. All data in the original relation R should be preserved after decomposition
2. Join of the decomposed relations( $R_1, R_2..R_n$ )= original relation R

# LET'S START WITH DBMS :).

## Lossy and Lossless decomposition

### Lossless Decomposition

So if the table is decomposed and we want to query the attributes present in both the tables we will use the join operation.

#### Natural Join :

- The natural join operation combines tuples(rows) from two relations based on common attributes.
- It only includes those combinations of tuples that have the same values for the common attributes.

# LET'S START WITH DBMS :).

## Lossy and Lossless decomposition

### Lossless decomposition

There is a relation R(A,B,C) with CK as A

Step 1: Lets decompose the relation based on the CK or SK of the given R

Decomposed relations: R1(A,B) and R2(A,C)

A	B
1	2
4	5

R1

A	C
1	3
4	6

R2

A	B	C
1	2	3
4	5	6

R

Basically break it by using N column which is a PK / SK. The point

# LET'S START WITH DBMS :).

## Lossy and Lossless decomposition

### Lossless decomposition

There is a relation R(A,B,C) with CK as A

A	B	C
1	2	3
4	5	6

R

Step 2: Lets perform a natural join between R1 and R2

When we do R1 natural join R2 we get the original relation back(lossless)

A	B	C
1	2	3
4	5	6

R1 natural join R2

# LET'S START WITH DBMS :).

## Dependency Preserving decomposition

**Dependency preserving decomposition** ensures that the functional dependencies are preserved/maintained after decomposing a relation into two or more smaller relations.

Consider a relation  $R(A, B, C)$  with FD :  $A \rightarrow B$ ,  $B \rightarrow C$ , find if its dependency preserving when divided into  $R1(AB)$  and  $R2(BC)$

- $R1(AB) : A \quad B$
- $R2(BC) : B \quad C$

*Preserve the FD at last in the main point.*

The decomposition is dependency preserving because the functional dependencies  $A \rightarrow B$  and  $B \rightarrow C$  are preserved in  $R1$  and  $R2$

# LET'S START WITH DBMS :).

## Dependency Preserving decomposition

Consider a relation  $R(A, B, C, D)$  with FD :  $A \rightarrow B$ ,  $A \rightarrow C$ ,  $C \rightarrow D$ , find if its dependency preserving when divided into  $R1(A, B, C)$  and  $R2(C, D)$

$R1(A, B, C)$ :  $A \rightarrow B$ ,  $A \rightarrow C$ (Functional dependency preserved)

$R2(C, D)$ :  $C \rightarrow D$ (Functional dependency preserved)

The decomposition is dependency preserving because the functional dependencies  $A \rightarrow B$  and  $B \rightarrow C$  are preserved in  $R1$  and  $R2$



# LET'S START WITH DBMS :).

## Normalisation and its types

### 4NF(Fourth Normal Form)

A relation is in 4NF if it satisfies the following conditions:

1. It is in BCNF
2. It has no multi-valued dependencies

Multivalued dependency :

A multi-valued dependency  $X \rightarrow Y \cup Z$  in a relation  $R(X,Y,Z)$  implies that for each value of  $X$ , there is a set of values for  $Y$  and a set of values for  $Z$  that are independent of each other.

→ It should be BCNF  
→ No multi Valued dependencies

# LET'S START WITH DBMS :).

## Normalisation and its types

### Fourth Normal Form (4NF)

Student (StudentID, Course, PhoneNbr)

- StudentID $\rightarrow\rightarrow$ Course (A student can take multiple courses)
- StudentID $\rightarrow\rightarrow$ PhoneNbr (A student can have multiple PhoneNbr)

StudentId	Course	PhoneNbr
1	Math	123
1	Sci	123
1	Math	345
1	Sci	345
2	Hin	678
2	Eng	678
2	Hin	910
2	Eng	910

# LET'S START WITH DBMS :).

StudentId	Course	PhoneNbr
1	Math	123
1	Sci	123
1	Math	345
1	Sci	345
2	Hin	678
2	Eng	678
2	Hin	910
2	Eng	910

4NF

StudentId	Course
1	Math
1	Sci
2	Hin
2	Eng

StudentId	PhoneNbr
1	123
1	345
2	678
2	910

# LET'S START WITH DBMS :).

## Normalisation and its types

### 5NF(Fifth Normal Form) or Project-Join Normal Form (PJNF)

A relation is in 5NF if it satisfies the following conditions:

1. It is in 4NF
2. The decomposition should be lossless.

Lossless decomposition : It ensures that when a relation R is decomposed/breaked into two or more relations, no data is lost, and the original relation R can be again reconstructed by joining these decomposed relations.

# LET'S START WITH DBMS :).

## Normalisation and its types

### 5NF(Fifth Normal Form)

There is a relation R(A,B,C) with CK as A

Step 1: Lets decompose the relation based on the CK or SK of the given R

Decomposed relations: R1(A,B) and R2(A,C)

A	B
1	2
4	5

R1

A	C
1	3
4	6

R2

A	B	C
1	2	3
4	5	6

R

Always decompose it with  
CK or SK for  
lossless decomposition.

# LET'S START WITH DBMS :).

## Normalisation and its types

### 5NF(Fifth Normal Form)

There is a relation R(A,B,C) with CK as A

A	B	C
1	2	3
4	5	6

R

Step 2: Lets perform a natural join between R1 and R2

When we do R1 natural join R2 we get the original relation back(lossless)

A	B	C
1	2	3
4	5	6

R1 natural join R2

# LET'S START WITH DBMS :).

## Normalisation and its types

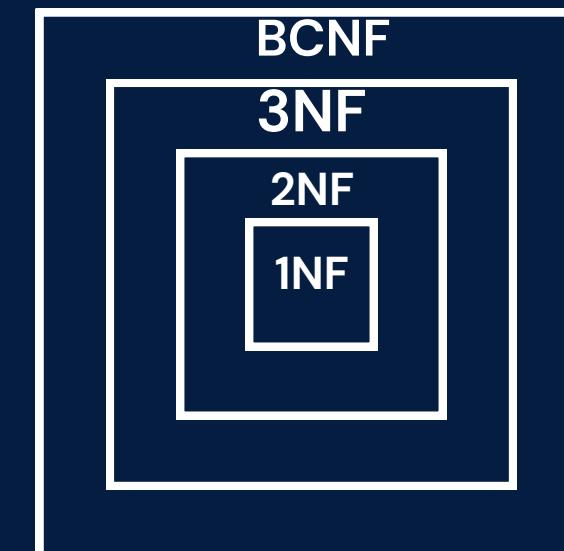
How to find the highest Normal form of a given relation?

Step 1: Identify the candidate key for the given relation using FD and closure method.

Step 2: Find the prime and non-prime attributes.

Step 3: Start checking for normal forms one by one according to their rule

Check from  
1NF → 2NF → 3NF  
BCNF



# LET'S START WITH DBMS :).

## Normalisation and its types

For a given relation R(A,B,C) with the following functional dependencies:  
A → BC, B → C, A → B, AB → C, B → A , find the highest normal form.

1. Find the CK for the given relation

C.K : A,B

2. Find prime and non-prime attributes

P.A={A,B}

N.P.A={C}

3. Checking for normal forms one by one according to their rule

# LET'S START WITH DBMS :).

## 1. First Normal Form (1NF)

A relation is in 1NF if it contains only atomic values (no multivalued attributes).

Since we are assuming our relation R is in a standard relational model, it is **already in 1NF**.

## 2. Second Normal Form (2NF)

A relation is in 2NF if it is in 1NF and every non-prime attribute is fully functionally dependent on every candidate key of the relation(P.D  $\rightarrow$ LHS is a proper subset of Candidate key AND RHS is a non-prime attribute).

- A BC = no partial dependency (A is a CK)
  - B C = no partial dependency (B is a CK)
  - A B= no partial dependency (A is a CK)
  - AB C= no partial dependency (AB is a combination of candidate keys, Its SK)
  - B A= no partial dependency (B is a CK) , **R is in 2NF.**
- CK are A & B  
JK is AB.*

# LET'S START WITH DBMS :).

## ◆ 3. Third Normal Form (3NF)

A relation is in 3NF if it is in 2NF and no transitive dependency exists.

$X \rightarrow Y$  ( $X$  is a superkey OR  $Y$  is a prime attribute if true no transitive dependency)

- A BC = no transitive dependency (A is a CK)
- B C = no transitive dependency (B is a CK)
- A B= no transitive dependency (A is a CK)
- AB C= no transitive dependency (AB is a combination of candidate keys, Its SK)
- B A= no transitive dependency (B is a CK) , R is in 3NF.

# LET'S START WITH DBMS :).

## 4. BCNF

A relation is in BCNF if it is in 3NF and for every functional dependency  $X \rightarrow Y$ , X is a superkey.

A  $BC = A$  is a CK

B  $C = B$  is a CK

A  $B = A$  is a CK

AB  $C = AB$  is a combination of candidate keys, Its SK

B  $A = B$  is a CK , R is in BCNF.

The highest normal form for the given relation R(A,B,C,D) is BCNF.

# LET'S START WITH DBMS :).

## How to normalise table

In normalisation we generally break/decompose the table into 2 or more tables.

Steps to normalize a table

1. Write down all the attributes of table, CK, Prime and non-prime attributes and start analyzing with the FD.
2. For table to be in 1NF : Table should have atomic (indivisible) values and a primary key
3. For table to be in 2NF : No Partial dependency(LHS proper subset of CK and RHS non-prime attribute should be false)
4. For table to be in 3NF : No transitive dependency(LHS must be a CK or RHS a prime attribute should be true)
5. For table to be in BCNF : LHS must be a CK or SK
6. If it fails at any of these steps decompose the table on a common attribute which is CK (lossless)

# LET'S START WITH DBMS :).

## How to normalise table

R(A,B,C,D) and assume we have the following functional dependencies:

A   B, B   C, C->D

**Step 1** : ABCD , CK-> A , Prime Attribute={A} , Non-Prime Attribute ={B,C,D}

**Step 2**: ABCDE, Since we are assuming our relation R is in a standard relational model, it is already in 1NF

**Step 3** : Check for 2NF

A   B=(no pd as A is not a proper subset of CK and B is non prime(False and True=false))

B   C=(no pd as B is not a proper subset of CK and C is non prime (False and True=false))

C->D=(no pd as C is not a proper subset of CK and D is non prime (False and True=false))

1. Write down all the attributes of table, CK, PA,NPA and start analyzing with the FD.
2. For table to be in 1NF : Table should have atomic (indivisible) values and a primary key
3. For table to be in 2NF : No Partial dependency(LHS proper subset of CK and RHS non-prime attribute should be false)
4. For table to be in 3NF : No transitive dependency(LHS must be a CK or RHS a prime attribute should be true)
5. For table to be in BCNF : LHS must be a CK or SK
6. If it fails at any of these steps decompose the table on a common attribute which is CK (lossless)

# LET'S START WITH DBMS :).

## How to normalise table

ABCD , CK-> A , Prime Attribute={A} , Non-Prime Attribute ={B,C,D}

1. Write down all the attributes of table, CK, PA, NPA and start analyzing with the FD.
2. For table to be in 1NF : Table should have atomic (indivisible) values and a primary key
3. For table to be in 2NF : No Partial dependency (LHS proper subset of CK and RHS non-prime attribute should be false)
4. For table to be in 3NF : No transitive dependency (LHS must be a CK or RHS a prime attribute should be true)
5. For table to be in BCNF : LHS must be a CK or SK
6. If it fails at any of these steps decompose the table on a common attribute which is CK (lossless)

### Step 4 : Check for 3NF

A    B=(no td as LHS is a CK)

B    C=(td is there as LHS is not CK and RHS non-prime)

C->D=(td is there as LHS is not CK and RHS non-prime)

So lets decompose the table

R1(A,B), R2(B,C), R3(C,D)

1. Write down all the attributes of table, CK, PA, NPA and start analyzing with the FD.
2. For table to be in 1NF : Table should have atomic (indivisible) values and a primary key
3. For table to be in 2NF : No Partial dependency (LHS proper subset of CK and RHS non-prime attribute should be false)
4. For table to be in 3NF : No transitive dependency (LHS must be a CK or RHS a prime attribute should be true)
5. For table to be in BCNF : LHS must be a CK or SK
6. If it fails at any of these steps decompose the table on a common attribute which is CK (lossless)

ABCD , CK  $\rightarrow$  A , Prime Attribute = {A} , Non-Prime Attribute = {B,C,D}

#### Step 4 : Check for BCNF

R1(AB) A    B=(A is a candidate key OR a super key, so R1 is in BCNF)

R2(BC) B    C=(B is a candidate key OR a super key, so R2 is in BCNF)

R3(CD) C  $\rightarrow$  D=(C is a candidate key OR a super key, so R3 is in BCNF)

Now, all decomposed relations R1, R2, and R3 are in BCNF

*If they are not in BCNF combined, Decompose them and then separately they will be in BCNF.*

# LET'S START WITH DBMS :).

To check the equivalence of  
FD's of  $R_1$  &  $R_2$

## Equivalence of Functional Dependencies

### Equivalence of Functional Dependencies

Functional Dependency: A functional dependency  $X \rightarrow Y$  holds on a relation  $R$  if, for any two tuples  $t_1$  and  $t_2$  in  $R$ , whenever  $t_1[X] = t_2[X]$ , then  $t_1[Y] = t_2[Y]$

Two sets of functional dependencies,  $F$  and  $G$ , are equivalent if the following conditions hold:

1.  $F$  implies  $G$ : Every functional dependency in  $G$  can be derived from  $F$ .  $F \supseteq G$
  2.  $G$  implies  $F$ : Every functional dependency in  $F$  can be derived from  $G$ .  $G \supseteq F$
- If both conditions are true, then we say that  $F$  and  $G$  are equivalent.

# LET'S START WITH DBMS :).

## Equivalence of Functional Dependencies

### How to find if two FD are equivalent

Step 1: Compute the Closure of both the sets

Step 2: Ensure that every functional dependency in set1 is in set2 closure

Step 3: Ensure that every functional dependency in set2 is in set1 closure

Step 4: If both subset checks pass, then set1 and set2 are equivalent.

# LET'S START WITH DBMS :).

- Step 1: Compute the Closure of both the sets
- Step 2: Ensure that every functional dependency in set1 is in set2 closure
- Step 3: Ensure that every functional dependency in set2 is in set1 closure
- Step 4: If both subset checks pass, then set1 and set2 are equivalent.

## Equivalence of Functional Dependencies

Consider two sets of functional dependencies:  $F=\{A \rightarrow B, B \rightarrow C\}$ ,  $G=\{A \rightarrow C, A \rightarrow B\}$

Check if F and G are equivalent

$$F=\{A \rightarrow B, B \rightarrow C\}$$

Closure of F, attributes  $\rightarrow A, B, C$

$$A^+=\{A, B, C\}$$

$$B^+=\{B, C\}$$

$$C^+=\{C\}$$

$$G=\{A \rightarrow C, A \rightarrow B\}$$

Closure of G, attributes  $\rightarrow A, B, C$

$$A^+=\{A, C, B\}$$

$$B^+=\{B\}$$

$$C^+=\{C\}$$

F and G are not equivalent because  $B \rightarrow C$  is not implied by G

# LET'S START WITH DBMS :).

## Minimal cover of Functional Dependency

### Why Do We Need to Find Minimal Cover?

It is a simplified version of the original set of functional dependencies

1. It helps to remove redundant functional dependencies.
2. It reduces the complexity of the functional dependencies.
3. It ensures that there are no unnecessary dependencies, which can lead to anomalies in database operations (insertion, deletion, and update).

Basically if we have a FD which can be derived by using other FD then it's redundant & should be removed.

$A \rightarrow B$   
 $B \rightarrow C$

$(A \rightarrow C)$  X

we can derive  
using the other 2 FD.  
↳ Redundant

# LET'S START WITH DBMS :).

## Minimal cover of Functional Dependency

### How to Find Minimal Cover?

Step 1: Decompose FDs (RHS) i.e  $X \rightarrow AB$  can be written as  $X \rightarrow A$ ,  $X \rightarrow B$

Step 2: Remove Redundant FD.

- Make a new FD set excluding the one you feel is redundant
- Now find the closure of LHS from the rest of the FD and see if it determines all the attributes of a table, if yes you can remove that, if no jump to the next one.

Step 3: Remove unnecessary attributes from LHS, if the determinant is a super key, it can be reduced to CK (minimal super key)

# LET'S START WITH DBMS :).

Find Minimal Cover FD: A BC , B C, A B, AB C

**Step 1:** A $\rightarrow$ B, A $\rightarrow$ C, B $\rightarrow$ C, A $\rightarrow$ B, AB $\rightarrow$ C

FD: A $\rightarrow$ B, A $\rightarrow$ C, B $\rightarrow$ C, AB $\rightarrow$ C

**Step 2 :**

1. For A $\rightarrow$ B

FD: A $\rightarrow$ C, B $\rightarrow$ C, AB $\rightarrow$ C

A $^+ = \{A, C\}$  since A $^+$  doesn't have all the attributes we shouldn't discard this

2. For A $\rightarrow$ C

FD: A $\rightarrow$ B, B $\rightarrow$ C, AB $\rightarrow$ C

A $^+ = \{A, B, C\}$ , since A $^+$  have all the attributes we can discard this

Step 1: Decompose FDs (RHS) i.e X $\rightarrow$ AB can be written as X $\rightarrow$ A, X $\rightarrow$ B

Step 2: Remove Redundant FD.

- Make a new FD set excluding the one you feel is redundant
- Now find the closure of LHS from the rest of the FD and see if it determines all the attributes of a table, if yes you can remove that, if no jump to the next one.

Step 3: Remove unnecessary attributes from LHS, if the determinant is a super key, it can be reduced to CK (minimal super key)

3. For B $\rightarrow$ C

FD: A $\rightarrow$ B, AB $\rightarrow$ C

B $^+ = \{B\}$ , since B $^+$  doesn't have all the attributes we shouldn't discard this

4. For AB $\rightarrow$ C

FD: A $\rightarrow$ B, B $\rightarrow$ C

AB $^+ = \{A, B, C\}$  since AB $^+$  have all the attributes we can discard this

Therefore, the minimal cover of the given functional dependencies is:

{A B, B C}

# LET'S START WITH DBMS :).

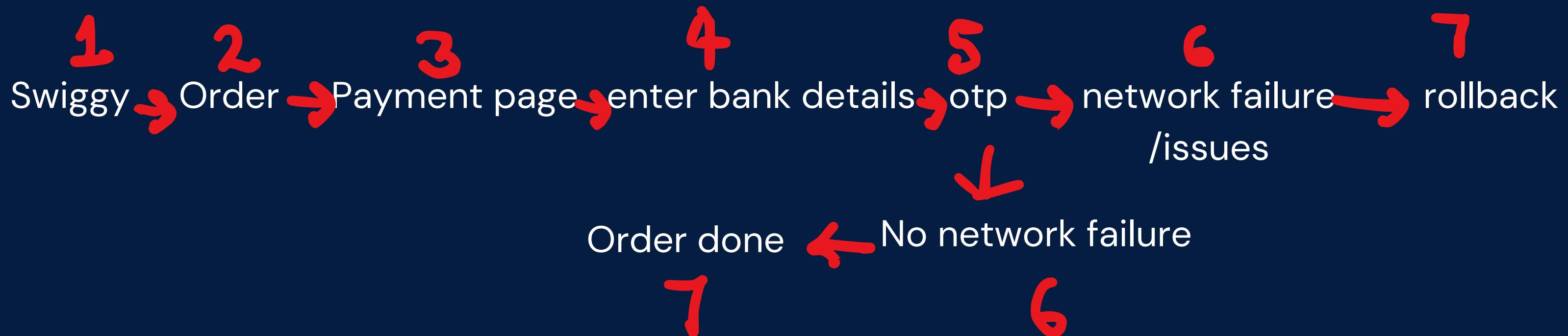
A  $\xrightarrow{50}$  B

## Transaction And Concurrency\_control

**Transaction** is a logical unit of work that comprises one or more database operations (like Read/write/commit/rollback) . In a transaction both read and write operations are fundamental actions that ensure ACID properties of transactions (data consistency and integrity)

**Read(R)**-> A read operation involves retrieving/fetching data from the database.

**Write(W)**->A write operation in a transaction involves modifying data in the database



# LET'S START WITH DBMS :).

- Consider an example of a banking application where a customer Ram wants to transfer 100rs to Shyam

**Step 1:** Begin Transaction

**Step 2:** The application will fetch/read the current bal of the Ram-> **R(Ram)**

**Step 3:** The application will calculate the new balances after the transfer ->

**Ram=Ram-100**

**Step 4:** Update it in the temporary Storage/transaction logs-> **W(Ram)**

**Step 5:** The application will fetch/read the current bal of the Shyam-> **R(Shyam)**

**Step 6:** The application will calculate the new balances after the transfer->

**Shyam=Shyam+100**

**Step 7:** Update it in the temporary Storage /transaction logs-> **W(Shyam)**

**Step 8:** If all updates are successful and there are no errors, commit the transaction to make the changes permanent. Logs are maintain till the occurrence of commit, after that log files are deleted and database is updated till step7 ->**COMMIT → if everything ok**

**Step 9:** If an error occurs during any step rollback the transaction to revert changes made within the transaction.-> **ROLLBACK → if error occurs**

# LET'S START WITH DBMS :).

## Transaction states

BEGIN  
R(Ram)  
Ram=Ram-100  
W(Ram)  
R(Shyam)  
Shyam=Shyam+100  
W(Shyam)  
COMMIT  
ROLLBACK

Case-1

ACTIVE

PARTIALLY COMMITTED

SUCCESSFUL

COMPLETED

Case-2

ACTIVE

FAILED

CANCELLED/ROLLBACK

COMPLETED

# LET'S START WITH DBMS :).

## Transaction And Concurrency control

**Concurrency control** ensures that multiple transactions can run concurrently without compromising data consistency.

Example : Consider a banking system where two transactions are happening concurrently

- 1.Ram giving Shyam 100rs
- 2.Shyam giving Ram 50rs , the data should be consistent for both transactions

Some techniques used here are:

- Locking
- Two-Phase Locking (2PL)
- Timestamp Ordering

# LET'S START WITH DBMS :).

## ACID Properties

**ACID** properties are the properties which ensures that transactions are processed reliably and accurately, even in complex situations(system failures/network issues)

- A-> **Atomicity** (Either execute all operations or none)
- C-> **Consistency** (Read should fetch upto date data and write shouldn't violate integrity constraints)
- I-> **Isolation** (One transaction should be independent of other transaction)
- D-> **Durability** (The committed transaction should remain even after a failure/crash)

# LET'S START WITH DBMS :).

## ACID Properties

**A-> Atomicity:** It ensures that a transaction is treated as a single unit of work. Either all operations are completed successfully (commit) or none of them are applied (rollback). → Rollback

This guarantees that the database remains in a consistent state despite any failures or interruptions during the transaction.

Ex : Consider Ram is transferring money to Shyam. The transaction must deduct the amount from the Ram's account and add it to the Shyam's account as a single operation.

If at any moment or at any part, this transaction fails (e.g., due to insufficient funds/system error/network error), the entire transaction is rolled back, ensuring that none of the accounts is affected partially.

# LET'S START WITH DBMS :).

## ACID Properties

C-> **Consistency**: It ensures that

1. **Read operations** retrieve consistent and up-to-date data from the database, and
2. **Write operations** ensure that data modifications maintain database constraints( such as foreign key relationships or unique constraints so that that data remains accurate)

*'Data should be following the Integrity constraints.'*

It guarantees that the database remains in a consistent state before and after the execution of each transaction.

Ex: Consider you had 100rs in your account but you want 50rs cash, so you transferred 50rs to a person X and he gave you 50rs cash.

Before transaction- 100rs(in acc)

After transaction- 100rs( 50rs in acc+ 50 rs cash)

*Retrieves upto from DB  
data*

# LET'S START WITH DBMS :).

$T_1$  cannot modify the other until  
 $T_1$  commits.

## ACID Properties

I-> **Isolation** : It ensures that if there are two transactions 1 and 2, then the changes made by Transaction 1 are not visible to Transaction 2 until Transaction 1 commits.

While the transaction is reading data, the dbms ensures that the data is consistent and isolated from other transactions. This means that other transactions cannot modify the data being read by the current transaction until it is committed or rolled back.

Ex : A= 40rs (in DB)

Transaction 1: Update value of A to 50rs

Transaction 2 : Read/get/Fetch value of A

If Transaction 1 is committed acc to Transaction 2 the value of A=50rs

If Transaction 1 is PENDING/RUNNING acc to Transaction 2 the value of A=40rs

# LET'S START WITH DBMS :).

## ACID Properties

D-> **Durability** : It ensures that once you save your data (commit a transaction), it stays saved, even if the system crashes or there is a power failure. Your data is always safe and won't disappear after you save as committed transactions are not lost.

Most DBMS use a technique called Write-Ahead Logging (WAL) to ensure durability. Before modifying data in the database, the DBMS writes the changes to a transaction log (often stored on disk) in a sequential manner. This ensures that if there is a failure event, the database can recover to a consistent state.

Ex : Consider if you are transferring 100rs to your friend and there is a sudden power outage or the system crashes right after the transaction is committed, the changes (the transfer of 100) will still be saved in the database. When the system is back up, both your account and your friend's account will reflect the updated balances.

After commit  
we can recover  
data.

# LET'S START WITH DBMS :).

lets say  $A = 10$  &

$T_1$        $T_2$   
 $W(A)$        $R(A)$   
 $A = A + 10$

Now it says Isolation levels and its types  
 $A = A + 10$   
 $= 10 + 10$   
 $= 20$   
But it's not  
Committed yet.  
But when  $T_2$  happens it reads the updated value 20  
which is not committed yet.

This is called  
dirty  
Read.

## Why do we need to learn about isolation level?

In systems where multiple transactions are executed concurrently, isolation levels manage the extent to which the operations of one transaction are isolated from those of other transactions.

Isolation levels help prevent common transactional anomalies:

- Dirty Read: Reading uncommitted data from another transaction.
- Non-repeatable Read: Data changes after it has been read within the same transaction.
- Phantom Read: New rows are added or removed by another transaction after a query.

In the above Ex - At the time of reading & writing same data A we can face this anomaly If we read its value 10 then again after sometime it is 20 and so on.

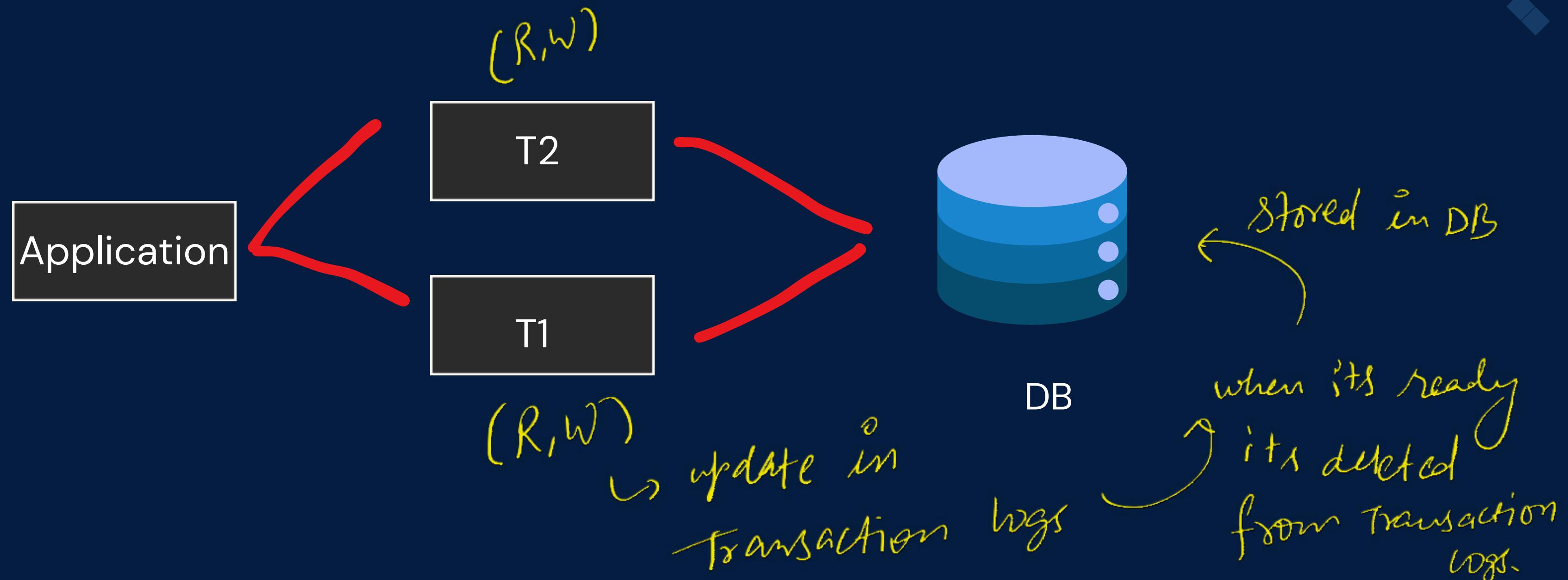
$E_1$  ), Add  $E_3$  → Gives different result.  
 $E_2$  ) Remove  $E_3$  (

T1	T2
R	R
R	W
W	R
W	W

# LET'S START WITH DBMS :).

## Isolation levels and its types

**Isolation level:** It determines the degree to which the operations in one transaction are isolated from those in other transactions.



# LET'S START WITH DBMS :).

## Isolation levels and its types

### Anamolies/Voilations to Isolation level

1. **Dirty Read** : Reading data written by a transaction that has not yet committed  
Consider if T2 reads the data written by T1 and if T1 fails, it becomes irrelevant.

T1	T2
W(A)	
R(A)	

*T<sub>1</sub> writes*  
↓  
*T<sub>2</sub> reads* → *dirty Read.*  
↓  
*But T<sub>1</sub> fails*  
↳ *Anomaly*

# LET'S START WITH DBMS :).

# Isolation levels and its types

# Anomalies/Violations to Isolation leve

**2. Non-Repeatable Read :** Reading the same row twice within a transaction and getting different values because another transaction modified the row and committed.

Consider if T2 modifies the data which T1 already Read and if T1 continue the transaction the data will be changed



Let's say  $A = 10$

$A = A + 10$

$A = 20$

Now in Ti  
It reads value  
of  $A$  as 10  
2 20 both

Causing Anomaly.

# LET'S START WITH DBMS :).

## Isolation levels and its types

### Anamolies/Voilations to Isolation level

3. **Phantom Read** : Getting different sets of rows in subsequent queries within the same transaction because another transaction inserted or deleted rows and committed.

T1(Query(id)) -> Fetch the name

T2(Query(id)) -> Insert a new entry

T1(Query(id)) -> Fetch the name

let's say we have 2 emp

2 emp details.

3 emp details

Phantom Read.

# LET'S START WITH DBMS :).

## Isolation levels and its types

There are 4 isolation levels which helps us with these anomalies:

### **Types of Isolation Levels**

- Read Uncommitted
- Read Committed
- Repeatable Read
- Serializable

# LET'S START WITH DBMS :).

## Isolation levels and its types

**Read Uncommitted:** The lowest isolation level where transactions can see uncommitted changes made by other transactions. If Transaction T1 is writing a value to a table, Transaction T2 can read this value before T1 commits.

- Dirty Reads: Yes
- Non-Repeatable Reads: Yes
- Phantom Reads: Yes

All  
problems  
still exist

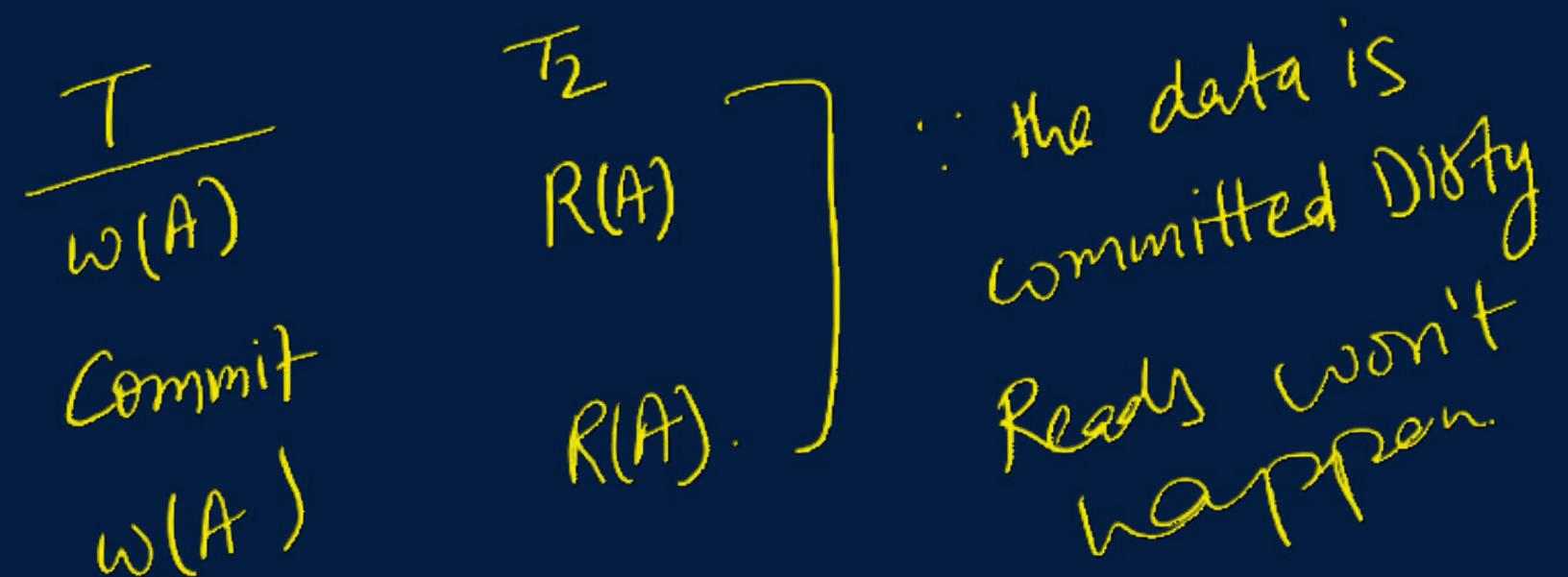
→ lowest level of Isolation.  
→  $T_1$  can see uncommitted changes in  $T_2$ .  
→  $T_1$   $w(A)$   $\xrightarrow{\quad}$   $T_2$   $R(A)$   
    ↳ Problems Exist

# LET'S START WITH DBMS :).

## Isolation levels and its types

**Read Committed:** It ensures that any data read during the transaction is committed at the moment it is read. If T1 has done some write operation T2 can only read the data when T1 is committed

- Dirty Reads: No
- Non-Repeatable Reads: Yes
- Phantom Reads: Yes



But still non-repeatable reads will exist along with phantom reads.

# LET'S START WITH DBMS :).

## Isolation levels and its types

**Repeatable Read:** It ensures that if a transaction reads a row, it will see the same values for that row during the entire transaction, even if other transactions modify the data and commit. If Transaction T1 reads a value, Transaction T2 cannot modify that value until T1 completes. But T2 can insert new rows that T1 can see on subsequent reads.

- Dirty Reads: No
- Non-Repeatable Reads: No
- Phantom Reads: Yes

Here T<sub>1</sub> takes care of non-repeatable  
Reads - But not phantom reads.

# LET'S START WITH DBMS :).

## Isolation levels and its types

**Serializable:** It ensures a serial transaction execution, so that there is complete isolation.

If Transaction T1 is executing, Transaction T2 must wait until T1 completes

- Dirty Reads: No
- Non-Repeatable Reads: No
- Phantom Reads: No

No Anomaly will  
occur in this case

# LET'S START WITH DBMS :).

## Schedule and its Types

**Schedule** : It refers to the sequence in which a set of concurrent/multiple transactions are executed. You can also say it as a sequence in which the operations (such as read, write, commit, and abort) of multiple transactions are executed. It is really helpful to ensure data consistency and integrity.

If there are T1, T2, T3....TN (n) transactions then the possible schedules=  $n!$  ( n factorial)

Ex : Schedule sc1

T1	T2
R(A)	
	R(A)
	W(A)
Commit	
	Commit

if commits are available  $\rightarrow$  complete schedule

if not  $\rightarrow$  incomplete schedule.

# LET'S START WITH DBMS :).

## Schedule and its Types

**Incomplete schedule :** An incomplete schedule is one where not all transactions have reached their final state of either commit or abort.

T1:Read(A)

T1:Write(A)

T2:Read(B)

T2:Write(B)

T2:COMMIT

Here, T1 is still in progress as there is no COMMIT for transaction T1.

T1	T2
R(A)	
W(A)	
	R(B)
	W(B)
	Commit

# LET'S START WITH DBMS :).

## Schedule and its Types

**Complete schedule** : A complete schedule is one where all the transactions in the schedule have either committed or aborted.

T1:Read(A)

T1:Write(A)

T1:COMMIT

T2:Read(B)

T2:Write(B)

T2:COMMIT

T1	T2
R(A)	
W(A)	
Commit	
	R(B)
	W(B)
	Commit

# LET'S START WITH DBMS :).

## Schedule and its Types

### Types of Schedule

1. Serial Schedule
2. Concurrent or Non-Serial Schedule
3. Conflict–Serializable Schedule
4. View–Serializable Schedule
5. Recoverable Schedule
6. Irrecoverable Schedule
7. Cascadeless Schedule
8. Cascading Schedule
9. Strict Schedule

# LET'S START WITH DBMS :).

## Schedule and its Types

**1.Serial Schedule** : A serial schedule is one where transactions are executed one after another. We can say it like if there are two transactions T1 and T2, T1 should commit to completion before T2 starts.

Example : T1->T2

T1:Read(A)

T1:Write(A)

T1:COMMIT(T1)

T2:Read(B)

T2:Write(B)

T2:COMMIT(T2)

T2 starts only after T1 is completed/ committed.

T1	T2
R(A)	
W(A)	
Commit	
	R(B)
	W(B)
	Commit

# LET'S START WITH DBMS :).

## Schedule and its Types

### Serial Schedule

#### Advantages :

1. It follows the ACID properties. Transactions are isolated from each other.
2. It maintains consistency.

#### Challenges:

1. Since there is poor throughput (no of transactions completed per unit time) and memory utilisation, this is not suggested as it can be inefficient.
2. Since wait time is high, less no of transactions are completed.

Wait Time ↑ ⇒ No. of transactions ↓  
poor throughput

# LET'S START WITH DBMS :).

## Schedule and its Types

**2. Non-Serial/Concurrent Schedule** : A non-serial schedule is one where multiple transactions can execute simultaneously (operations of multiple transactions are alternate/interleaved executions). We can say it like if there are two transactions T1 and T2, T2 doesn't need to wait for T1 to commit, it can start at any point.

Example : T1 ,T2, T3

T2 didn't waited for T1 to commit

T1	T2	T3
R(A)		
	R(A)	
		R(B)
	W(A)	
COMMIT		
		COMMIT

# LET'S START WITH DBMS :).

## Schedule and its Types

Resource Utilization ↑  
Throughput ↑

ACID issues ↑  
Consistency ↑

### Non-Serial Schedule

#### Advantages :

1. Better resource utilization
2. Wait time is not involved. One transaction doesn't need to wait for the running one to finish.
3. Throughput (no of transactions completed per unit time) is high

#### Challenges:

1. Consistency issue may arise because of non-serial execution. It requires robust concurrency control mechanisms to ensure data consistency and integrity.
2. We can use Serializability and Concurrency Control Mechanisms to ensure consistency.

# LET'S START WITH DBMS :).

## Schedule and its Types

**3. Conflict-Serializable Schedule :** A schedule is conflict-serializable if it can be transformed into a serial schedule by swapping adjacent non-conflicting operations.

R(A) T1 & R(A) T2: No conflict (both reads)

R(A) T1 & W(A) T2: RW conflict (T2 reads A after T1 writes A)

W(A) T1 & W(A) T2: WW conflict (both write to A)

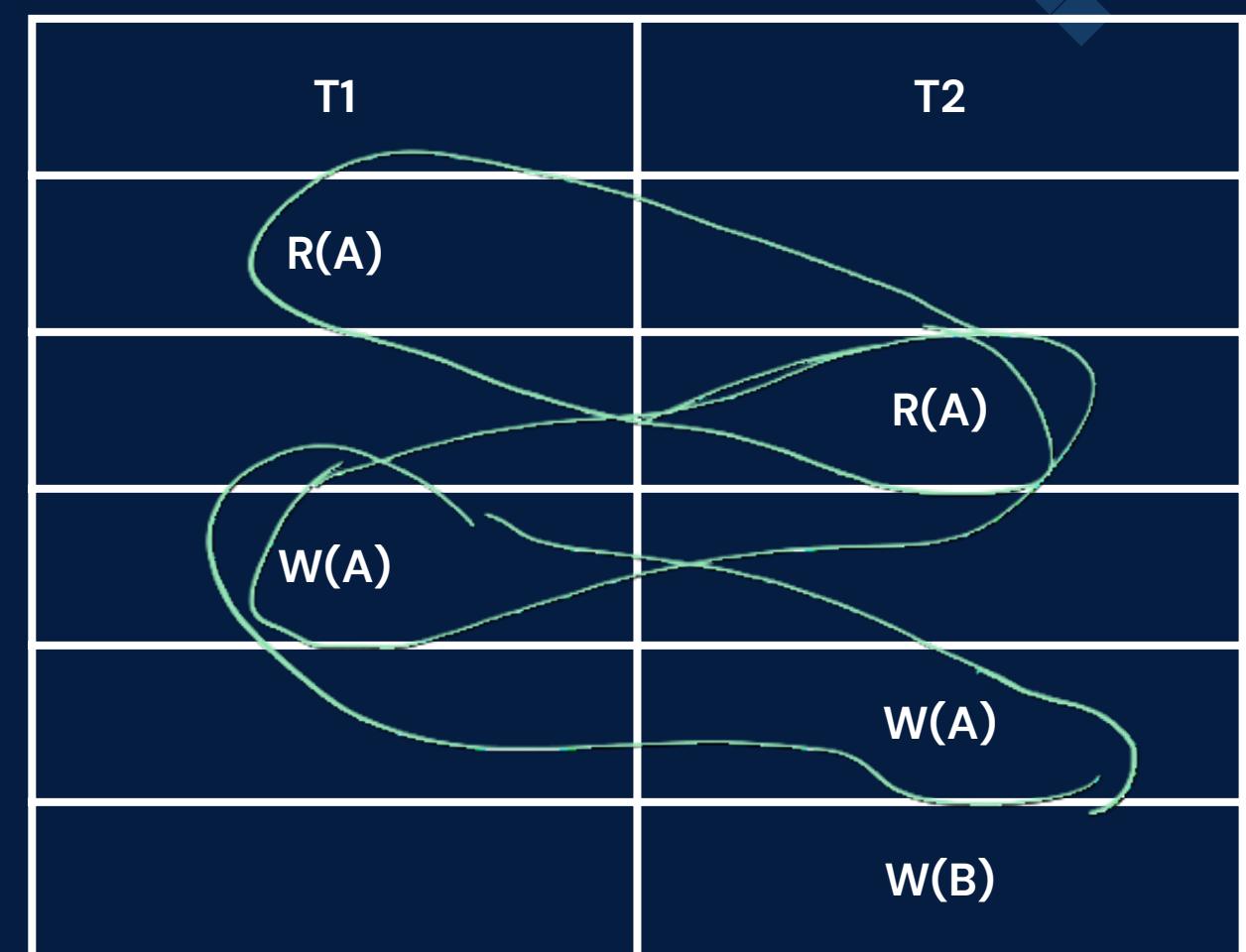
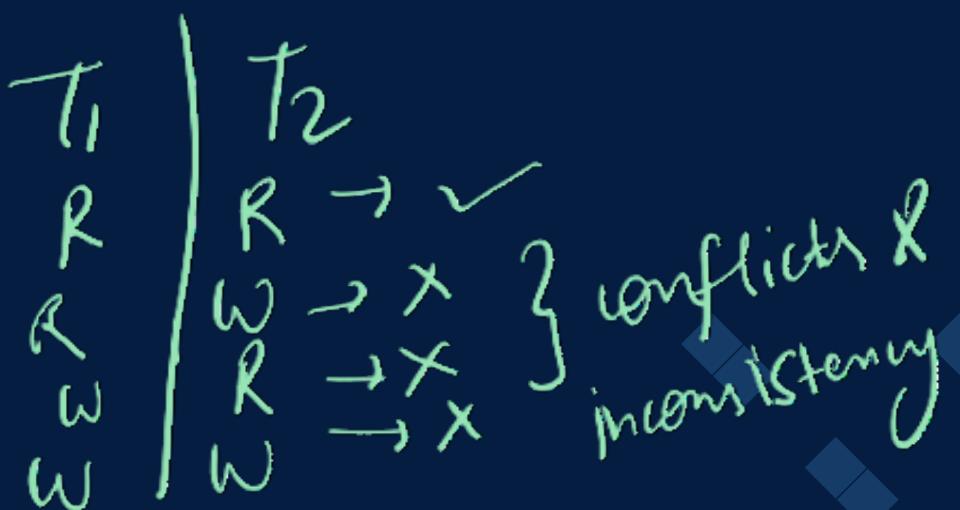
**Conflict pairs :** Read-Write, Write-Read , Write-Write(perfomed on the same data item)

**Non-conflict pairs :**

Read-Read (perfomed on the same data item),

Read-Read (perfomed on the different data item),

Write-Write(perfomed on the different data item)



# LET'S START WITH DBMS :).

## Schedule and its Types

### 4. View-Serializable Schedule :

View serializability ensures that the database state seen by transactions in a concurrent schedule can be replicated by some serial execution of those transactions.

When we find cycle in our conflict graph, we don't know if our schedule is serializable or not, so we use the view serializability here.

A schedule is view serializable if its view equivalent is equal to a serial schedule/execution.

T1	T2
R(A)	
	R(A)
W(B)	
	W(A)

# LET'S START WITH DBMS :).

## Schedule and its Types

**5. Recoverable Schedule** : A recoverable schedule ensures that if a transaction reads data from another transaction, it should not commit until the transaction from which it read has committed. This helps in maintaining the integrity of the database in case a transaction fails.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
COMMIT	
	COMMIT

Recoverable Schedule

*It means if  
T1 is R(A) from  
then it waits  
T1 to commit else  
for T1 to commit else  
it can face inconsistency.*

# LET'S START WITH DBMS :).

## Schedule and its Types

6. **Irrecoverable Schedule** : An irrecoverable schedule allows a transaction to commit even if it has read data from another uncommitted transaction. This can lead to inconsistencies and make it impossible to recover from certain failures.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	COMMIT
FAIL	

Irrecoverable Schedule

Opp. of Recoverable  
Inconsistency ↑  
fail to recover  
ever.

# LET'S START WITH DBMS :).

*T<sub>1</sub> cause all other to abort, this is cascading.*

## Schedule and its Types

**7. Cascading Schedule :** This schedule happens when the failure or abort of one transaction causes a series of other transactions to also abort.

- T1 Writes to A: T1 writes to data item A
- T2 Reads A: T2 reads the uncommitted value of A written by T1

Now, if T1 fails and aborts, T2 must also abort because it has read an uncommitted value from T1. Consequently, T3 must abort as well

Issues:

1. Performance degradation because multiple transactions need to be rolled back
2. Improper CPU resource utilisation

T1	T2	T3
R(A)		
W(A)		
	R(A)	
		R(A)
Fail		

Cascading Schedule

# LET'S START WITH DBMS :).

## Schedule and its Types

**8. Cascadeless Schedule :** It ensures transactions only read committed data, such that the abort of one transaction does not lead to the abort of other transactions that have already read its uncommitted changes.

- T1 Writes to A: T1 writes to data item A
- T2 Reads A: T2 reads the committed value of A

Ensuring there is no write-read problem (dirty read)  
Also, T2 does not read any uncommitted data,  
there are no cascading aborts in this schedule.

Issues:

1. Write-Write problem is encountered.
2. Performance overhead is there

Only  
Allows  
Reading  
of Committed  
Data.

T1	T2	T3
R(A)		
W(A)		
Commit		
	R(A)	
		R(A)

Cascadeless Schedule

# LET'S START WITH DBMS :).

## Schedule and its Types

**9. Strict Schedule :** It ensures transaction is not allowed to read or write a data item that another transaction has written until the first transaction has either committed or aborted. It prevents cascading aborts.

- T2 cannot read or write the value of A until T1 has committed.
- This ensures that T2 only sees the committed value of A

*I am as Serial Schedule  
& prevent cascading  
Aborts.*

T1	T2
R(A)	
W(A)	
Commit	
	R(A)
	W(A)
	COMMIT

Strict Schedule

# LET'S START WITH DBMS :).

## Concurrent VS Parallel Schedule

For	Concurrent	Parallel
Defination	Concurrent scheduling <u>manages</u> multiple tasks at the same time. Tasks can <u>overlap</u> in their execution periods but do not run exactly simultaneously.	Parallel scheduling <u>runs</u> multiple tasks <u>simultaneously</u> using multiple cores or processors. Each task is executed on a separate core or processor at the same time.
Execution	Achieved on a <u>single-core processor</u> through <u>context switching</u> , where the <u>CPU</u> rapidly alternates between tasks, creating the illusion of simultaneous execution.	Requires a multi-core processor or multiple processors, with each core/processor handling a different task concurrently.
Example	Multi-threading on a single-core CPU, where threads take turns using the CPU.	Multi-threading on a multi-core CPU, where threads run concurrently on different cores.

# LET'S START WITH DBMS :).

## Serializability and its types

**Serializability:** It ensures that concurrent transactions yield results that are consistent with some serial execution i.e the final state of the database after executing a set of transactions concurrently should be the same as if the transactions had been executed one after another in some order.

In case of concurrent schedule consistency issue may arise because of non-serial execution and we do serializability there, serial schedules are already serial

*Ensures Consistency .*

# LET'S START WITH DBMS :).

- Consider nodes as transactions, and edges represent conflicts and detect if the resulting graph has cycles.

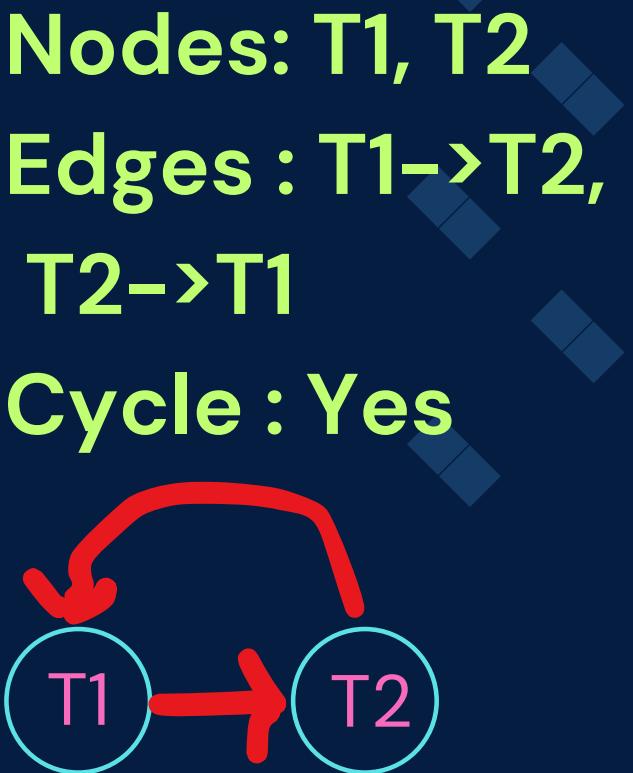
T1	T2
R(A)	
W(A)	
Commit	
	R(A)
	W(A)
	Coomit

SERIAL SCHEDEULE

for checking if any schedule is  
Serializable or not we check both  
Conflict & View Serializability.  
If cycle ✗ → Serializable ✓  
cycle ✓ → view equivalent  
follows condition ✓ ✓  
else ✗

T1	T2
R(A)	
	R(A)
	W(A)
W(A)	

CONCURRENT SCHEDEULE

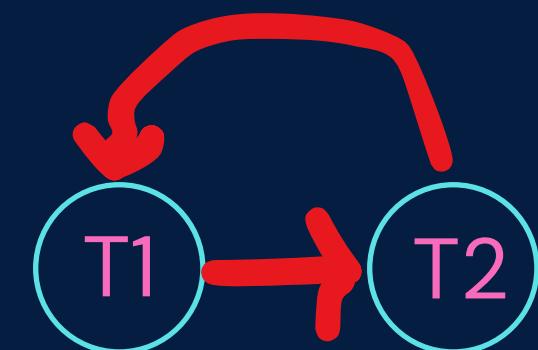


A concurrent schedule does not always have a cycle.

A concurrent schedule can be conflict-serializable, meaning that it is equivalent to some serial schedule of transactions and its conflict graph does not have any cycles.

# LET'S START WITH DBMS :).

T1	T2
R(A)	
	R(A)
	W(A)
W(A)	



## CONCURRENT SCHEDULE

**Nodes:** T1, T2

**Edges :** T1→T2, T2→T1

**Cycle :** Yes

Now, since a cycle is detected we need to serialize them  
So, we use the serializability here

- **Conflict-Serializability→** to detect the cycle using conflict graph
- **View-Serializability→** to check if schedule is serializable after a cycle is detected.

Why serializing them?

- To avoid consistency issue which may arise because of non-serial execution

# LET'S START WITH DBMS :).

## Serializability and its types

### Types of Serializability

- Conflict-Serializability
- View-Serializability

# LET'S START WITH DBMS :).

## Conflict-Serializability

**Serializability** : Serializability is a property of a schedule where the outcome is equivalent to some serial execution of the transactions.

**Conflict-Serializability** : A schedule is said to be conflict serializable when one of its conflict equivalent is serializable. So basically, if a schedule can be transformed into a serial schedule by swapping non-conflicting operations, then the schedule is conflict serializable.

**Conflict equivalent** : If a schedule S1 is formed after swapping adjacent non-conflicting operations/pairs in a given schedule S, then S1 and S are conflict equivalent.

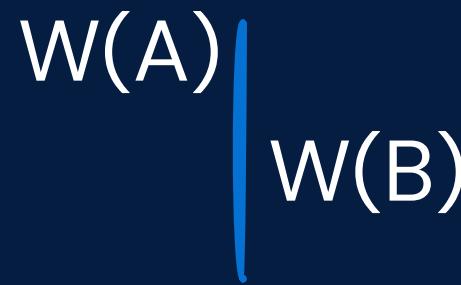
Conflict pairs : Read-Write, Write-Read , Write-Write(perfomed on the same data item)

Non-conflict pairs : Read-Read (perfomed on the same data item), Read-Read (perfomed on the different data item), Write-Write(perfomed on the different data item)

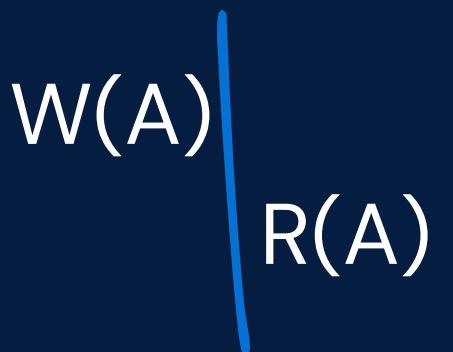
# LET'S START WITH DBMS :).

## Conflict-Serializability

Non- Conflict pairs



Conflict pairs



# LET'S START WITH DBMS :).

## Conflict-Serializability

**How to achieve conflict equivalent schedule :**

When a schedule can be transformed into a serial schedule by swapping adjacent non-conflicting operations. Conflicts arise when two transactions access the same data item, and at least one of them is a write operations.

**Now, why are we only swapping the non-conflict pairs and not the conflict ones?**

So if we swap the conflict pairs, the order of execution if it was

T1: R(A)

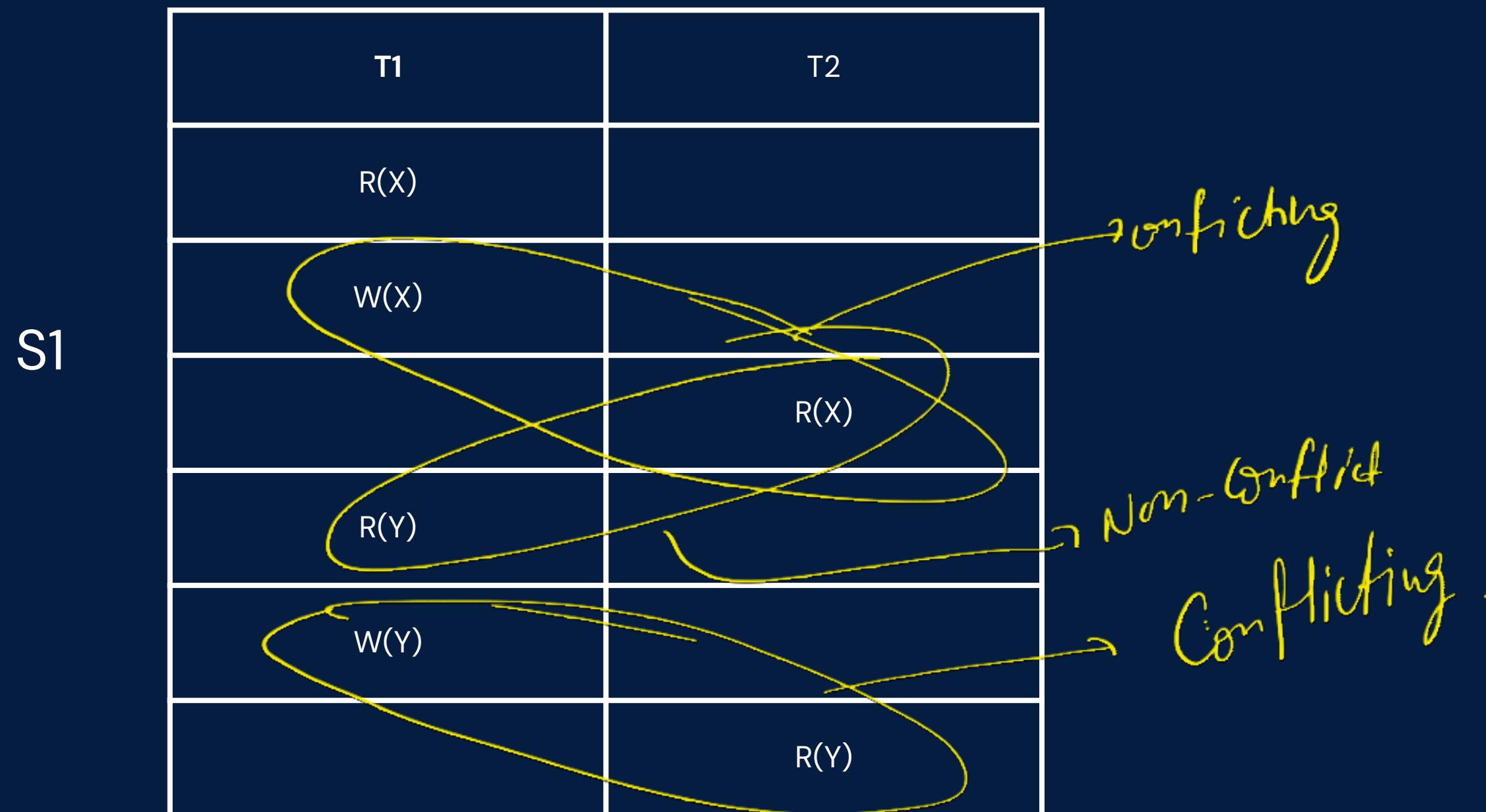
T2: W(A)

the results values may change as first we were reading A and then writing/modifying it, but now it will be writing A and then reading the modified value so the result might change if we change the order of execution.

# LET'S START WITH DBMS :).

## Conflict-Serializability

Q. Find a conflict equivalent for a schedule S1



# LET'S START WITH DBMS :).

## Conflict-Serializability

Q. Find a conflict equivalent for a schedule S1

1. Find the adjacent non-conflicting pairs and do a swap

T2 : R(X) T1: R(Y)

T1	T2
R(X)	
w(X)	
R(Y)	R(X)
w(Y)	
	R(Y)

T1	T2
R(X)	
w(X)	
	R(X)
R(Y)	
w(Y)	
	R(Y)

S1

→ This is  
After swapping  
→ Non-conflicting.  
→ Conflicting

# LET'S START WITH DBMS :).

## Conflict-Serializability

Q. Find a conflict equivalent for a schedule S1

2. After the first swap again search for adjacent non-conflicting pairs and swap if any

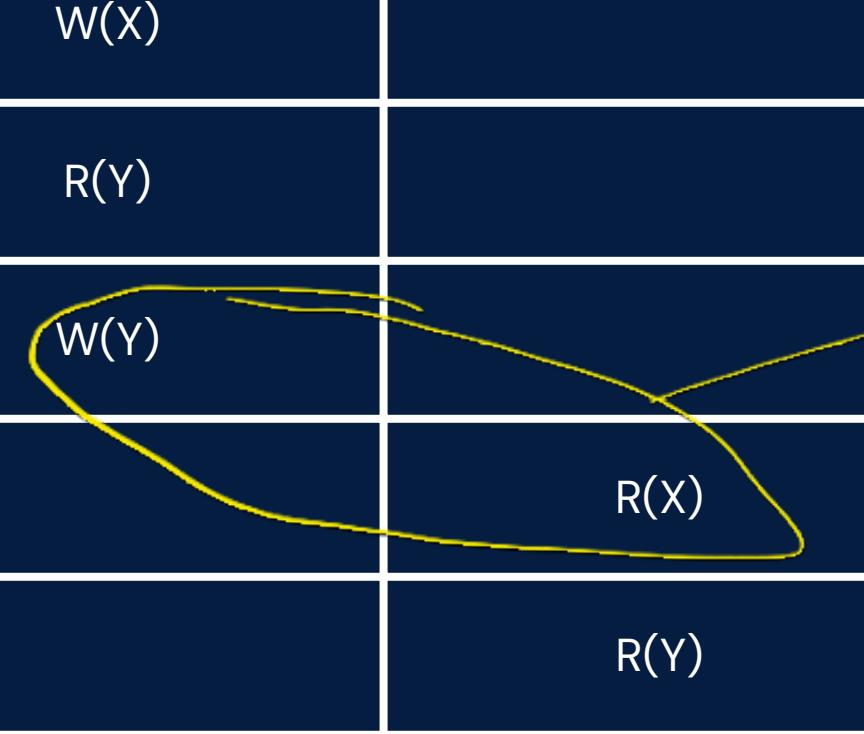
T2 : R(X) T1: W(Y)

So, S1 is a serializable schedule as S1' has a serial execution(i.e its conflict equivalent)

T1	T2
R(X)	
W(X)	
R(Y)	
W(Y)	R(X)
	R(Y)

T1	T2
R(X)	
W(X)	
R(Y)	
	R(X)
W(Y)	
	R(Y)

S1 after swap



After swapping  
Now this is conflict  
equivalent

When done on the same data item

# LET'S START WITH DBMS :).

For complex schedules  
we don't use above method.

## Conflict-Serializability

How to check whether a schedule is conflict-serializable or not?

Conflicts occur when two operations from different transactions access the same data item and at least one of them is a write operation.

**Conflict graph/ precedence graph** : A conflict graph, or precedence graph, is a directed graph used to determine conflict serializability. The nodes represent transactions, and the edges represent conflicts between transactions

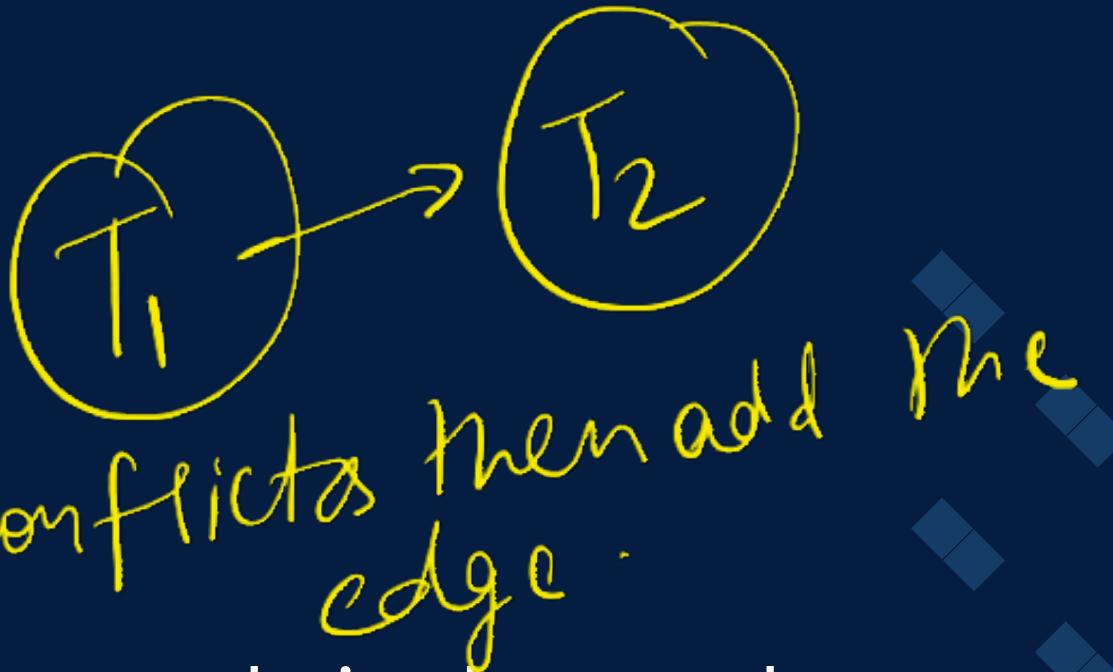
# LET'S START WITH DBMS :).

## Conflict-Serializability

**Conflict graph/ precedence graph:**

- **Nodes:** Each transaction in the schedule is represented as a node in the graph.
- **Edges:** An edge from transaction  $T(X)$  to transaction  $T(Y)$ (denoted  $T(X) \rightarrow T(Y)$ ) is added if
  - a. an operation of  $T(X)$  conflicts with an operation of  $T(Y)$  and
  - b.  $T(X)$  operation precedes  $T(Y)$  operation in the schedule.
- **Cycle Detection:** The schedule is conflict-serializable if and only if the conflict graph is acyclic. If there are no cycles in the graph, it means that the schedule can be serialized without violating the order of conflicting operations.

No cycle  $\rightarrow$  Means Conflict Serializable



# LET'S START WITH DBMS :).

## Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

T1	T2	T3
R(A)		
	R(A)	
W(A)		
		W(A)
	W(B)	
		R(B)

- T1 reads A
- T2 reads A
- T1 writes A
- T3 writes A
- T2 writes B
- T3 reads B

# LET'S START WITH DBMS :).

## Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

### Step 1: Find the conflict in the schedule

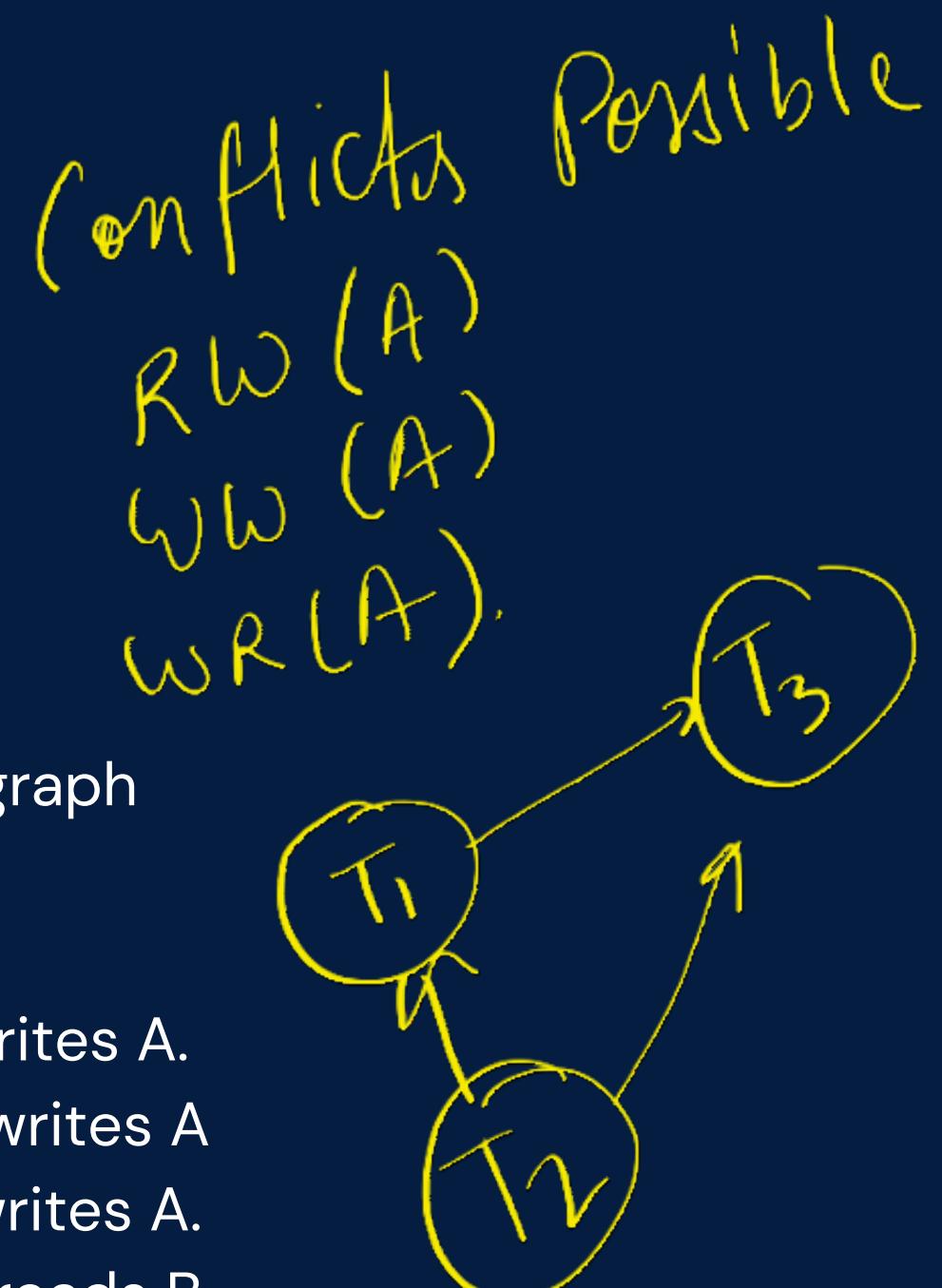
- RW Conflict (T1, T3) on A
- RW Conflict (T2, T3) on A
- RW Conflict (T2,T1) on A
- WW Conflict (T1, T3) on A
- WR Conflict( T2.T3) on B

### Step 2: Find the nodes

Each transaction T1 T2, T3 is a node in the graph

### Step 3: Find the edges/conflicts

- T1 → T3: Because T1 reads A before T3 writes A.
- T2 → T3: Because T2 reads A before T3 writes A
- T1 → T3: Because T1 writes A before T3 writes A.
- T2 → T3: Because T2 writes B before T3 reads B.
- T2→T1 : Because T2 reads A before T1 writes A.



Write-Write conflict  
W(B) - W(B)  
W(A)-W(A)

Write-Read conflict  
W(B) - R(B)  
W(A)-R(A)

Read-write conflict  
R(B) - W(B)  
R(A)-W(A)

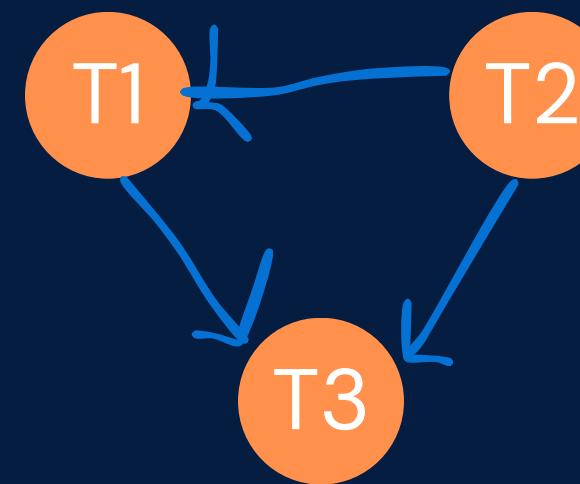
T1	T2	T3
R(A)		
	R(A)	
W(A)		
		W(A)
		W(B)
		R(B)

# LET'S START WITH DBMS :).

## Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

Step 4 : Draw the graph



edges

- T2 T1
- T1 T3
- T2 T3

T1	T2	T3
R(A)		
	R(A)	
W(A)		
		W(A)
		W(B)
		R(B)

Step 5 : If the graph has cycle it is not conflict-serializable or serializable, if not lets find the serial execution of transactions

Since the conflict graph is acyclic, the schedule is conflict-serializable

# LET'S START WITH DBMS :).

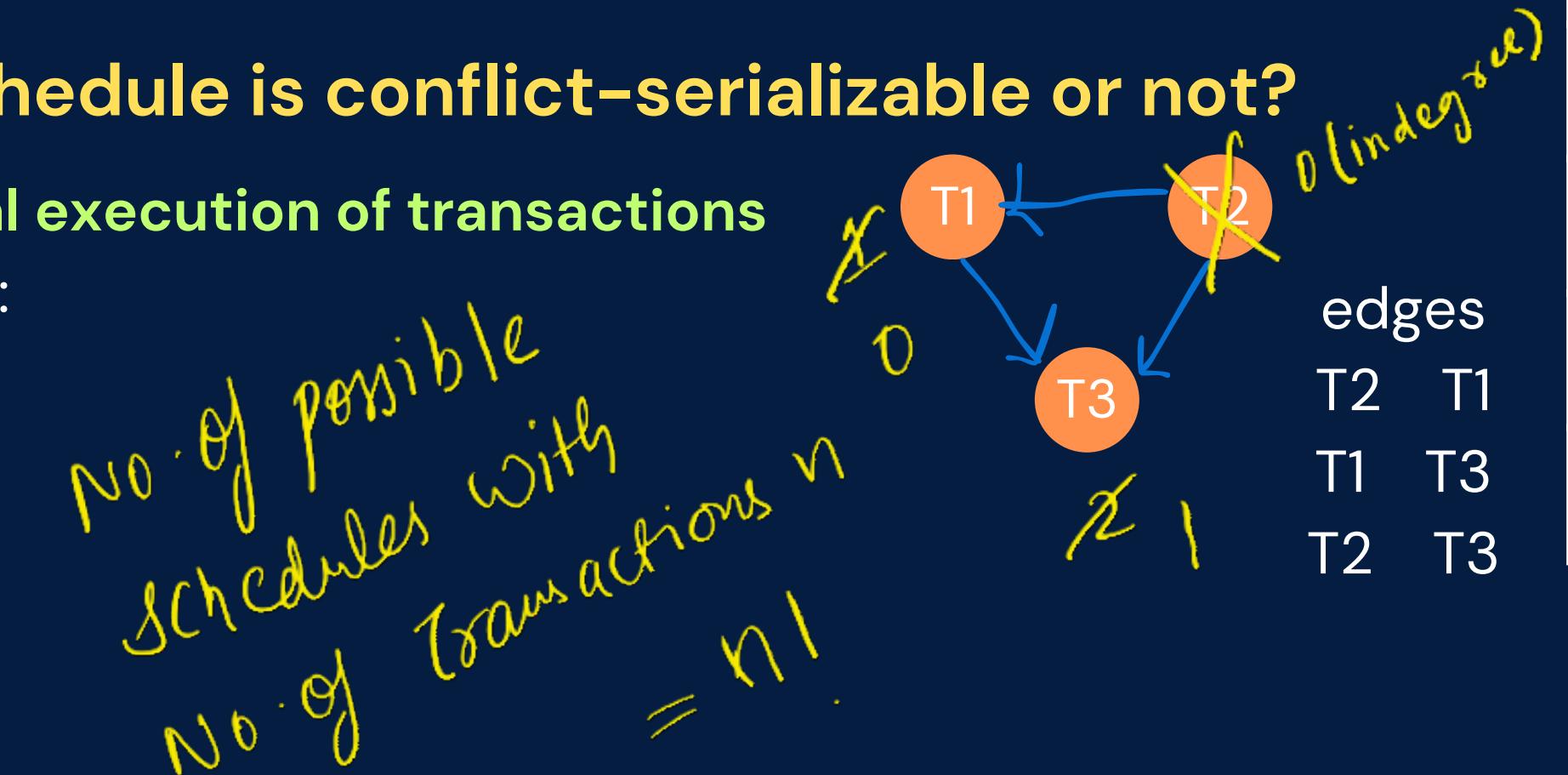
## Conflict-Serializability

Q. Check if the given schedule is conflict-serializable or not?

Step 6 : Lets find the serial execution of transactions

Possible combinations are :

T1→T2→T3  
T1→T3→T2  
T2→T1→T3  
T2→T3→T1  
T3→T2→T1  
T3→T1→T2



Find the indegree(the number of edges directed into that node) and if its 0 it can be the first in serial execution

T1 - 1, T2 - 0, T3 - 2 , T2 would be the first as indegree is 0

- T2 must precede T1
- T1 must precede T3

Therefore, one possible equivalent serial schedule is T2 T1 T3.

T1	T2	T3
R(A)		
	R(A)	
W(A)		
		W(A)
		W(B)
		R(B)

# LET'S START WITH DBMS :).

## View-Serializability

View serializability ensures that the database state seen by transactions in a concurrent schedule can be replicated by some serial execution of those transactions.

When we find cycle in our conflict graph, we don't know if our schedule is serializable or not, so we use the view serializability here.

A schedule is view serializable if its view equivalent is equal to a serial schedule/execution.

T1	T2
R(A)	
	R(A)
W(A)	
	W(A)

# LET'S START WITH DBMS :).

## View-Serializability

Conditions for View Equivalent schedule

For a schedule to be view-Equivalent it should follow the following conditions

### 1. Initial Read Values:

- An initial read of both schedules must be identical. For example, consider two schedules, S and S'. If, in schedule S, transaction T1 reads the data item A, then in schedule S', transaction T1 should also read A.

T1	T2	T3
R(A)		
	W(A)	
		W(B)

S

T1	T2	T3
R(A)	W(A)	
		W(B)

S'

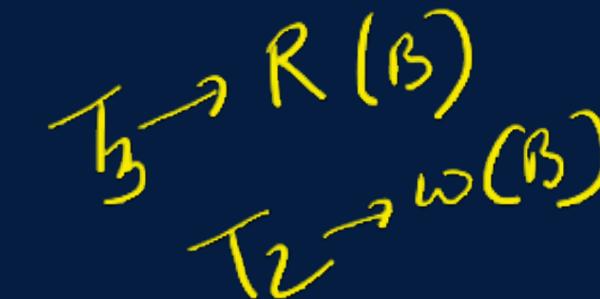
# LET'S START WITH DBMS :).

## View-Serializability

Conditions for View Equivalent schedule

### 2. Updated Read:

- In schedule S, if  $T_i$  is reading B which is updated by  $T_j$  then in  $S'$  also,  $T_i$  should read B which is updated by  $T_j$ .



T1	T2	T3
	w(B)	
		R(B)
R(A)		

S

T1	T2	T3
R(A)		
	w(B)	
		R(B)

S'

# LET'S START WITH DBMS :).

## View-Serializability

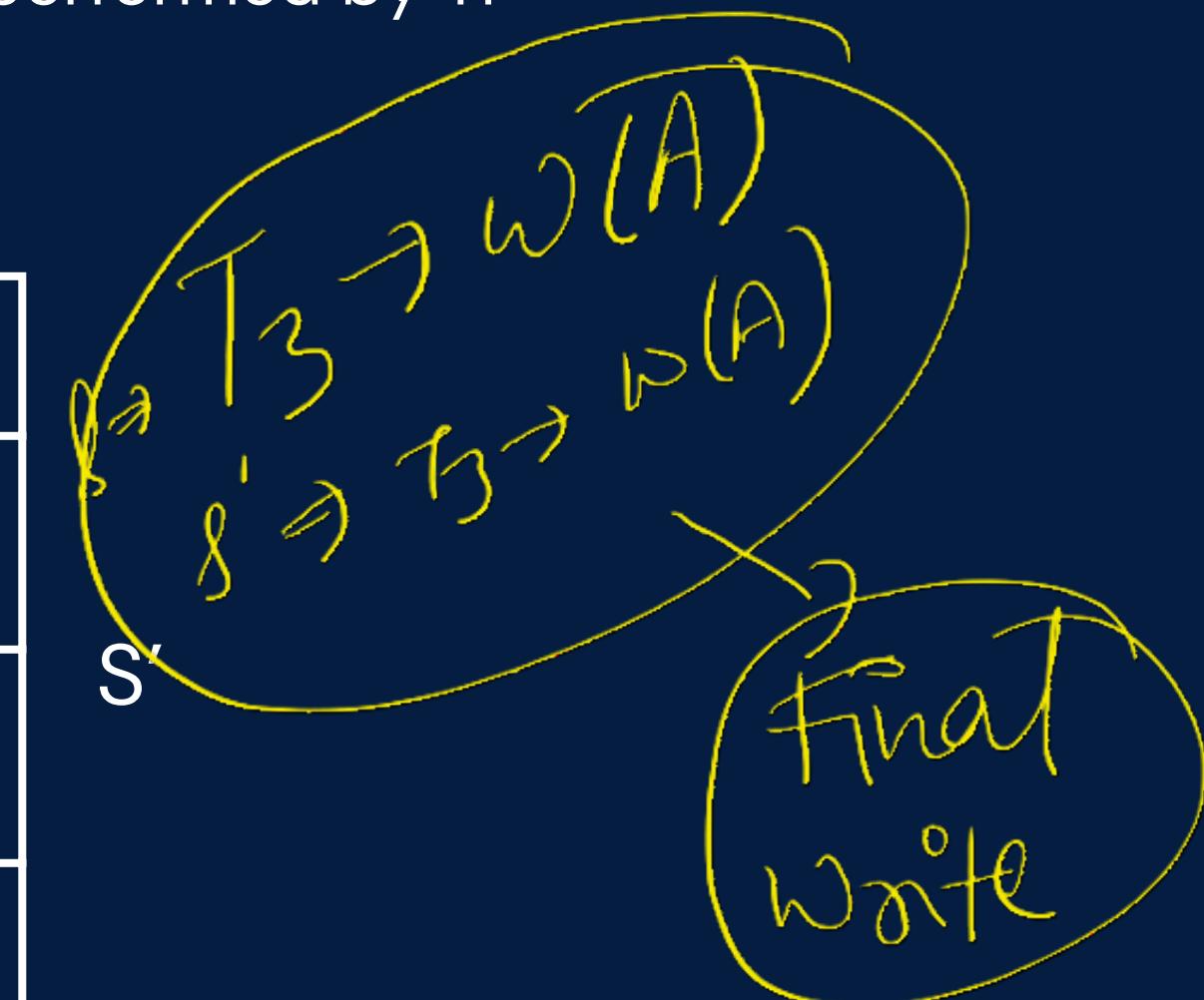
Conditions for View Equivalent schedule

### 3. Final Update

- A final write must be identical in both schedules. If, in schedule S, transaction  $T_i$  is the last to update A, then in schedule  $S'$ , the final write operation on A should also be performed by  $T_i$

T1	T2	T3
W(A)		
	R(A)	
		W(A)

T1	T2	T3
		R(A)
W(A)		
		W(A)



# LET'S START WITH DBMS :).

## View-Serializability

The number of possible serial schedules for n transactions is given by the number of permutations of the transactions:  $n!$

**Q.Find the view equivalent schedule**

T1	T2	T3
R(A)		
	W(A)	
W(A)		
		W(A)

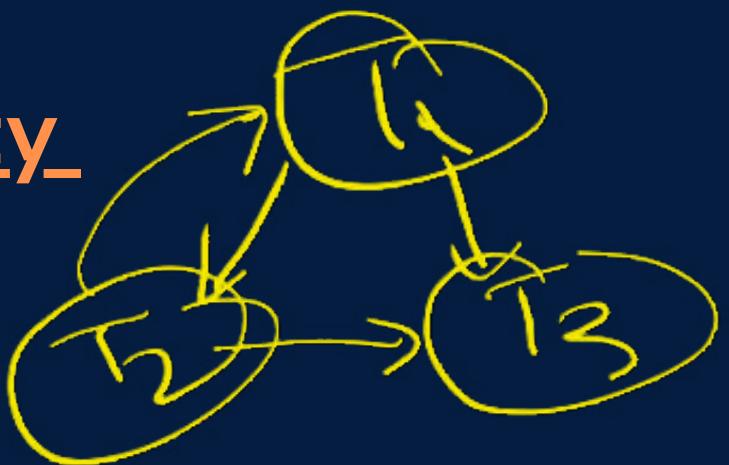
# LET'S START WITH DBMS :).

## View-Serializability

**Step 1:** Find if conflict serializable or not.

**Step 2:** Find the possible serial schedules  $\rightarrow 3!$

**Step 3:** Choose one possibility and check for view equivalent conditions ( $T_1 \rightarrow T_2 \rightarrow T_3$ )



T1	T2	T3
R(A)		
	W(A)	
W(A)		
		W(A)

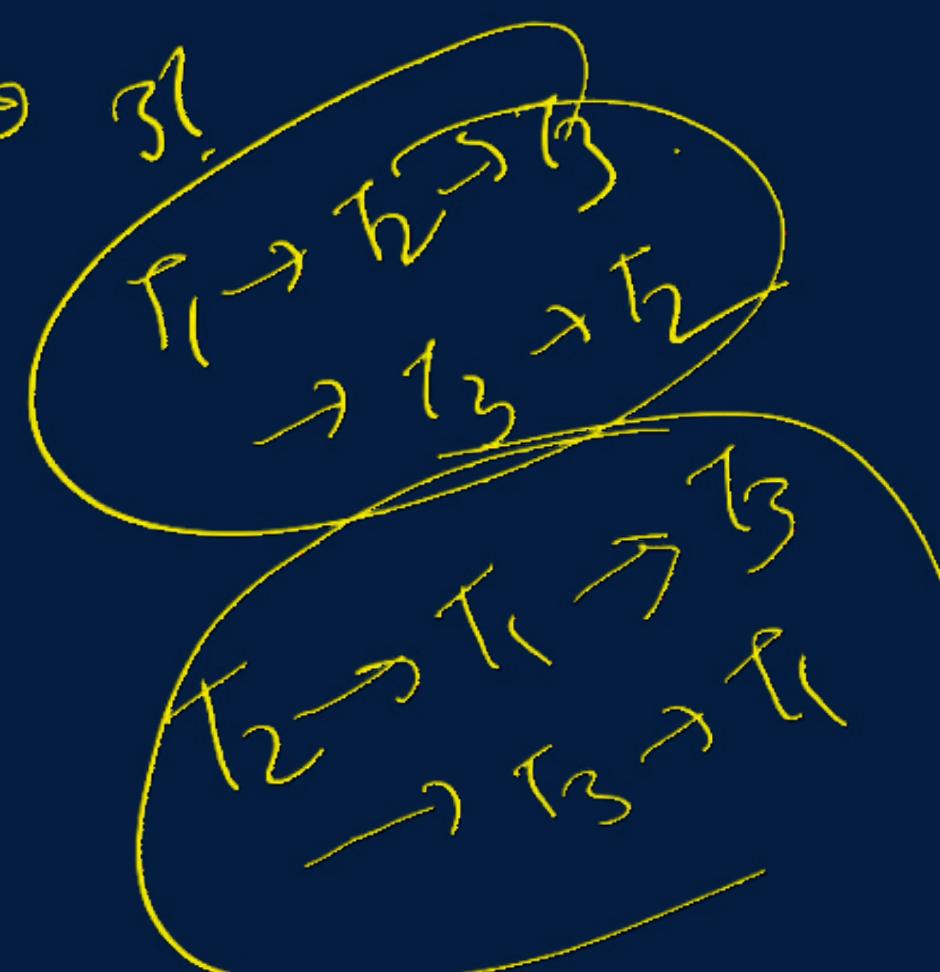
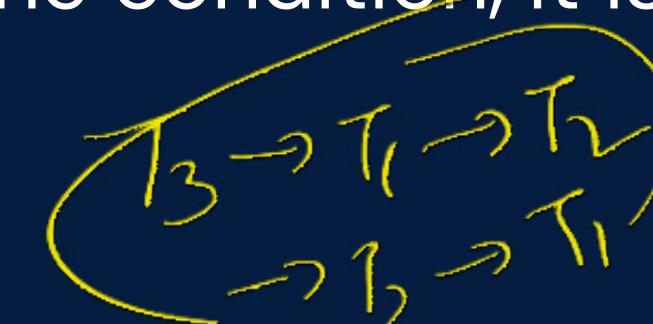
1. Initial Read  $\rightarrow T_1$

2. Update Read  $\rightarrow$  No read performed after update so no need to check

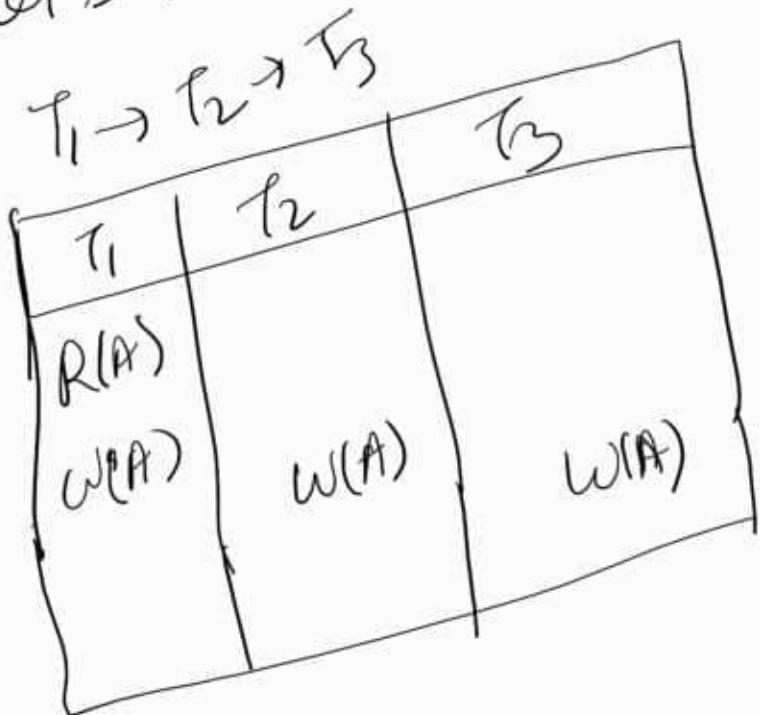
3. Final Update  $\rightarrow T_3$

**Step 4:** If the view equivalent schedule matches the condition, it is view serializable.

~~cycle exists~~ view equivalent  
check here  $\rightarrow$  3!



let's take



Initial Read  $\rightarrow T_1$  ✓  
update Read  $\rightarrow$  no read  
performed after update  $\rightarrow$  no need  
final update  $\rightarrow T_3$  ✓  
so its view equivalent

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

Concurrency control -> It is a process of managing multiple users access and modification in data simultaneously in shared or multi-user database systems.

How it helps?

1. **Data Consistency**: Ensures that data remains accurate and reliable despite concurrent access.
2. **Isolation**: Maintains the isolation property of transactions, so the outcome of a transaction is not affected by other concurrently executing transactions.
3. **Serializability**: Ensures that the result of concurrent transactions is the same as if the transactions had been executed serially

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

**Concurrency control mechanisms** are techniques used in DBMS to ensure that transactions are executed concurrently without leading to inconsistencies in the database. In serializability we check if a schedule is serializable or not but in concurrency control we use certain techniques to make a schedule serializable.

Why its needed?

- **Lost Updates:** When two or more transactions update the same data simultaneously, one of the updates might be lost. For example, if two users modify the same record at the same time, the changes made by one user could overwrite the changes made by the other. (WW)
- **Dirty Reads:** When a transaction reads data that has been modified by another transaction but not yet committed. If the first transaction rolls back, the other transaction will have read invalid data. (WR)

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

- **Uncommitted Dependency (Dirty Writes):** When a transaction modifies data that has been read by another transaction, leading to inconsistencies if one of the transactions rolls back.
- **Inconsistent Retrievals (Non-repeatable Reads):** Happens when a transaction reads the same data multiple times and gets different values each time because another transaction is modifying the data in between the reads.
- **Phantom Reads:** Occurs when a transaction reads a set of rows that satisfy a condition, but another transaction inserts or deletes rows that affect the set before the first transaction completes. This results in the first transaction reading different sets of rows if it re-executes the query.

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

Some common concurrency control mechanisms are :

- **Lock-Based Protocols**(2PL, Strict 2PL, Rigorous 2PL , Conservative 2PL)
- **Timestamp-Based Protocols** (Basic Timestamp Ordering (TO))
- **Optimistic Concurrency Control (OCC)**
- **Multiversion Concurrency Control (MVCC)**

# LET'S START WITH DBMS :).

## Database recovery management

It involves strategies and processes to restore a database to a consistent state after a failure or crash.

### Types of Database Failures

- **Transaction Failure:** Occurs when a transaction cannot complete successfully due to logical errors or system issues (like deadlocks).
- **System Failure:** Occurs when the entire system crashes due to hardware or software failures, leading to loss of in-memory data.
- **Media Failure:** Occurs when the physical storage (e.g., hard drives) is damaged, resulting in data loss or corruption.

# LET'S START WITH DBMS :).

## Database recovery management

### Recovery Phases

- **Analysis Phase:** Identifies the point of failure and the transactions that were active at that time.
- **Redo Phase:** Reapplies changes from committed transactions to ensure the database reflects all completed operations.
- **Undo Phase:** Reverts the effects of incomplete transactions to maintain consistency.



# LET'S START WITH DBMS :).

## Database recovery management

### Recovery Techniques

- **Backup and Restore:** Regular backups are taken to ensure data can be restored. Full, incremental, and differential backups are common types.
- **Logging:** Keeps a record of all transactions. The Write-Ahead Logging (WAL) protocol ensures that logs are written before any changes are applied to the database.
- **Shadow Paging:** Maintains two copies of the database pages; one is updated, and the other remains unchanged until the transaction commits.

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

Some common concurrency control mechanisms are :

- **Lock-Based Protocols**(2PL, Strict 2PL, Rigorous 2PL , Conservative 2PL)
- **Timestamp-Based Protocols** (Basic Timestamp Ordering (TO))
- **Optimistic Concurrency Control (OCC)**
- **Multiversion Concurrency Control (MVCC)**

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

- **Lock-Based Protocols**

### Types of Locks

a. **Binary Locks**: A simple mechanism where a data item can be either locked (in use) or unlocked. If a thread tries to acquire the lock when it's already locked, it must wait until the lock is released by the thread currently holding it.

### b. Shared and Exclusive Locks

- **Shared Lock (S-lock)**: Allows multiple transactions to read a data item simultaneously but prevents any of them from modifying it. Multiple transactions can hold a shared lock on the same data item at the same time.
- **Exclusive Lock (X-lock)**: Allows a transaction to both read and modify a data item. When an exclusive lock is held by a transaction, no other transaction can read or modify the data item.

T1	T2
X(A)	
R(A)	
W(A)	
U(A)	
	S(B)
	R(B)
	U(B)

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

**Note :** When a transaction acquires a shared lock on a data item, other transactions can also acquire shared locks on that same item, enabling concurrent reads. However, no transaction can acquire an exclusive lock on that item as long as one or more shared locks are held.

When a transaction acquires an exclusive lock on a data item, it has full control over that item, meaning it can both read and modify it. No other transaction can acquire a lock on the same data item until the exclusive lock is released.

Shared lock -> any no of transactions

Exclusive lock -> only one transaction should hold it at one time

While shared and exclusive locks are vital for maintaining data integrity and consistency in concurrent environments, they can introduce significant challenges in terms of performance, deadlocks, reduced concurrency, and system complexity.

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

Drawbacks of shared-exclusive locks

**Performance issues** : Managing locks requires additional CPU and memory resources. The process of acquiring, releasing, and managing locks can introduce significant overhead

**Concurrency issues** : Exclusive locks prevent other transactions from accessing locked data, which can significantly reduce concurrency.

**Starvation**: Some transactions may be delayed if higher-priority transactions consistently acquire locks before them, leading to starvation where a transaction never gets to proceed.

**Deadlocks** : Shared and exclusive locks can lead to deadlocks, where two or more transactions hold locks that the other transactions need.

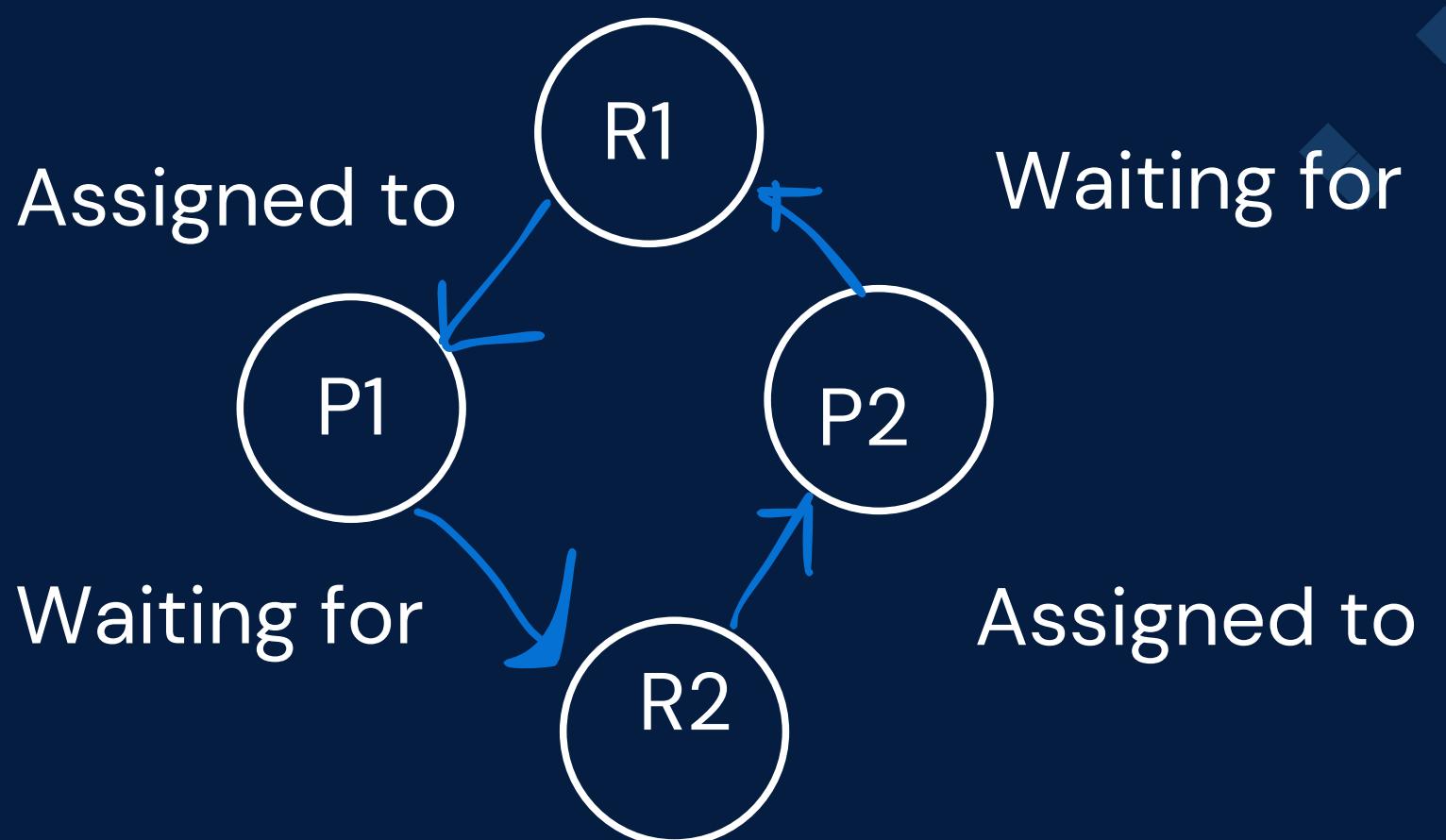
**Irrecoverable** : If Transaction B commits after the lock is released based on a modified value in transaction A which fails after sometime.

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

**Deadlock** : It is a situation when 2 or more transactions wait for one another to give up the locks.

T1	T2
X(A)	
R(A)	
W(A)	
	X(B)
	R(B)
	W(B)
X(A)	
R(A)	
W(A)	
	X(B)
	R(B)
	W(B)



# LET'S START WITH DBMS :).

## Concurrency control mechanisms

**Two-Phase Locking (2PL)**: This protocol ensures serializability by dividing the execution of a transaction into two distinct phases

- **Growing Phase**: A transaction can acquire locks but cannot release any. This phase continues until the transaction has obtained all the locks it needs.
- **Shrinking Phase**: After the transaction releases its first lock, it can no longer acquire any new locks. During this phase, the transaction releases all the locks it holds.

T1	T2
X(A)	
R(A)	
W(A)	
	R(A)
S(B)	
R(B)	
U(A)	
U(B)	

Any transaction which is following 2PL locking achieves serializability and consistency.

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

### Two-Phase Locking (2PL)

#### Advantages :

1. It guarantees that the schedule of transactions will be serializable, meaning the results of executing transactions concurrently will be the same as if they were executed in some serial order.
2. By ensuring that transactions are serializable, 2PL helps maintain data integrity and consistency, which is critical in environments where data accuracy is essential.

#### Disadvantages :

1. Deadlocks, starvation and cascading rollbacks
2. Transactions must wait for locks to be released by other transactions. This can lead to increased waiting times and lower system throughput.
3. In case of a system failure, recovering from a crash can be complex

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

### Strict Two-Phase Locking (Strict 2PL):

A stricter variant where exclusive locks are held until the transaction commits or aborts. This helps prevent cascading rollbacks (where one transaction's rollback causes other transactions to roll back).

### Advantages:

- Prevents Cascading Aborts
- Ensures Strict Serializability

T1	T2
X(A)	
R(A)	
W(A)	
	R(A)
Commit	
U(A)	

### Disadvantages:

- Since write locks are held until the end of the transaction, other transactions may be blocked for extended periods
- Transactions may experience longer wait times to acquire locks
- Deadlocks and starvation is there

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

### **Rigorous Two-Phase Locking:**

An even stricter version where all locks (both shared and exclusive) are held until the transaction commits. This guarantees strict serializability.

### **Advantages:**

- Since all locks are held until the end of the transaction, the system can easily ensure that transactions are serializable and can be recovered
- Prevents Cascading Aborts and Dirty Reads
- 

### **Disadvantages:**

- Performance bottlenecks
- Increased Transaction Duration
- Deadlocks and starvation is there

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

T1	T2
R(A)	
W(B)	
	W(A)
	R(B)

### **Conservative Two-Phase Locking:**

Conservative Two-Phase Locking(Static Two-Phase Locking) is a variant of the standard 2PL protocol that aims to prevent deadlocks entirely by requiring a transaction to acquire all the locks it needs before it begins execution.

If the transaction is unable to acquire all the required locks (because some are already held by other transactions), it waits and retries. The transaction only starts execution once it has successfully acquired all the necessary locks.

Since a transaction never starts executing until it has all the locks it needs, deadlocks cannot occur because no transaction will ever hold some locks and wait for others

In this scenario, deadlocks cannot occur because neither T1 nor T2 starts execution until it has all the locks it needs.

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

### **Timestamp-Based Protocols: It assign a unique timestamp**

- Every transaction is assigned a unique timestamp when it enters the system. It is used to order the transactions based on their Timestamps.
- There are two important timestamps for each data item:
  - **Read Timestamp (RTS):** The last timestamp of any transaction that has successfully read the data item.
  - **Write Timestamp (WTS):** The last timestamp of any transaction that has successfully written the data item.

Transaction with smaller timestamp(Old) → Transaction with larger timestamp(young)

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

**Timestamp-Based Protocols:** It assign a unique timestamp

**Rules :** Consider if there are transactions performed on data item A.

R\_TS(A) -> Read time-stamp of data-item A.

W\_TS(A)-> Write time-stamp of data-item A.

1. Check the following condition whenever a transaction  $T_i$  issues a Read (X) operation:

- If  $W\_TS(A) > TS(T_i)$  then the operation is rejected. (rollback  $T_i$ )
- If  $W\_TS(A) \leq TS(T_i)$  then the operation is executed. (set  $R\_TS(A)$  as the max of  $(R\_TS(A), TS(T_i))$ )

2. Check the following condition whenever a transaction  $T_i$  issues a Write(X) operation:

- If  $TS(T_i) < R\_TS(A)$  then the operation is rejected. (rollback  $T_i$ )
- If  $TS(T_i) < W\_TS(A)$  then the operation is rejected and  $T_i$  is rolled back otherwise the operation is executed. Set  $W\_TS(A)=TS(T_i)$

# LET'S START WITH DBMS :).

## Concurrency control mechanisms

### Timestamp-Based Protocols: It assign a unique timestamp

Let's assume there are two transactions T1 and T2. Suppose the transaction T1 has entered the system at 7:00PM and transaction T2 has entered the system at 7:05pm then T1 has the higher priority, so it executes first as it is entered the system first. T1->T2

If younger has done a Read/Write and older wants to do Read/Write, that's a rollback case.



# LET'S START WITH DBMS :).

## Query Optimization

Whenever we want to improve the efficiency of a query, make it execute faster and consume fewer resources we use query optimization techniques.

Optimizing SQL queries is essential for improving the performance of database applications

**1. Use Indexes Efficiently:** Index the columns frequently used in WHERE, JOIN, ORDER BY, and GROUP BY clauses to speed up data retrieval.

**2. Select Only Necessary Columns :** Avoid SELECT \*, specify only the columns you need in your query. This reduces the amount of data transferred and processed.

# LET'S START WITH DBMS :).

## Query Optimization

3. **Optimize JOIN Operations** : Choose the Right JOIN Type and use indexed columns in Join conditions.
4. **Partition Large Tables**: For very large tables, consider partitioning them to improve query performance. Partitioning allows the DBMS to scan only relevant partitions, reducing I/O and improving response times.
5. **Cache results** : Cache the results of frequently executed queries to avoid redundant calculations.

# LET'S START WITH DBMS :).

## Physical Storage And File Organization

### Physical Storage

Storage device : For storing the data, there are different types of storage options available.

- Primary Storage -> Main memory and cache (fastest and expensive and as soon as the system leads to a power cut or a crash, the data also get lost. )
- Secondary Storage -> Flash Memory and Magnetic Disk Storage (non-volatile) save and store data permanently.
- Tertiary Storage-> Optical disk and Magnetic tape (used for data backup and storing large data offline)

# LET'S START WITH DBMS :).

## Physical Storage And File Organization

Databases store data in files on disk. Each table or index may correspond to one or more files. Each file has sequence of records.

Accessing data from RAM is much faster than accessing data from a hard disk or SSD.

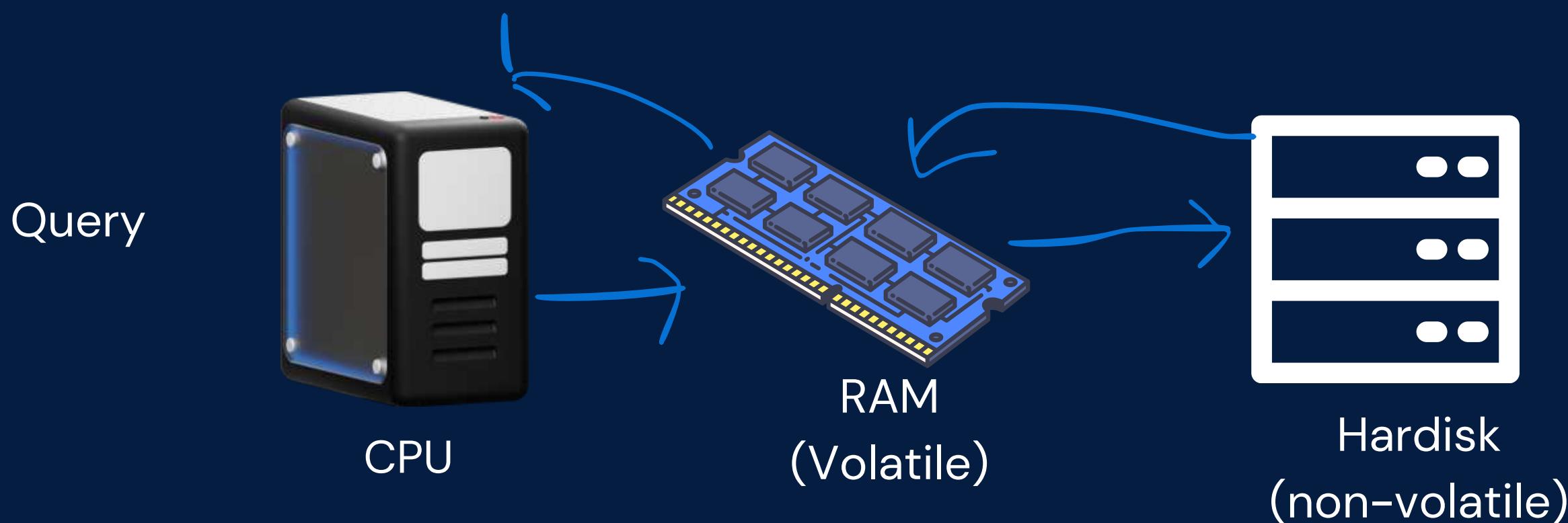
This is because RAM is designed for high-speed read and write operations. Data retrieval in RAM typically involves fetching data in nanoseconds.

- **RAM** : Provides fast, volatile memory for quick data retrieval. Data is accessed via addressable locations and cache memory enhances speed.
- **Hard Disk**: Provides slower, non-volatile storage with mechanical components that impact access speed. Data retrieval involves moving the read/write head and waiting for disk rotation.

# LET'S START WITH DBMS :).

## Physical Storage And File Organization

When a program or application needs to access data, it first checks if the data is available in RAM. If the data is not in RAM, it must be loaded from a slower storage device like a hard disk or SSD. Index files in a Database Management System (DBMS) are stored on disk but are also managed in RAM to optimize performance



# LET'S START WITH DBMS :).

## Physical Storage And File Organization

How file is organized?

Rollno	Sname	SAge

DB



HARDISK

No of records stored in each block= Size of block/size of record

Records can be stored in 2 ways-> sorted(seraching is fast) or unsorted(searching is slow, insertion is fast)

Files are allocated on the hard disk in contiguous blocks to reduce fragmentation.

# LET'S START WITH DBMS :).

## Indexing and its types

**Why do we need indexing?**

**Imagine you have a 1,000-page textbook and you want to search a topic "xyz".**

- When indexing is not present-> You need to manually search for each term in the book, which could be frustrating and time-consuming.
- When indexing is present->The index allows you to quickly locate each topic. You can look up "xyz," find that it's covered on pages 448-490. This makes it more efficient and less time-consuming.

Indexing is a critical technique in database management that significantly improves the performance of query operations by minimizing the no of disk access. Index table is always sorted

# LET'S START WITH DBMS :).

## Indexing and its types

### Why do we need indexing?

- In the same way how we have used indexing in book , it helps us to find specific records or data entries in a large table or dataset.

Search key	Data access
It is used to find the record	It contains the address where the data item is stored in memory for the provided search key

It helps in finding what a user is looking for  
(P.K OR C.K)

### Index table

It tell where the record is stored in the memory  
(set of pointer holding address of block)

# LET'S START WITH DBMS :).

## Indexing and its types

### How it helps?

- **Improved Query Performance:** Indexes allow the database management system (DBMS) to locate and retrieve data much more quickly than it could by scanning the entire table.
- **Indexes can be used to optimize queries that sort data (ORDER BY) or group data (GROUP BY) :** When an index is available on the columns being sorted or grouped, the DBMS can retrieve the data in the desired order directly from the index, eliminating the need for additional sorting operations.
- We have certain indexing methods like Dense(index entry for all the records) and Sparse(index entries for only a subset of records) index.

# LET'S START WITH DBMS :)

## Indexing and its types

**Sparse Index:** A sparse index is a type of database indexing technique where the index does not include an entry for every single record in the database. Instead, it contains entries for only some of the records, typically one entry per block (or page) of data in the storage. This makes sparse indexes smaller and faster to search through compared to dense indexes, which have an entry for every record.

It is used when we have an ordered data set.

EX: If a table has 1,000 rows divided into 100 blocks, and you create a sparse index, the index might only have 100 entries, with each entry pointing to the first record in each block.

## ◆ Index table

Search key	data ref
1	B1
4	B2

Sparse Index

no of records in index  
file=no of blocks

1 Ram	
2 Shyam	
3	
4	B1
5	
6	

disk

RollNo	Name
1	Ram
2	Shyam
3	Raghu
4	Riti
5	Raj
6	Rahul

## How Sparse Indexing Works:

- **Primary Index:** Sparse indexes are often used with primary indexes, where the table is sorted on the indexed column. The sparse index only includes an entry for the first record in each block or page.
- **Searching:** When searching for a specific value, the database uses the sparse index to quickly locate the block where the record might reside. It then searches within the block to find the exact record.

Index table

Search key	data ref
1	B1
4	B2

Sparse Index

no of records in index  
file=no of blocks

1	B1
2	
3	B2
4	
5	

disk

RollNo	Name
1	Ram
2	Shyam
3	Raghav
4	Riti
5	Raj
6	Rahul

## Disadvantages of Sparse Indexes

- **Additional I/O Operations:** After using the index to find the correct block, an additional search within the block is required to find the specific record.
- **Less Efficient for Random Access:** If queries often need to retrieve random, non-sequential records, a dense index might be more efficient.

## Use Cases:

- **Primary Indexing on Large Tables:** Sparse indexes are ideal for primary indexes on large tables where records are sequentially stored.
- **Range Queries:** Sparse indexes can be effective for range queries where the query needs to retrieve a range of records rather than a single record.

# LET'S START WITH DBMS :)

## Indexing and its types

**Dense Index:** A dense index is a type of indexing technique used in databases where there is an index entry for every single record in the data file. This means that the index contains all the search keys and corresponding pointers (or addresses) to the actual data records, making it highly efficient for direct lookups.

It is used when we have an un-ordered data set.

EX : Consider a table with 1,000 rows, and you create a dense index on a non-primary key column. The dense index will have no ofunique non key records, each pointing to a specific row in the table.

These are useful in scenarios where random access to individual records is quiet frequent .

Search key	data ref
18	B1
19	B1
20	B2
21	B2

Index table

18 Ram	B1
18. Shyam	
19	
20	B2
20	
21	

### Dense Index

no of records in index  
 file=no of blocks unique non  
 key records

Age	Name
18	Ram
18	Shyam
19	Raghu
20	Riti
20	Raj
21	Rahul

## How Dense Indexing Works:

In a dense index, each index entry includes:

- **A search key** (the value of the indexed column).
- **A pointer** (or address) to the actual record in the data file.

When a query searches for a specific value, the database uses the dense index to quickly locate the corresponding record.

## Disadvantages of Dense Indexes

- **Storage Intensive:** Requires significant storage space because it maintains an entry for every record in the table.
- **Maintenance Overhead:** Inserting, updating, or deleting records requires updating the dense index, which can be costly in terms of performance, especially for large tables.

## Use Cases:

- **Primary Indexing:** Dense indexes are often used for primary indexing when the table is small to medium-sized, and quick access to individual records is required.
- **Exact Match Queries:** Ideal for situations where queries frequently request individual records based on an exact key match.

Search key	data ref
18	B1
19	B1
20	B2
21	B2

Index table

18 Ram 18. Shyam 19	B1
20 20 21	B2

Dense Index  
no of records in index  
file=no of blocks unique non  
key records

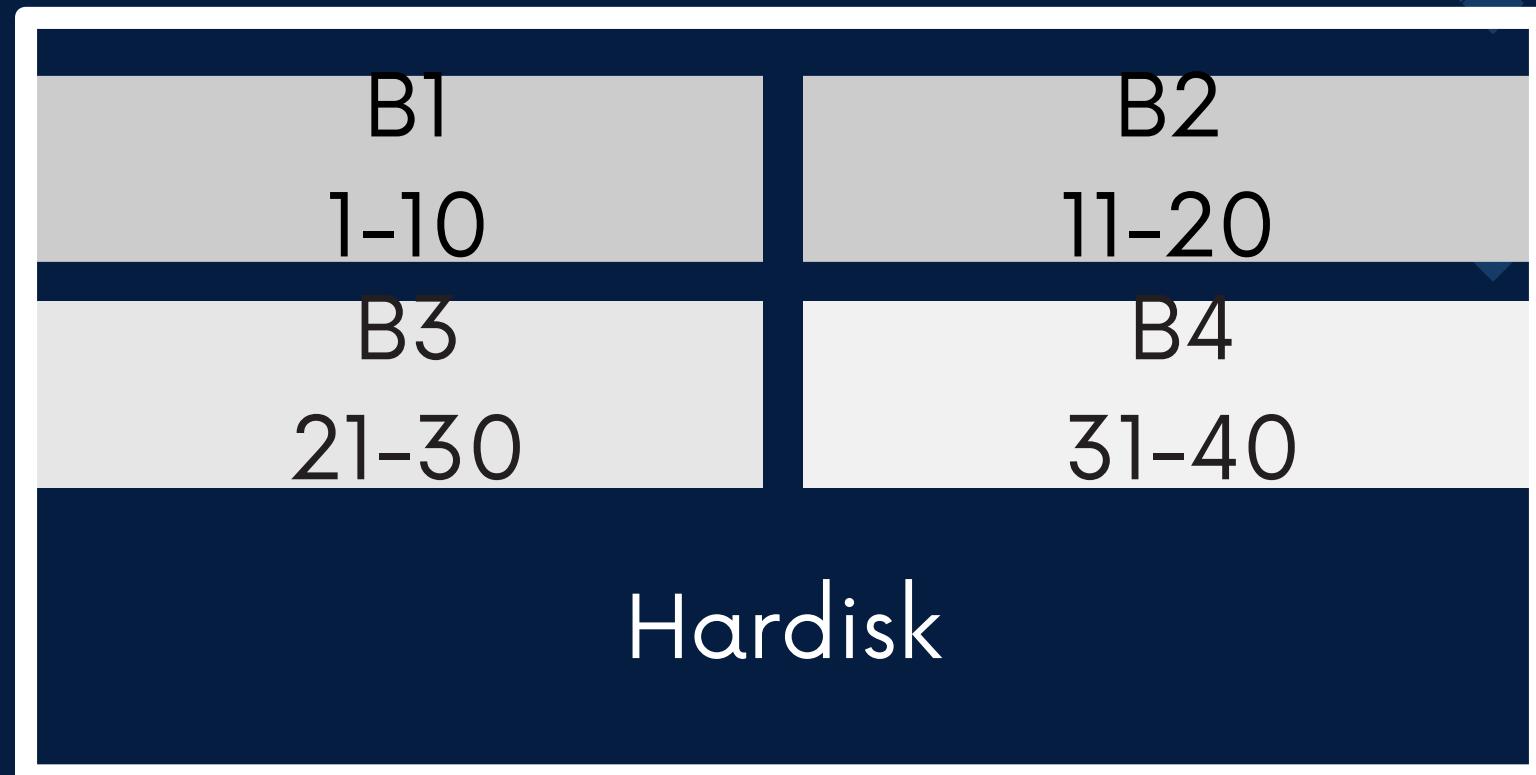
Age	Name
18	Ram
18	Shyam
19	Raghav
20	Riti
20	Raj
21	Rahul

# LET'S START WITH DBMS :).

## Indexing and its types

Index table

Search key	data ref
1	B1
11	B2
21	B3



When a query is run, the database engine checks the index to find the pointers to the rows that contain the desired data. It then retrieves these rows directly, without scanning the entire table.

# LET'S START WITH DBMS :).

## Indexing and its types

How indexing works?

It takes a search key as input and then returns a collection of all the matching records. Now in index there are 2 columns, the first one stores a duplicate of the key attribute/ non-key attribute(search key) from table while the second one stores pointer which hold the disk block address of the corresponding key-value.

# LET'S START WITH DBMS :).

## Indexing and its types



# LET'S START WITH DBMS :).

## Indexing and its types

Key data: Unique identifiers for each record. Often used to distinguish one record from another.

Non-key data: All other data in the record besides the key.

- **Single-Level Indexing** is straightforward and works well for smaller databases, offering a direct approach to speeding up query performance.
- **Multi-Level Indexing** is more complex but necessary for larger databases, as it effectively manages large indexes by creating additional layers of indexing, thereby improving data retrieval times.

# LET'S START WITH DBMS :).

## Primary Index

It enhances the efficiency of retrieving records by using their primary key values. It establishes an index structure that associates primary key values with disk block addresses. This index is composed of a sorted list of primary key values paired with their corresponding disk block pointers, thereby accelerating data retrieval by reducing search time. It is especially beneficial for queries based on primary keys. It is done on sorted data

Search key	data ref
1	B1
4	B2

It creates an index structure that maps primary key values to disk block addresses.  
Index table

1		
2		
3		B1
4		
5		
6		B2

Sparse Index  
no of records in index file = no of blocks

RollNo	Name
1	Ram
2	Shyam
3	Raghu
4	Riti
5	Raj
6	Rahul

# LET'S START WITH DBMS :).

## Primary Index

Example : In a Users table with a UserID column as the primary key, a primary index on UserID allows for quick retrieval of user records when you know the UserID

The primary index ensures that the database can immediately locate the row associated with a given UserID, making operations like SELECT, UPDATE, and DELETE highly efficient.

# LET'S START WITH DBMS :).

## Cluster Index

The cluster index is generally on a non-key attribute. The data in the table is typically ordered according to the order defined by the clustered index. Mostly each index entry points directly to a row. It is mostly used where we are using GROUP BY. It physically order records in a table based on the indexed columns.

Search key	data ref
18	B1
19	B1
20	B2
21	B2

Index table

18		B1
18		B1
19		
20		B2
20		B2
21		

Dense Index  
no of records in index file = no of blocks unique non key records

Age	Name
18	Ram
18	Shyam
19	Raghu
20	Riti
20	Raj
21	Rahul

# LET'S START WITH DBMS :).

## Cluster Index

Example : In an Employees table, if you often need to produce lists of employees ordered by LastName, creating a clustered index on LastName can make these queries faster.

The clustered index keeps the rows in LastName order, reducing the need for the database to perform additional sorting when executing queries.

# LET'S START WITH DBMS :).

## Multilevel Index

A multilevel index is an advanced indexing technique used in database management systems to manage large indexes more efficiently. When a single-level index becomes too large to fit into memory, multilevel indexing helps by breaking down the index into multiple levels, reducing the number of disk I/O operations needed to search through the index.

- First Level (Primary Index): The first level is the original index, where each entry points to a block of data or a page in the table.
- Second Level (Secondary Index): If the first level index is too large to fit in memory, a second-level index is created. This index points to blocks of the first-level index, effectively indexing the index itself.

# LET'S START WITH DBMS :).

## Multilevel Index

Eid	Pointer
E101	
E201	
E301	

Outer index

Eid	Pointer
E101	B1
E151	B2

Eid	Pointer
E201	B3
E251	B4

Eid	Pointer
E301	B5
E351	B6

Inner index

Eid	Data
E101	
E102	
.	
E151	
E152	
.	
E201	
E202	
.	

# LET'S START WITH DBMS :).

## Secondary Index

Secondary indexing speeds up searches for non-key columns in a database. Unlike primary indexing, which uses the primary key, secondary indexing focuses on other columns. It creates an index that maps column values to the locations where the records are stored, making it faster to find specific records without scanning the entire table.

The data is mostly unsorted and it can be performed on both key and non-key attributes.

# LET'S START WITH DBMS :).

## Secondary Index

1. when search key is key attribute and data is unsorted

Dense Index

no of records in index

file = no of blocks unique  
non key records

Search key	data ref
18	B1
20	B1
22	B2
24	B2
.	.
.	.
.	.

Index table

Age	
18	
20	
45	
67	
68	
	B1
22	
24	
30	
	B2

# LET'S START WITH DBMS :).

## Secondary Index

2. when search key is non-key attribute and data is unsorted

Dense Index

no of records in index

file = no of blocks unique

non key records

Search key	data ref
18	B1
20	B1
22	B1->B2
24	B2
.	.
.	.
.	.
.	.

Index table

Age	
18	
18	B1
20	
67	
22	
22	B2
24	
30	

# LET'S START WITH DBMS :).

## Indexing and its types

- How to create indexes

### Non-Clustered index:

(On one column)

**CREATE INDEX index\_name**

**ON table\_name (column\_name);**

(On multiple column)

**CREATE INDEX index\_name**

**ON table\_name (column\_name1, column\_name2,....)**

- Remove an index

**DROP INDEX index\_name ON table\_name**

### Clustered index:

Automatically created with the primary key. MySQL does not support explicit creation of additional clustered indexes.

# LET'S START WITH DBMS :)

## B and B+ trees

Clustered and non-clustered indexes are concepts that describe how data is stored and accessed in a database, but B-trees (and B+ trees) are the data structures that actually implement these indexes. Knowing B-trees gives you insight into how these indexes work under the hood.

Understanding B-trees helps you understand why certain queries perform well or poorly based on the structure of the index. For example, how a B-tree's balanced nature affects search times or why range queries are efficient with B-tree indexes

B-trees provide the balanced, efficient structure that makes these types of indexes performant, ensuring that operations like search, insert, delete, and update are done in logarithmic time ( $O(\log n)$ ). This makes B-tree indexing crucial for optimizing database queries, whether in clustered or non-clustered index scenarios.

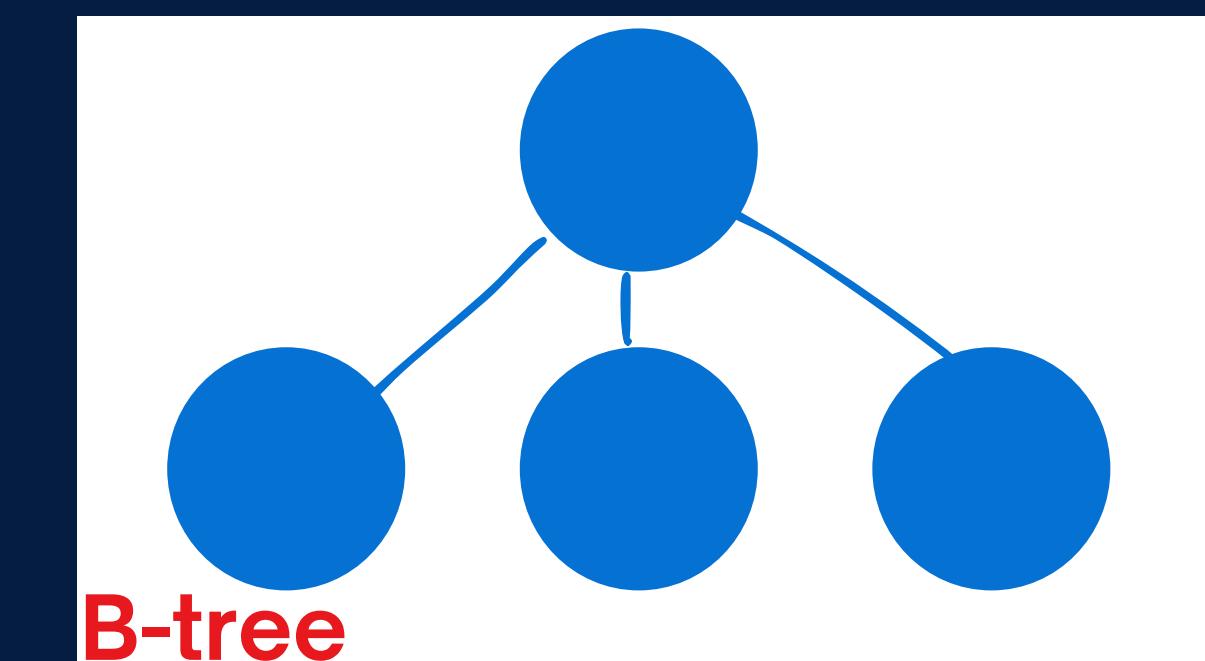
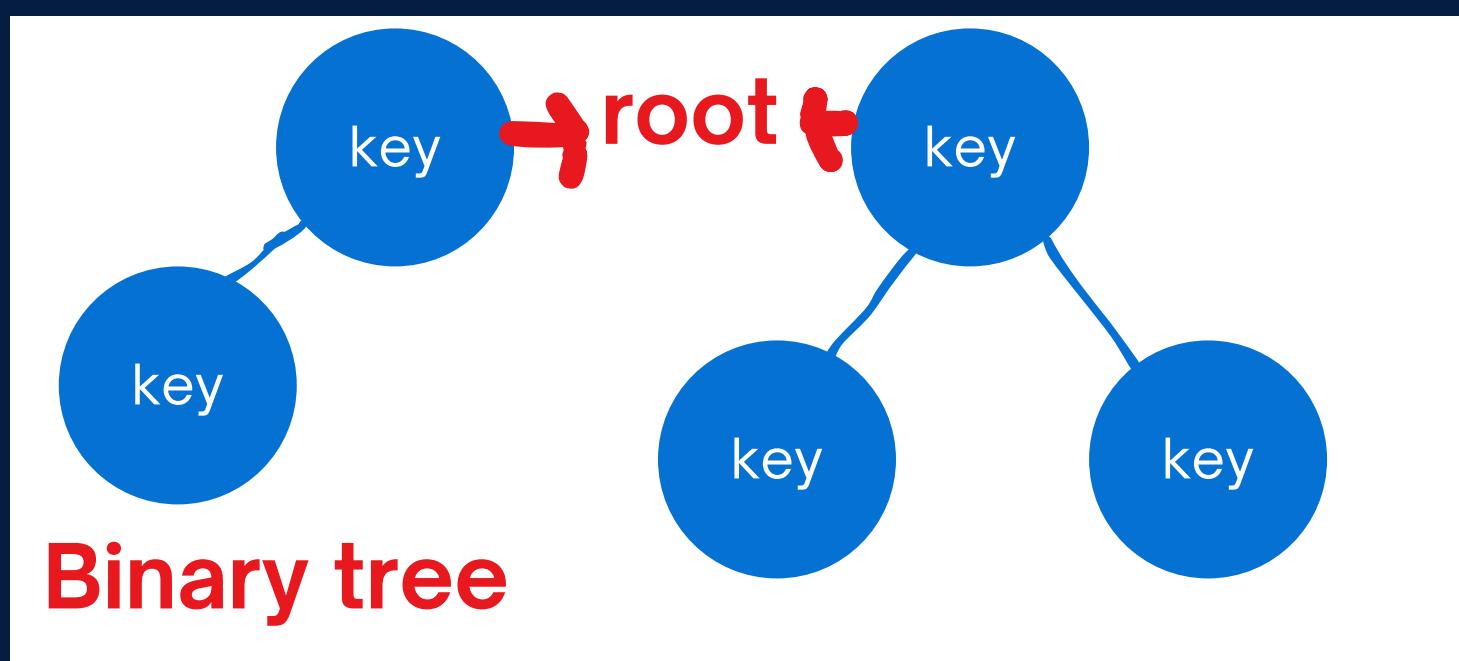
# LET'S START WITH DBMS :)

## B trees(M-way tree)

B-trees are self-balancing tree data structures that maintain sorted data and allow searches, sequential access, insertions, and deletions in logarithmic time. All leaf nodes are at the same level.

In B trees you can have minimum 2 children and max x children.

Now B-tree is a generalisation of Binary Search trees. In BST every node can have atmost 2 children(0,1,2) and only one key



# LET'S START WITH DBMS :)

## B trees

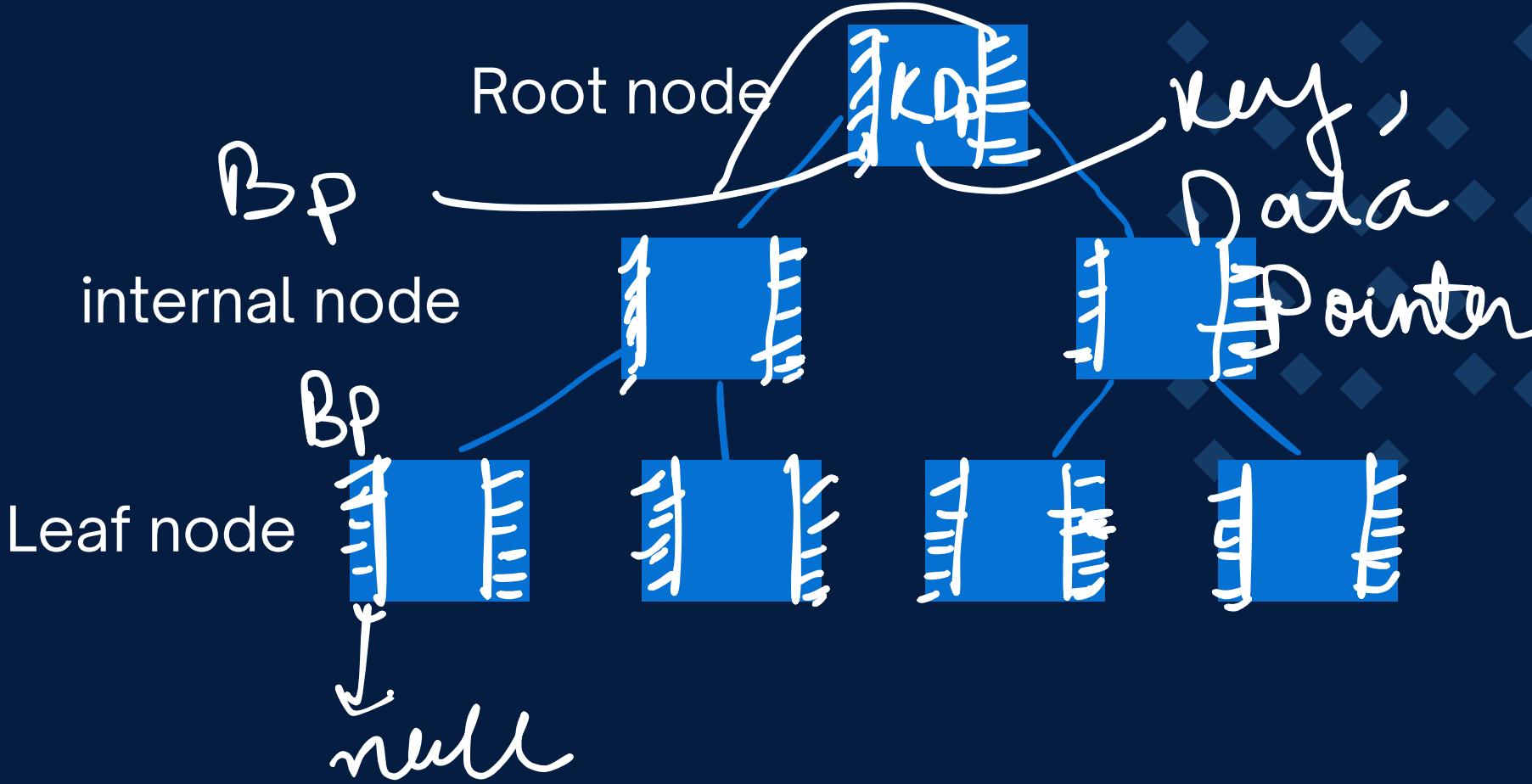
(Rp) Dp -> record/data pointer (where the record is present)

in secondary memory (disk)

(Bp) Tp -> block/tree pointer (links to the children nodes)

### Structure of B-Tree:

- Nodes: A B-tree is composed of nodes, each containing keys and pointers (references) to child nodes. The keys within a node are sorted in ascending order.
- Root, Internal Nodes, and Leaves:
  1. The root node is the topmost node in the tree.
  2. Internal nodes contain keys and child pointers.
  3. Leaf nodes contain keys and possibly pointers to records or other data.
- In indexing, each key in a B-tree node typically represents a value or range of values, and the associated pointer directs to a data block where records corresponding to the key(s) can be found.
- For example, in a database, the key might be a value in a column, and the pointer might direct to the location of a row or a set of rows in a table.



# LET'S START WITH DBMS :)

## B trees

A B-tree of order  $x$  can have:

- **The max no. of children**

For every node  $\rightarrow x$  i.e the order of tree.

- **The max no. of keys**

For every node  $\rightarrow x-1$

- **The min no of keys**

a. root node  $\rightarrow 1$

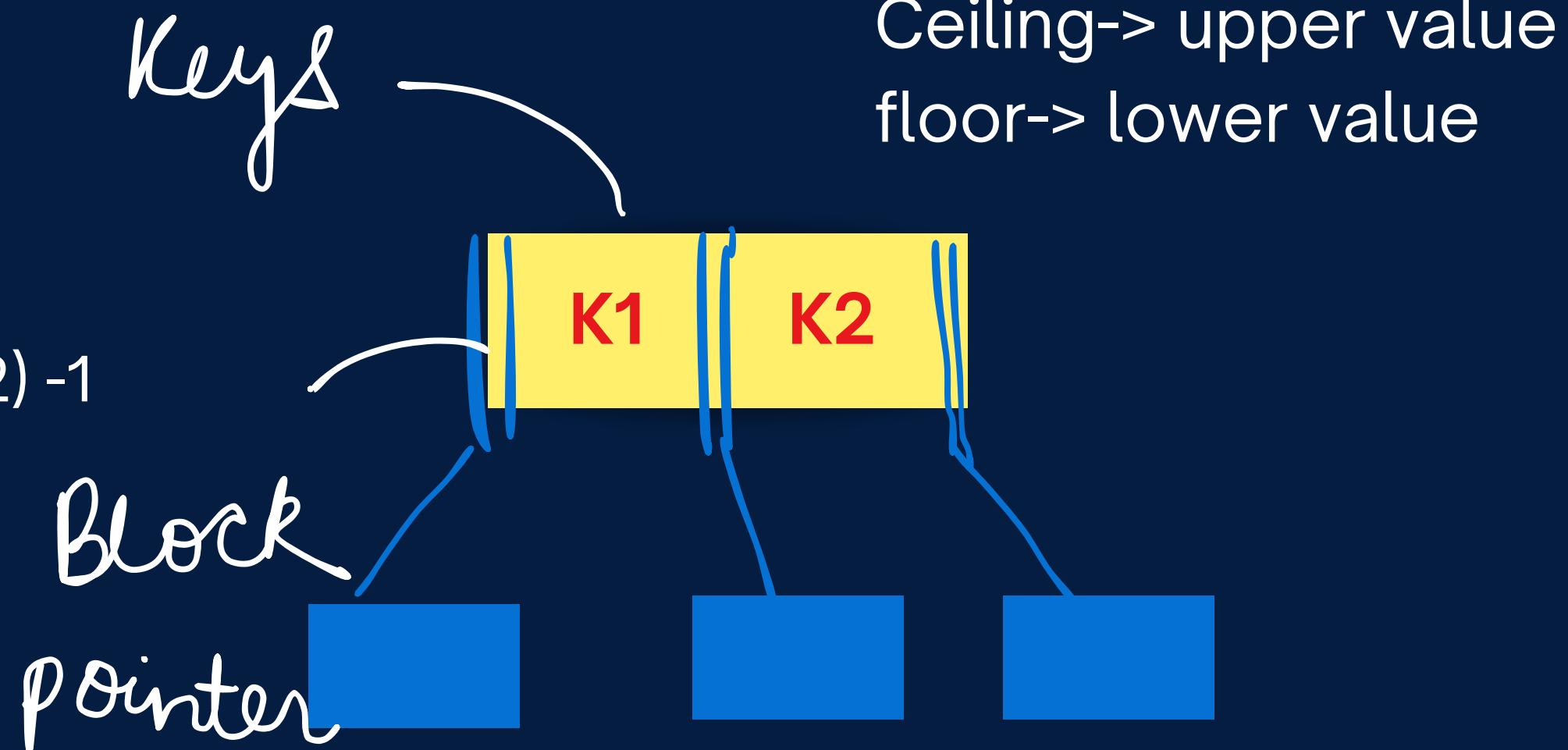
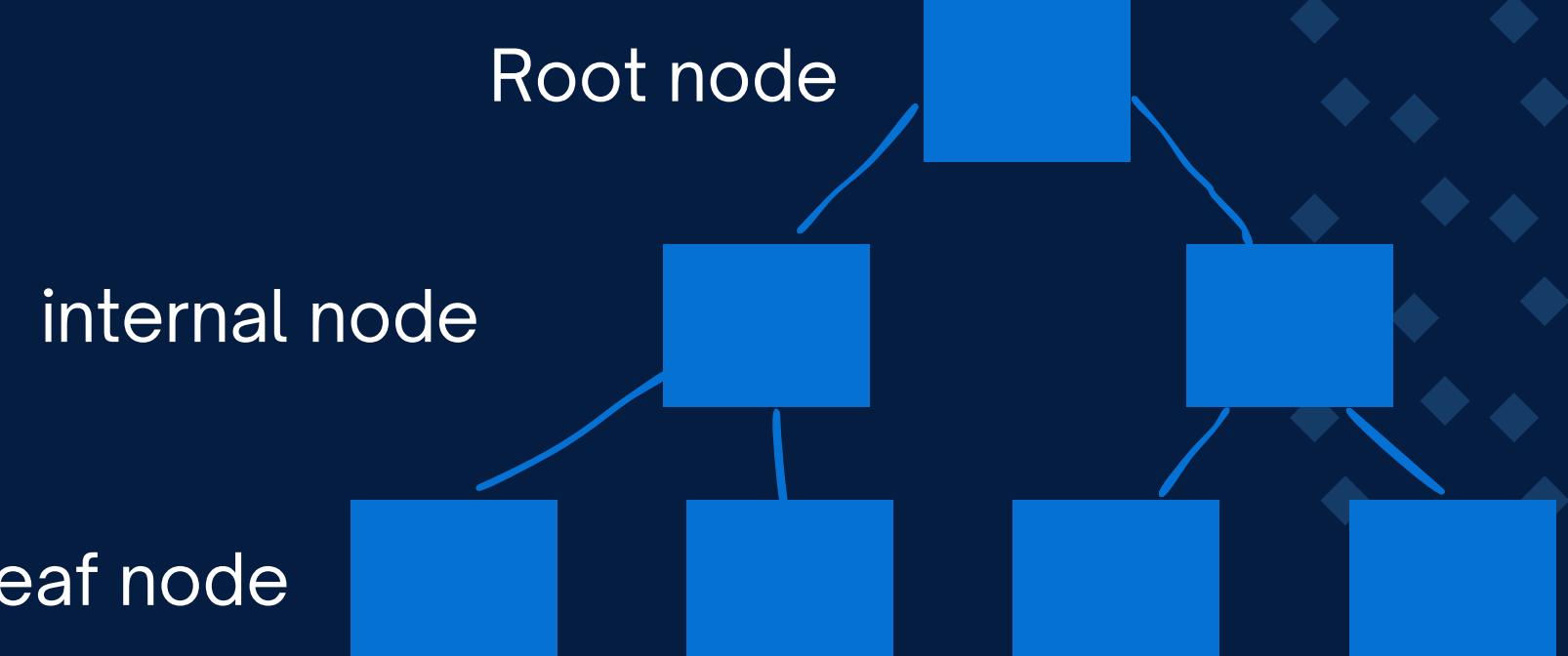
b. other nodes apart from root  $\rightarrow \text{ceiling}(m/2) - 1$

- **The min no. of children**

a. root node  $\rightarrow 2$

b. Leaf node  $\rightarrow 0$

c. internal node  $\rightarrow \text{ceiling}(m/2)$



Ceiling-> upper value  
floor-> lower value

**Insertion in B-tree always happens from leaf node.**

# LET'S START WITH DBMS :)

## B trees

To determine the order of a B-tree when the block size, block pointer size, and data pointer size are given :

$$m \times Pb + (m-1) \times (K+Pd) = B$$

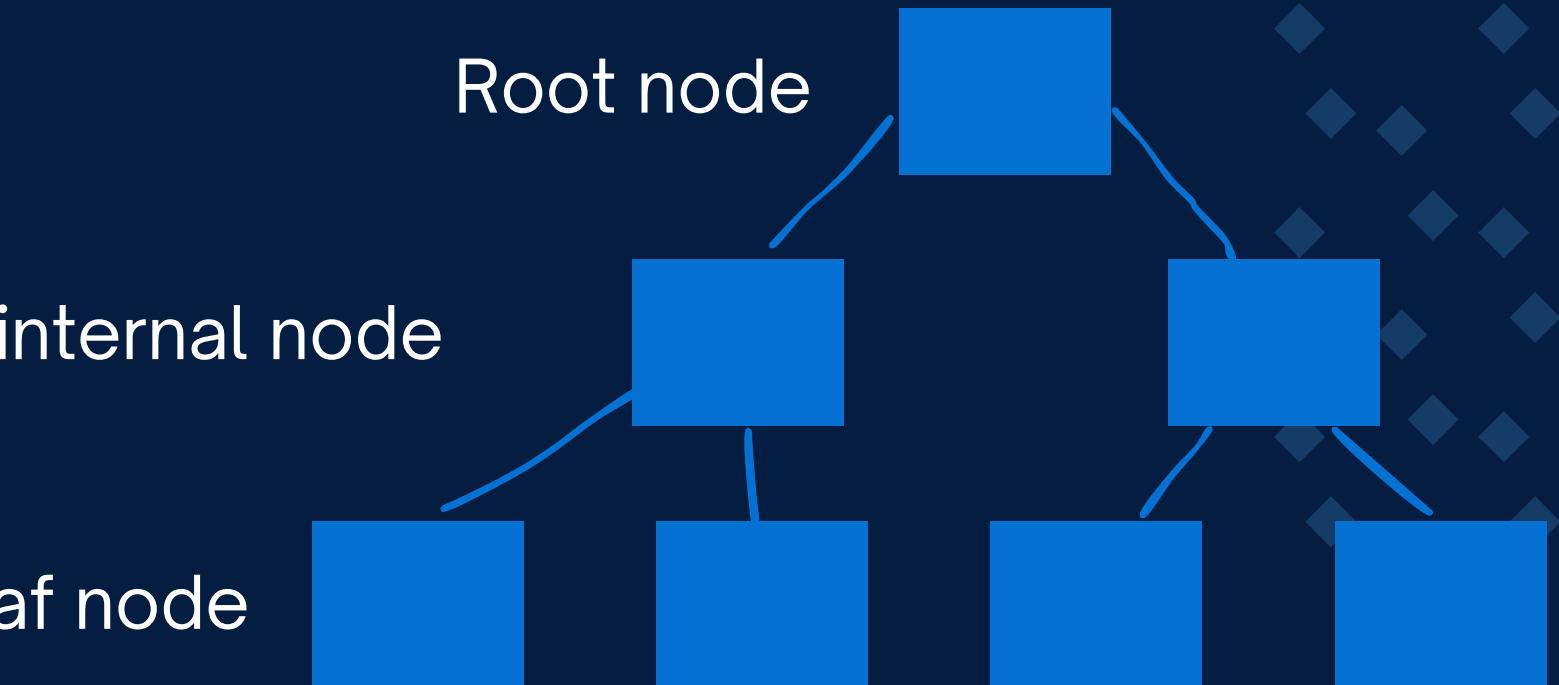
B- Block size

m- Order of tree

Pb- Block pointer size

K- Key size

Pd- Data pointer size



For a B-tree node with m children:

- Number of Keys: A node with m children can have a maximum of m-1 keys.
- Number of Block Pointers: Each node has m block pointers (pointers to child nodes).
- Number of Data Pointers: Each key has an associated data pointer, so there are m-1 data pointers.

# LET'S START WITH DBMS :)

## B trees

Let's say you have the following:

- Block size (B) = 1024 bytes
- Block pointer size (Pb) = 8 bytes
- Data pointer size (Pd) = 12 bytes
- Key size (K) = 16 bytes

Find the order of the tree.

Formula to find order :  $m \times Pb + (m-1) \times (K+Pd) \leq B$

$$m \times 8 + (m-1) \times (16+12) \leq 1024$$

So, the order of the B-tree is  $m = 29$

To determine the order of a B-tree when the block size, block pointer size, and data pointer size are given :

$$m \times Pb + (m-1) \times (K+Pd) \leq B$$

B- Block size

m- Order of tree

Pb- Block pointer size

K- Key size

Pd- Data pointer size

# LET'S START WITH DBMS :)

## Insertion in B-TREE

### Steps to insert values in B-tree

1. Start at the root and recursively move down the tree to find the appropriate leaf node where the new value should be inserted.
2. Insert the value into the leaf node in sorted order. If the leaf node has fewer than the maximum allowed keys (order - 1), this step is simple.
3. If the leaf node contains the maximum number of keys after the insertion, it causes an overflow.  
Split the Node:  
Divide the node into two nodes. The middle key (median) is pushed up to the parent node.
  - The left half of the original node stays in place, while the right half forms a new node.
  - Insert the Median into the Parent:
  - If the parent node also overflows after this insertion, recursively split the parent node and propagate the median up the tree.
4. If the root node overflows (which can happen if it already has the maximum number of keys), split it into two nodes, and the median becomes the new root. This increases the height of the B-tree by one.

A B-tree of order x can have:

- Every node will have max x children i.e the order of tree.
- Every node can have max  $(x-1)$  keys
- The min no of keys
  - a. root node  $\rightarrow 1$
  - b. other nodes apart from root  $\rightarrow \text{ceiling}(m/2) - 1$
- Min children
  - a. root node  $\rightarrow 2$
  - b. Leaf node  $\rightarrow 0$
  - c. internal node  $\rightarrow \text{ceiling}(m/2)$

# LET'S START WITH DBMS :)

## Insertion in B-TREE

Create a B-tree of Order 3 and insert values from 1,4,6,8,10,12,14,16

Max children= order of tree=3

Max keys node can have=order-1=3-1=2

Insertion in B-tree happens from leaf node and values are also inserted in sorted order. The element in left of root would be less than root and the element in right would be greater than root

**Step 1:** Start at the root and recursively move down the tree to find the appropriate leaf node where the new value should be inserted. Insert the value into the leaf node in sorted order. If the leaf node has fewer than the maximum allowed keys (order - 1), this step is simple.

1

4

Max children= order of tree=3

Max keys node can have=order-1=3-1=2

# LET'S START WITH DBMS :)

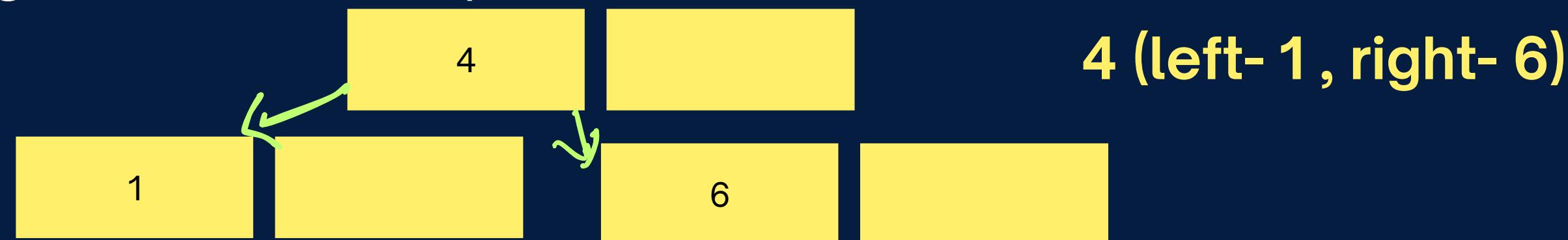
## Insertion in B-TREE

Create a B-tree of Order 3 and insert values from 1,4,6,8,10,12,14,16

Step 2: If the leaf node contains the maximum number of keys after the insertion, it causes an overflow.

Split the Node: Divide the node into two nodes. The middle key (median) is pushed up to the parent node.

- The left half of the original node stays in place, while the right half forms a new node.
- Insert the Median into the Parent:
- If the parent node also overflows after this insertion, recursively split the parent node and propagate the median up the tree.



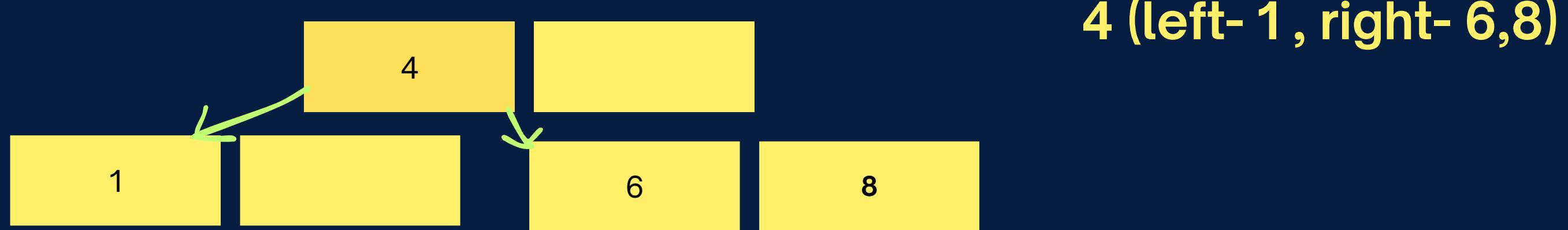
# LET'S START WITH DBMS :)

Max children= order of tree=3

Max keys node can have=order-1=3-1=2

## Insertion in B-TREE

Create a B-tree of Order 3 and insert values from 1,4,6,8,10,12,14,16



Follow step-2 again



Max children= order of tree=3

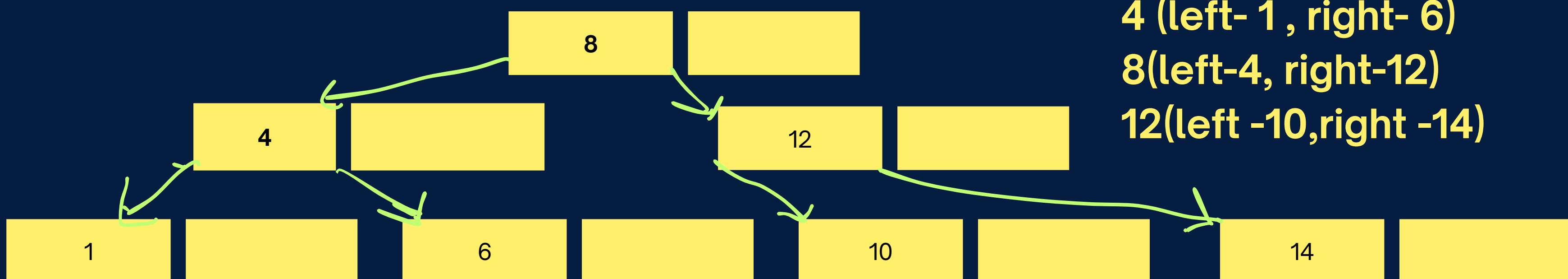
Max keys node can have=order-1=3-1=2

# LET'S START WITH DBMS :)

## Insertion in B-TREE

Create a B-tree of Order 3 and insert values from 1,4,6,8,10,12,14,16

Step 3: If the root node overflows (which can happen if it already has the maximum number of keys), split it into two nodes, and the median becomes the new root. This increases the height of the B-tree by one.



# LET'S START WITH DBMS :)

## Deletion in B-TREE

Consider a B-tree of order 4

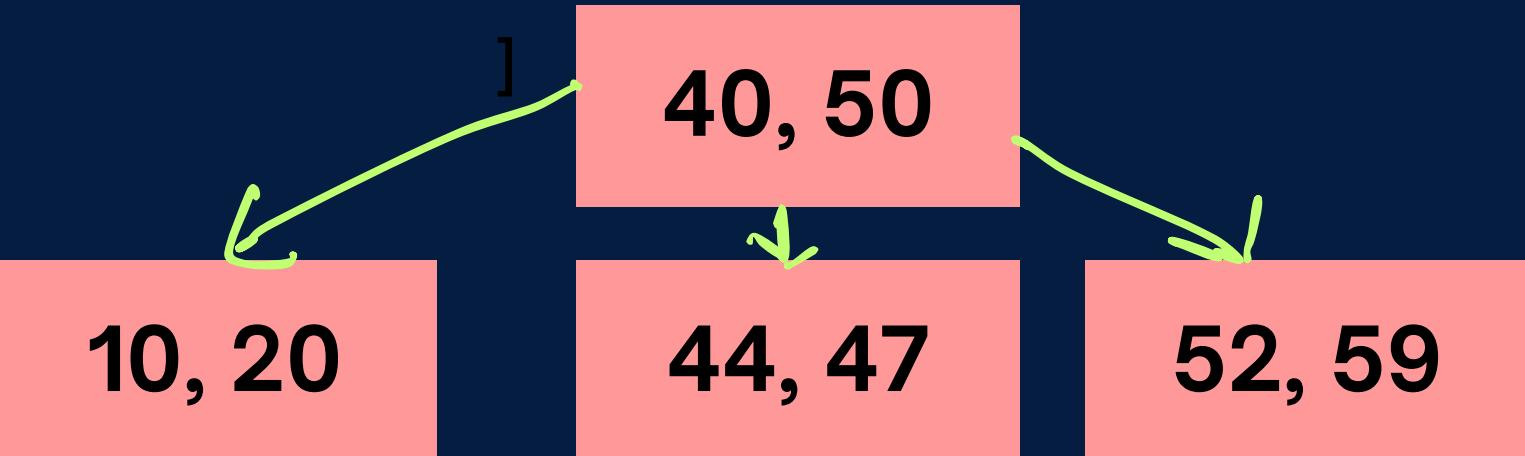
Step 1: Begin at the root and recursively move down the tree to find the node that contains the key to be deleted.

Step 2 :

Case 1: The Key is in a Leaf Node (**Delete 10**)

- Simply remove the key from the leaf node.
- If the node still has the minimum required number of keys (i.e., at least  $\text{ceil}(\text{order}/2) - 1$  keys), the deletion is complete.

maximum of  $(m-1)=3$  keys  
and minimum of  $(\text{ceil}(m/2)-1)=1$  key



# LET'S START WITH DBMS :)

## Deletion in B-TREE

Consider a B-tree of order 4

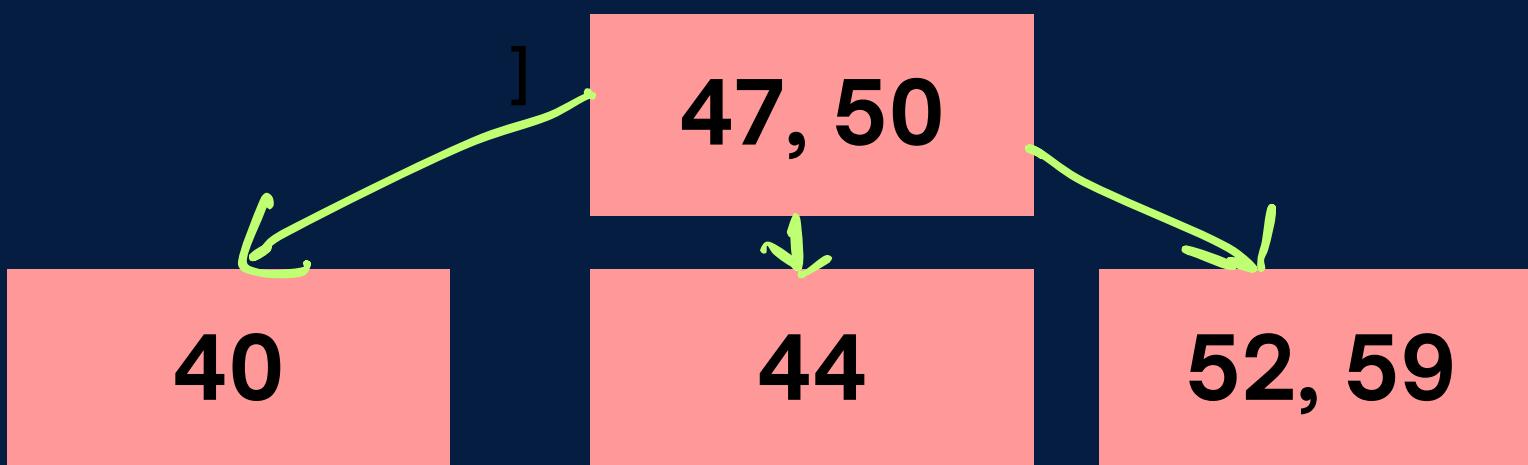
Step 2 :

Case 2: The Key is in a Leaf Node (**Delete 20**)

- If the deletion causes the node to have fewer than the minimum number of keys, proceed to the Borrowing or Merging step.

1. If the node has a sibling with more than the minimum number of keys, you can borrow a key from this sibling. The parent key between the node and the sibling moves down to the node, and a key from the sibling moves up to the parent.

maximum of  $(m-1)=3$  keys  
and minimum of  $(\text{ceil}(m/2)-1)=1$  key



# LET'S START WITH DBMS :)

## Deletion in B-TREE

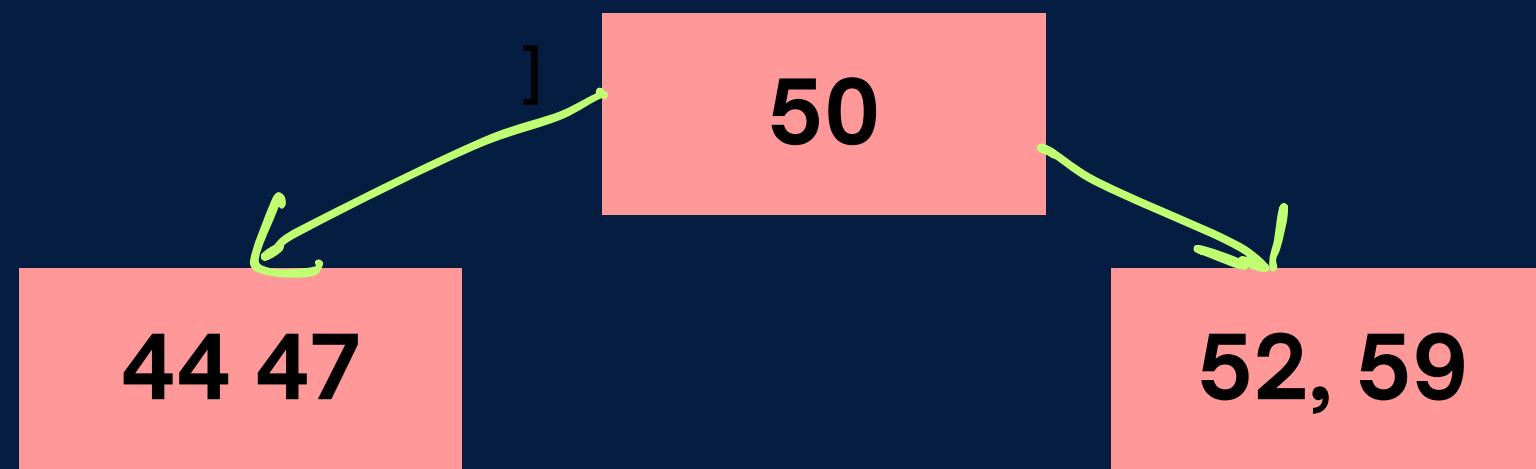
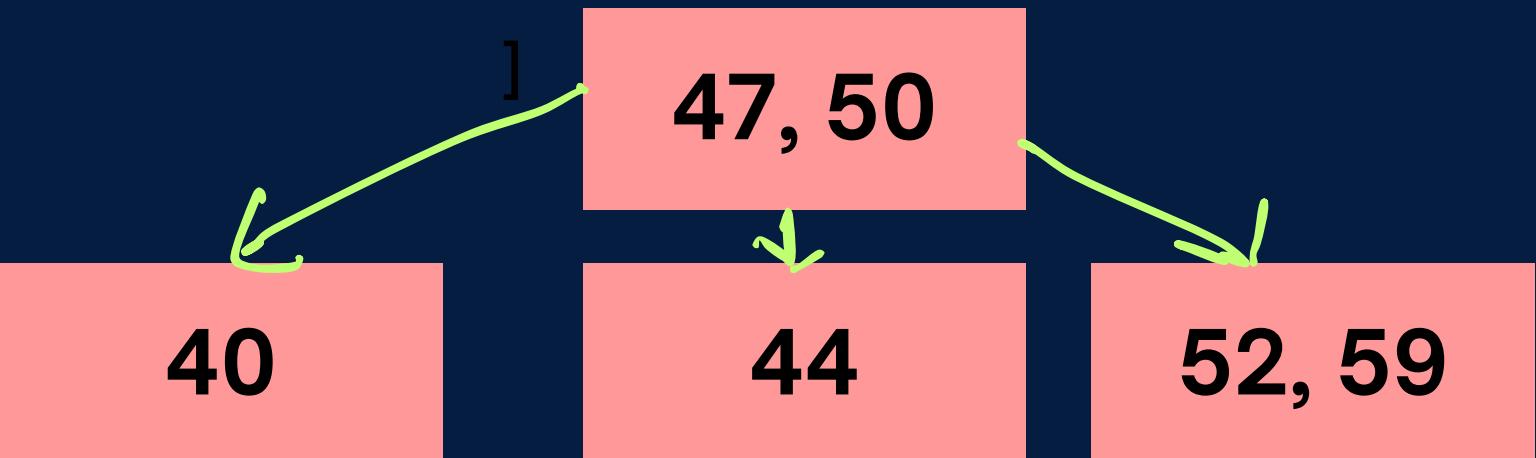
Consider a B-tree of order 4

Step 2 :

Case 2: The Key is in a Leaf Node (**Delete 40**)

2. If borrowing is not possible (i.e., the sibling also has the minimum number of keys), merge the node with a sibling. The key from the parent that separates the two nodes moves down into the newly merged node. If this causes the parent to have too few keys, repeat the borrowing or merging process at the parent level.

maximum of  $(m-1)=3$  keys  
and minimum of  $(\text{ceil}(m/2)-1)=1$  key



# LET'S START WITH DBMS :)

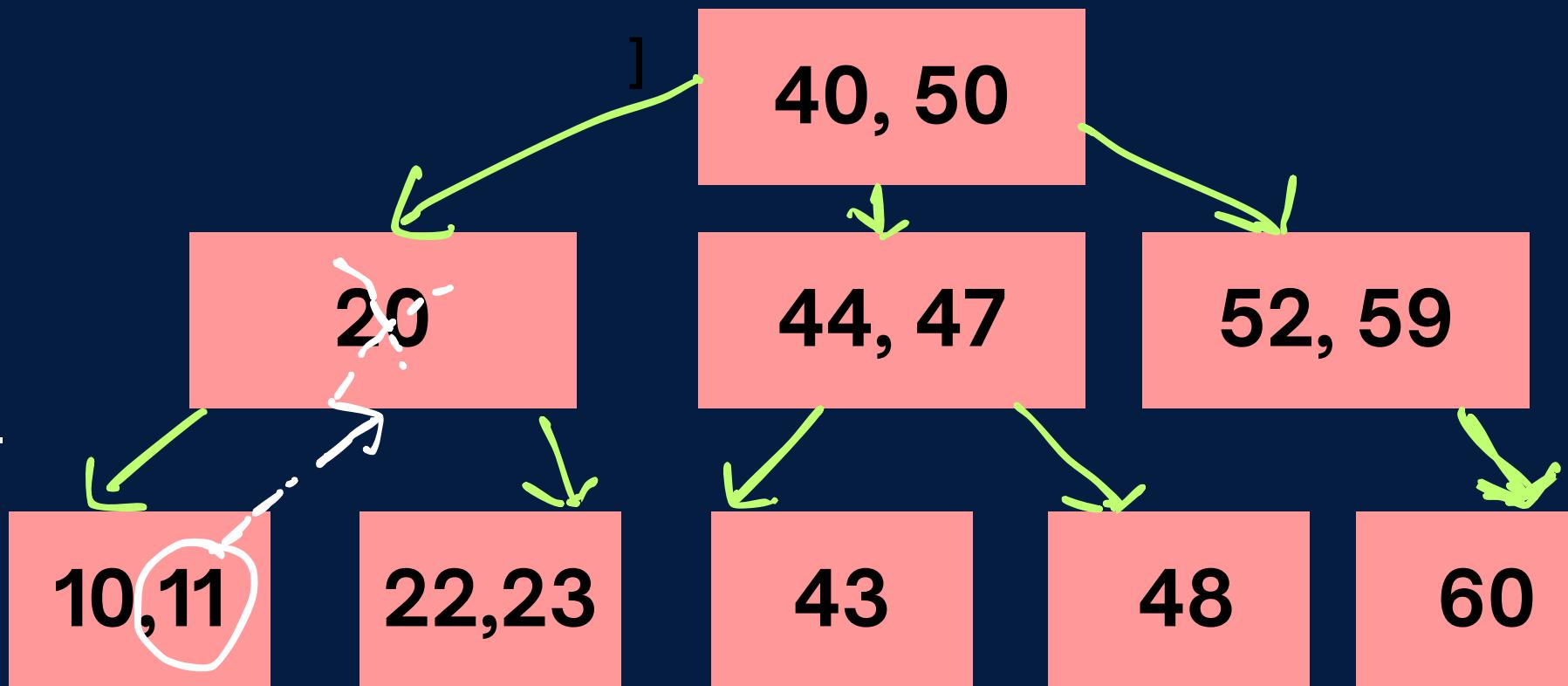
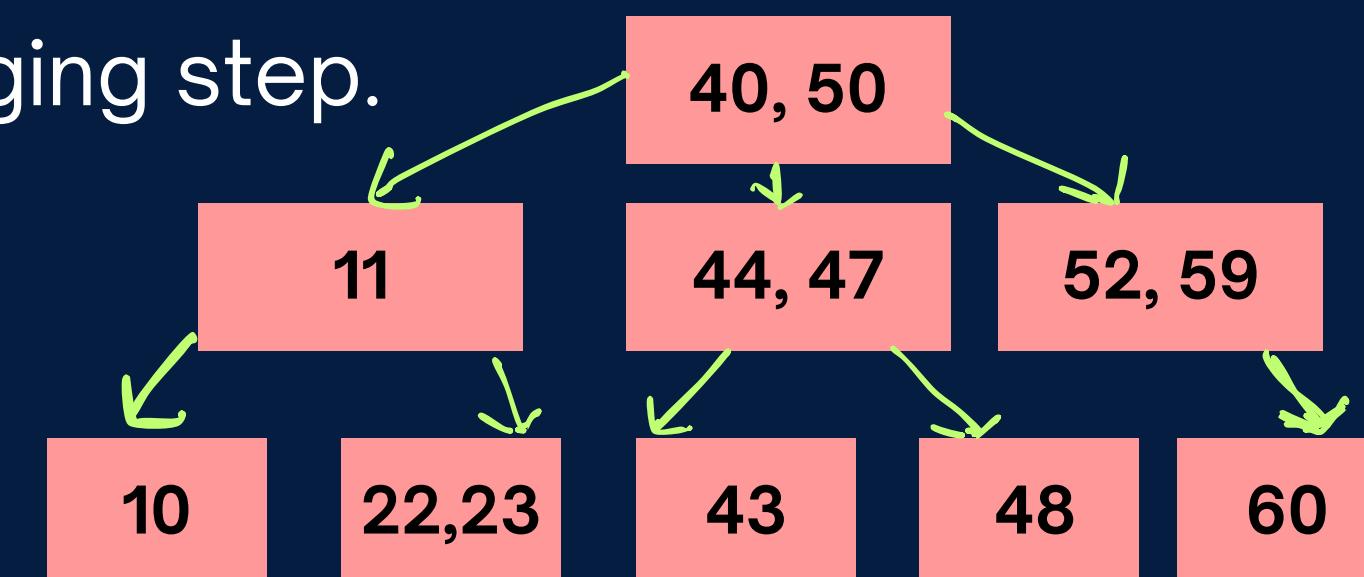
## Deletion in B-TREE

Steps on how we can delete elements in B-tree

Step 2:

Case 2: The Key is in an Internal Node (**Delete 20**)

- Replace the key with its predecessor (the largest key in the left subtree) or its successor (the smallest key in the right subtree).
- Delete the predecessor or successor key from the corresponding subtree.
- If this causes an underflow (i.e., a node has too few keys), proceed to the Borrowing or Merging step.



# LET'S START WITH DBMS :)

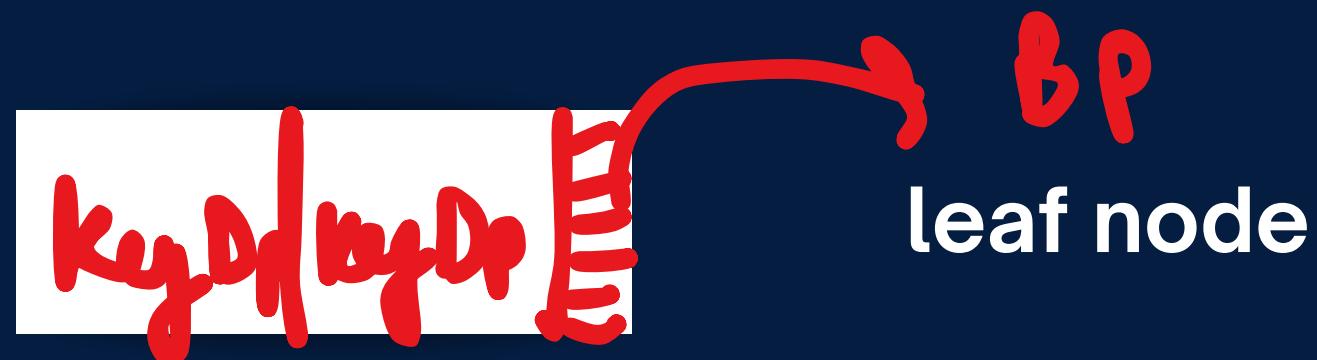
## B+ Tree

A B+ tree is an extension of the B-tree and is commonly used in databases and file systems to maintain sorted data and allow for efficient insertion, deletion, and search operations. B+ tree is a balanced tree, meaning all leaf nodes are at the same level

The key difference between a B+ tree and a B-tree lies in how they store data and how leaf nodes are structured.

1. In a B+ tree, all actual data (or references to data) are stored in the leaf nodes.

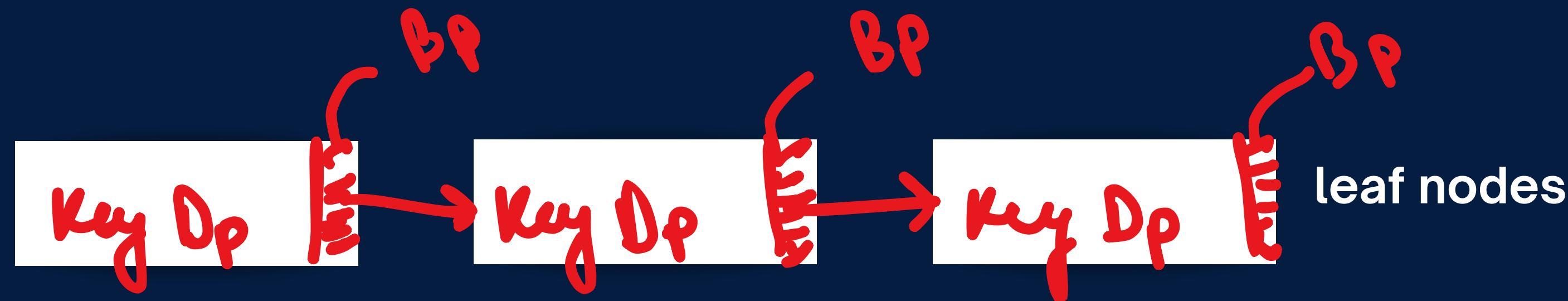
Internal nodes only store keys



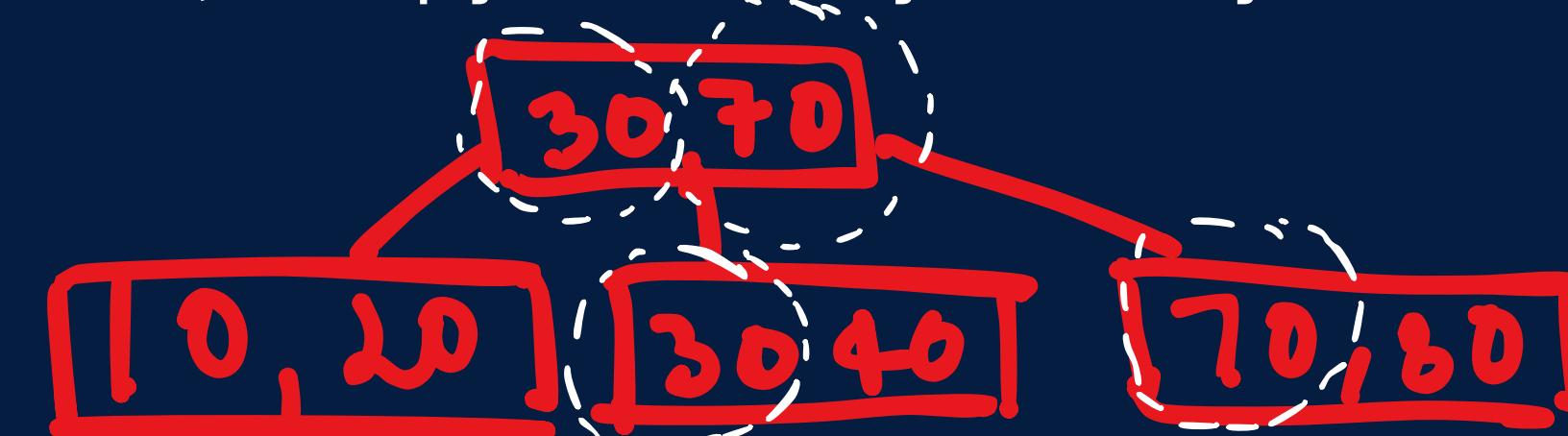
# LET'S START WITH DBMS :)

## B+ Tree

2. In a B+ tree, Leaf nodes are linked together in a linked list fashion, allowing for efficient sequential access, one leaf node will have only 1 Bp.



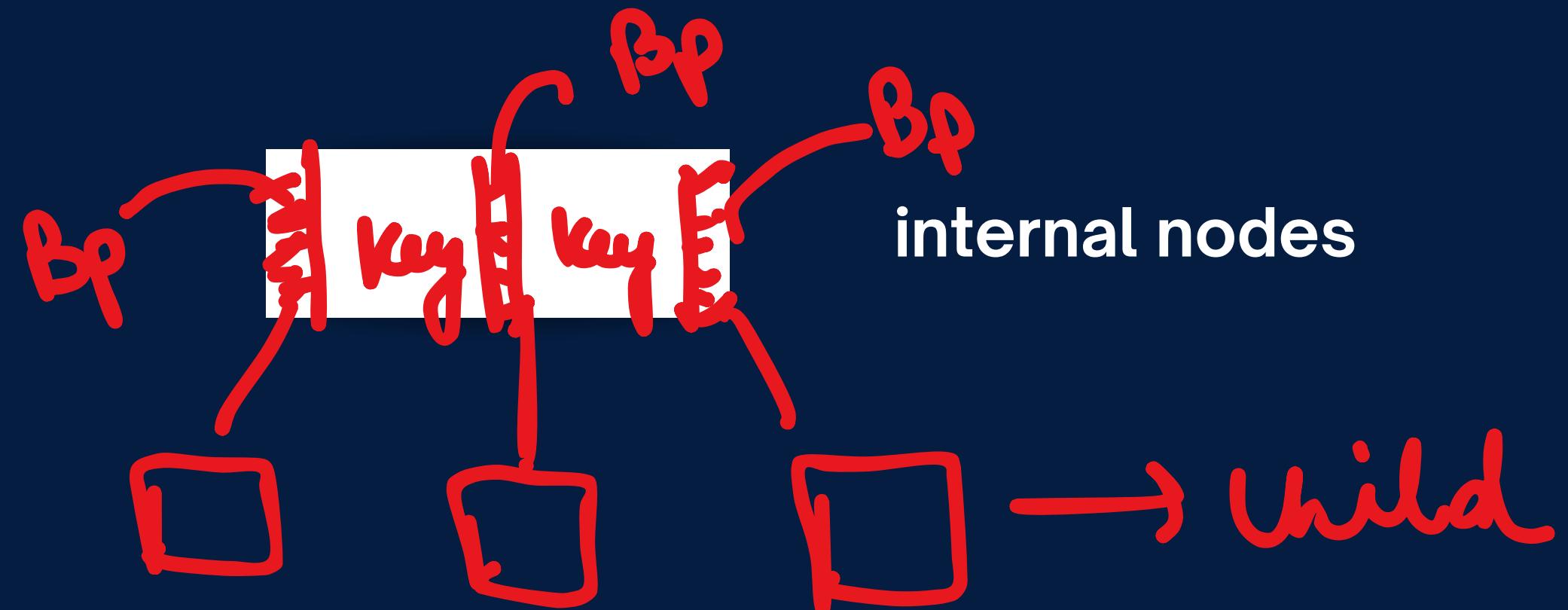
During inserting value in B+ tree, a copy of the key is always stored in the leaf node.



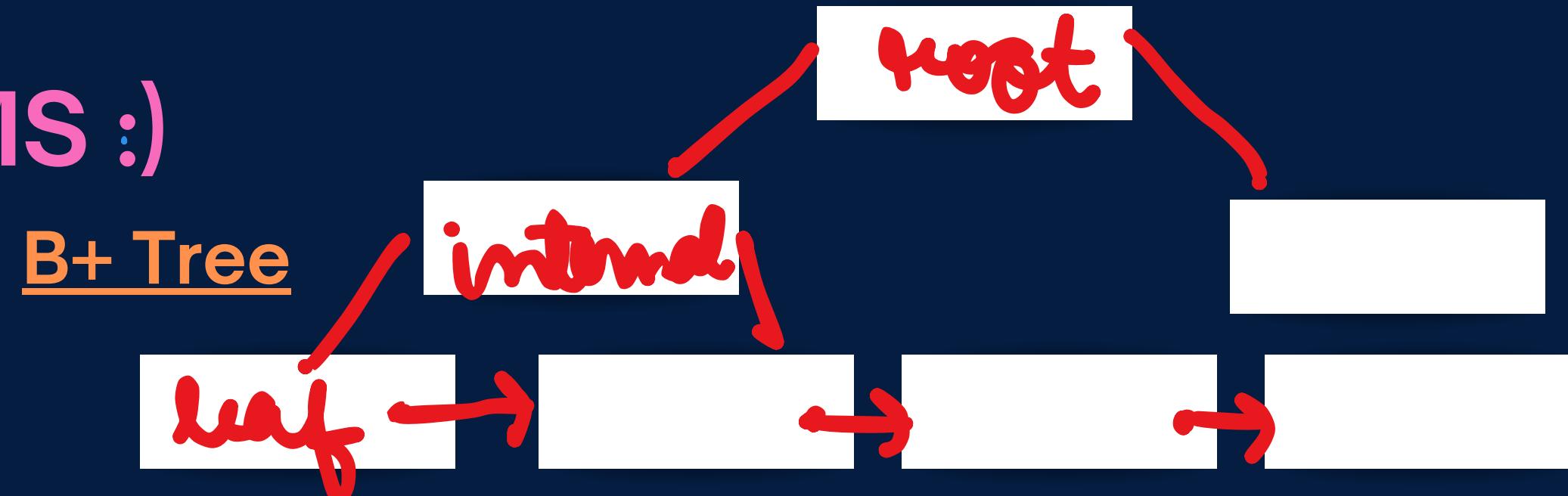
# LET'S START WITH DBMS :)

## B+ Tree

3. Internal nodes do not store data pointers, only keys and child pointers. This allows more keys to be stored in each internal node, leading to a lower height and more efficient operations.



# LET'S START WITH DBMS :)



Structure of a B+ Tree:

- Root Node:
  - The top node of the B+ tree, which points to the first level of internal nodes or directly to leaf nodes if the tree has only one level.
- Internal Nodes:
  - These nodes contain only keys and pointers to child nodes. They guide the search process down to the correct leaf node.
- Leaf Nodes:
  - These nodes contain keys and data pointers. Each leaf node stores a pointer to the next leaf node, enabling quick traversal of records.

# LET'S START WITH DBMS :)

## B+ Tree

Advantages :

1. B+ trees have a balanced structure, meaning all leaf nodes are at the same level. This balance ensures that search operations require logarithmic time relative to the number of keys, making it very efficient even for large datasets.
2. The structure of the B+ tree allows for direct access to data. Since the internal nodes contain only keys, searching for a specific value can be done quickly by navigating through the tree down to the leaf node where the data is stored.

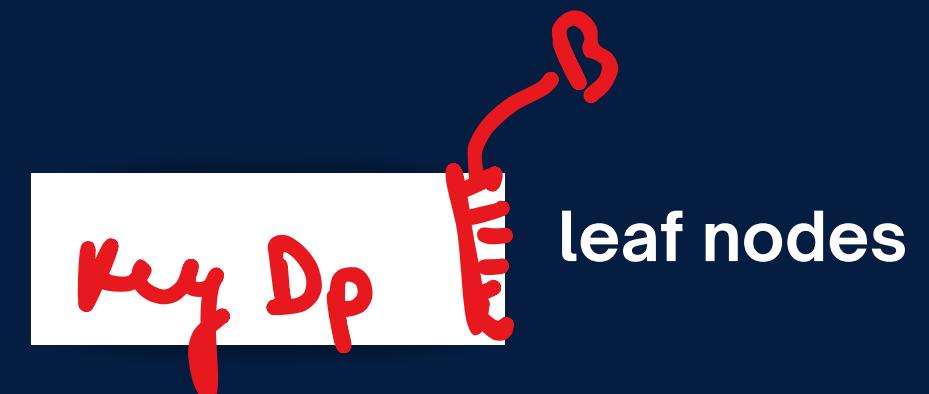
# LET'S START WITH DBMS :)

## B+ Tree

Order of B+ tree

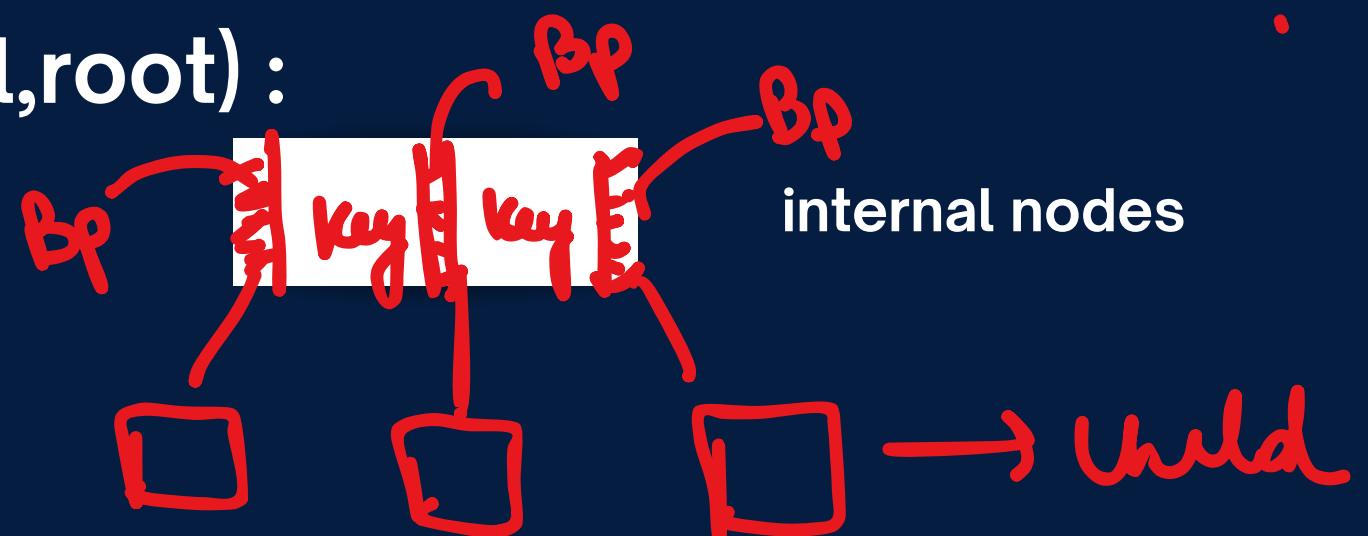
Order of Leaf node :

$$1 \times Pb + M(k + Pd) \leq B$$



Order of non-Leaf node(internal,root) :

$$m \times Pb + (m-1)k \leq B$$



B- Block size

m- Order of tree

Pb- Block pointer size

K- Key size

Pd- Data pointer size