



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

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)

8

Columnar Databases

2

NoSQL Databases

9

XML Databases

3

Object-Oriented Databases

10

NewSQL Databases

4

In-Memory Databases

11

Blockchain Databases

5

Time-Series Databases

SQL

6

Spatial Databases

7

Multimedia Databases

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 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)

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.

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.

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.

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.

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.

DISADVANTAGE OF DBMS

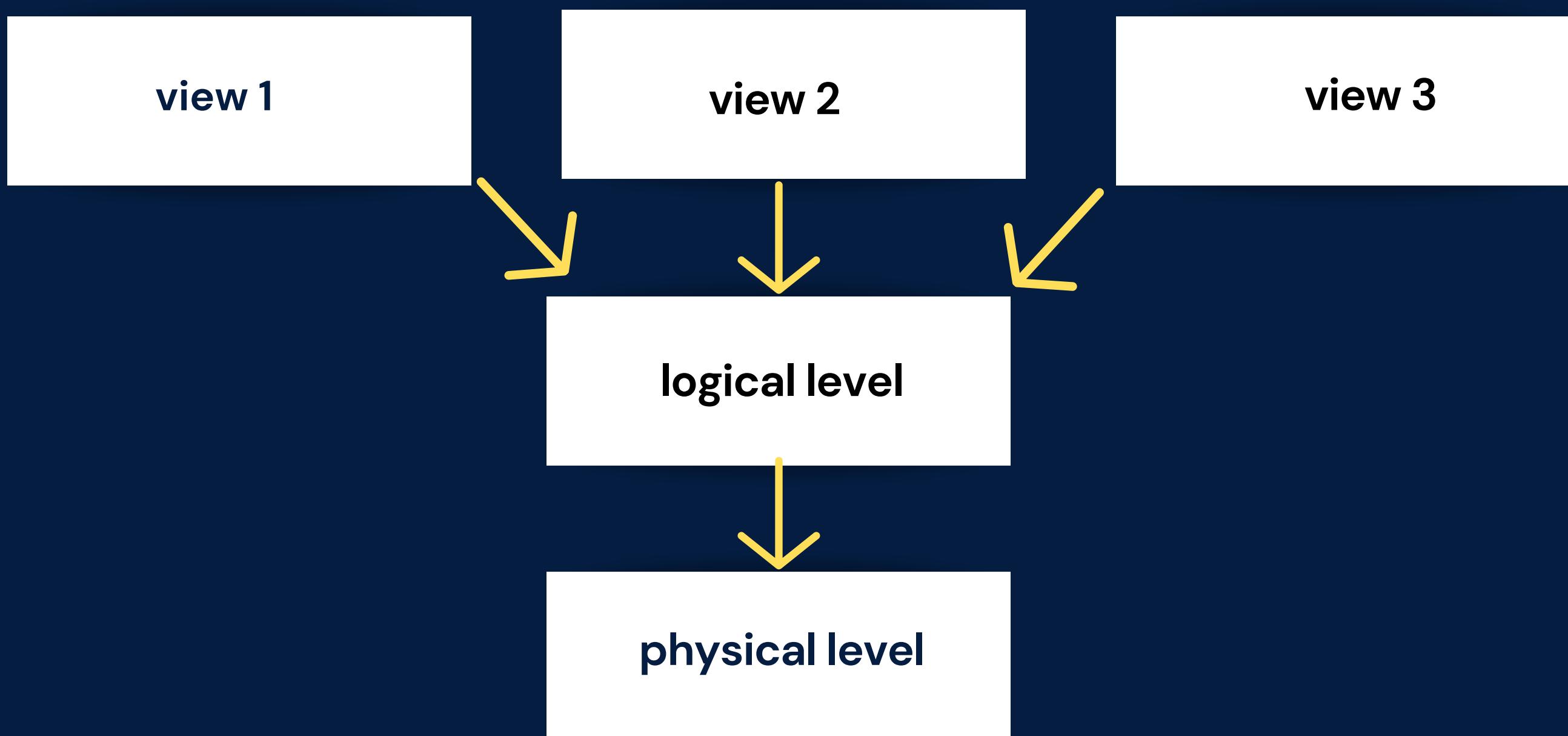
- Vendor Lock-In- Once you've chosen a specific DBMS, it can be challenging to switch to a different one due to differences in data formats, query languages, and other technical aspects. This can lead to vendor lock-in, where you are dependent on a particular vendor's technology and pricing.

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.

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.

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.

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.

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.



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.

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.

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**

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



Client (User/Application)

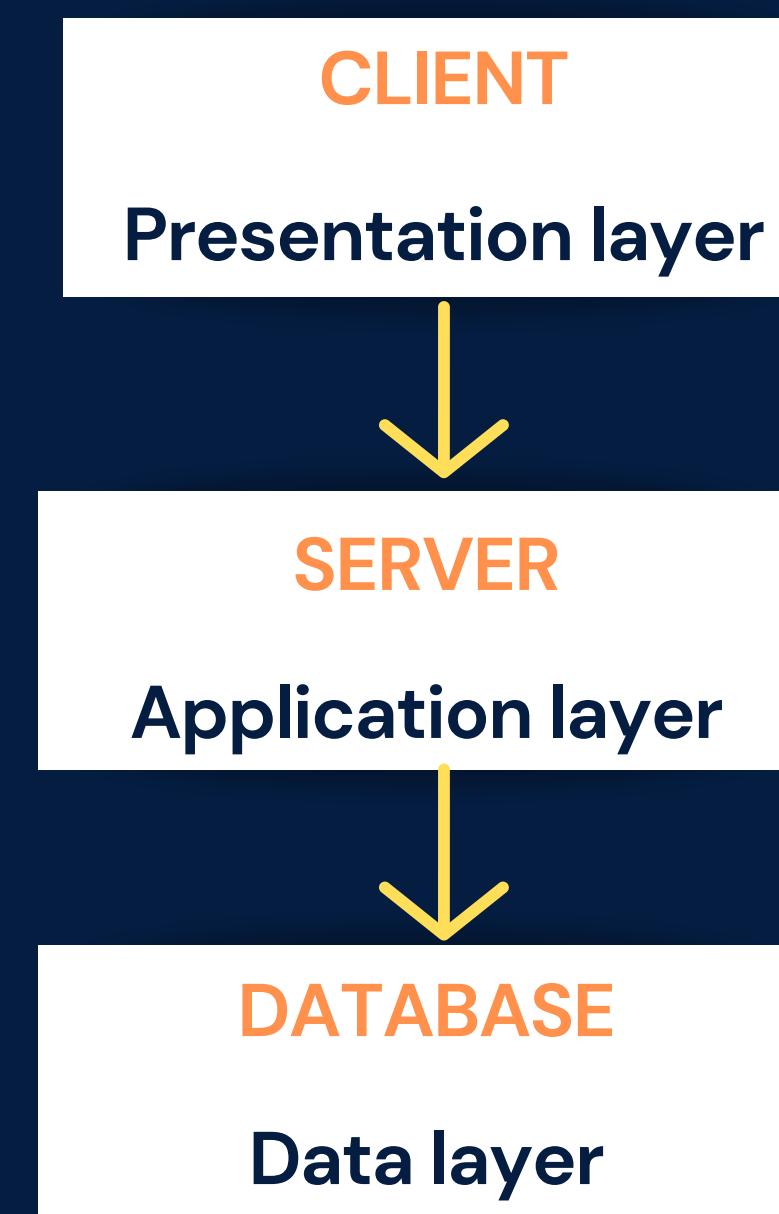
Server(db)

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

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.

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.

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.

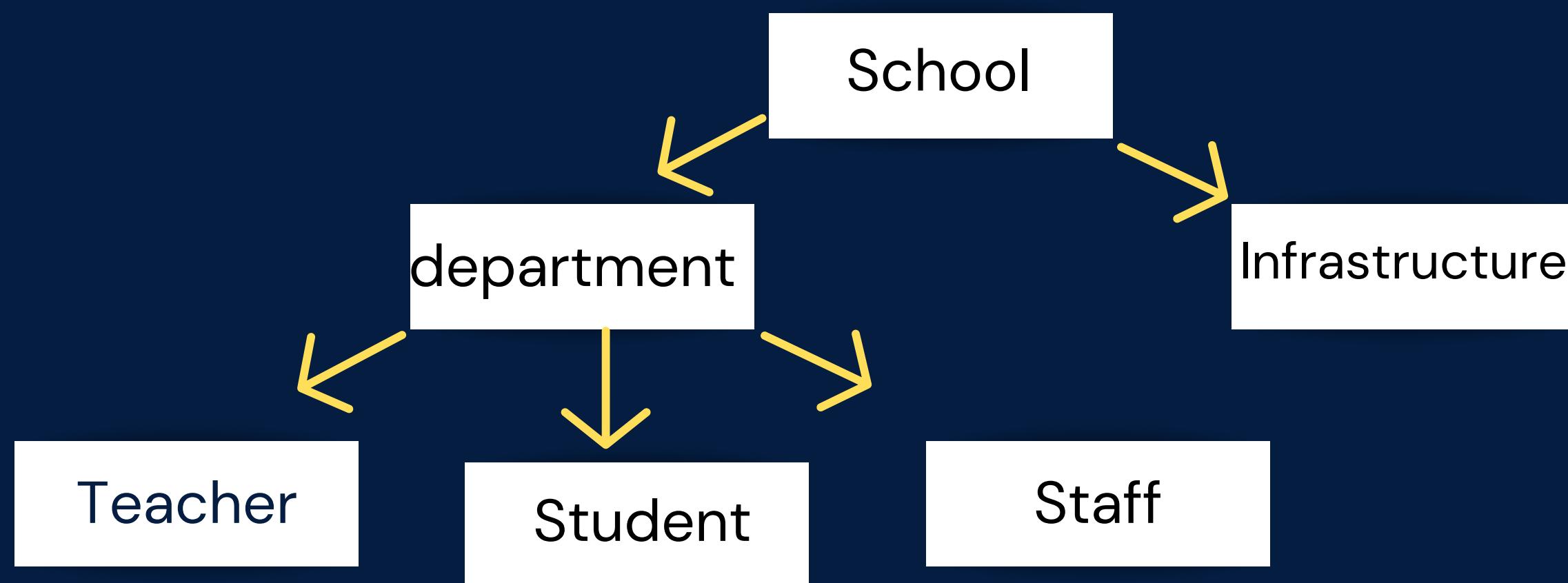
DATA MODEL

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

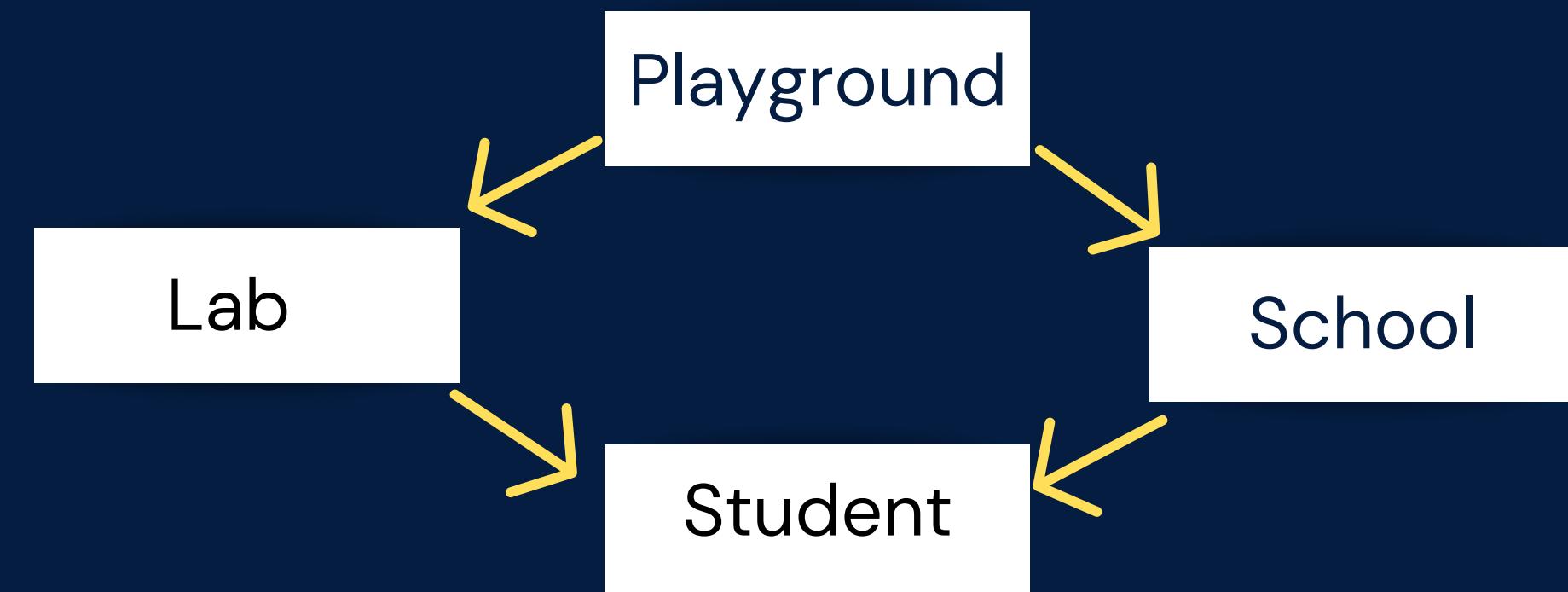
TYPES OF DATA MODEL

- **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.



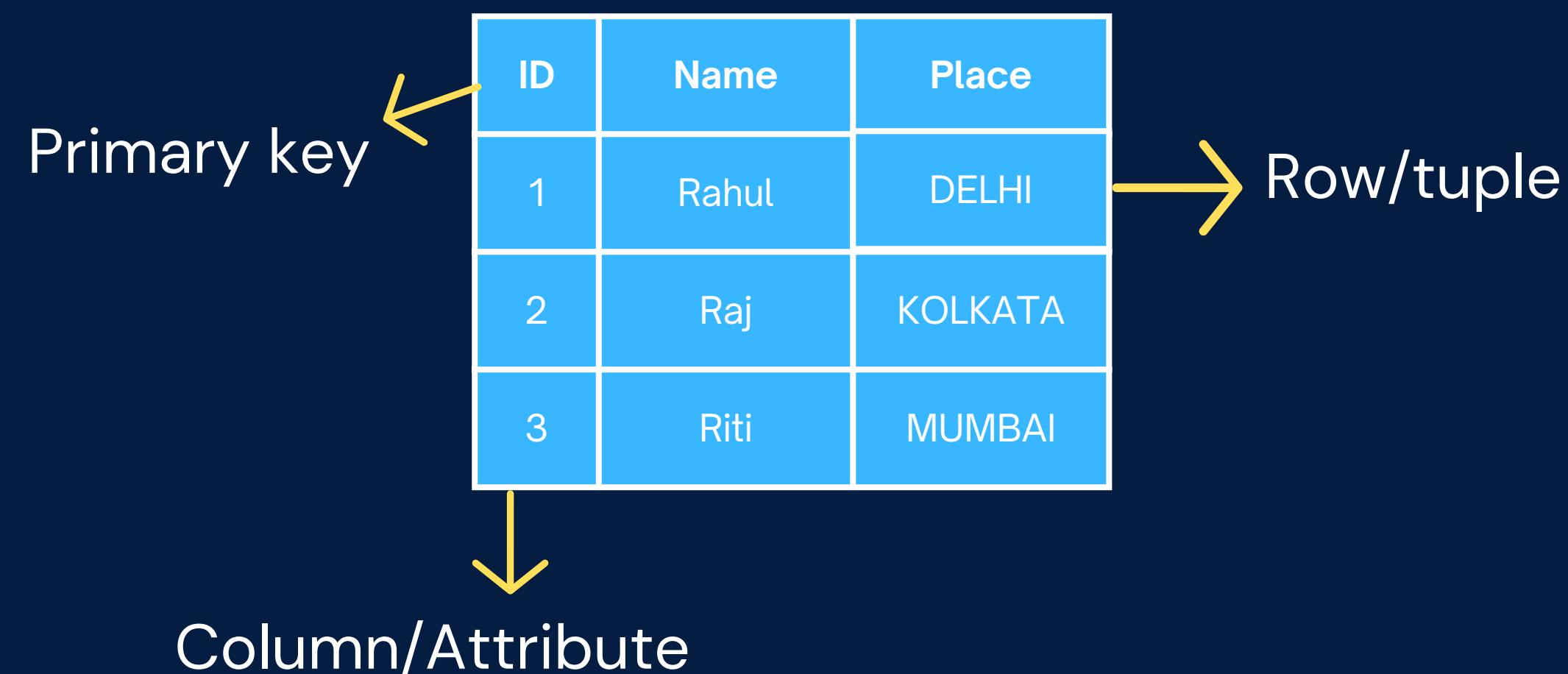
TYPES OF DATA MODEL

- **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.



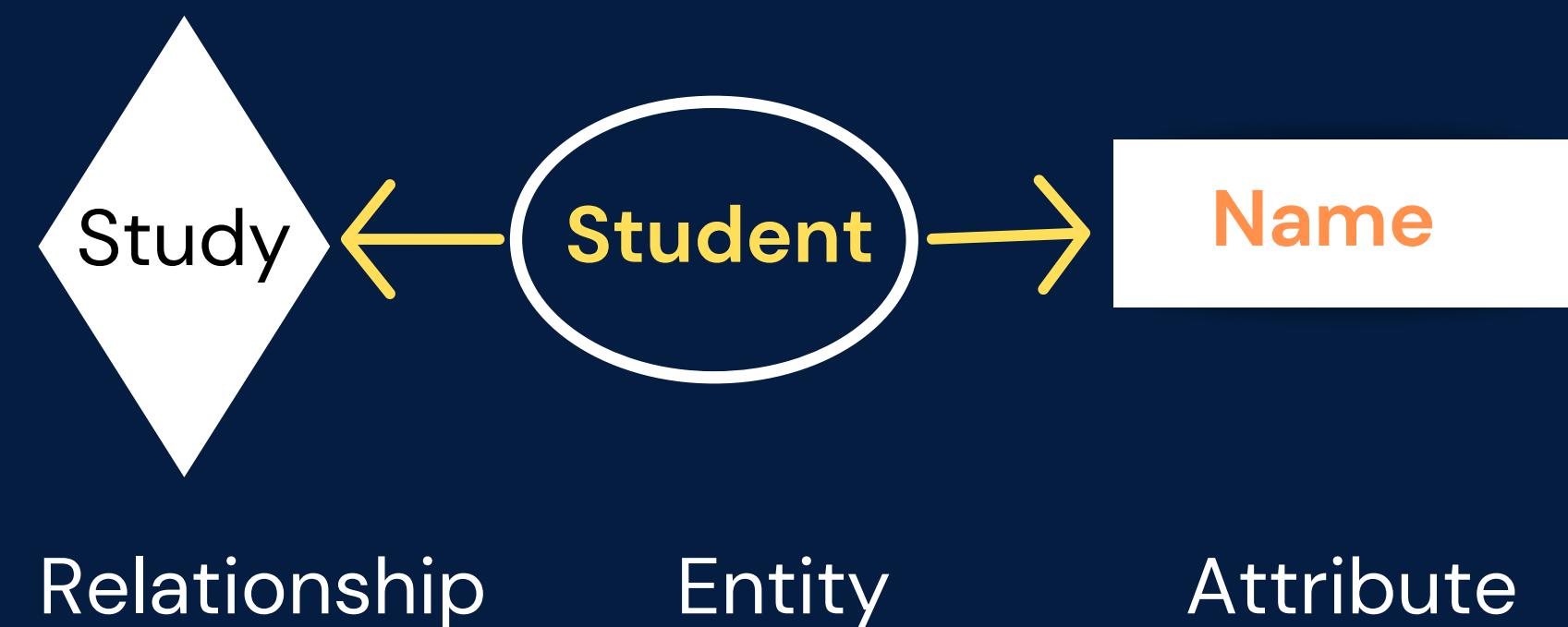
TYPES OF DATA MODEL

- **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.



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

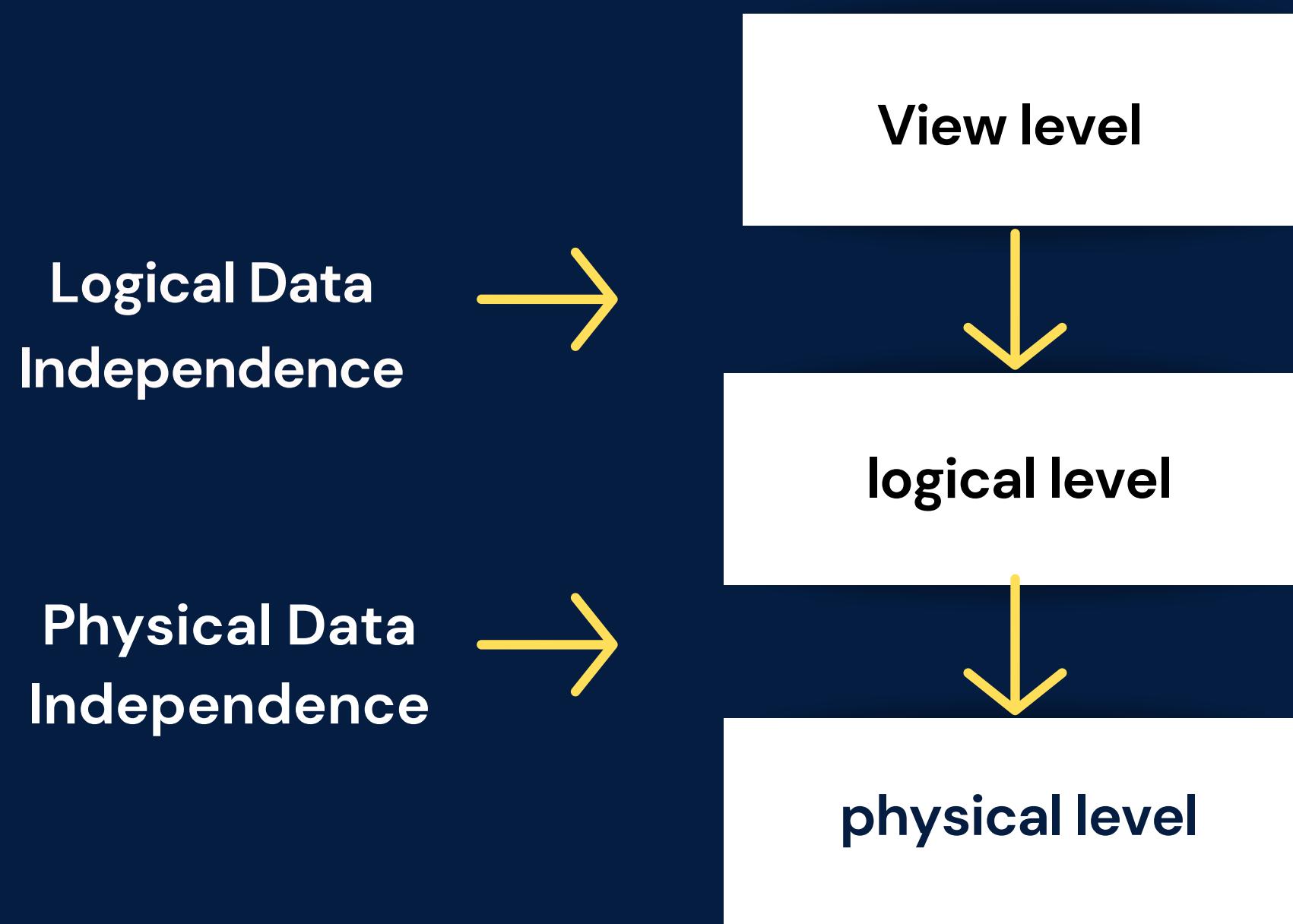
- **Object-Oriented Data Model:** Extending the principles of object-oriented programming into the database domain, this model depicts data as objects complete with attributes and methods, fostering support for inheritance and encapsulation.
- **NoSQL Data Models:** NoSQL databases encompass a diverse 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.

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.

DATA INDEPENDENCE

There are three levels of Abstraction



ESSENTIAL COMPONENTS OF TABLES

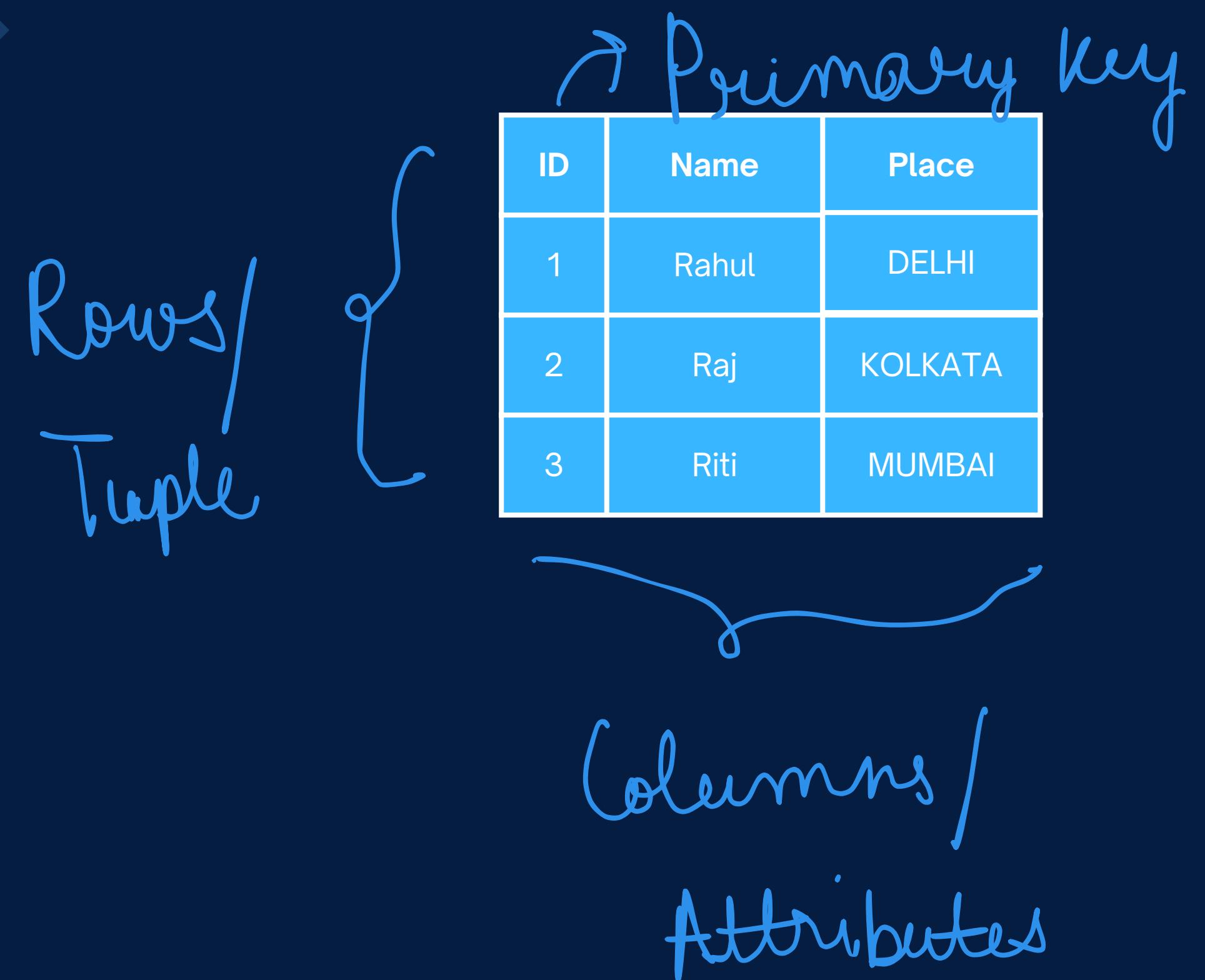
Row/Tuple - Rows, also known as records or tuples, represent individual entries or instances of data within the table.

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



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.

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
1	Raj	456	blr	123	18
2	Ravi	123	delhi	124	21
3	Ram	789	hyd	345	22

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**

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

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

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

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.

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

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.

KEYS IN DBMS

Referential Integrity in Foreign key.

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.

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".

KEYS IN DBMS

Referential Integrity in Foreign key.

- Insertion in Referencing table

May cause violation

KEYS IN DBMS

Referential Integrity in Foreign key.

- Deletion in Referencing table

No violation

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

INTEGRITY CONSTRAINT IN DBMS

Integrity constraints help to ensure that data remains reliable and meaningful throughout its lifecycle.

Types of Integrity Constraint:

- Domain Integrity Constraint
- Entity Integrity Constraint
- Referential Integrity Constraint
- Key Constraint
- Check Constraint
- Null Constraint
- Unique Constraint
- Default Constraint

INTEGRITY CONSTRAINT IN DBMS

Domain Integrity Constraint

It ensures the validity and appropriateness of data values
(i.e valid data types, ranges, and formats for columns)
within a specific column or attribute of a table.

Ex-> Check for date column so that it contains valid date values

INTEGRITY CONSTRAINT IN DBMS

Entity Integrity Constraint

It ensures that each row/record in a table is uniquely identified by a primary key.

It also helps in preventing duplicate or null values in the primary key.

INTEGRITY CONSTRAINT IN DBMS

Referential Integrity Constraint

It ensures that values in a foreign key column match with the values in the corresponding primary key column in another table.

INTEGRITY CONSTRAINT IN DBMS

Key Constraint

It ensures uniqueness for the primary key.

INTEGRITY CONSTRAINT IN DBMS

Check Constraint

It checks for a condition that each row in a table must satisfy.

If the condition is not met, the insertion or update of the row is rejected.

INTEGRITY CONSTRAINT IN DBMS

Null Constraint

It determines whether a column in a table can have null (i.e., missing or unknown) values or not.

INTEGRITY CONSTRAINT IN DBMS

Unique Constraint

It ensures that values in a specified column or combination of columns are unique across a table.

This constraint prevents duplicate values from being inserted into the specified column(s), maintaining data consistency and integrity.

INTEGRITY CONSTRAINT IN DBMS

Default Constraint

It ensures a default value for a column, which is used if no other value is provided

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).

ER MODEL IN DBMS

Introduction to ER Model

• Entity

Things/Object

Ex-person

Attributes

Properties of entity

Ex-name,age

Relationship

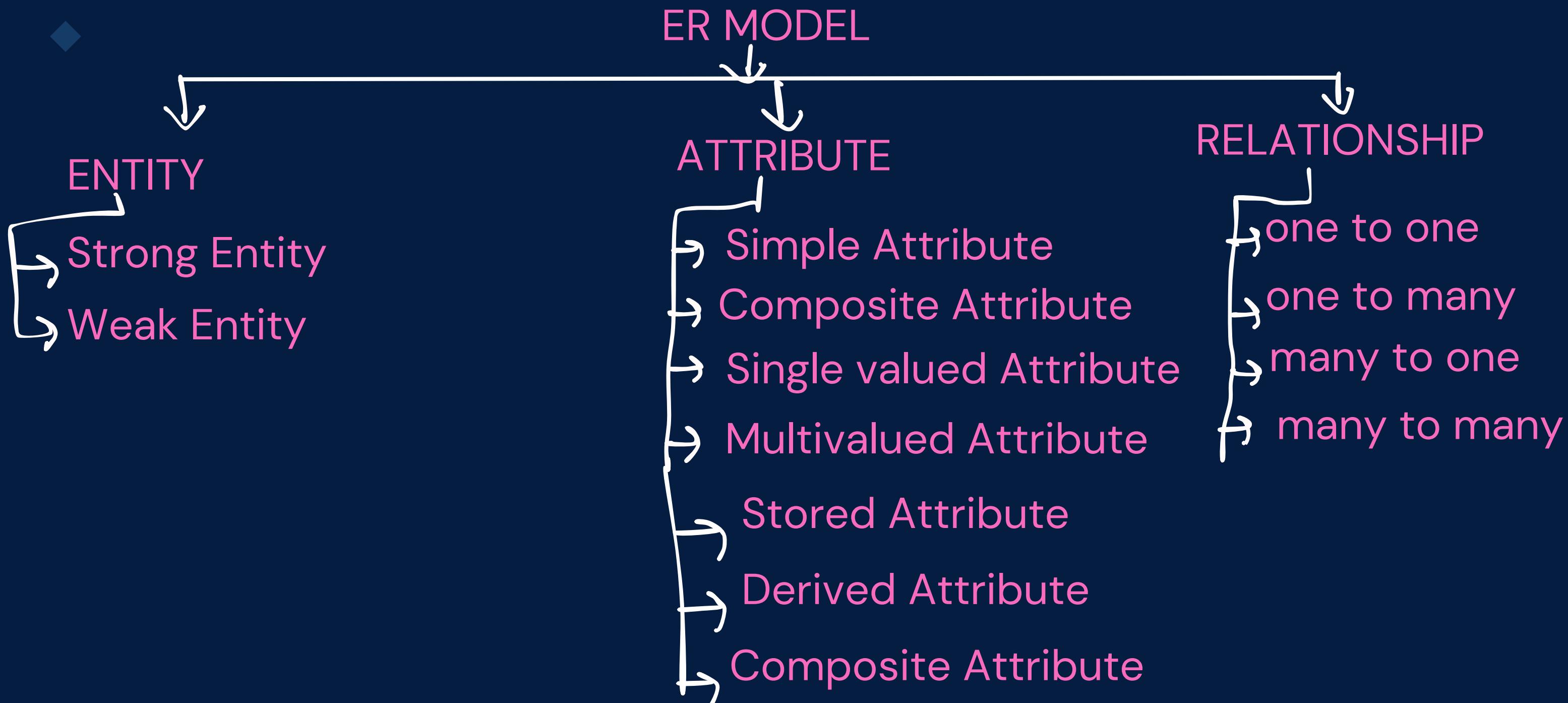
association among entities

Ex-Works for

ER MODEL IN DBMS

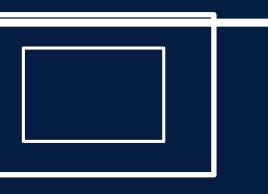
- The Entity-Relationship (ER) model stands as a prevalent conceptual modeling approach within the realm of database design.
- Its primary role is to offer a visual representation of a database's architecture by illustrating the entities, their respective attributes, and the interconnections between 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

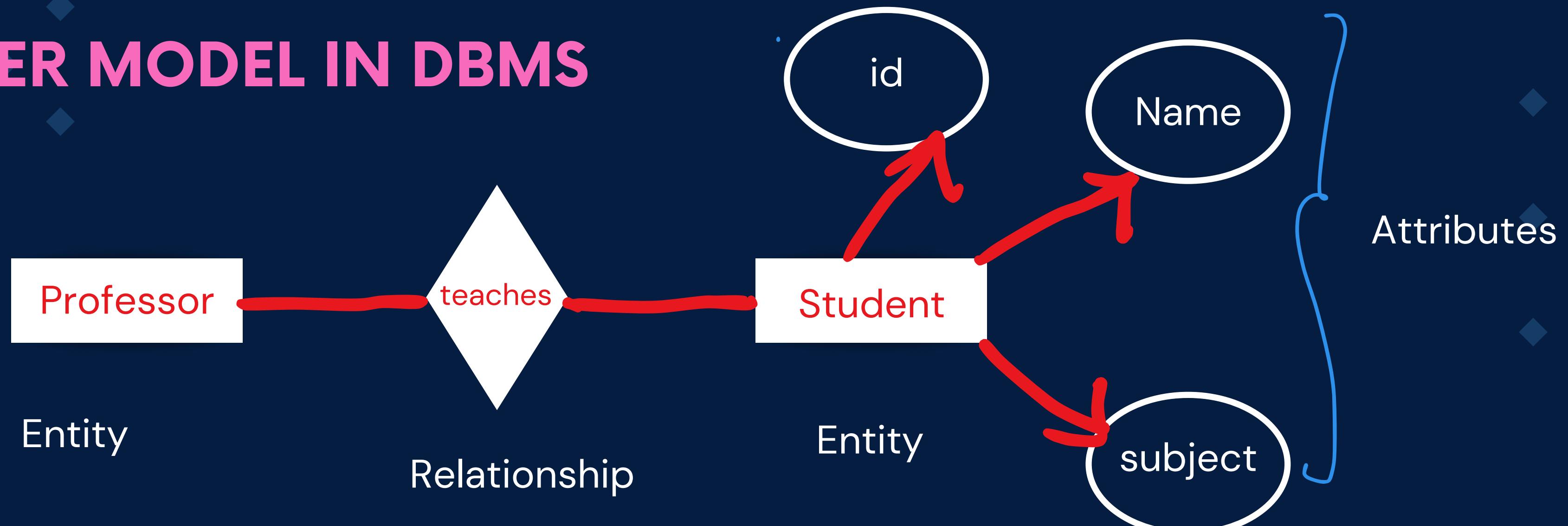


ER MODEL IN DBMS

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

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

Attribute

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

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)

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

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)

ER MODEL IN DBMS

Types of Attributes

Stored Attribute

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

Ex- Date of birth

ER MODEL IN DBMS

Types of Attributes

Derived Attribute

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

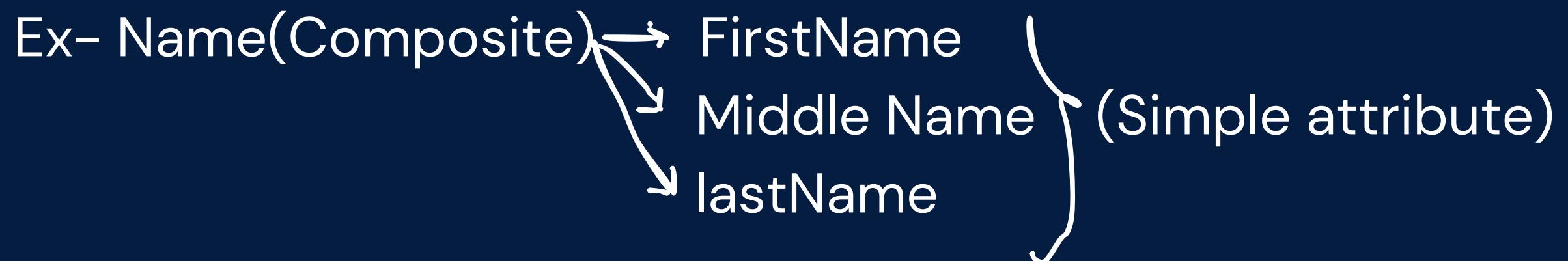
Ex- Age derived from dob

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.

Types of Relationship

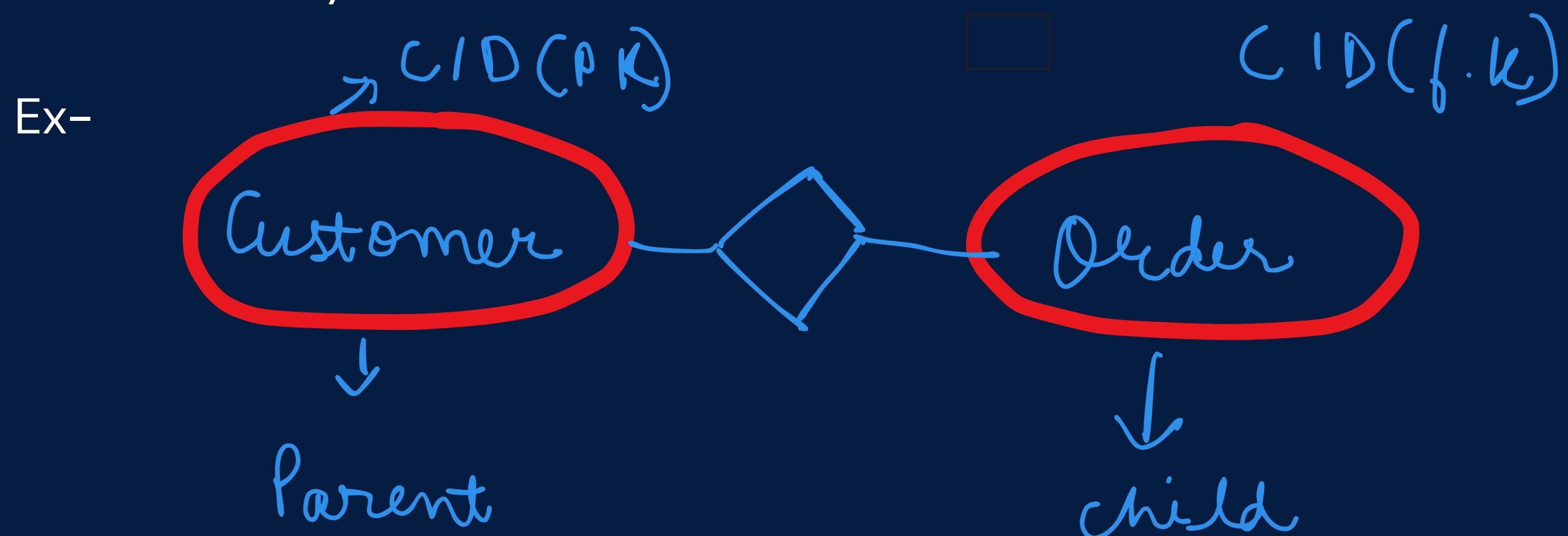
Strong Relationship

Weak Relationship

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.

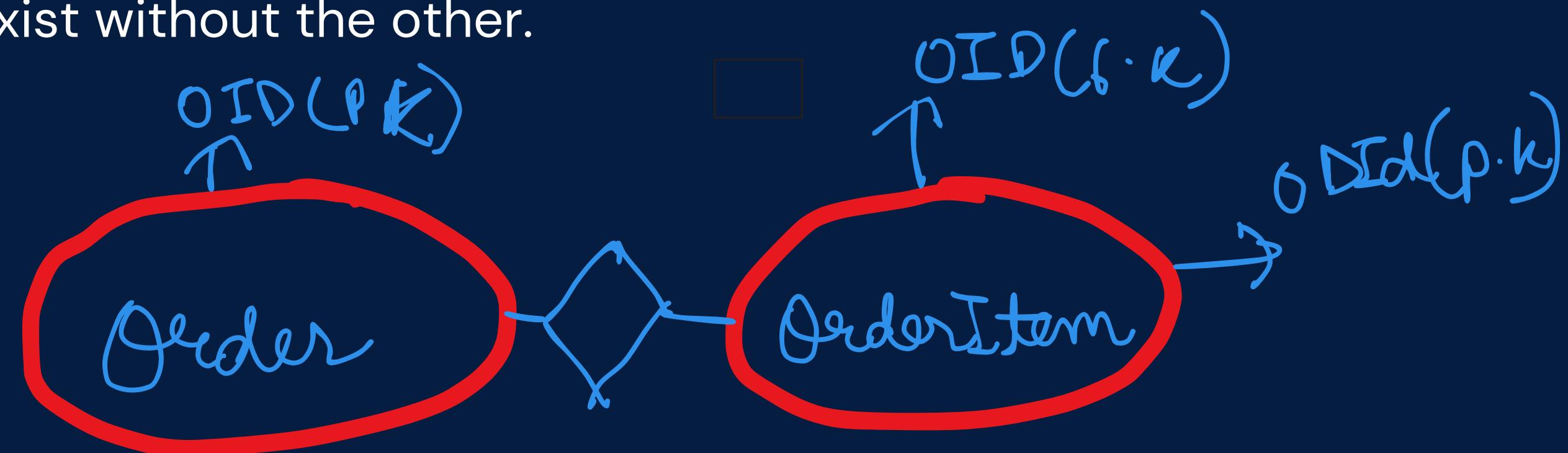


ER MODEL IN DBMS

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

ER MODEL IN DBMS

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

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.

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.

ER MODEL IN DBMS

Types of relationship in dbms (Based on degree)

There are 4 types of relationship:

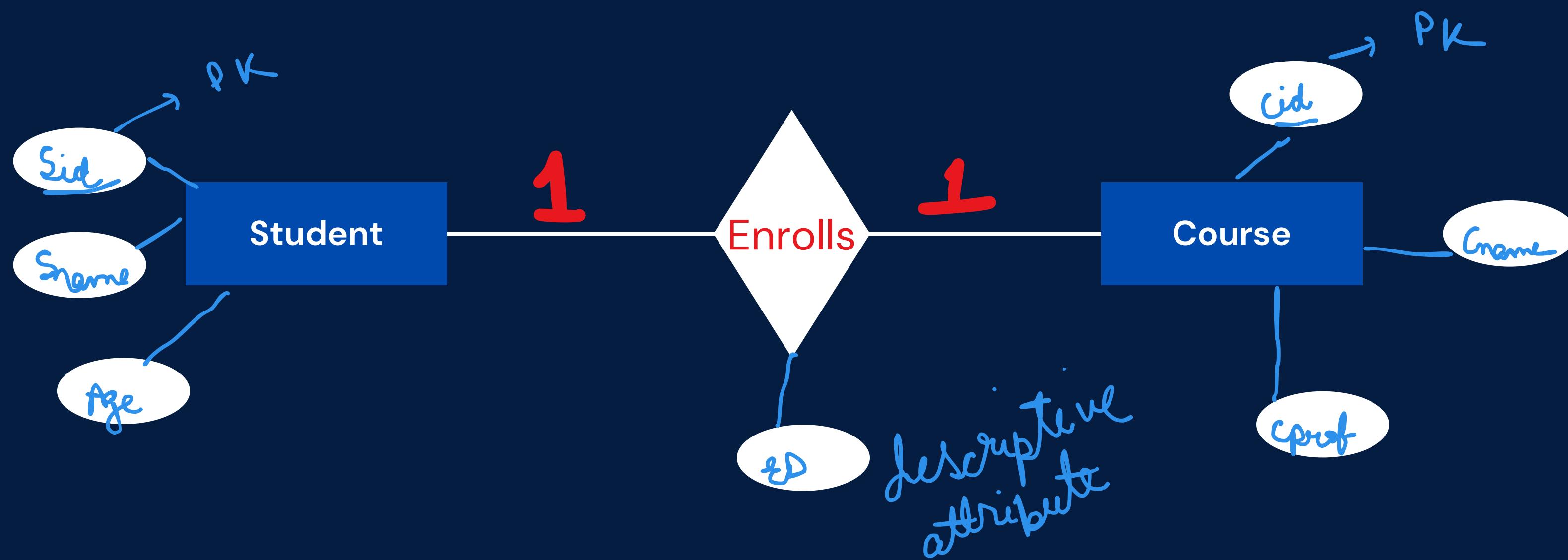
- one to one (1-1)
- one to many (1-N)
- many to one (N-1)
- many to many (N-N)

ER MODEL IN DBMS

Types of Relationship(Cardinality)

1 to 1 Relationship(1:1)

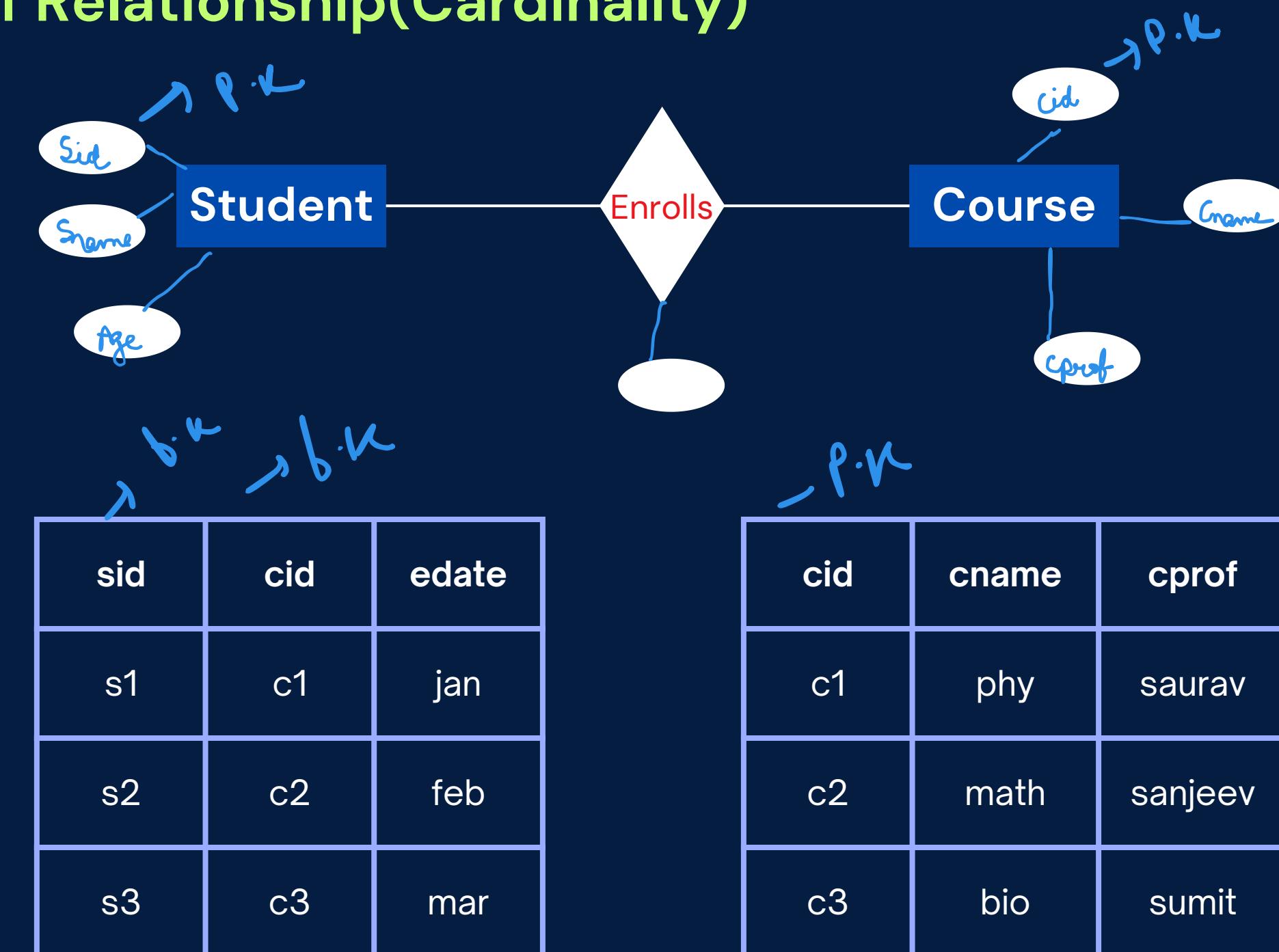
A complex attribute is an attribute that is made up of multiple smaller attributes



ER MODEL IN DBMS

Types of Relationship(Cardinality)

1 to 1 Relationship(1:1).



ER MODEL IN DBMS

P.K

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

PK

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

P.K

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

P.K

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

P.K

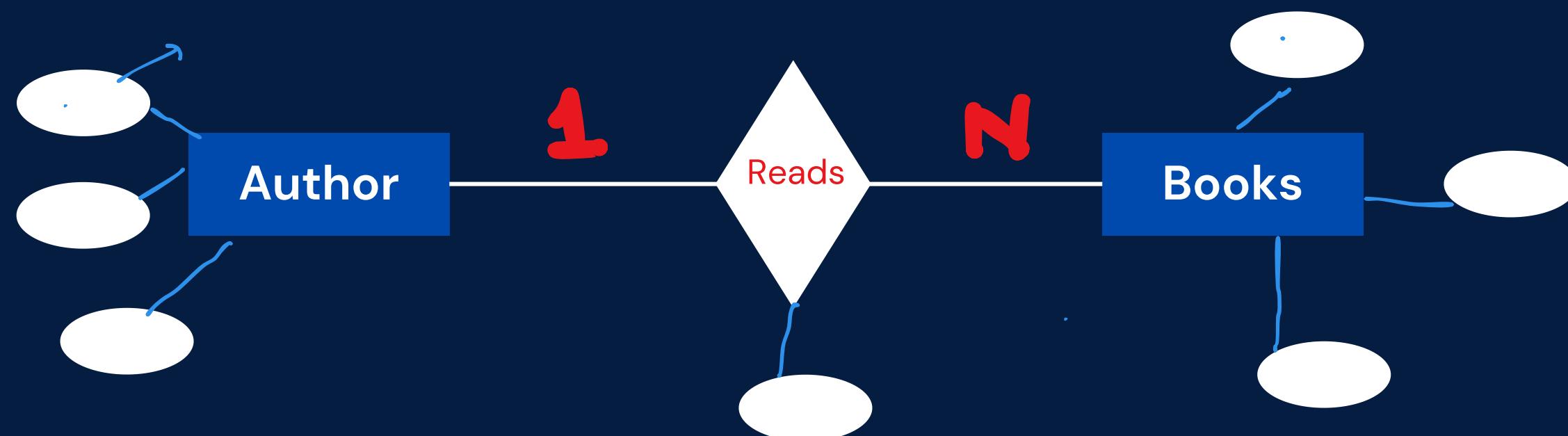
cid	cname	cprof
c1	phy	saurav
c2	math	sanjeev
c3	bio	sumit

ER MODEL IN DBMS

Types of Relationship

1 to Many Relationship(1:N).

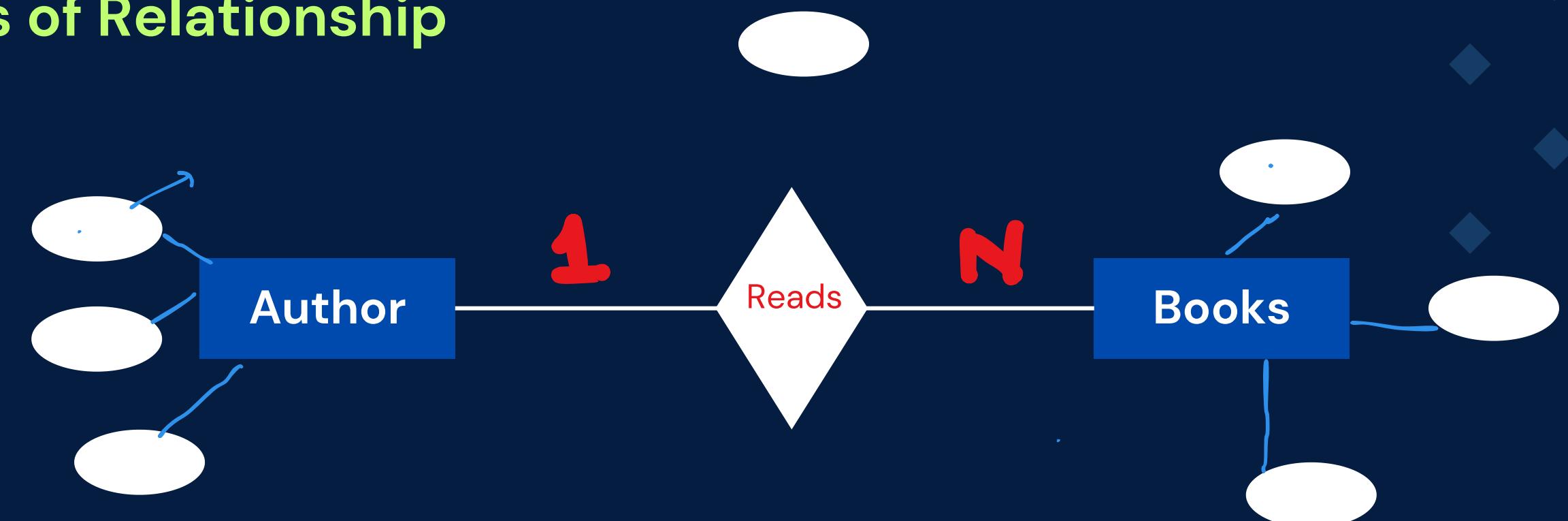
A database model where one entity (record) on one side of the relationship is associated with multiple entities (records) on the other side



ER MODEL IN DBMS

Types of Relationship

1 to Many Relationship(1:N).



aid	aname	aage
a1	ram	14
a2	raj	15
a3	riti	16

aid	bid	bdate
a1	b1	jan
a2	b2	feb
a1	b3	mar

bid	bname	btype
b1	ab	fiction
b2	cd	thrill
b3	ef	drama

ER MODEL IN DBMS

aid	aname	aage
a1	ram	14
a2	raj	15
a3	riti	16

PK *FK*

aid	bid	bdate
a1	b1	jan
a2	b2	feb
a1	b3	mar

PK

bid	bname	btype
b1	ab	fiction
b2	cd	thrill
b3	ef	drama

PK

aid	aname	aage
a1	ram	14
a2	raj	15
a3	riti	16

PK

bia	bname	btype	aid	bdate
b1	ab	fiction	a1	jan
b2	cd	thrill	a2	feb
b3	ef	drama	a1	mar

ER MODEL IN DBMS

Types of Relationship

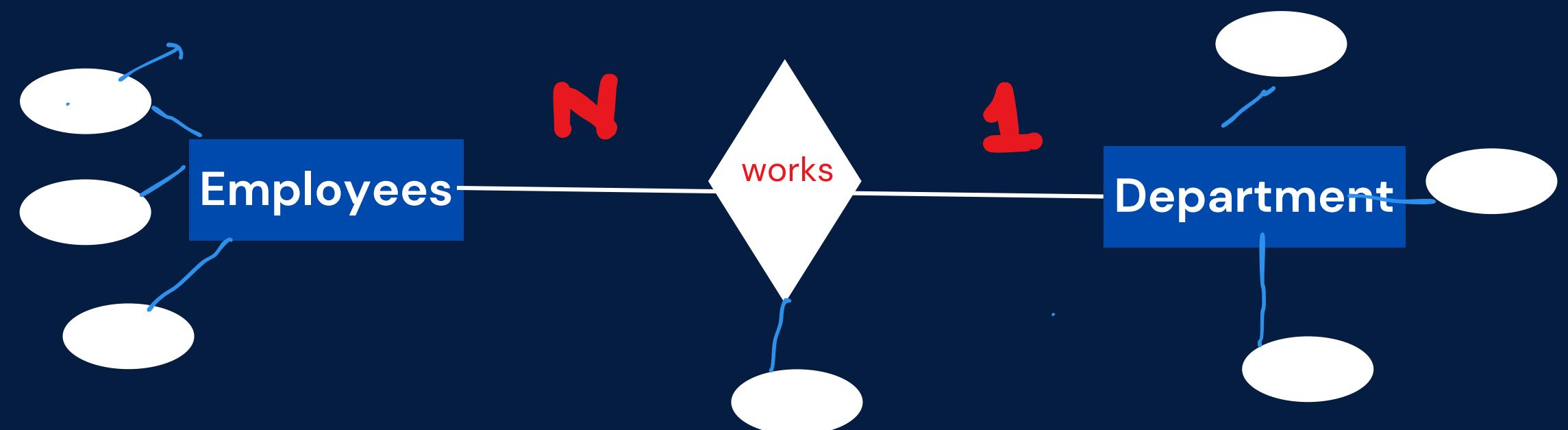
Many to 1 Relationship(N:1).

A database model where multiple entities (records) on one side of the relationship are associated with a single entity (record) on the other side.

ER MODEL IN DBMS

Types of Relationship

Many to 1 Relationship(N:1).

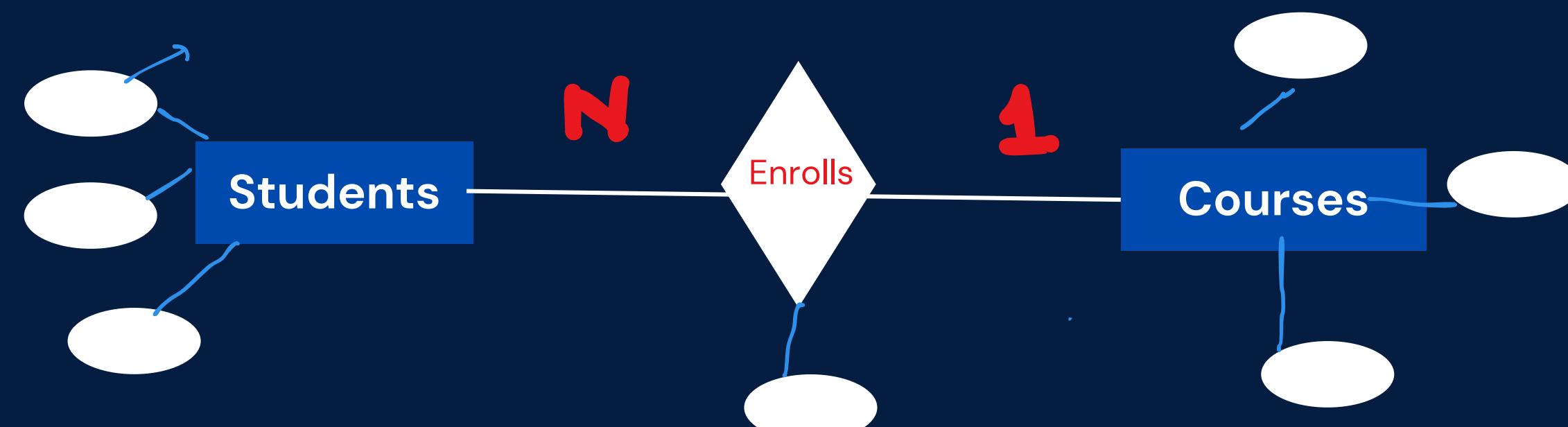


ER MODEL IN DBMS

Types of Relationship

Many to many Relationship(N:N)

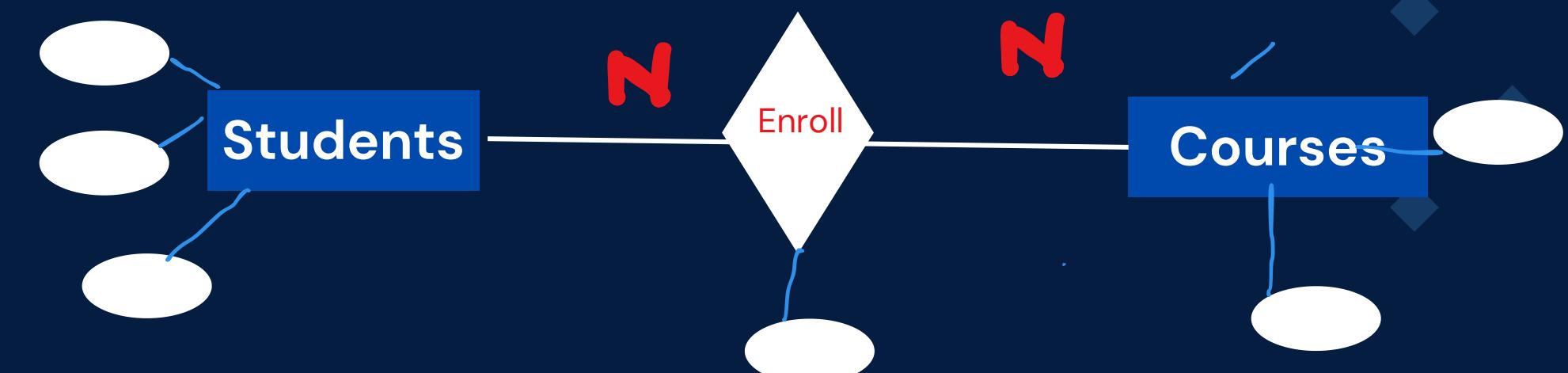
A database model where multiple entities (records) on one side of the relationship are associated with multiple entities on the other side.



ER MODEL IN DBMS

Types of Relationship

Many to many Relationship(N:N)

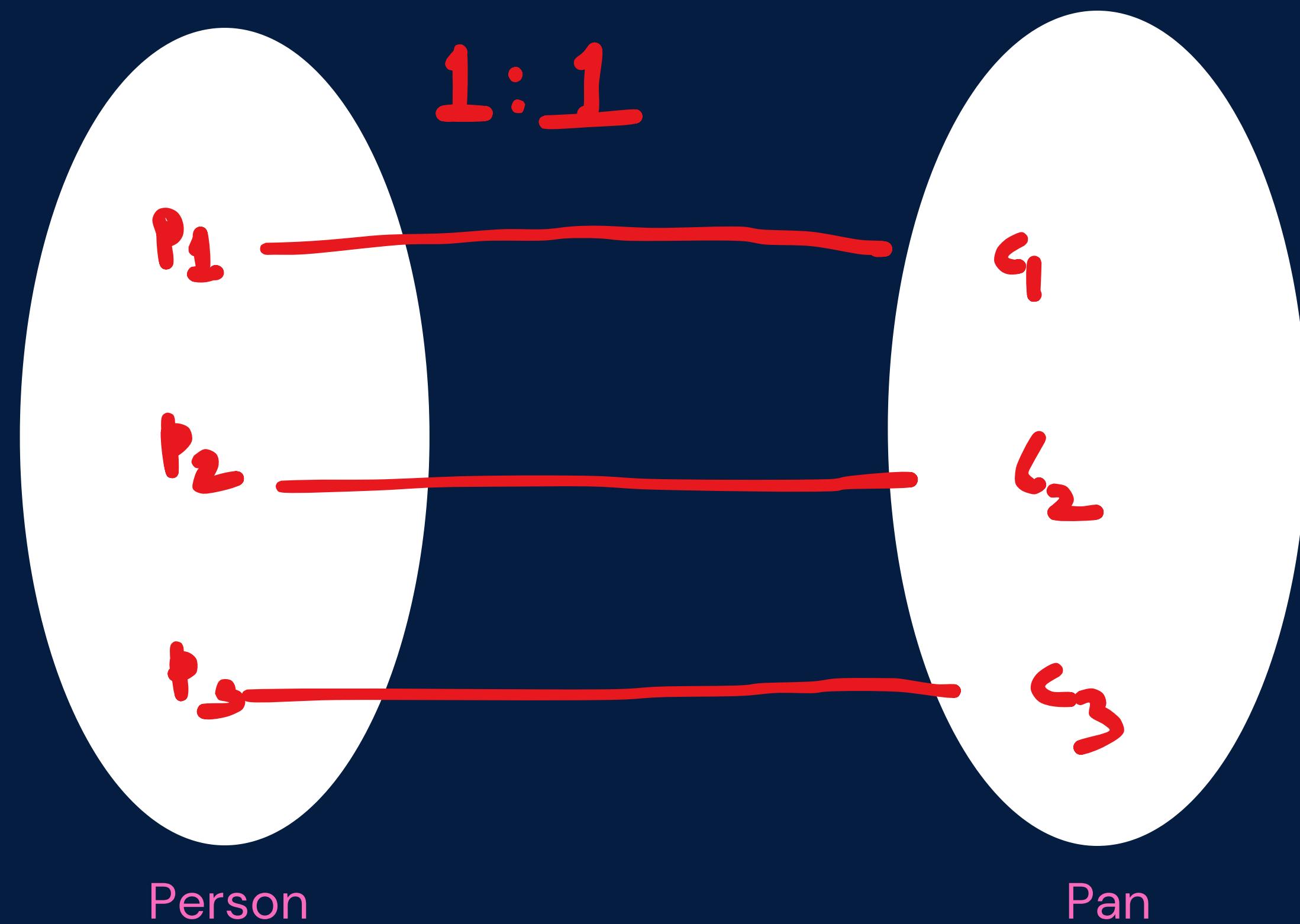


sid	sname	sage
s1	ram	14
s2	raj	15
s3	riti	16

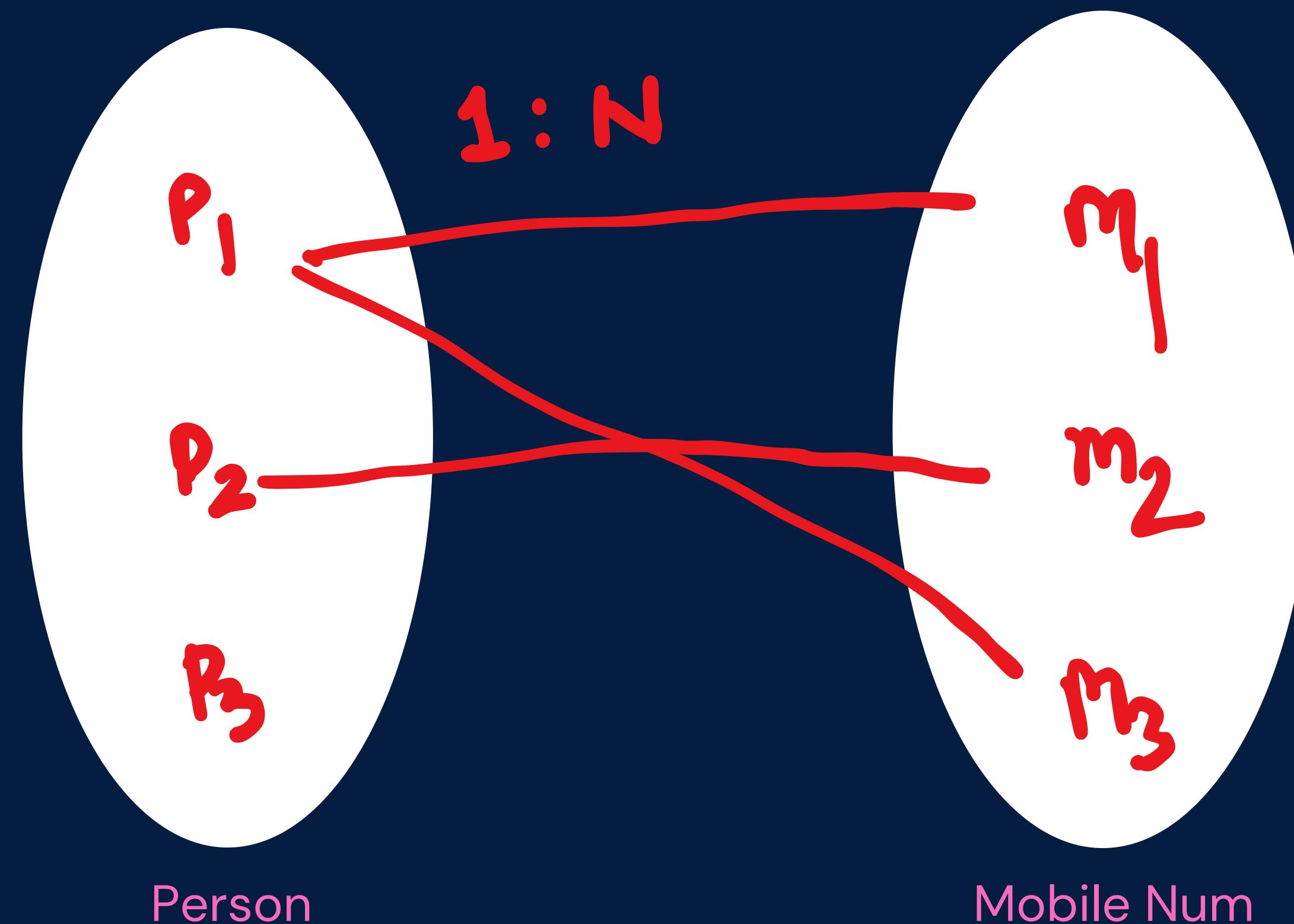
sid	bid	edate
s1	c1	jan
s2	c2	feb
s3	c3	mar

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

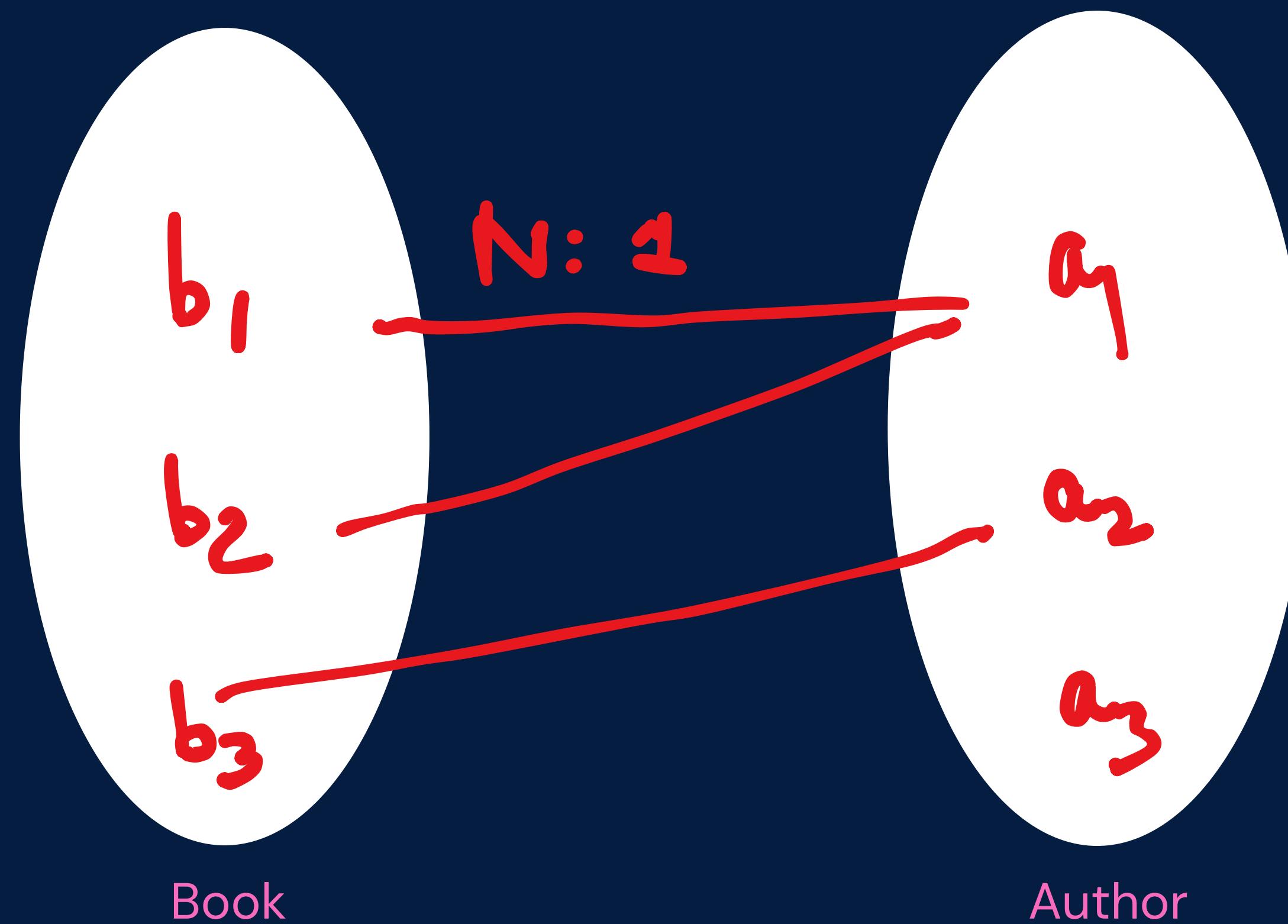
ER MODEL IN DBMS



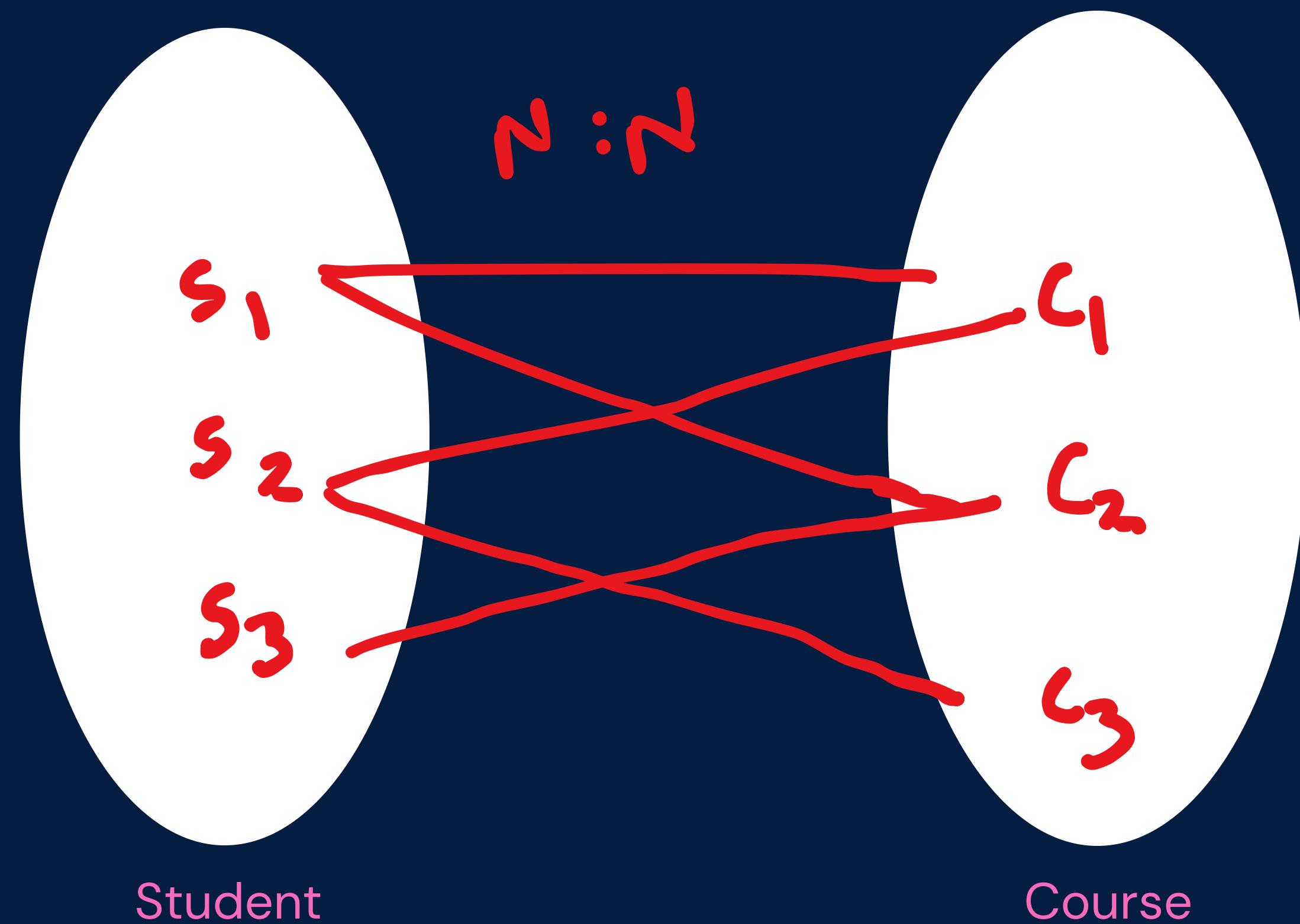
ER MODEL IN DBMS



ER MODEL IN DBMS



ER MODEL IN DBMS



ER MODEL IN DBMS

Participation Constraints

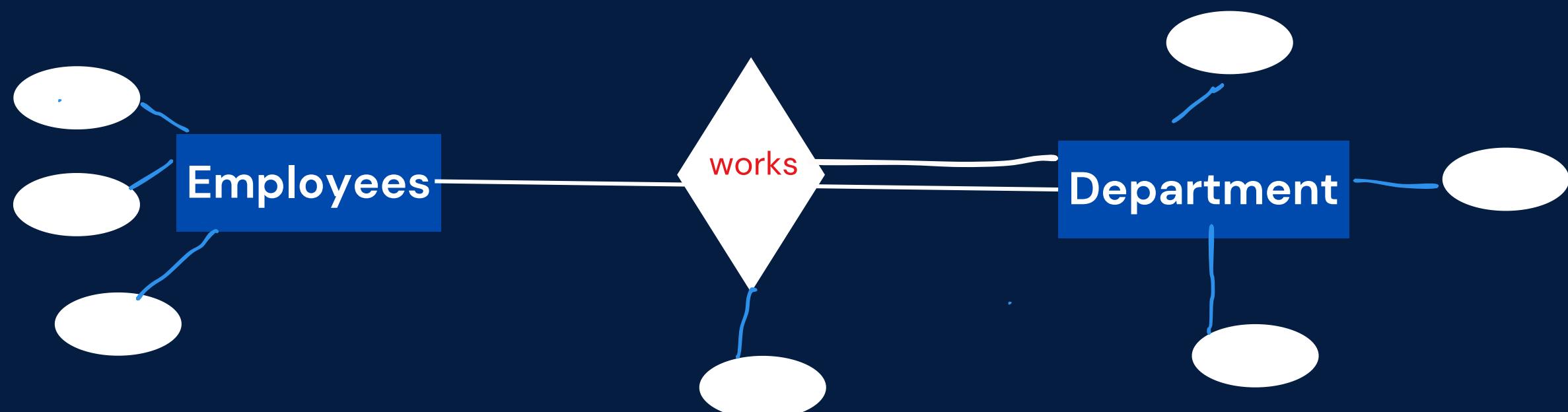
Participation Constraints in an ER model define whether every entity in one group must be connected with at least one entity in another group or if the connection is optional.

ER MODEL IN DBMS

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.

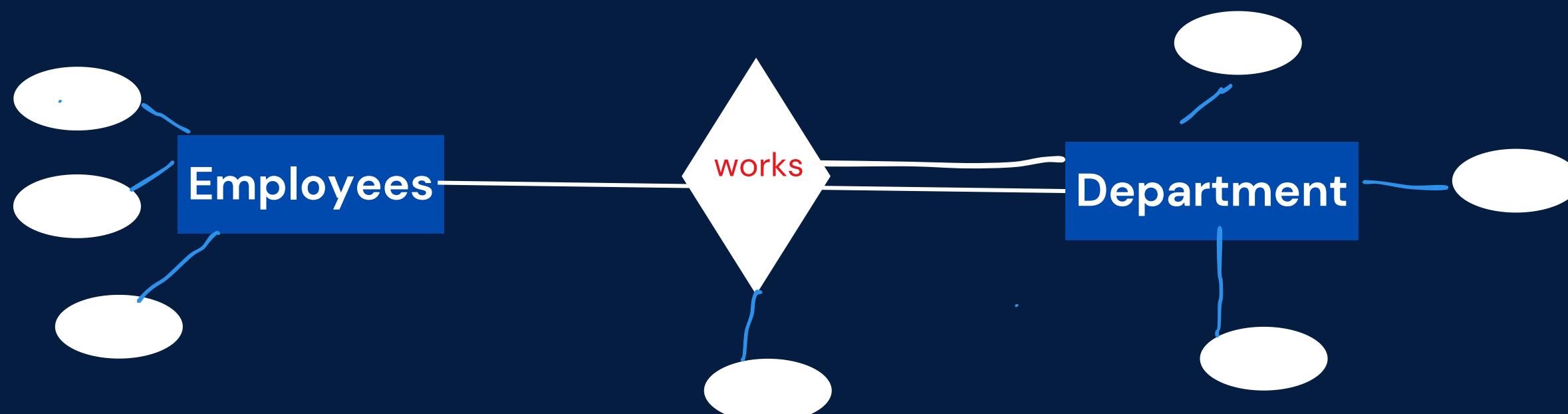


ER MODEL IN DBMS

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

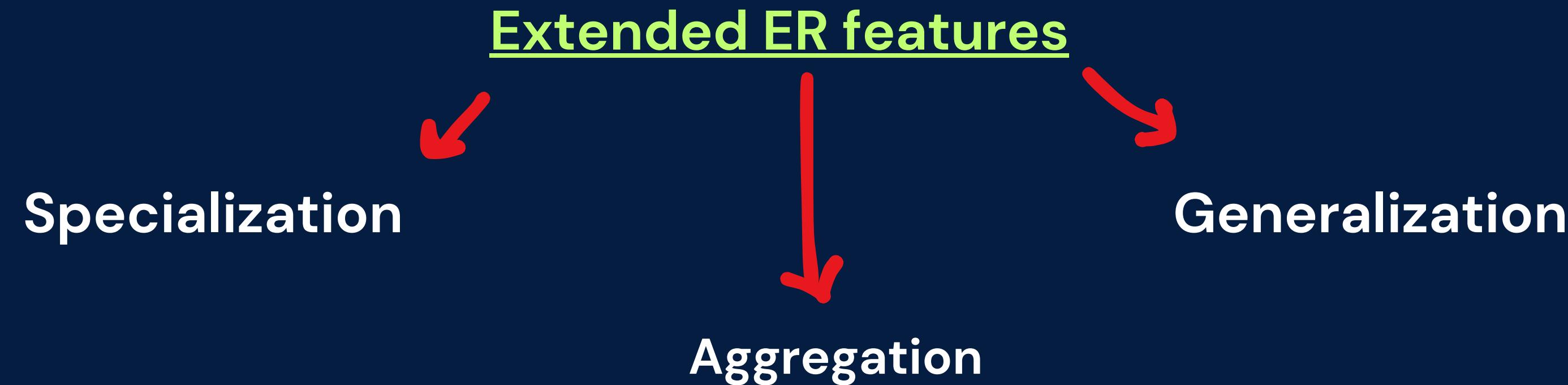
Extended ER features

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



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

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**.

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

ER MODEL IN DBMS

Extended ER features

Inheritance

Attribute



Participation

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

Steps to draw an ER model

1. Recognize entities.
2. Specify entity characteristics/attributes.
3. Discover connections/relationships(also constraints like mapping/participation)
4. Define the connection type (how entities connect)/cardinality.
5. Construct an ERD (Entity–Relationship Diagram).
6. Annotate relationships and attributes.
7. Review and refine the model.
8. Document the model.
9. Validate with stakeholders.
10. Implement the database schema.

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

age- derived attribute

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

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

ER MODEL IN DBMS

ER Model of Instagram

Step-2 : Discover connections/relationships(also constraints like mapping/participation)

1.userProfile have userFriends (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-2 : 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

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

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**

RELATIONAL MODEL

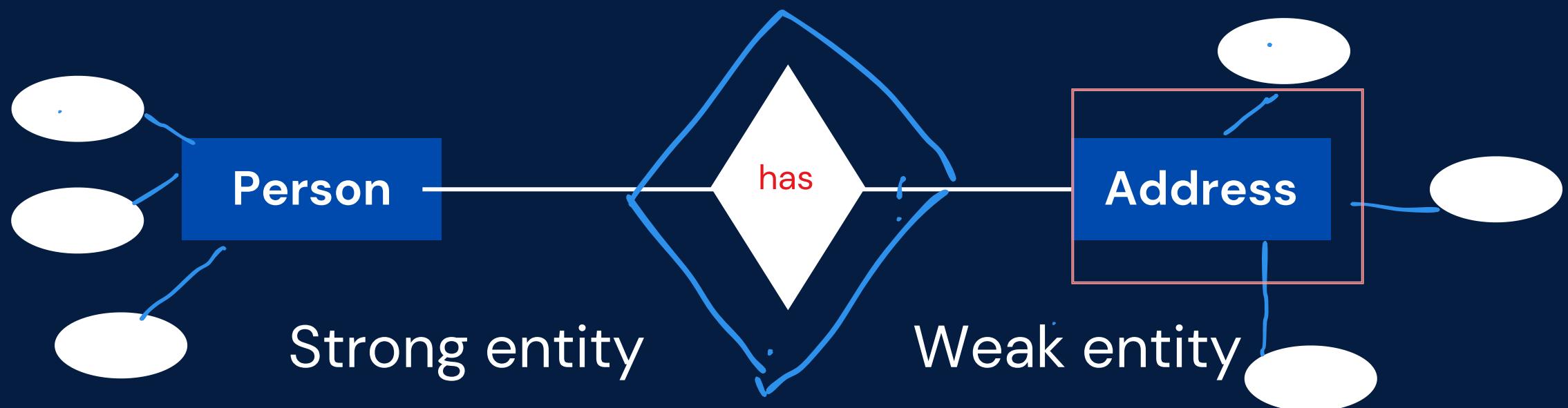
In relational model we take care of different things like:

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.



Person (id, name , age) -> id (p.k)

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

Composite attribute

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 LEARN
COMPLETE SQL IN ONE VIDEO

-By Riti Kumari



LET'S START WITH SQL :)

Database : Collection of data is called as database.

DBMS: A software application to manage our data.



LET'S START WITH SQL :)

Database

Relational(Use tables to store data)

MySQL

Oracle

MariaDB,

Non-relational(Data is not stored in tables)

MongoDb

LET'S START WITH SQL :)

Why SQL?

We need a language to interact with databases.

So we use SQL to interact with DB, do some CRUD operations on the data.

Then what is MySQL?

MySQL is a specific Relational Database Management System (RDBMS) that uses SQL as its querying language.

LET'S START WITH SQL :)

History of SQL (Structured Query Language)

SQL originated in the 1970s from IBM's research on relational databases. It started as SEQUEL, later renamed SQL due to trademark issues.

LET'S START WITH SQL :)

SQL (Structured Query Language)

SQL is a programming language that is used to communicate and manipulate data in databases.

It helps user in performing CRUD (Create, Read, Update, Delete) operations in DB.

LET'S START WITH SQL :)

How SQL helps us ?

SQL allows users to perform a variety of tasks related to databases

- **Retrieving Data**: Extracting precise information from a database through queries.
- **Manipulating Data**: Adding, modifying, or removing records within a database.
- **Defining Data**: Creating and adjusting the structure of a database, including tables, views, and indexes.
- **Controlling Data**: Managing database access by granting or revoking permissions.

LET'S START WITH SQL :)

Installation of MySQL

MySQL Server : Database server where data is stored, managed, and accessed.

MySQL WorkBench : It is a visual tool which is used for database design, development, administration, and management.

It provides a user interface (UI) to interact with MySQL Server.

LET'S START WITH SQL :)

Lets install the Server first:

- Go to the MySQL Official website: <https://www.mysql.com/>
- Go to Downloads
- Select MySQL Community (GPL) Downloads at the bottom of the page.
- Choose MySQL Community Server, select the version and click on download
- Follow the instructions and set the root password. This password would be asked while creating a new connection.

LET'S START WITH SQL :)

Lets install the WorkBench:

- Go to the MySQL Official website: <https://www.mysql.com/>
- Go to Downloads
- Select MySQL Community (GPL) Downloads at the bottom of the page.
- Choose MySQL Workbench, select the version and click on download
- Follow the instructions .

LET'S START WITH SQL :)

Types of SQL Commands

SQL commands are divided into different categories based on their functionalities.

1. **Data Query Language (DQL) Commands**
2. **Data Manipulation Language (DML) Commands**
3. **Data Definition Language (DDL) Commands**
4. **Data Control Language (DCL) Commands**
5. **Transaction Control Language (TCL) Commands**

LET'S START WITH SQL :)

Types of SQL Commands

1. Data Query Language (DQL) Commands

DQL is used to retrieve data from the database

Commands: SELECT

2. Data Manipulation Language (DML) Commands

DML is used to manipulate data stored in the database.

Commands: INSERT, UPDATE, DELETE

LET'S START WITH SQL :)

Types of SQL Commands

3. Data Definition Language (DDL) Commands

DDL is used to define the structure and schema of the database.

Commands: CREATE, ALTER, DROP, TRUNCATE, RENAME

4. Data Control Language (DCL) Commands

DCL deals with the control and security of data within the database.

Commands: GRANT, REVOKE

LET'S START WITH SQL :)

Types of SQL Commands

5. Transaction Control Language (TCL) Commands

TCL is used to manage transactions within a database.

Commands: COMMIT, ROLLBACK, SAVEPOINT

LET'S START WITH SQL :)

Creation of Database

Lets understand database design from an example, Consider a college database.

Database- School

table1- Student (Sname, Rollno)

table2-Teacher(Tname,Tid)

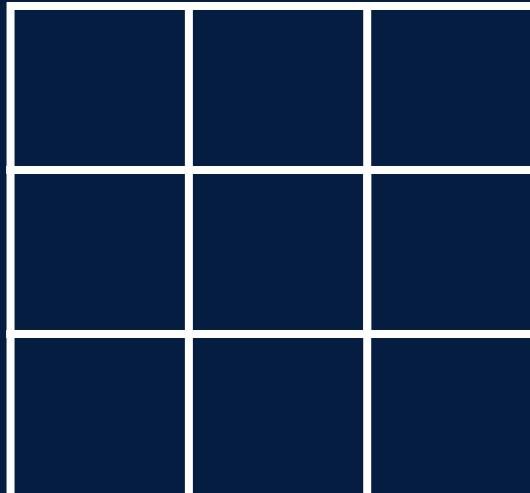
Sname, Rollno, Tname, Tid->attributes(characteristics)

LET'S START WITH SQL :)

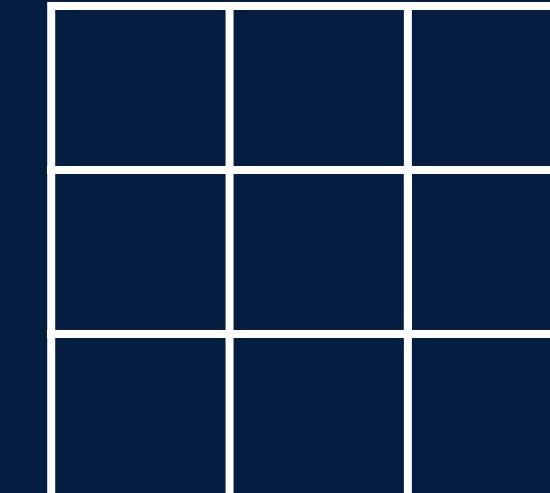
Creation of Database



School



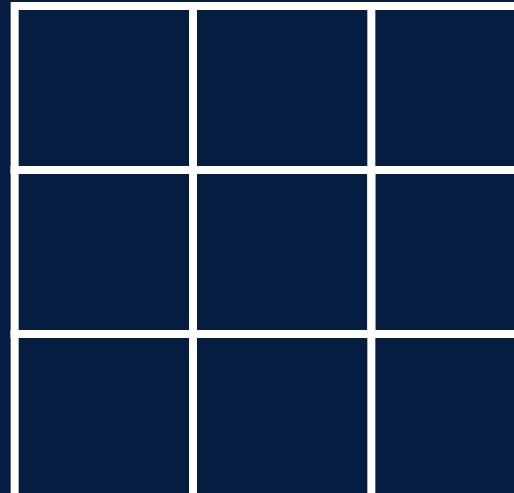
Course



Fees



Hospital



Patient

LET'S START WITH SQL :)

Creation of Database

Steps to create a Database :

1. Choose a DBMS(Database Management System)
2. Connect to the server using a command-line tool or a graphical user interface.
3. Create a new Database
4. Once the database is created, you can use the USE statement to create tables in the database.
5. Create Tables and Insert Data

LET'S START WITH SQL :)

Creation of Database

Creating a new Database

We use the **CREATE DATABASE** statement to create a new database

These commands are not case-sensitive.

Command: **CREATE DATABASE dbName;**

Also to avoid errors we can use:

Command: **CREATE DATABASE IF NOT EXISTS dbName;**

IF NOT EXISTS and **IF EXISTS** clauses are commonly used in conjunction with the **CREATE TABLE** and **DROP TABLE** statements to avoid errors

LET'S START WITH SQL :)

Deletion of Database

Deleting a Database

We use the **DROP DATABASE** statement to delete a database.

Dropping a database means deleting the entire database, including all tables, data, and other objects within it. DROP Is a DDL Command.

These commands are not case-sensitive.

Command: **DROP DATABASE databaseName;**

Also to avoid errors we can use:

Command : **DROP DATABASE IF EXISTS databaseName;**

LET'S START WITH SQL :)

Using a Database

Using a Database

We use the USE DATABASE statement to use a database

These commands are not case-sensitive.

Command: **USE databaseName;**

LET'S START WITH SQL :)

Showing all the Database

Showing a Database

We use the SHOW DATABASES statement to see all the databases present in a server.

Command: **SHOW DATABASES;**

LET'S START WITH SQL :)

Table

Creating a table

We use the **CREATE TABLE** statement to craete a table in DB.

Command:

```
CREATE TABLE TableName (
    Column1 DataType1 Constraint1,
    Column2 DataType2 Constraint2,
    Column3 DataType3 Constraint3,
    -- additional columns if needed
);
```

LET'S START WITH SQL :)

Creating a table

CREATE- DDL Command

Example:

```
CREATE TABLE employee (
    emplId INT PRIMARY KEY,
    name VARCHAR(50),
    salary INT
);
```

employee		
empld	name	salary

LET'S START WITH SQL :)

Inserting values into table

INSERT- DML Command

```
INSERT INTO tableName (Column1, Column2... ColumnN)  
VALUES (value1,value2.....valuen)
```

LET'S START WITH SQL :)

Inserting values into table

INSERT- DML Command

Example:

INSERT INTO employee

(empld,name,salary)

VALUES

(1,"Raj",1200),

(2,"Rahul",1100),

(3,"Riti",1100);

employee

empld	name	salary
1	Raj	1200
2	Rahul	1100
3	Ram	1100

LET'S START WITH SQL :)

Inserting values into table

INSERT- DML Command

Example:

```
INSERT INTO employee VALUES  
(1,"Raj",1200),  
(2,"Rahul",1100),  
(3,"Riti",1100);
```

employee		
empld	name	salary
1	Raj	1200
2	Rahul	1100
3	Ram	1100

LET'S START WITH SQL :)

Seeing tables in a given Database

SHOW

Example:

SHOW TABLES;

It helps us to see all the tables in a given database.

employee

empld	name	salary
1	Raj	1200
2	Rahul	1100
3	Ram	1100

LET'S START WITH SQL :)

To see all the values in table

SELECT

Example:

To see specific values of a column:

SELECT empld FROM employee;

To see all the values or the entire table

SELECT * FROM employee;

employee

empld	name	salary
1	Raj	1200
2	Rahul	1100
3	Ram	1100

LET'S START WITH SQL :)

- Let's Create a Database for Instagram

Step 1: Create a database

```
CREATE DATABASE IF NOT EXISTS instagramDb;
```

Step 2: Use the database to create tables

```
USE instagramDb;
```

LET'S START WITH SQL :)

Step 3 : Create tables into the db

```
CREATE TABLE IF NOT EXISTS users (
    userId INT PRIMARY KEY,
    userName VARCHAR(50),
    email VARCHAR(100)
);
```

```
CREATE TABLE IF NOT EXISTS posts (
    postId INT PRIMARY KEY,
    userId INT,
    caption VARCHAR(100),
);
```

LET'S START WITH SQL :)

Step 3 : Insert Values in the tables

```
INSERT INTO users (userId, userName, email)  
VALUES  
(1, "riti", "abc@gmail.com),  
(1, "raj", "xyz@gmail.com),  
(1, "rahul", "abc2@gmail.com);
```

```
INSERT INTO posts (postId, userId, caption)  
VALUES  
(101, 561, "light"),  
(102, 562, "air"),  
(103, 563, "water");
```

LET'S START WITH SQL :)

Step 4 : You can see all the tables in the db

```
USE DATABASE instagramDb;  
SHOW TABLES;
```

Step 5: All the values in a specific table

```
SELECT * FROM users;  
SELECT * FROM posts;
```

LET'S START WITH SQL :)

Datatypes in SQL

Data types are used to specify the type of data that a column can store.

Numeric	Character/ String	Date & Time	Boolean	Binary
<ul style="list-style-type: none">• INTEGER/ INT• SMALLINT• BIGINT• DECIMAL• FLOAT• DOUBLE	<ul style="list-style-type: none">• CHAR(n)• VARCHAR(n)• TEXT	<ul style="list-style-type: none">• DATE• TIME• DATETIME• TIMESTAMP	<ul style="list-style-type: none">• BOOLEAN	<ul style="list-style-type: none">• BINARY(n)• VARBINARY(n)• BLOB

LET'S START WITH SQL :)

Datatypes

Numeric Datatypes

1. INT - Used for storing whole numbers without decimal points.
(-2,147,483,648 to 2,147,483,647 (signed integer))
2. BIGINT - Used for storing large whole numbers. (-9,223,372,036,854,775,808
to 9,223,372,036,854,775,807)
3. FLOAT- Used for storing decimal numbers. (4-byte)

LET'S START WITH SQL :)

Numeric Datatypes

Datatypes

3. FLOAT- Used for storing decimal numbers. (4-byte)
4. DOUBLE- Used for storing decimal numbers. (8-byte)
5. DECIMAL(p,s)- Used for exact numeric representation. p is the precision and s is the scale.

Command :

```
CREATE TABLE example1(  
    id INT  
)
```

LET'S START WITH SQL :)

Numeric Datatypes

By default all the numeric datatypes can have negative as well as positive values. This restrict the range so if we know there is only +ve values which is stored we use UNSIGNED attribute (0-255).

for eg- salary can never be in negative or age

Command :

```
CREATE TABLE example1(  
    id INT UNSIGNED  
)
```

Datatypes

LET'S START WITH SQL :)

Datatypes

Character Datatypes

1. CHAR(n)- Fixed-length character strings can be stored. (0-255)
2. VARCHAR(n)- Variable-length character strings can be stored.(0-255)
3. TEXT- Variable-length character string with no specified limit.

Command :

```
CREATE TABLE example1 (
    name VARCHAR(50)
);
```

LET'S START WITH SQL :)

Datatypes

Date & Time Datatypes

1. DATE- Used for storing date values. (YYYY-MM-DD)
2. TIME - Used for storing time values. (hh:mm:ss)
3. DATETIME/TIMESTAMP- Used for storing date and time values. (yyyy-mm-dd hh:mm:ss)

Command :

```
CREATE TABLE example1(  
createdTs TIMESTAMP  
);
```

LET'S START WITH SQL :)

Datatypes

Boolean Datatypes

1. BOOLEAN- Used to store a true or false value.

Command :

```
CREATE TABLE example1 (  
isActive BOOLEAN  
);
```

LET'S START WITH SQL :)

Datatypes

Binary Datatypes

1. **BINARY(n)**- Used for fixed-length binary data.
2. **VARBINARY(n)**- Used for storing variable-length binary data.
3. **BLOB (Binary Large Object)**- Used for storing large amounts of binary data.(var len)

Command :

```
CREATE TABLE document (
  data BLOB
);
```

LET'S START WITH SQL :)

Constraints in SQL

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.
- Primary key : Enforces the uniqueness of values in one or more columns
- Foreign key: Enforces a link between two tables by referencing a column in one table that is a primary key in another table.

LET'S START WITH SQL :)

Constraints in SQL

Unique constraint:

```
CREATE TABLE example1(  
phoneNbr INT UNIQUE);
```

Not null constraint:

```
CREATE TABLE example1(  
address VARCHAR(50) NOT NULL );
```

Check constraint:

```
CREATE TABLE example1(  
age INT CHECK (age >= 18));
```

Default constraint:

```
CREATE TABLE example1(  
enrolled VARCHAR(20) DEFAULT 'no' );
```

LET'S START WITH SQL :)

Constraints in SQL

Primary key constraint:

```
CREATE TABLE employee ( id INT PRIMARY KEY, name VARCHAR(255) );
```

or

```
CREATE TABLE employee (
    id INT ,
    name VARCHAR(255)
    PRIMARY KEY (id)
);
```

Foreign key constraint:

```
CREATE TABLE orders (
    orderItemNo INT PRIMARY KEY,
    custId INT,
    FOREIGN KEY (custId) REFERENCES customer(custId) );
```

LET'S START WITH SQL :)

Keys in SQL

Primary key- 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.

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

LET'S START WITH SQL :)

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

LET'S START WITH SQL :)

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

LET'S START WITH SQL :)

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

LET'S START WITH SQL :)

Foreign Key

id	Name	course_id
1	Rahul	100
2	Raj	101
3	Riti	102

Student

Base/referenced/parent table

course_id	name	teacher	id
100	Hindi	Ram	1
101	Maths	Mohan	2
102	English	Priya	3

Course

Refrencing/child table

LET'S START WITH SQL :)

Foreign Key

Foreign key helps to perform operations related to the parent table, such as joining tables or ensuring referential integrity.

Query :

```
CREATE TABLE childtableName (
    childId INT PRIMARY KEY,
    baselId INT,
    FOREIGN KEY (baselId) REFERENCES baseTableName(baselId)
);
```

LET'S START WITH SQL :)

Cascading in Foreign Key

Cascading are a set of rules which dictate what actions should be taken automatically when a referenced row in the parent table is modified or deleted.

1. **CASCADE**: If a row in the parent table is updated or deleted, all related rows in the child table will be automatically updated or deleted.

2. **SET NULL**: If a row in the parent table is updated or deleted, all corresponding foreign key values in the child table will be set to NULL.

3. **RESTRICT or NO ACTION**: Blocks the modification or deletion of a referenced row in the parent table if related rows exist in the child table, thus maintaining referential integrity.

LET'S START WITH SQL :)

Cascading in Foreign Key

These cascading actions help maintain the integrity of the data across related tables in the database.

- 1.ON DELETE CASCADE
- 2.ON UPDATE CASCADE

LET'S START WITH SQL :)

Cascading in Foreign Key

1. **ON DELETE CASCADE** : The ON DELETE CASCADE clause indicates that if a row in the parent table (parent_table) is deleted, all corresponding rows in the child table (child_table) will be automatically deleted as well.

QUERY:

```
CREATE TABLE childtableName (
    childId INT PRIMARY KEY,
    baseld INT,
    FOREIGN KEY (baseld) REFERENCES baseTableName(baseld)
    ON DELETE CASCADE
);
```

LET'S START WITH SQL :)

Cascading in Foreign Key

2. **ON UPDATE CASCADE** : The ON UPDATE CASCADE clause indicates that if a row in the parent table (parent_table) is updated, all corresponding rows in the child table (child_table) will be automatically updated as well.

QUERY :

```
CREATE TABLE childtableName (
    childId INT PRIMARY KEY,
    baseld INT,
    FOREIGN KEY (baseld) REFERENCES parenttableName(childId)
    ON UPDATE CASCADE
);
```

LET'S START WITH SQL :)

Lets make a database for all SQL commands

Let's make a Database for a Company

Requirements :

1. Make a database for a company xyz

CREATE DATABASE xyz;

2. Make an employee table in the xyz database.

CREATE TABLE employee(
id INT PRIMARY KEY,
name VARCHAR(50),
age INT,
department VARCHAR(50)
city VARCHAR(50),
salary INT);

LET'S START WITH SQL :)

Retrieving data from table

3. Fill details in the table

```
INSERT INTO employee(id,name,age,department,city,salary)  
VALUES  
(1, "rahul" , 25 , "IT" , "Mumbai", 1500),  
(2, "afsara" , 26 , "HR" , "Pune, 2000),  
(3, "abhimanyu" , 27 , "IT" , "Mumbai" , 2500),  
(4, "aditya" , 25 , "Marketing" , "Surat" , 2400),  
(5, "raj" , 24, "Finance" , "Indore", 1500);
```

4. See all the data in the table

LET'S START WITH SQL :)

UPDATE Command

The UPDATE command in SQL is used to modify existing records in a table.
If you get a safe mode error while executing queries run this query

QUERY: **SET SQL_SAFE_UPDATES=0;**

QUERY :

UPDATE table_name

SET columnName1= value1(to be set) , columnName2 =value2(to be set)

WHERE condition;

LET'S START WITH SQL :)

UPDATE Command (Practice Question)

1. Write a query to update the salary for all employees in the 'HR' department to 50000.

QUERY :

```
UPDATE employee  
SET salary = 50000  
WHERE department = "HR";
```

LET'S START WITH SQL :)

UPDATE Command (Practice Question)

2. Write a query to update the name of an employee from raaj to raj .

QUERY :

```
UPDATE employee  
SET name = "raj"  
WHERE name = "raaj";
```

LET'S START WITH SQL :)

DELETE Command

The DELETE command in SQL is used to remove records from a table.

QUERY:

```
DELETE FROM table_name  
WHERE condition;
```

LET'S START WITH SQL :)

DELETE Command (Practice Question)

1. Write a query to DELETE all records from the employee table where the department is 'HR'

QUERY :

```
DELETE FROM employee  
WHERE department = "HR";
```

LET'S START WITH SQL :)

DELETE Command (Practice Question)

2. Write a query to DELETE the record of an employee having name as raj

QUERY :

```
DELETE FROM employee  
WHERE name = "raj";
```

LET'S START WITH SQL :)

Retrieving data from table

SELECT command – Select is a DQL(Data Query Language) Command. It is used to retrieve data from a database.

We can provide specific columns from which we can retrieve data.

SELECT column1, column2 FROM tableName; –> to retrieve data present in specific column in a table

SELECT * FROM tableName; –> to retrieve all the data present in table

LET'S START WITH SQL :)

Filtering data using the WHERE clause

WHERE clause - It filters the rows based on specified conditions.

QUERY : **SELECT col1 col2 FROM tableName WHERE condition;**

ex : **SELECT * FROM employee WHERE age > 20;**

LET'S START WITH SQL :)

SQL Commands

DQL

SELECT

DML

INSERT
UPDATE
DELETE

DDL

CREATE
ALTER
DROP
TRUNCATE
RENAME

LET'S START WITH SQL :)

ALTER Command

ALTER command - ALTER is a DDL command used to modify(change) existing database objects, such as tables, indexes, or constraints(schema)

Let's see all the things ALTER can help us to do. So mostly it is used to modify the schema, so we will mostly see how it can help in modification of columns like - addition of new column, deletion of column, modification of column and much more

LET'S START WITH SQL :)

ALTER Command

1. ADD a column

Query :

```
ALTER TABLE tableName  
ADD columnName datatype constraint ;
```

2. Drop a column

Query :

```
ALTER TABLE tableName  
DROP COLUMN columnName ;
```

LET'S START WITH SQL :)

ALTER Command

3. Modify the data type of an existing column

MODIFY clause : The MODIFY clause is oftenly used within an ALTER TABLE statement in SQL. It allows us to change the definition or properties of an existing column in a table.

Query :

```
ALTER TABLE tableName  
MODIFY columnName newdatatype ;
```

The above command modifies columnName to a new dataType.

LET'S START WITH SQL :)

ALTER Command

4. Change the name of an existing column

CHANGE: The CHANGE command is oftenly used within an ALTER TABLE statement in SQL. It helps to change the name or data type of a column within a table.

Query :

ALTER TABLE tableName

CHANGE oldcolumnName newcolumnName newdatatype;

The above command changes the oldcolumnName to newcolumnName and also its datatype

LET'S START WITH SQL :)

ALTER Command

4. Rename the name of an existing column

RENAME COMMAND : RENAME command is used to change the name of an existing database object, such as a table, column, index, or constraint.

Query :

ALTER TABLE tableName

RENAME COLUMN oldcolumnName TO newcolumnName ;

The above command renames the oldcolumnName to newcolumnName

LET'S START WITH SQL :)

RENAME Command

RENAME : RENAME command is used to change the name of an existing database object, such as a table, column, index, or constraint.

Query (Table Renaming) :

```
RENAME TABLE oldTableName TO newTableName ;
```

The above command renames the oldTableName to newTableName

LET'S START WITH SQL :)

RENAME Command

Query (Column Renaming) :

```
ALTER TABLE tablename  
RENAME COLUMN oldcolumnname TO newcolumnname;
```

The above command renames the oldcolumnName to newcolumnName

Query (Database Renaming) :

```
RENAME DATABASE olddatabasename TO newdatabasename;
```

LET'S START WITH SQL :)

TRUNCATE Command

TRUNCATE command – This command removes all rows from the given table, leaving the table empty but preserving its structure,

QUERY :

TRUNCATE TABLE tableName;

LET'S START WITH SQL :)

Difference Between TRUNCATE, DELETE and DROP

TRUNCATE	DELETE	DROP
remove all rows from a table	Used to remove specific rows from a table based on a condition	Used to completely remove table
TRUNCATE TABLE tablename;	DELETE FROM tablename WHERE condition;	DROP TABLE tablename;

LET'S START WITH SQL :)

Using DISTINCT to retrieve unique values

DISTINCT – DISTINCT keyword is used within the SELECT statement to retrieve unique values from a column or combination of columns.

Query :

```
SELECT DISTINCT col1  
FROM tableName;
```

→ retrieve a list of unique values for col1

```
SELECT DISTINCT col1, col2  
FROM tableName;
```

→ return unique combinations of col1 & col2

LET'S START WITH SQL :)

Operators in SQL

To perform operations on data in SQL we use operators.

QUERY : **SELECT col1 col2 FROM tableName WHERE condition(use operator);**

Types of operators in SQL:

- **Arithmetic Operators**: addition (+) ,subtraction (-), multiplication (*), division (/) , modulus (%)

QUERY : **SELECT * FROM employee WHERE age+1 =60;**

LET'S START WITH SQL :)

Operators in SQL

- Comparison Operators : equal to (=) , not equal to (<> or !=) , greater than (>) less than (<), greater than or equal to (>=), less than or equal to (<=)

QUERY : **SELECT * FROM employee WHERE age > 20;**

LET'S START WITH SQL :)

Operators in SQL

- Logical Operators

1. **AND** : It combines two conditions and returns true if both are true

QUERY: **SELECT * FROM employee WHERE city= 'Pune' AND age > 18;**

2. **OR** : It combines two conditions and returns true if either is true

QUERY: **SELECT * FROM employee WHERE city= 'Pune' OR age > 18;**

3. **NOT**: It reverses the result of a condition, returns true if the condition is false

QUERY: **SELECT * FROM employee WHERE department NOT IN ('IT', 'HR');**

LET'S START WITH SQL :)

Operators in SQL

- IN Operator: IN(Checks if a value matches in a list of values)

QUERY : **SELECT * FROM employee WHERE department IN ('IT', 'HR');**

- IS NULL / IS NOT NULL Operators : IS NULL (checks for null values) , IS NOT NULL(chcks for not null values)

QUERY : **SELECT * FROM employee WHERE department IS NOT NULL;**

- Bitwise Operators: AND(&), OR(|)

LET'S START WITH SQL :)

Operators in SQL

- **LIKE & Wildcard Operators**: LIKE operator is used to search for a specified pattern in a column. It uses wildcard operators for matching patterns.

1. % (percent sign): It matches for any sequence of zero or more characters.

QUERY : **SELECT * FROM employee WHERE name LIKE 'A%';**

2. _ (underscore): It matches for any single character.

QUERY : **SELECT * FROM employee WHERE name LIKE '_A%';**

LET'S START WITH SQL :)

Operators in SQL

- BETWEEN Operator: Checks if a value is within a range of values.

QUERY : **SELECT * FROM employee WHERE salary BETWEEN 1200 AND 1500;**

LET'S START WITH SQL :)

Clauses in SQL

Clauses are like tools/conditions that helps us to make queries more specific or decide what data to fetch.

Ex- **WHERE, GROUP BY, HAVING , ORDER BY, LIMIT**

QUERY : `SELECT col1,col2
FROM tableName
clause condition;`

LET'S START WITH SQL :)

WHERE clause

WHERE clause - It filters the rows based on specified conditions.

QUERY : **SELECT col1 ,col2
FROM tableName
WHERE condition;**

ex : **SELECT * FROM employee WHERE age > 20;**

LET'S START WITH SQL :)

LIMIT CLAUSE

LIMIT clause – The LIMIT clause in SQL is used to restrict the number of rows returned by a query.

This query retrieves the first n rows from the table.

QUERY :

```
SELECT col1 , col2 FROM tableName  
LIMIT noOfRows;
```

ex : `SELECT * FROM employee LIMIT 2;`

LET'S START WITH SQL :)

Sorting data with the ORDER BY clause.

ORDER BY clause - It is used to sort the results in ascending or descending order. By default it returns the result in ascending order

This query retrieves the first n rows from the table.

QUERY :

```
SELECT col1 , col2 FROM tableName  
ORDER BY col1 (ASC/DESC), col2 (ASC/DESC)
```

ex : `SELECT * FROM employee ORDER BY salary DESC;`

LET'S START WITH SQL :)

Practice question

Write a SQL Query to fetch the details of employees having id as 1

QUERY :

```
SELECT * FROM employee  
WHERE id=1;
```

LET'S START WITH SQL :)

Practice question

Write a SQL Query to fetch the details of employees having id as 1 and city as MUMBAI

QUERY :

```
SELECT * FROM employee  
WHERE id=1 AND city = "MUMBAI";
```

LET'S START WITH SQL :)

Practice question

Write a SQL Query to fetch the details of employees having salary greater than 1200 and city as MUMBAI.

QUERY :

```
SELECT * FROM employee  
WHERE salary>1200 AND city = "MUMBAI";
```

LET'S START WITH SQL :)

Practice question

Write a SQL Query to fetch the details of employees who are not from MUMBAI.

QUERY :

```
SELECT * FROM employee  
WHERE city NOT IN ( "MUMBAI");
```

LET'S START WITH SQL :)

Practice question

Write a SQL Query to fetch the details of employees having the maximum salary.

QUERY :

```
SELECT * FROM employee  
ORDER BY salary DESC;
```

LET'S START WITH SQL :)

Practice question

Write a SQL Query to fetch the details of 2 employees having the maximum salary.

QUERY :

```
SELECT * FROM employee  
ORDER BY salary DESC  
LIMIT 2;
```

LET'S START WITH SQL :)

Aggregate Functions

Aggregate functions performs some operations on a set of rows and then returns a single value summarizing the data. These are used with SELECT statements to perform calculations

Types of **Aggregate functions** :

- COUNT()
- SUM()
- AVG()
- MIN()
- MAX()
- GROUP_CONCAT()

LET'S START WITH SQL :)

Aggregate Functions

COUNT() - It counts the number of rows in a table or the number of non-null values in a column.

This counts how many things are in a list or a group.

Query : `SELECT count(name) FROM employee ;` -> this will tell the number of employees in a company

LET'S START WITH SQL :)

Aggregate Functions

SUM() - It calculates the sum of all values in a numeric column.
This adds up all the numbers in a list.

Query : **SELECT SUM(salary) FROM employee ;** -> this will tell the total amount company is paying to its employees

LET'S START WITH SQL :)

Aggregate Functions

AVG() - It computes the average of all values in a numeric column.
It finds the average, or the "middle" number, of all the numbers in a list.

Query : **SELECT AVG(salary) FROM employee ;** -> this will tell the avg amount
company is paying to its employees

LET'S START WITH SQL :)

Aggregate Functions

MIN() - It helps to find the smallest number in a list.

Query : **SELECT MIN(salary) FROM employee ;** -> this will tell the minimum salary company is paying to its employees

LET'S START WITH SQL :)

Aggregate Functions

MAX() - It finds the maximum value in a column.

Query : **SELECT MAX(salary) FROM employee ;** -> this will tell the max salary company is paying to its employees

LET'S START WITH SQL :)

Grouping data with the GROUP BY clause.

GROUP BY clause - This is used to group rows that have the same values into together. It helps to organize data into groups so that you can do calculations, like finding totals or averages, for each group

This query retrieves the first n rows from the table.

QUERY :

```
SELECT col1, aggregateFun(col2)  
FROM tableName  
GROUP BY col1 ;
```

ex: `SELECT department, AVG(salary) AS avgsal FROM employee
GROUP BY department;`

LET'S START WITH SQL :)

HAVING clause.

HAVING clause - The HAVING clause is just like clause but the main difference is it works on aggregated data. It is used with the GROUP BY clause. It helps to filter groups based on given conditions.

QUERY :

```
SELECT col1, col2 aggregateFun(col3)
FROM tableName
GROUP BY col1 col2
HAVING condition;
```

ex :

```
SELECT department, AVG(salary) AS avgSal
FROM employee
GROUP BY department
HAVING avgSal > 1500;
```

LET'S START WITH SQL :)

GROUP BY and HAVING clause.

These queries demonstrate how to use

- a. GROUP BY to categorize data and
- b. HAVING to filter grouped data based on specific conditions in SQL.

LET'S START WITH SQL :)

Difference Between WHERE and HAVING Clause

WHERE	HAVING
used to filter rows from the result based on condition applied to a row before the aggregation	used to filter rows from the result based on condition applied to a row after the aggregation
It is used with SELECT , UPDATE , or DELETE commands	It is used with GROUP BY and aggregate functions
<code>SELECT * FROM tableName WHERE condition;</code>	<code>SELECT col1, col2 aggregateFun(col3) FROM tableName GROUP BY col1 col2 HAVING condition;</code>

LET'S START WITH SQL :)

Practice Questions

1. Write a query to find the total number of employees in each city

Query :

```
Select city, COUNT(name) AS no_of_emp  
FROM employee  
GROUP BY city;
```

LET'S START WITH SQL :)

Practice Questions

2. Write a query to find the maximum salary of employees in each city in descending order

Query :

```
Select city, max(salary) AS max_salary  
FROM employee  
GROUP BY city  
ORDER BY DESC;
```

LET'S START WITH SQL :)

Practice Questions

3. Write a query to display the department names alongside the total count of employees in each department, sorting the results by the total number of employees in descending order.

Query :

```
SELECT department, COUNT(id) AS totalemployees  
FROM employee  
GROUP BY department  
ORDER BY totalemployees DESC;
```

LET'S START WITH SQL :)

Practice Questions

4. Write a query to list the departments where the average salary is greater than 1200, also display the department name and the average salary.

Query :

```
SELECT department, AVG(salary) AS avgsalary  
FROM employee  
GROUP BY department  
HAVING AVG(salary) > 50000;
```

LET'S START WITH SQL :)

The general order of SQL commands

Sno	Command	Usecase
1.	SELECT	Retrieve from the database
2.	FROM	Identify the table
3.	WHERE	Filter rows based on some conditions
4.	GROUP BY	Group rows that have the same values
5.	HAVING	Filter groups based on some conditions
6.	ORDER BY	Sort the result set either aesc/desc
7.	LIMIT	Limit the number of rows returned

LET'S START WITH SQL :)

Joins in SQL

Joins are used to combine rows from two or more tables based on a related or shared or common column between them. There are commonly 4 types of joins including **INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN, SELF JOIN , CROSS JOIN.**

id	Name	Age
1	Riya	17
2	Rahul	18
3	Ram	17

Student

id	course_id	course_name
1	101	Eng
2	102	Hin
3	103	PhE

Course

LET'S START WITH SQL :)

Joins in SQL

Q. Is Foreign Key important for performing joins?

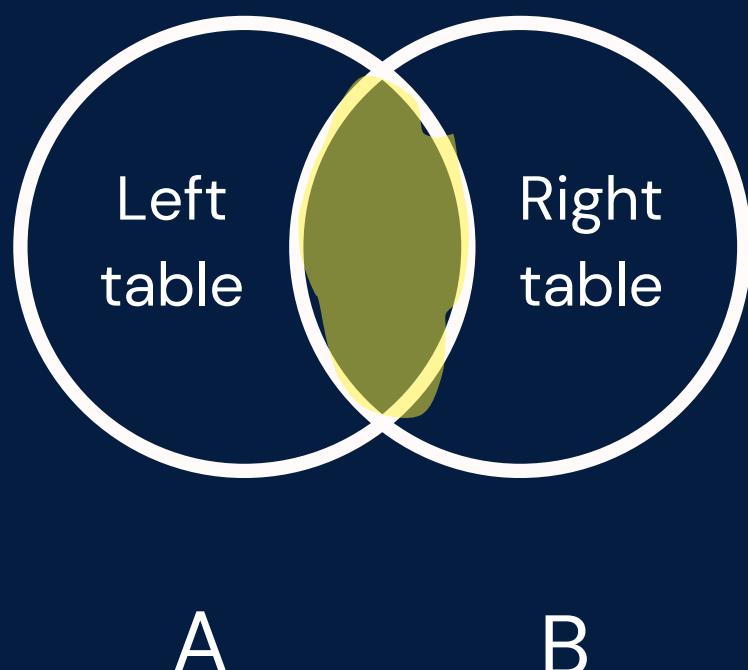
->Joins can be performed based on any columns that establish a relationship between tables, not just FK constraints, so its not necessary.

LET'S START WITH SQL :)

Joins in SQL

Types of Joins in SQL

1. Inner Join



rollno	name
1	Ram
2	Rahul
3	Riti

rollno	c_name
2	Hindi
3	Eng
4	Maths

Student

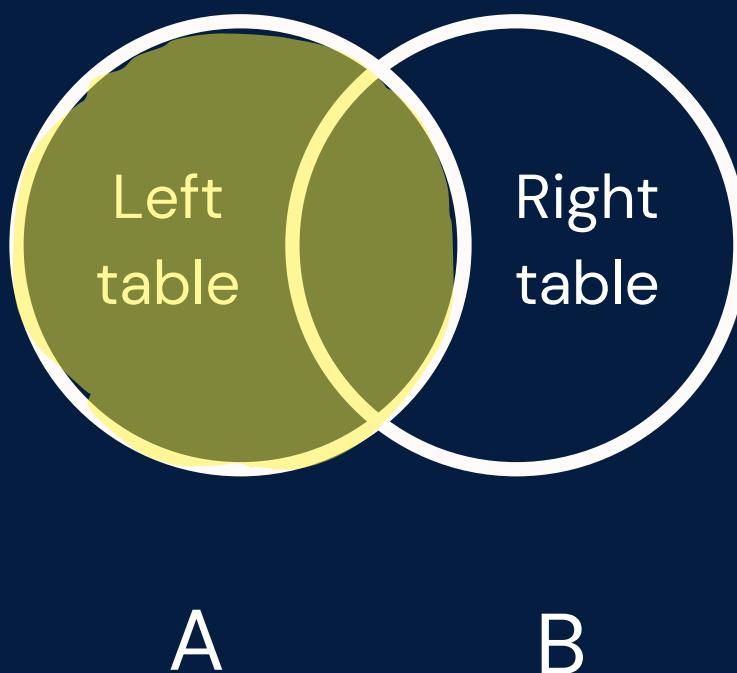
Course

LET'S START WITH SQL :)

Joins in SQL

Types of Joins in SQL

2. Left Join/Left Outer Join



rollno	name
1	Ram
2	Rahul
3	Riti

Student

rollno	c_name
2	Hindi
3	Eng
4	Maths

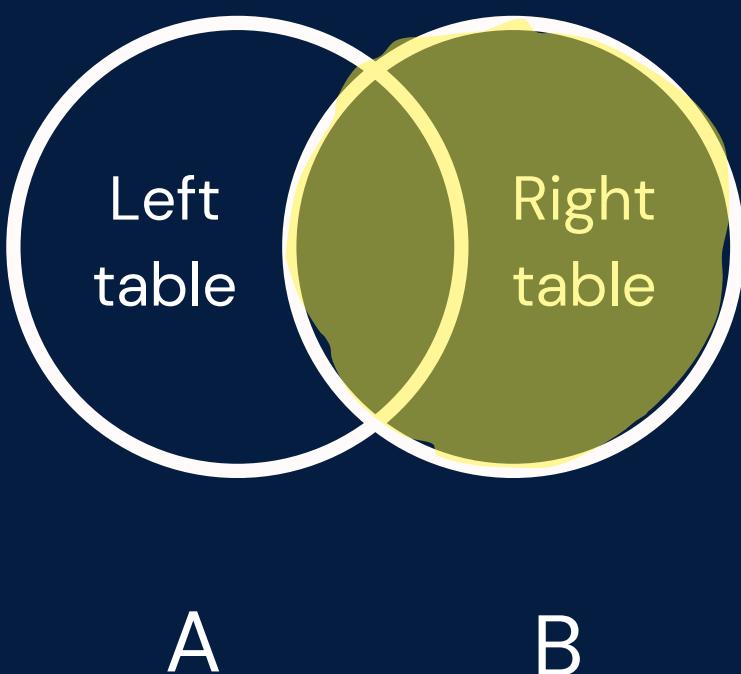
Course

LET'S START WITH SQL :)

Joins in SQL

Types of Joins in SQL

3. Right Join/ Right Outer Join



rollno	name
1	Ram
2	Rahul
3	Riti

Student

rollno	c_name
2	Hindi
3	Eng
4	Maths

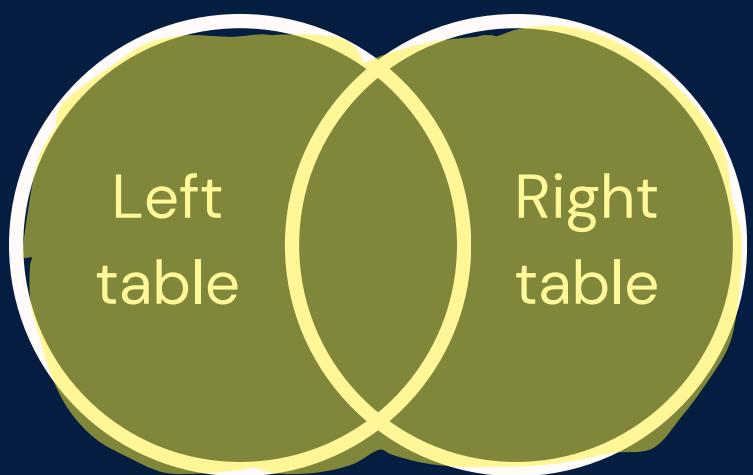
Course

LET'S START WITH SQL :)

Joins in SQL

Types of Joins in SQL

4. Full Join/Full Outer Join



A

B

rollno	name
1	Ram
2	Rahul
3	Riti

Student

rollno	c_name
2	Hindi
3	Eng
4	Maths

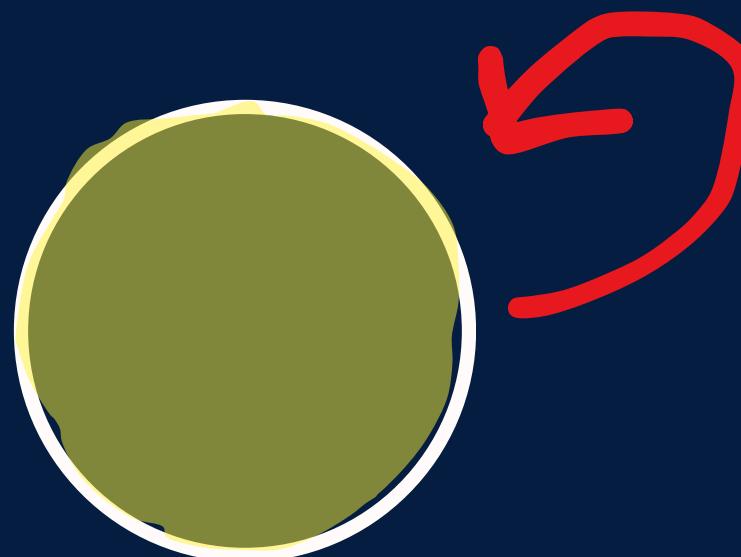
Course

LET'S START WITH SQL :)

Joins in SQL

Types of Joins in SQL

5. Self Join



rollno	name
1	Ram
2	Rahul
3	Riti

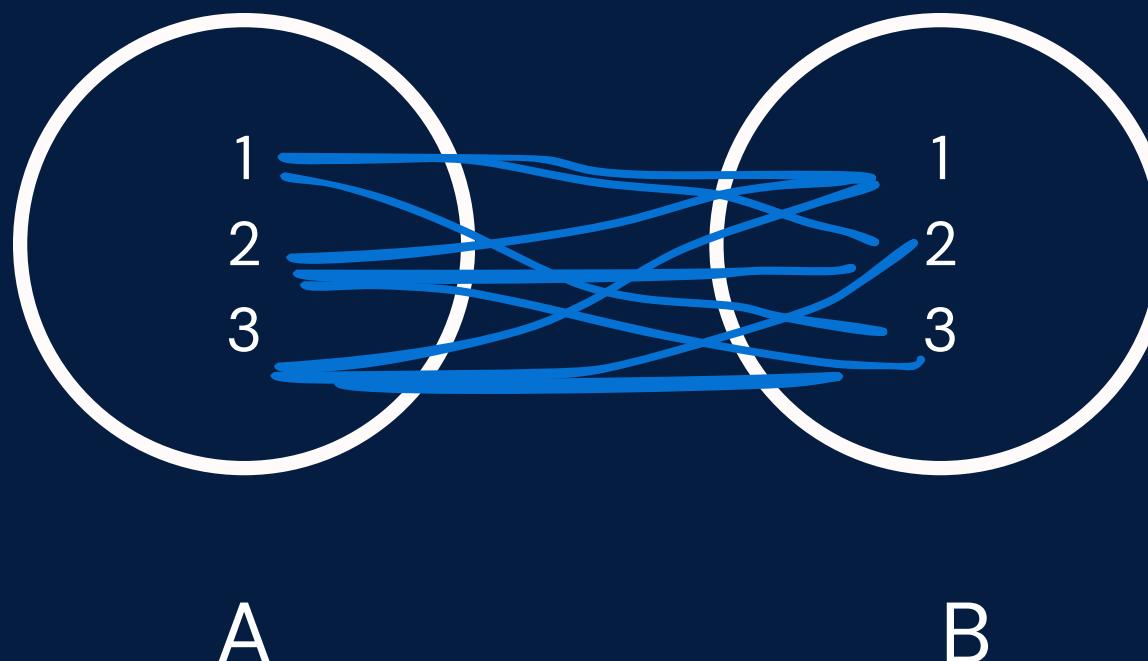
Student

LET'S START WITH SQL :)

Joins in SQL

Types of Joins in SQL

6. Cross Join



rollno	name
1	Ram
2	Rahul
3	Riti

Student

rollno	c_name
2	Hindi
3	Eng
4	Maths

Course

LET'S START WITH SQL :)

Joins in SQL

1. Inner Join: It helps us in getting the rows that have matching values in both tables, according to the given join condition.

Query:

```
SELECT columns  
FROM table1  
INNER JOIN table2  
ON table1.colName = table2.colName;
```

id	name
101	Ram
102	Rahul
103	Riti

Customer

id	o_name
102	Fruit
103	Ball
104	Utensils

Order

LET'S START WITH SQL :)

Joins in SQL

Query: It only returns rows where there is a matching id in both tables

```
SELECT *  
FROM customer  
INNER JOIN order  
ON customer.id = order.id;
```

id	name	id	o_name
102	Rahul	102	Fruit
103	Riti	103	Ball

id	name
101	Ram
102	Rahul
103	Riti

customer

id	o_name
102	Fruit
103	Ball
104	Utensils

Order

LET'S START WITH SQL :)

Joins in SQL

2. Left Join/Left Outer Join: It is used to fetch all the records from the left table along with matched records from the right table.

If there are no matching records in the right table, NULL values are returned for the columns of the right table.

Query:

```
SELECT columns  
FROM table1  
LEFT JOIN table2  
ON table1.colName = table2.colName;
```

Left table : the table specified before the LEFT JOIN keyword

Right table : the table specified after the LEFT JOIN keyword

LET'S START WITH SQL :)

Joins in SQL

Query:

```
SELECT *  
FROM customer  
LEFT JOIN order  
ON customer.id = order.id;
```

id	name	id	o_name
101	Ram	null	null
102	Rahul	102	Fruit
103	Riti	103	Ball

id	name
101	Ram
102	Rahul
103	Riti

customer

id	o_name
102	Fruit
103	Ball
104	Utensils

Order

LET'S START WITH SQL :)

Joins in SQL

3. Right Join/ Right Outer Join: It is used to fetch all the records from the right table along with matched records from the left table.

If there are no matching records in the left table, NULL values are returned for the columns of the left table.

Query:

```
SELECT columns  
FROM table1  
RIGHT JOIN table2  
ON table1.colName = table2.colName;
```

Left table : the table specified before the RIGHT JOIN keyword

Right table : the table specified after the RIGHT JOIN keyword

LET'S START WITH SQL :)

Joins in SQL

Query:

```
SELECT *
FROM customer
RIGHT JOIN order
ON customer.id = order.id;
```

id	o_name	id	name
102	Fruit	102	Rahul
103	Ball	103	Riti
104	utensils	null	null

id	name
101	Ram
102	Rahul
103	Riti

customer

id	o_name
102	Fruit
103	Ball
104	Utensils

Order

LET'S START WITH SQL :)

Joins in SQL

4. Full Join/Full Outer Join: It returns the matching rows of both left and right table and also includes all rows from both tables even if they don't have matching rows.

If there is no match, NULL values are returned for the columns of the missing table.

In MySQL, the syntax for a full join is different compared to other SQL databases like PostgreSQL or SQL Server.

MySQL does not support the FULL JOIN keyword directly. So we use a combination of LEFT JOIN, RIGHT JOIN, and UNION to achieve the result.

LET'S START WITH SQL :)

Joins in SQL

4. Full Join/Full Outer Join:

Query:

```
SELECT columns  
FROM table1  
LEFT JOIN table2  
ON table1.colName = table2.colName;
```

UNION

```
SELECT columns  
FROM table1  
RIGHT JOIN table2  
ON table1.colName = table2.colName;
```

LET'S START WITH SQL :)

Joins in SQL

Query:

```
SELECT *
FROM customer
LEFT JOIN order
ON customer.id = order.id;
UNION
SELECT *
FROM customer
RIGHT JOIN order
ON customer.id = order.id;
```

id	name
101	Ram
102	Rahul
103	Riti

customer

id	o_name
102	Fruit
103	Ball
104	Utensils

Order

LET'S START WITH SQL :)

Joins in SQL

Result :

id	name	id	o_name
101	Ram	null	null
102	Rahul	102	Fruit
103	Riti	103	Ball
null	null	104	Utensils

LET'S START WITH SQL :)

Joins in SQL

5. CrossJoin: It combines each row of the first table with every row of the second table.

Query:

```
SELECT *  
FROM table1  
CROSS JOIN table2;
```

id	name
101	Ram
102	Rahul

Customer

o_id	o_name
1	Fruit
2	Ball

Order

It results in a new table where the number of rows is equal to the product of the number of rows in each table. ($m \times n$)

LET'S START WITH SQL :)

Joins in SQL

Result :

id	name	o_id	o_name
101	Ram	1	Fruit
101	Ram	2	Ball
102	Rahul	1	Fruit
102	Rahu	2	Ball

LET'S START WITH SQL :)

Joins in SQL

6. Self Join: A self join in SQL is a type of join where a table is joined with itself. It is a type of inner join.

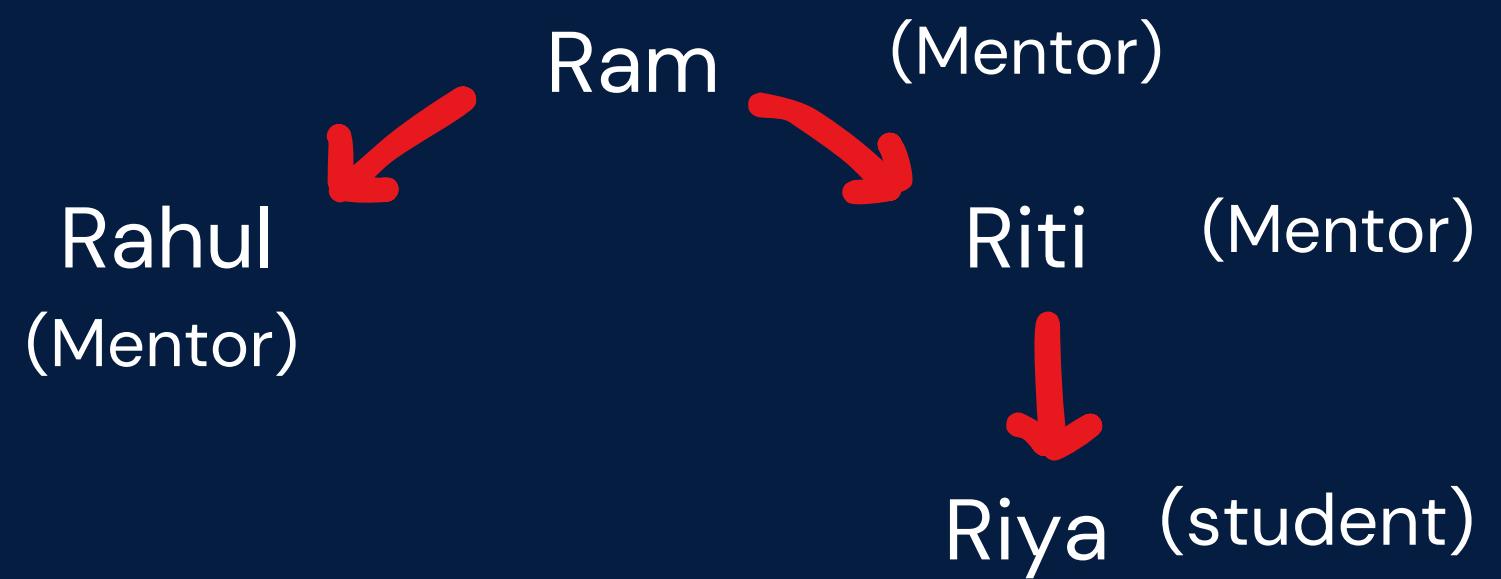
Query:

```
SELECT columns  
FROM table as t1  
JOIN table as t2  
ON t1.colName = t2.colName
```

t1 and t2 are aliases for the table, used to distinguish between the order rows.

LET'S START WITH SQL :)

Joins in SQL



Query :

```
SELECT s1.name as mentor_name, s2.name  
as name  
FROM student as s1  
JOIN student as s2  
WHERE s1.s_id=s2.mentor_id
```

s_id	name	mentor_id
1	Ram	null
2	Rahul	1
3	Riti	1
4	Riya	3

LET'S START WITH SQL :)

Joins in SQL

Result :

mentor_name	name
Ram	Riti
Ram	Rahul
Riti	Riya

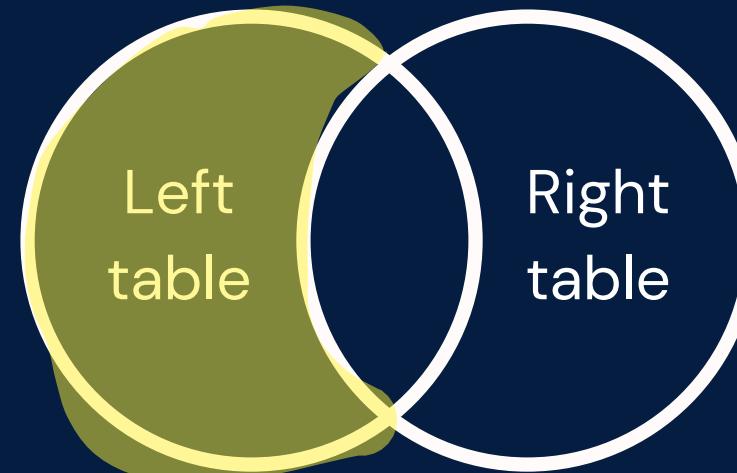
LET'S START WITH SQL :)

Exclusive Joins in SQL

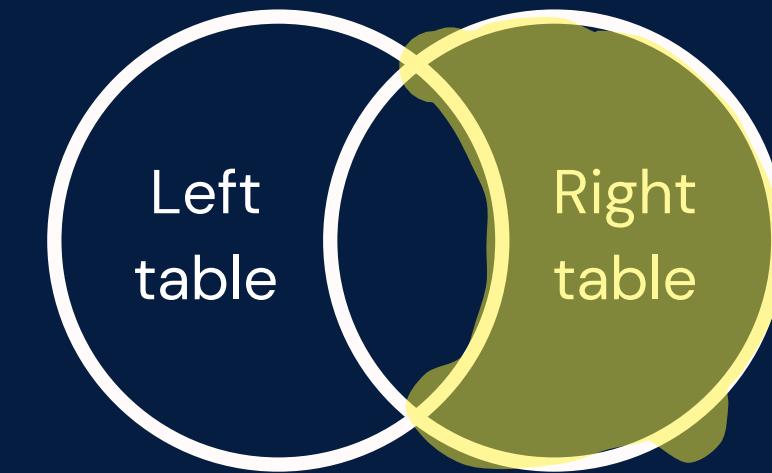
Exclusive joins are used when we want to retrieve data from two tables excluding matched rows. They are a part of outer joins or full outer join.

Types :

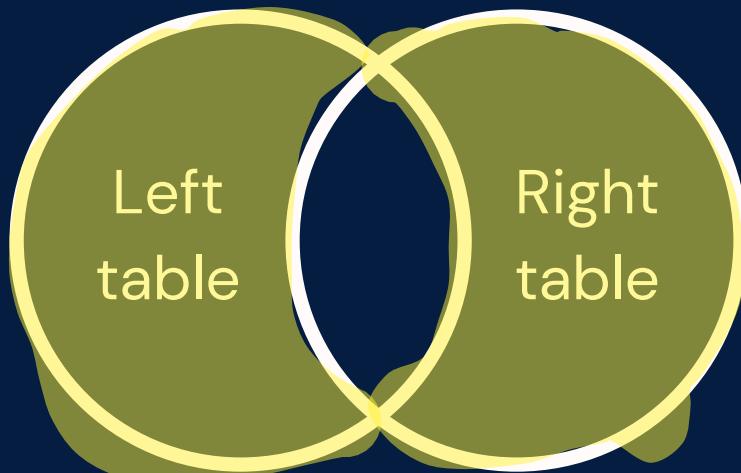
1.Left Exclusive JOIN



2.Right Exclusive JOIN



3.Full Exclusive JOIN



LET'S START WITH SQL :)

Exclusive Joins in SQL

Left Exclusive JOIN: When we retrieve records from the left table excluding the ones matching in both left and right table .

Query:

```
SELECT columns  
FROM table1  
LEFT JOIN table2  
ON table1.colName = table2.colName;  
WHERE table2.colName IS NULL;
```

id	name
101	Ram
102	Rahul
103	Riti

customer

id	o_name
102	Fruit
103	Ball
104	Utensils

Order

LET'S START WITH SQL :)

Exclusive Joins in SQL

Right Exclusive JOIN: When we retrieve records from the right table excluding the ones matching in both left and right table .

Query:

```
SELECT columns  
FROM table1  
RIGHT JOIN table2  
ON table1.colName = table2.colName;  
WHERE table1.colName IS NULL;
```

id	name
101	Ram
102	Rahul
103	Riti

customer

id	o_name
102	Fruit
103	Ball
104	Utensils

Order

LET'S START WITH SQL :)

Exclusive Joins in SQL

Full Exclusive JOIN: When we retrieve records from the right table and left table excluding the ones matching in both left and right table .

Query:

```
SELECT columns  
FROM table1  
LEFT JOIN table2  
ON table1.colName = table2.colName;  
WHERE table2.colName IS NULL;  
UNION  
SELECT columns  
FROM table1  
RIGHT JOIN table2  
ON table1.colName = table2.colName;  
WHERE table1.colName IS NULL;
```

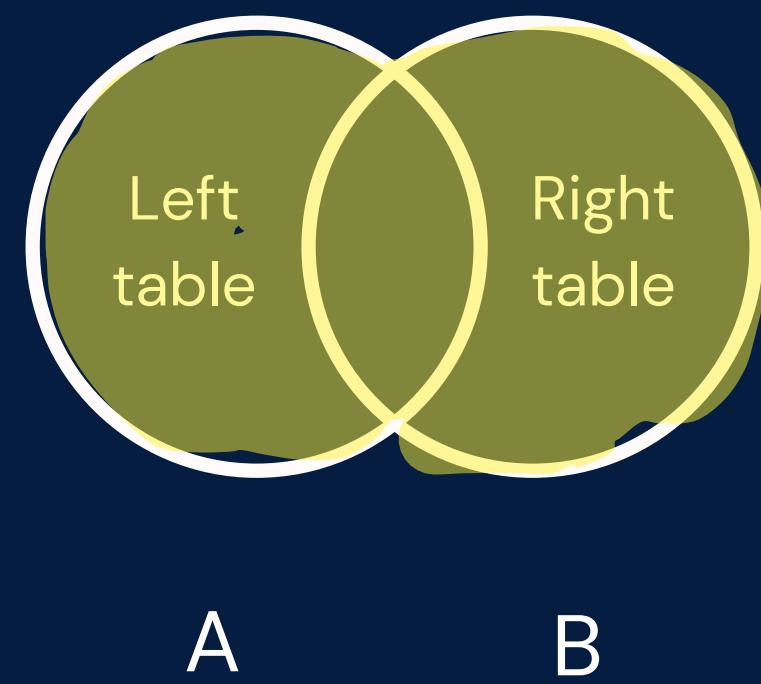
LET'S START WITH SQL :)

UNION Operator in SQL

UNION: UNION operator in SQL is used to combine the results of two or more SELECT queries into a single result set and gives unique rows by removing duplicate rows.

Things to keep in mind:

1. Each SELECT command within the UNION must retrieve the same number of columns.
2. The data types of columns in corresponding positions across SELECT statements should match.
3. Columns should be listed in the same order across all SELECT statements.

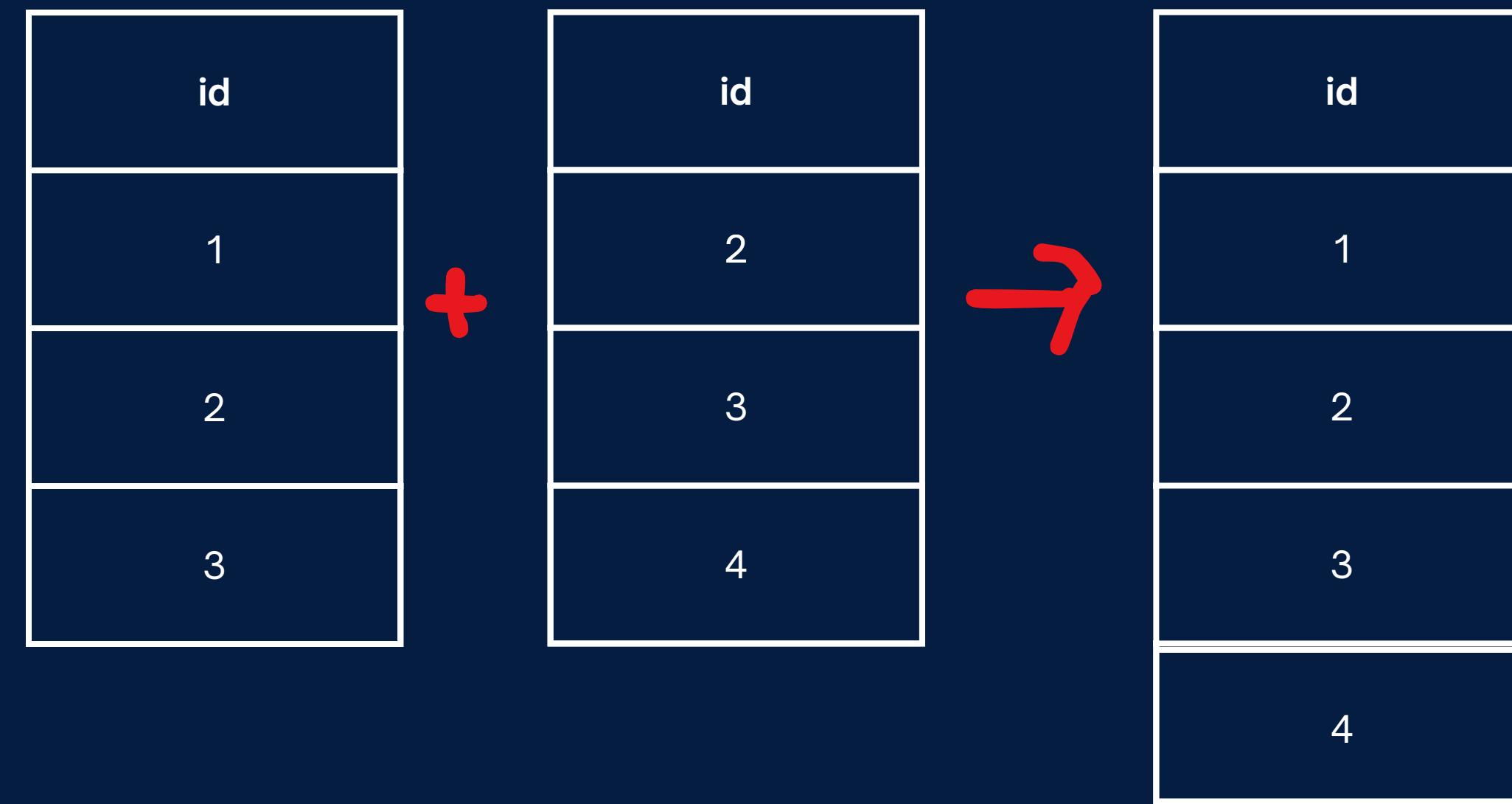


LET'S START WITH SQL :)

UNION Operator in SQL

QUERY:

```
SELECT columns  
FROM table1  
UNION  
SELECT columns  
FROM table2;
```



LET'S START WITH SQL :)

UNION ALL Operator in SQL

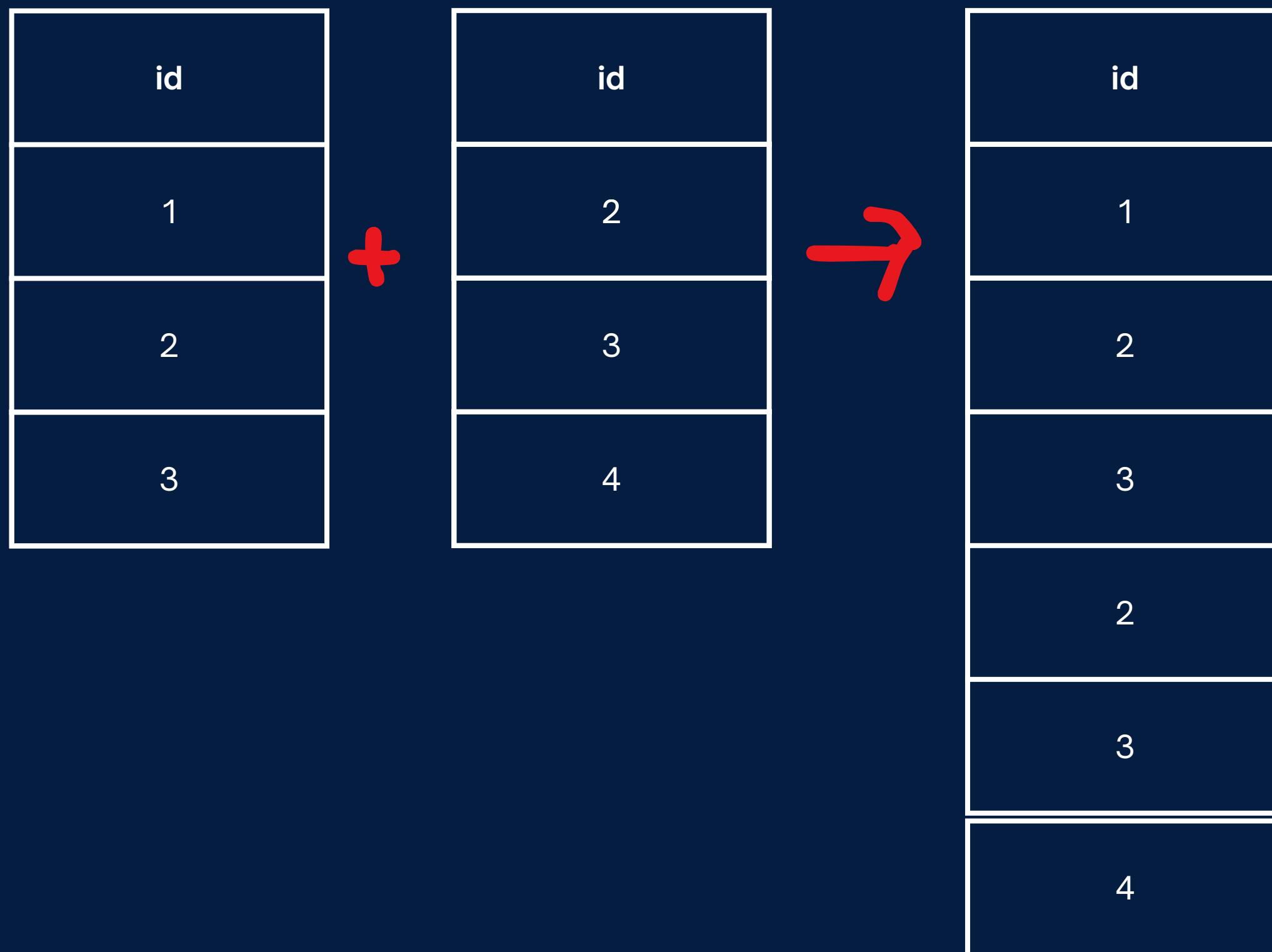
UNION ALL: UNION operator in SQL is used to combine the results of two or more SELECT queries into a single result set and gives all rows by not removing duplicate rows.

QUERY:

```
SELECT columns  
FROM table1  
UNION ALL  
SELECT columns  
FROM table2;
```

LET'S START WITH SQL :)

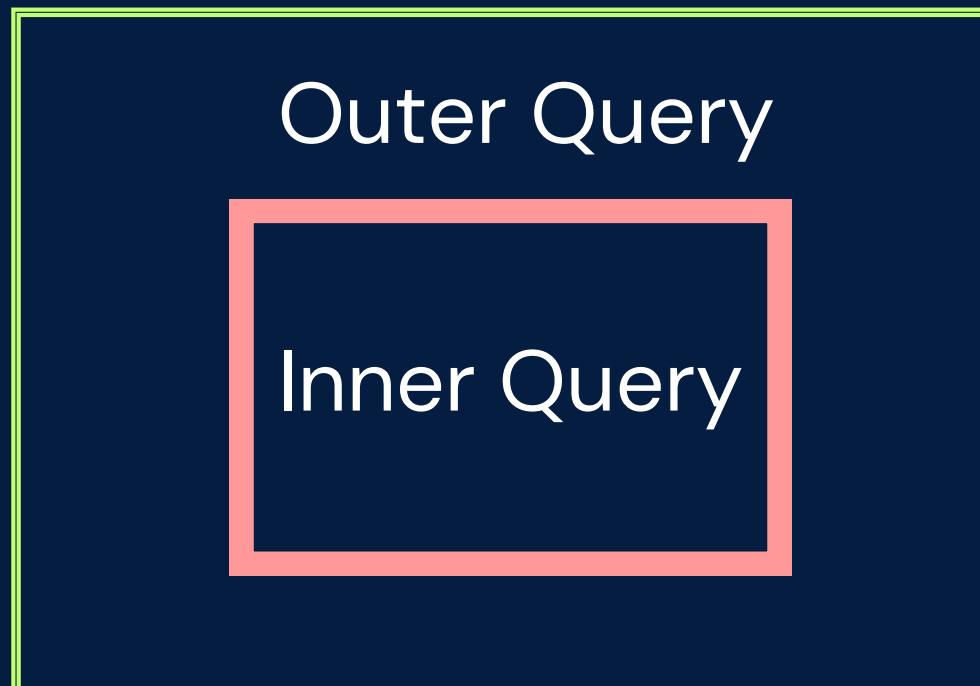
UNION ALL Operator in SQL



LET'S START WITH SQL :)

SQL Subqueries/Nested queries

Subqueries/Inner Queries/Nested Queries: SQL subquery is a query nested within another SQL statement. Whenever we want to retrieve data based on the result of another query we use nested queries.



LET'S START WITH SQL :)

SQL Subqueries/Nested queries

How can we use Subqueries?

Subqueries can be used in multiple ways :

- Subqueries can be used with clauses such as **SELECT, INSERT, UPDATE, or DELETE** to perform complex data retrieval.

QUERY:

```
SELECT columns, (subquery)  
FROM tableName;
```

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

How can we use Subqueries?

Subqueries can be used in multiple ways :

- Subqueries can be used with **WHERE** clause to filter data based on conditions.

QUERY:

```
SELECT *  
FROM tableName  
WHERE column name operator (subquery);
```

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

How can we use Subqueries?

Subqueries can be used in multiple ways :

- Subqueries can also be used in the **FROM** clause.

QUERY:

```
SELECT *  
FROM subquery AS altName ;
```

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

Let's understand from example of using subqueries in WHERE:

1. Find all the employees who have salary greater than the min salary

- Find the min salary
- Find employee having salary greater than min salary

id	name	age	departmen	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

Let's understand from example of using subqueries in WHERE:

- To find the min salary

QUERY:

```
SELECT AVG(salary) FROM employee
```

- To find all the employees having salary greater than min salary

QUERY:

```
SELECT name, salary  
FROM employee  
WHERE salary > (subquery)
```

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

Let's understand from example of using subqueries in WHERE:

2. Find the employees with the minimum age

- Find the min age
- Find employee having the min age

id	name	age	departmen	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

Let's understand from example of using subqueries in WHERE:

- To find the min age

QUERY:

```
SELECT MIN(age) FROM employee
```

- To find all the employees having min age

QUERY:

```
SELECT name, age  
FROM employee  
WHERE age =(subquery);
```

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

Let's understand from example of using subqueries in FROM:

1. Find the employees who is having age greater than min_age

- Find the min age
- Find employee having age > min age

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

Let's understand from example of using subqueries in WHERE:

- To Find the min age

QUERY:

```
SELECT min(age) AS min_age FROM employee;
```

- Find employee having age > min age

QUERY:

```
SELECT emp.name  
FROM employee emp, (subquery) AS subquery  
WHERE emp.age > subquery.min_age;
```

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

Let's understand from example of using subqueries in SELECT:

1. Print the employees with the average age and age of employees

- Find the avg age
- Print the employee age and avg_age

id	name	age	departmen	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

LET'S START WITH SQL :)

SQL Subqueries/Nested queries

Let's understand from example of using subqueries in SELECT:

- Find the avg age

QUERY:

```
SELECT AVG(age) FROM employee
```

- Print the employee age and avg_age

QUERY:

```
SELECT (subquery)AS avg_age , age  
FROM employee;
```

LET'S START WITH SQL :)

Nth Highest Salary

Q. Find the nth highest salary in a given dataset.

Steps to find the nth highest salary:

Step 1: Select the column which you want to show the final result i.e salary.

Step 2: Order the salary in descending order so that you have the max at the first.

Step 3: Now the value of n could 1,2,3....till n, so we have to make the query in such a way so that whatever be the value of n it can provide the result.

Step 4: So at the end of the query we will provide a LIMIT so that on the data set which we have got after ordering the salary in descending order, we can fetch the nth highest one.

LET'S START WITH SQL :)

Nth Highest Salary

Q. Find the nth highest salary in a given dataset.

LIMIT- LIMIT clause is used to restrict the number of rows returned by a query.

- **LIMIT n** - It helps to retrieve a maximum of n rows from the beginning of the result set.

- **LIMIT m, n**- It helps to retrieve a specific range of rows where

m- number of rows to skip from the beginning

n- number of rows to fetch after skipping

LET'S START WITH SQL :)

Nth Highest Salary

Q. Find the nth highest salary in a given dataset.

QUERY:

```
SELECT DISTINCT Salary  
FROM tableName  
ORDER BY Salary DESC  
LIMIT n-1, 1;
```

LET'S START WITH SQL :)

Stored Procedures

Stored Procedure- These are programs that can perform specific tasks based on the stored query. It is basically a collection of pre-written SQL statements grouped together under a specific name.

Query: (to create a procedure)

```
CREATE PROCEDURE procedureName()  
BEGIN  
Query  
END;
```

Query: (to call the procedure)

```
CALL procedureName();
```

LET'S START WITH SQL :)

Stored Procedures

Examples: Stored procedure without params

Query 1:

```
CREATE PROCEDURE getAllOrderDetails()
BEGIN
    Select * from orders;
END;
```

Query: (to call the procedure)

```
CALL getAllOrderDetails();
```

LET'S START WITH SQL :)

Stored Procedures

Sometimes we encounter an issue in SQL workbench so we use delimiter there

Query 1:

```
DELIMITER /
CREATE PROCEDURE getAllOrderDetails()
BEGIN
SELECT * FROM orders;
END/
DELIMITER ;
```

Query: (to call the procedure)

```
CALL getAllOrderDetails();
```

LET'S START WITH SQL :)

Stored Procedures

Examples: Return the details of the order by id (Stored procedure with params)

Query 2:

```
CREATE PROCEDURE getAllOrderDetailsById(IN id int)
BEGIN
SELECT *FROM Orders WHERE id = id;
END;
```

Query: (to call the procedure)

```
CALL getAllOrderDetailsById(2);
```

LET'S START WITH SQL :)

Views In SQL

A view is a virtual table in SQL. It helps in providing a filtered view of data for security purposes.

QUERY:

```
CREATE VIEW viewName AS  
SELECT columns FROM baseTableName; (Specify the columns to be  
included in the view)
```

It helps in Data Abstraction, Security and simplify complex queries.

LET'S START WITH SQL :)

Views In SQL

- To see all the data in view

QUERY:

```
SELECT * FROM viewName ;
```

- To drop a view

QUERY:

```
DROP VIEW IF EXISTS viewName;
```

LET'S START WITH SQL :)

CASE AND IF IN SQL

- **CASE:** It allows you to perform conditional logic within a query. It can be used in both SELECT and UPDATE statements to evaluate conditions and return specific values based on those conditions.

QUERY:

CASE

```
WHEN condition1 THEN result1 WHEN  
condition2 THEN result2 ... ELSE resultN  
END
```

LET'S START WITH SQL :)

CASE with Select statement

Q. Categorise the students on basis of their percentage to Top, Pass and fail in a new column category

QUERY:

```
SELECT sid, name, percentage,  
CASE  
    WHEN percentage > 90 THEN 'Top'  
    WHEN percentage BETWEEN 89 AND 34 THEN 'Pass'  
    ELSE 'Fail'  
END AS category  
FROM student;
```

LET'S START WITH SQL :)

CASE with Update statement

Q. Students have got some grace marks so update their grades. Where its A update to A+ and where its B update to A.

QUERY:

```
UPDATE student  
SET grade = CASE  
    WHEN grade= 'B' THEN 'A'  
    WHEN grade = 'A' THEN 'A+'  
END;
```

LET'S START WITH SQL :)

IF IN SQL

- **IF:** It is used to return one of two values depending on whether a condition is true or false. It is not supported in many DB but supported in MySQL

QUERY:

IF(condition, value_if_true, value_if_false)

LET'S START WITH SQL :)

IF with Select statement

Q. Categorise the students on basis of their percentage to Top, Pass and fail in a new column category

QUERY:

```
SELECT sid, name, percentage,  
IF(percentage > 90, 'Top' , IF(percentage BETWEEN 89 AND 34, 'Pass', 'Fail')) AS category  
FROM student;
```

LET'S START WITH SQL :)

IF with Update statement

Q. Swap all 'f' and 'm' values (i.e., change all 'f' values to 'm' and vice versa) with a single update statement and no intermediate temporary tables.

QUERY:

```
UPDATE employee  
SET gender = if(gender = 'm', 'f', 'm')
```

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

**Q. Write the SQL Query to 1.create a database Company, 2.create a table employee in it
delete/drop the database**

1.Create a Database Company

CREATE DATABASE company;

2. Create a table Employee

USE company; -> to tell the server to create table in this DB

```
CREATE TABLE employee(  
employee_id INT PRIMARY KEY,  
first_name VARCHAR(50),  
last_name VARCHAR(50));
```

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write the SQL Query to 1.create a database Company, 2.create a table employee in it, delete/drop the database

3. Delete the Database company

DROP DATABASE company;

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write the SQL Query to 1.create a table employee, 2.Insert data into the table employee
3. Update Salary for all people in HR department to 20000 4. Delete data for employee having empld =1 5. Delete the entire table

1. Create a table employee

```
CREATE TABLE employee(  
empld INT PRIMARY KEY,  
name VARCHAR(50),  
department VARCHAR(50),  
salary INT);
```

2. Insert data into the table employee

```
INSERT INTO employee(empld,name,department,salary)  
VALUES(1, 'Riti', 'IT', 30000),  
(2, 'Rahul', 'HR', 15000);
```

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write the SQL Query to 1.create a table employee, 2.Insert data into the table employee
3. Update Salary for all people in HR department to 20000 4. Delete data for employee having empld =1 5. Delete the entire table

3. Update Salary for all people in HR department to 20000

UPDATE employees

SET salary = 20000

WHERE department='HR'

4. Delete data for employee having empld =1

DELETE FROM employee

WHERE empld = 1;

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write the SQL Query to 1.create a table employee, 2.Insert data into the table employee
3. Update Salary for all people in HR department to 20000 4. Delete data for employee having emplId =1 5. Delete the entire table

5. Delete the entire table

DROP TABLE employee;

To delete all the data

DELETE FROM employee;

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write a query to find the total number of employees working in the 'IT' department.

Query :

```
SELECT COUNT(*) FROM employee  
WHERE department = 'IT';
```

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

COUNT(*) is a SQL aggregate function that returns the total number of rows in a specified table or query. It counts all the rows, regardless of whether they contain NULL values or not.

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write a query to find all the employees that have their name starting from 'R'

Query :

```
SELECT * FROM employee WHERE  
name LIKE 'R%';
```

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

LIKE- It is used to search for a specified pattern in a column
We use '%' and '_' for searching patterns

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Condition	Query
Name start with A	LIKE 'A%'
Name contain ra	LIKE '%ra%
Name start with 'A' and have exactly five characters	LIKE 'A_____'
Name has a as second character	LIKE '_a%

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write a query to find unique salaries in employee table

Query :

```
SELECT DISTINCT salary  
FROM employee;
```

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

DISTINCT- It is used to retrieve unique records from a table

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write a query to find the second highest salary in a table

Query :

```
SELECT MAX(salary)
FROM employee
WHERE salary <> (SELECT
MAX(salary) FROM employee);
```

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

MAX - gives the aggregated max value from a column

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write a query to find the nth highest salary in a table

Query :

```
SELECT DISTINCT Salary  
FROM employee  
ORDER BY Salary DESC  
LIMIT n-1, 1;
```

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

ORDER BY - Order the salary in descending/ascending order

LIMIT m, n - It helps to retrieve a specific range of rows

m - number of rows to skip from the beginning

n - number of rows to fetch after skipping

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write a query to find the top 2 salaries from a table

Query :

```
SELECT salary  
FROM employee  
ORDER BY salary DESC  
LIMIT 2;
```

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write a query to calculate the total salary and average salary in a department

Query :

```
SELECT department, SUM(salary) AS  
total_salary, AVG(salary) AS avg_salary  
FROM employee  
GROUP BY department;
```

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write a query to find the rows where a department has NULL values

Query :

```
SELECT *FROM employee  
WHERE department IS NULL;
```

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Write a query to find the duplicate rows in employee for column department.

Query :

```
SELECT department, COUNT(*)  
FROM employee  
GROUP BY department  
HAVING COUNT(*) > 1;
```

id	name	age	department	city	salary
1	Rahul	25	'IT'	'Mumbai'	1500
2	Afsara	26	'HR'	'Pune'	2000
3	Abhimanyu	27	'IT'	'Mumbai'	2500
4	Aditya	25	'Marketing'	'Surat'	2400
5	Raj	24	'Finance'	'Indore'	1500

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. What is SQL?

→ SQL stands for Structured Query Language
It is a standard language used for managing and manipulating databases.

Q. What is the difference between DELETE and TRUNCATE

→ DELETE removes rows from a table based on a condition and can be rolled back.
while, TRUNCATE removes all rows from a table without logging individual row deletions and cannot be rolled back.

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. What is UNION and UNION ALL.

→ UNION combines the results of two queries and removes duplicate rows.

while, UNION ALL combines the results of two queries and includes all duplicates.

Q. What is a stored procedure?

→ A stored procedure is a prepared SQL code that you can save and reuse in other queries.

Q. What is difference between CHAR() and VARCHAR()

→ CHAR is used when we have data with a fixed length

while, VARCHAR is used when we have data with variable length

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. Explain the types of JOINS in SQL?

- INNER JOIN: It returns all records that have matching values in both tables.
- LEFT OUTER JOIN: It returns all records from the left table, and the matched records from the right table.
- RIGHT OUTER JOIN : It returns all records from the right table, and the matched records from the left table.
- FULL OUTER JOIN: It returns all records when there is a match in either left or right table.
- CROSS JOIN: It returns the Cartesian product of the two tables.
- SELF JOIN : A join where a table is joined with itself.

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. What is NULL in SQL?

-> It is used to handle NULL values. It is used to indicate that a data value does not exist in the database.

Q. What is a foreign key?

-> A foreign key is a key that helps in establishing a relationship between the two tables.

It uniquely identifies a row of another table. A foreign key is a key in one table, that refers to the primary key in another table.

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. What is a primary key?

→ A primary key is a unique identifier/key which uniquely identifies all record in a table/relation.

It must contain unique values and cannot contain NULL values.(UNIQUE+NOT NULL)

Q. What is the difference between WHERE and HAVING

→ WHERE is used to filter records before any groupings are made.

while, HAVING is used to filter records after groupings are made.

LET'S START WITH SQL :)

Top SQL Questions asked in interviews

Q. What is the view in SQL?

→ Views are a kind of virtual table in SQL

Q. What is DEFAULT constraint

→ Whenever we need to fill a column with default and fixed values we use DEFAULT, like set the default salary as 0 where salary is null.

Q. What is an ALIAS command in SQL

→ These are temporary names given to a table or column which is just a temporary change i.e the table name does not change in the original database.

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2).

How to think on what SQL clause/operator/function to choose?

1. Data retrieval

- **SELECT:** Use for retrieving data from one or more tables.

2. Data Filtering

- **WHERE:** Filter records based on specific conditions
- **AND, OR, NOT:** Combine multiple conditions.
- **BETWEEN :** Range search
- **IN:** Checks whether a specified value matches any value in a subquery or a list
- **LIKE :** Pattern matching(%, _)

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2)

How to think on what SQL clause/operator/function to choose?

3. Aggregation on data

- **AVG()**: Returns the average value of a numeric column.
- **MIN()**: Returns the minimum value in a column.
- **MAX()**: Returns the maximum value in a column.
- **SUM()**: Returns the total sum of a numeric column.
- **COUNT()**: Counts the number of non-NULL values in a specified column
- **COUNT(*)**: Counts the total number of rows in a table, including rows with NULL values.

4. Grouping and Filtering Groups

- **GROUP BY**: Groups rows that have the same values in specified columns
- **HAVING**: Filters groups based on specified conditions (used with GROUP BY).

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2)

How to think on what SQL clause/operator/function to choose?

5. Sorting data

- **ORDER BY:** Sorts the result set by one or more columns.

6. Data retrieval from combination of 2 or more tables

- **INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL JOIN:** Combine rows from two or more tables based on related columns.

7. Conditional logic

- **CASE :** Perform conditional logic within a query using WHEN, THEN and ELSE
- **IF :** Return values depending on whether a condition is true or false.

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2).

How to solve SQL questions

Step 1: Understand what the question says?

Retrieving data, aggregating results, joining tables, or performing updates/deletes.

Step 2: Check data types, constraints ,primary key, foreign keys, and relationships between tables

Step 3: Use the Appropriate SQL Clauses and Functions

Step 4: Ensure appropriate indexes on columns used in WHERE, JOIN, and ORDER BY clauses.

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2).

Leetcode Questions

1. [Swap Salary](https://leetcode.com/problems/swap-salary/) : <https://leetcode.com/problems/swap-salary/>
2. [Duplicate Emails](https://leetcode.com/problems/duplicate-emails/) : <https://leetcode.com/problems/duplicate-emails/>
3. [Employees Earning More Than Their Managers](https://leetcode.com/problems/employees-earning-more-than-their-managers/) :
<https://leetcode.com/problems/employees-earning-more-than-their-managers/>
4. [Not Boring Movies](https://leetcode.com/problems/not-boring-movies/) : <https://leetcode.com/problems/not-boring-movies/>
5. [Classes More Than 5 Students](https://leetcode.com/problems/classes-more-than-5-students/): <https://leetcode.com/problems/classes-more-than-5-students/>

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2)

Leetcode Questions

Swap Salary

Q. Write a solution to swap all 'f' and 'm' values (i.e., change all 'f' values to 'm' and vice versa) with a single update statement and no intermediate temporary tables.

Input:			
Salary table:			
+	-----	-----	-----
	id	name	sex
	-----	-----	-----
1 A m 2500			
2 B f 1500			
3 C m 5500			
4 D f 500			
	-----	-----	-----
Output:			
	-----	-----	-----
id	name	sex	salary
-----	-----	-----	-----
1 A f 2500			
2 B m 1500			
3 C f 5500			
4 D m 500			
	-----	-----	-----

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2)

Leetcode Questions

Duplicate Emails

Q. Write a solution to report all the duplicate emails. Note that it's guaranteed that the email field is not NULL.

Return the result table in any order.

Input:	
Person table:	
+	-----+
id email	-----
+	-----+
1 a@b.com	-----
2 c@d.com	-----
3 a@b.com	-----
+	-----+

Output:

Email
a@b.com

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2).

Leetcode Questions

Employees Earning More Than Their Managers

Q. Write a solution to find the employees who earn more than their managers.

Employee table:			
id	name	salary	managerId
1	Joe	70000	3
2	Henry	80000	4
3	Sam	60000	Null
4	Max	90000	Null

Output:	
Employee	
Joe	

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2)

Leetcode Questions

Not Boring Movies

Q. Write a solution to report the movies with an odd-numbered ID and a description that is not "boring".

Return the result table ordered by rating in descending order.

Input:				
Cinema table:				
id	movie	description	rating	
1	War	great 3D	8.9	
2	Science	fiction	8.5	
3	irish	boring	6.2	
4	Ice song	Fantacy	8.6	
5	House card	Interesting	9.1	

Output:				
id	movie	description	rating	
5	House card	Interesting	9.1	
1	War	great 3D	8.9	

LET'S START WITH SQL :)

Top SQL Questions asked in interviews(Part-2)

Leetcode Questions

Classes More Than 5 Students

Write a solution to find all the classes that have at least five students.

Input:

Courses table:

student	class
A	Math
B	English
C	Math
D	Biology
E	Math
F	Computer
G	Math
H	Math
I	Math

Output:

class
Math

LET'S START WITH DBMS :).

Intension and Extension in DataBase

Intension in Database :

The intension defines what kind of data can be stored and the relationships between them. This is basically the blueprint or definition of the database structure. It doesn't change frequently and its the permanent definition of the database structure.

It includes:

- Table definitions(name of tables, their columns, and the data types allowed in each column)
- Constraints(Rules that govern the data, such as primary keys, foreign keys, and data validation rules)
- Relationships between tables(how tables are connected through shared columns)

LET'S START WITH DBMS :).

Intension and Extension in DataBase

Intension in Database

Example:

```
customer(  
id INT PRIMARY KEY,  
name VARCHAR(50))
```

LET'S START WITH DBMS :).

Intension and Extension in DataBase

Extension in Database :

The extension is the actual data stored in the database at a given instance in time.

Basically the data which is stored in tuples/rows at a given instance of time.

When there are more tuples added the data can change .

Employee		
id	name	departmen
1	Rahul	'IT'
2	Afsara	'HR'
3	Abhimanyu	'IT'

Data at instance t1

Employee		
id	name	departmen
1	Rahul	'IT'
2	Afsara	'HR'
3	Abhimanyu	'IT'
4	Aditya	'Marketing'

Data at instance t2

Employee		
id	name	departmen
1	Rahul	'IT'
2	Afsara	'HR'
3	Abhimanyu	'IT'
4	Aditya	'Marketing'
5	Raj	'Finance'

Data at instance t3

LET'S START WITH DBMS :).

RDBMS

What is RDBMS ?

Database : Collection of data is called as database.

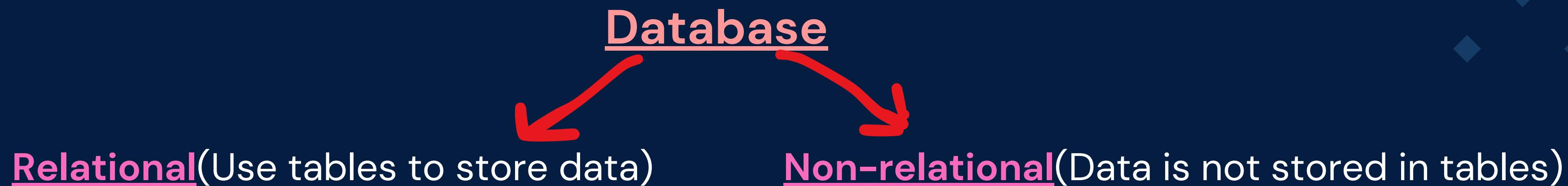
DBMS: A software application to manage our data.



LET'S START WITH DBMS :).

RDBMS

What is RDBMS ?



MySQL

Oracle

MariaDB,

MongoDb

LET'S START WITH DBMS :).

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

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)

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

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

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

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.

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

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?

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

LET'S START WITH DBMS :).

Normalisation and its types

Types of Normalisation

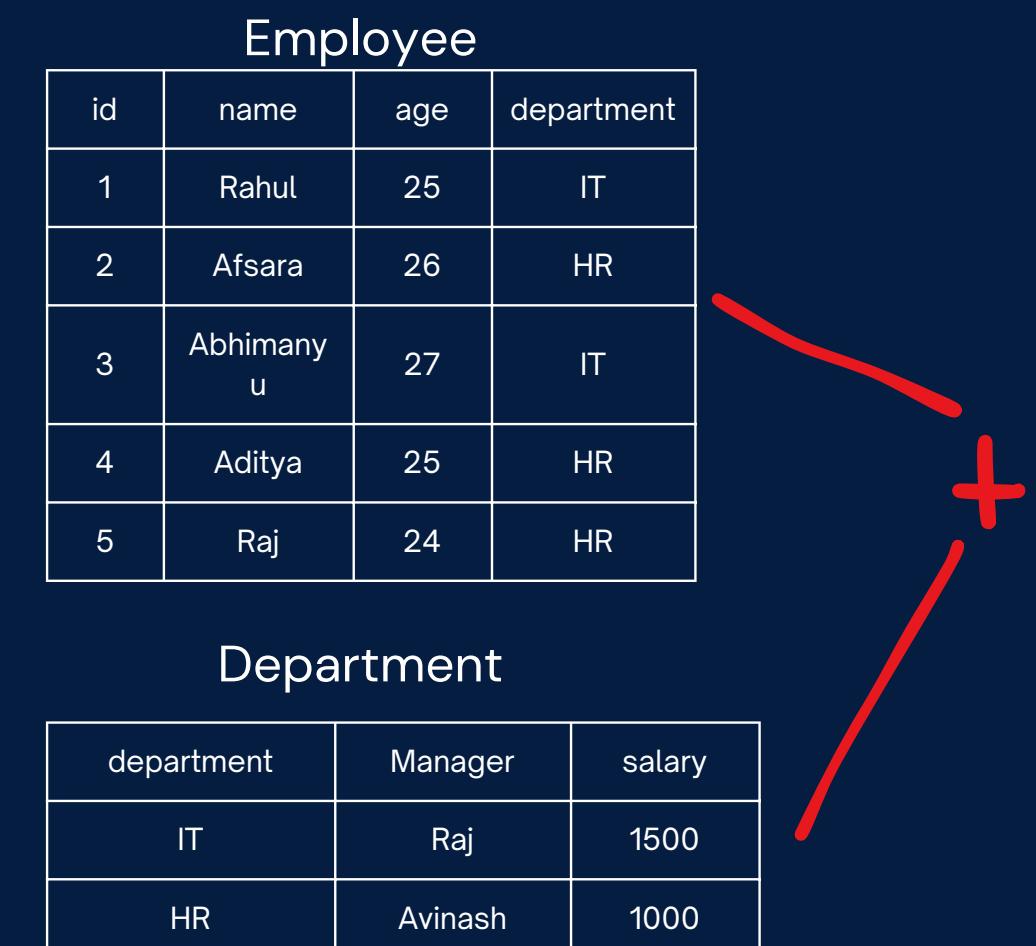
- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form(BCNF)

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	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 :).

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.

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	EmpFirstName	EmpLastNmae
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 .

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

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 .
3. Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.
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 $X \rightarrow Z$.

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

$$X \setminus Y = Y$$

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 $X \neq Y$.
 $\{EmplID\} \rightarrow \{EmpFirstName\}$ is trivial because $\{EmpFirstName\}$ is not a subset of $\{EmplID\}$.

$X \cap Y = \text{empty}$

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

LET'S START WITH DBMS :).

Attribute closure/closure set

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}

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.

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:

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

2. Analyze the functional dependencies and find closure.

From $A \rightarrow B$ and $B \rightarrow C$, we can infer $A \rightarrow C$

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

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 $2^{3=8}$

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

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

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

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)

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.

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.

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 :).

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)

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

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

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

PK - ID

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

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.

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}

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.

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

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)

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.

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→C, AB→D

CK : AB

- a. AB→C (LHS i.e AB is a CK)
- b. AB→D (LHS i.e AB is a CK)

It is in BCNF.

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 $R2(BC)$

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

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 :).

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(R1, R2..Rn) is not equal to the original relation

R

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)

A	B	C
1	2	3
4	5	6

R1 natural join R2

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. $R1 \cup R2 = R$
 - b. $R1 \cap R2 = \text{common attribute}$
3. To ensure that a decomposition is lossless, a common approach is to use the dependency preservation property

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	C
1	2	3
4	5	6

R

A	B
1	2
4	5

R1

A	C
1	3
4	6

R2

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 :).

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 $X \rightarrow\rightarrow Y$ $X \rightarrow\rightarrow Z$ 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.

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	C
1	2	3
4	5	6

R

A	B
1	2
4	5

R1

A	C
1	3
4	6

R2

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 :).

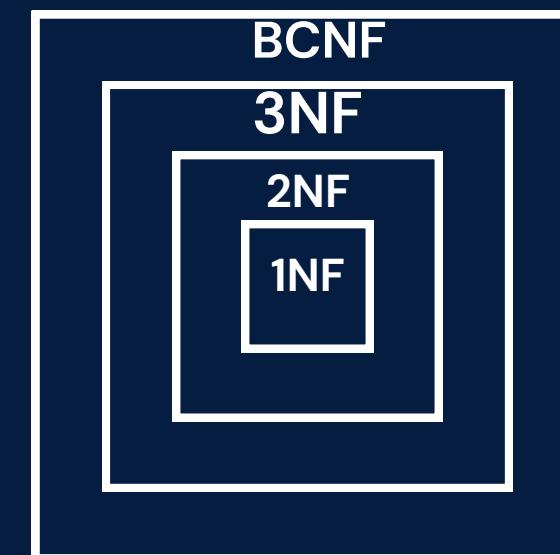
Normalisation and its types

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



LET'S START WITH DBMS :).

Normalisation and its types

For a given relation R(A,B,C) with the following functional dependencies:
 $A \rightarrow BC$, $B \rightarrow C$, $A \rightarrow B$, $AB \rightarrow C$, $B \rightarrow A$, 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).

$A \rightarrow BC$ = no partial dependency (A is a CK)

$B \rightarrow C$ = no partial dependency (B is a CK)

$A \rightarrow B$ = no partial dependency (A is a CK)

$AB \rightarrow C$ = no partial dependency (AB is a combination of candidate keys, Its SK)

$B \rightarrow A$ = no partial dependency (B is a CK) , **R is in 2NF.**

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)

$A \rightarrow BC$ = no transitive dependency (A is a CK)

$B \rightarrow C$ = no transitive dependency (B is a CK)

$A \rightarrow B$ = no transitive dependency (A is a CK)

$AB \rightarrow C$ = no transitive dependency (AB is a combination of candidate keys, Its SK)

$B \rightarrow A$ = 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.

$A \rightarrow BC = A$ is a CK

$B \rightarrow C = B$ is a CK

$A \rightarrow B = A$ is a CK

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

$B \rightarrow A = B$ 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

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)

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

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

Step 1 : ABCD , CK $\rightarrow A$, 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

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

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

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

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

$A \rightarrow B$ =(no td as LHS is a CK)

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

$C \rightarrow D$ =(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)

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 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→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

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).

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

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

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

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

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

LET'S START WITH DBMS :).

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

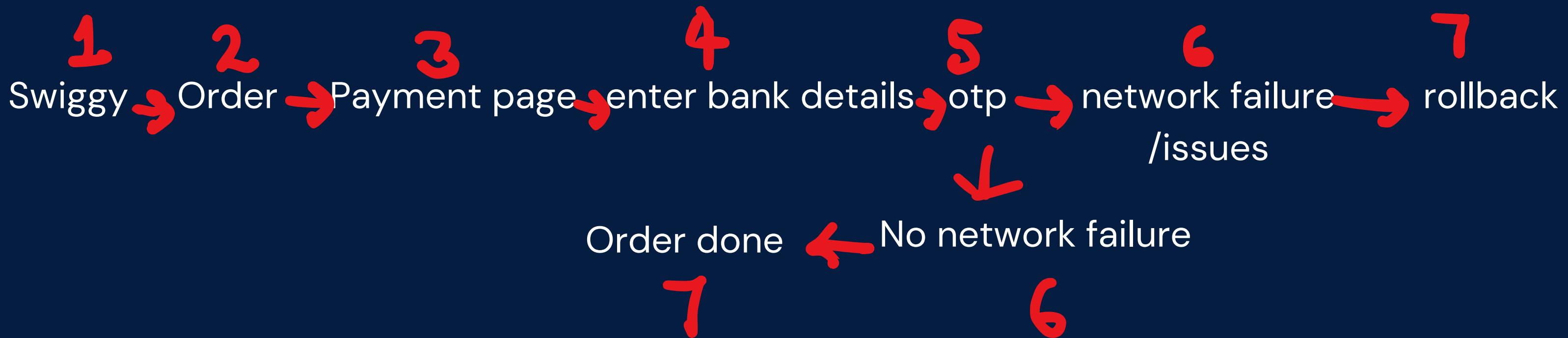
LET'S START WITH DBMS :).

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**

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

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).

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)

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)

LET'S START WITH DBMS :).

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.

LET'S START WITH DBMS :).

Isolation levels and its types

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.

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.

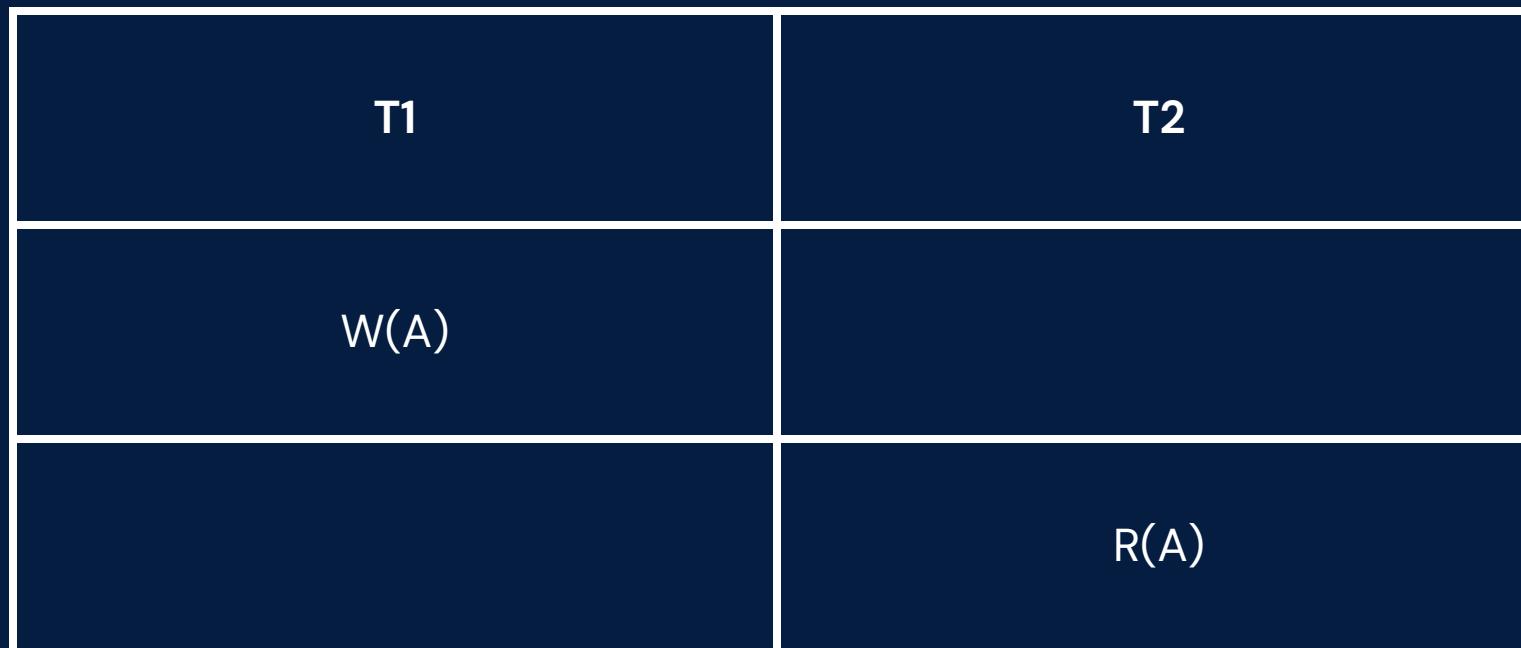


LET'S START WITH DBMS :).

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.



LET'S START WITH DBMS :).

Isolation levels and its types

Anamolies/Voilations to Isolation level

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

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

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 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

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

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

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

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

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)

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

T2 starts only after T1 is completed/ committed.

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.

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

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)

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

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

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

LET'S START WITH DBMS :).

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

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

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 manages multiple tasks at the same time. Tasks can overlap in their execution periods but do not run exactly simultaneously.	Parallel scheduling runs multiple tasks simultaneously using multiple cores or processors. Each task is executed on a separate core or processor at the same time.
Execution	Achieved on a single-core processor through context switching, where the CPU 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

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

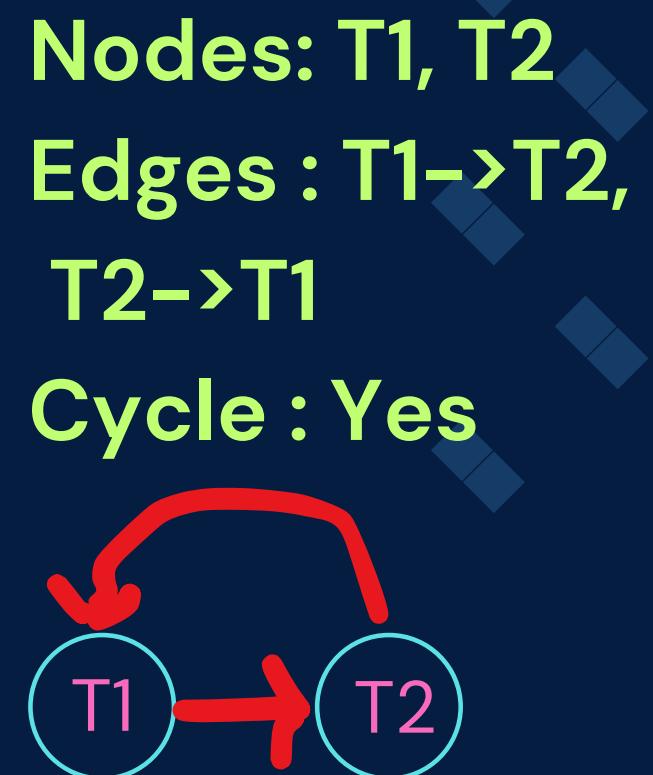
SERIAL SCHEDEULE

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

CONCURRENT SCHEDEULE

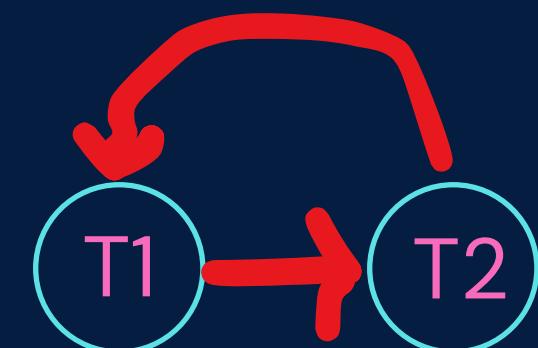
A concurrent schedule does not always have a cycle.

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.

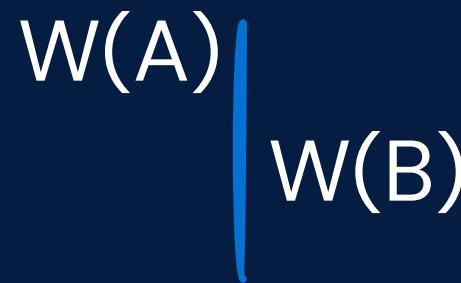
Conflict pairs : Read-Write, Write-Read , Write-Write(perfomed on the same data item)

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)

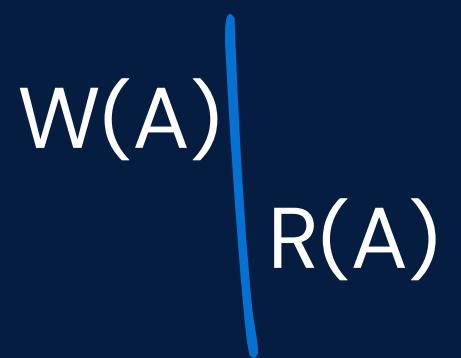
LET'S START WITH DBMS :).

Conflict-Serilizability

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

S1		T1	T2
		R(X)	
		W(X)	
			R(X)
		R(Y)	
		W(Y)	
			R(Y)

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

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

When done on the same data item

LET'S START WITH DBMS :).

Conflict-Serializability

- Read-Write (RW) Conflict
- Write-Read (WR) Conflict
- Write-Write (WW) Conflict

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.

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

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

Step 2: Find the nodes

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

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

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.

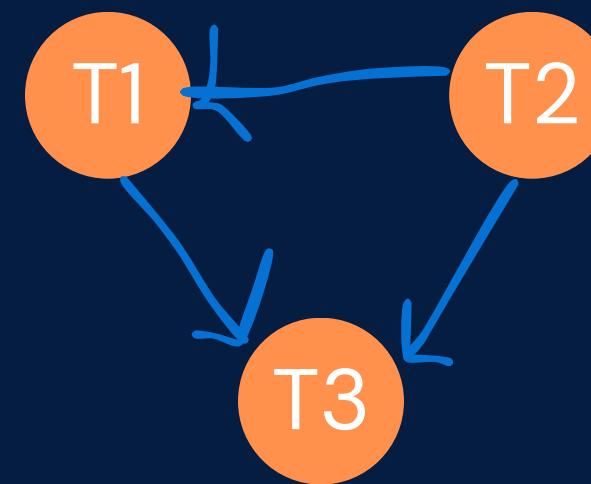
Read-write conflict
R(B) - W(B)
R(A)-W(A)

LET'S START WITH DBMS :).

Conflict-Serializability

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

Step 4 : Draw the graph



edges

- $T_2 \rightarrow T_1$
- $T_1 \rightarrow T_3$
- $T_2 \rightarrow T_3$

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

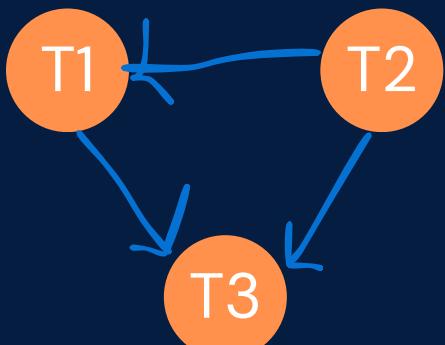
T1→T3→T2

T2→T1→T3

T2→T3→T1

T3→T2→T1

T3→T1→T2



edges
T2→T1
T1→T3
T2→T3

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

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**.

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 Ti is reading B which is updated by Tj then in S' also, Ti should read B which is updated by Tj.

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

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

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 Ti is the last to update A, then in schedule S', the final write operation on A should also be performed by Ti

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

S

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

S'

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.

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 :).

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.

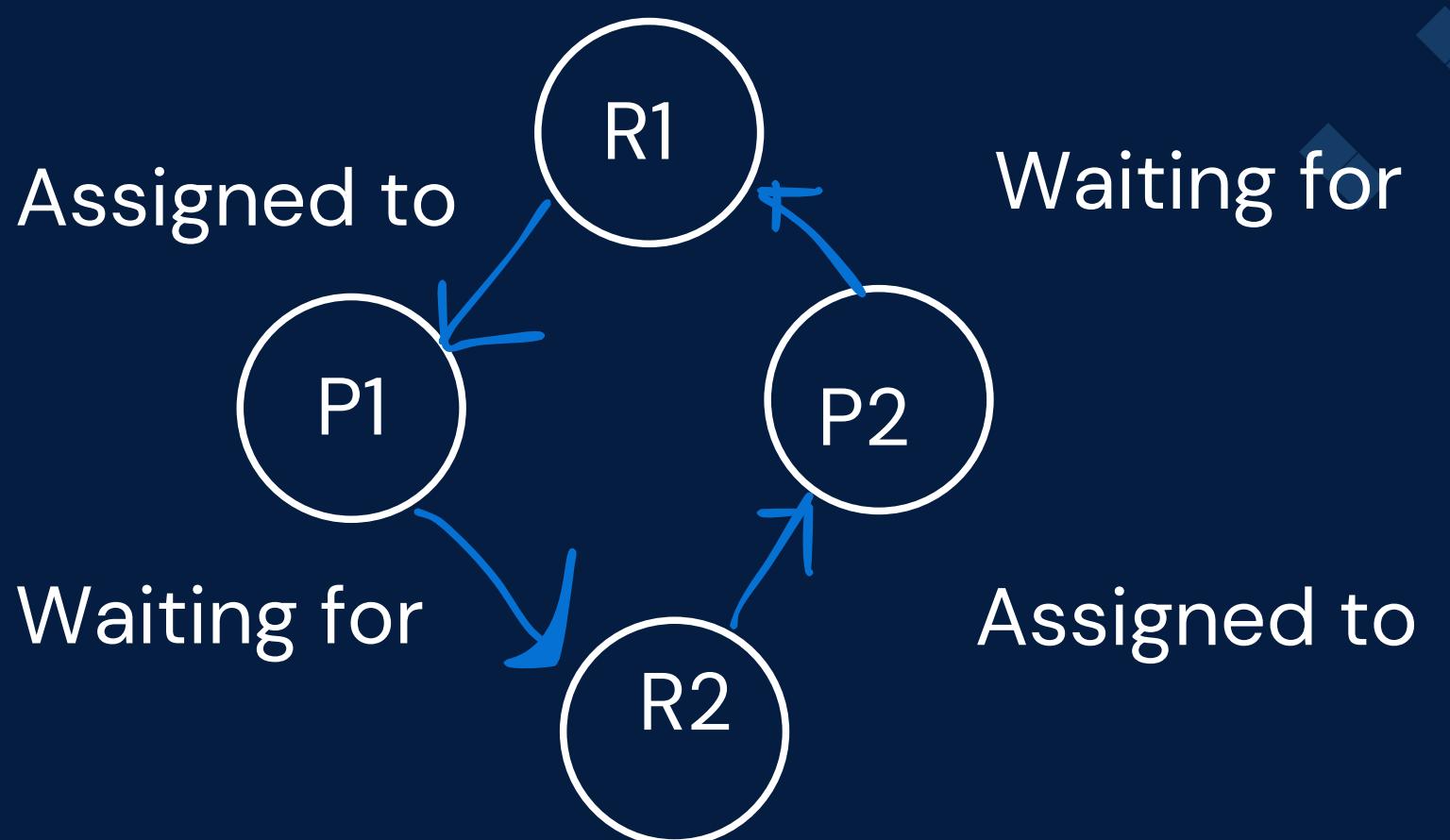
Irrecoverable : If Transaction B commits after the lock is released based on a modified value in transaction A which fails after sometime.

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.

T1	T2
R(A)	
	W(A)

T1	T2
W(A)	
	R(A)

T1	T2
W(A)	
	W(A)

T1	T2
	R(A)
	W(A)

T1	T2
	W(A)

T1	T2
	W(A)
	R(A)

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.