

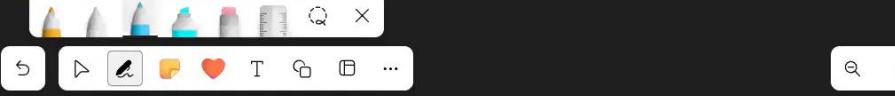
# SQL & DBMS

DATE - > 13/02/2024

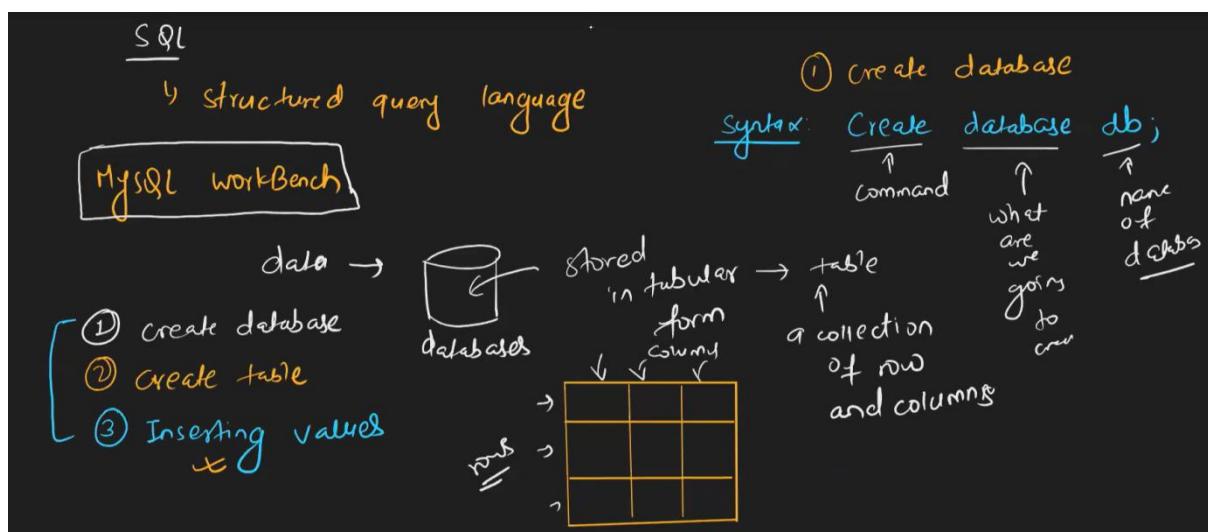
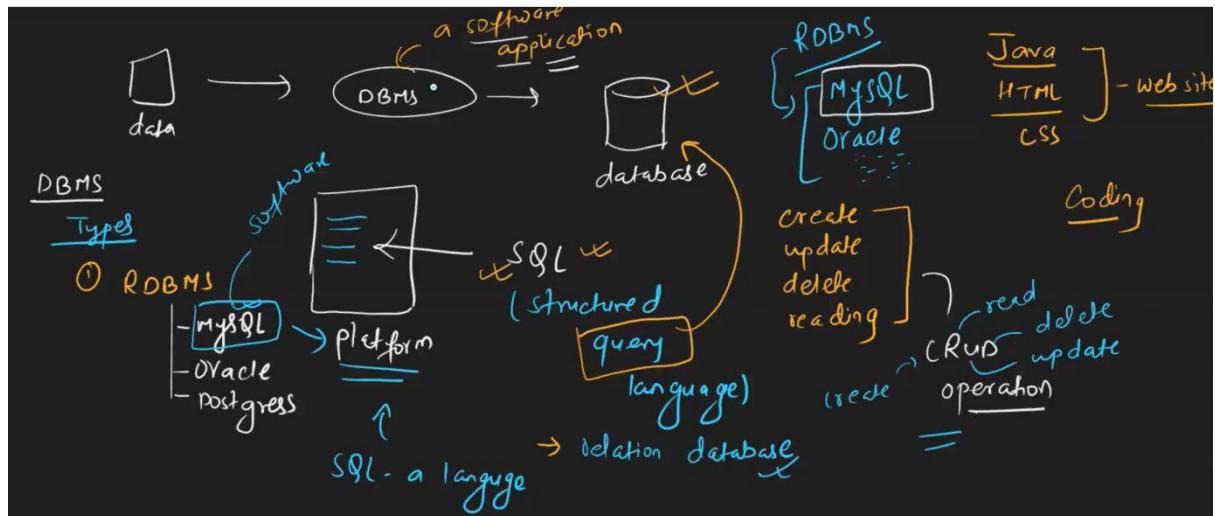
[DBMS Notes - Google Docs](#)

DBMS → a medium  
↳ Database Management System  
↳ a software that allows users to interact with databases.  
Examples  
MySQL, Oracle, SQL Server, MongoDB etc.

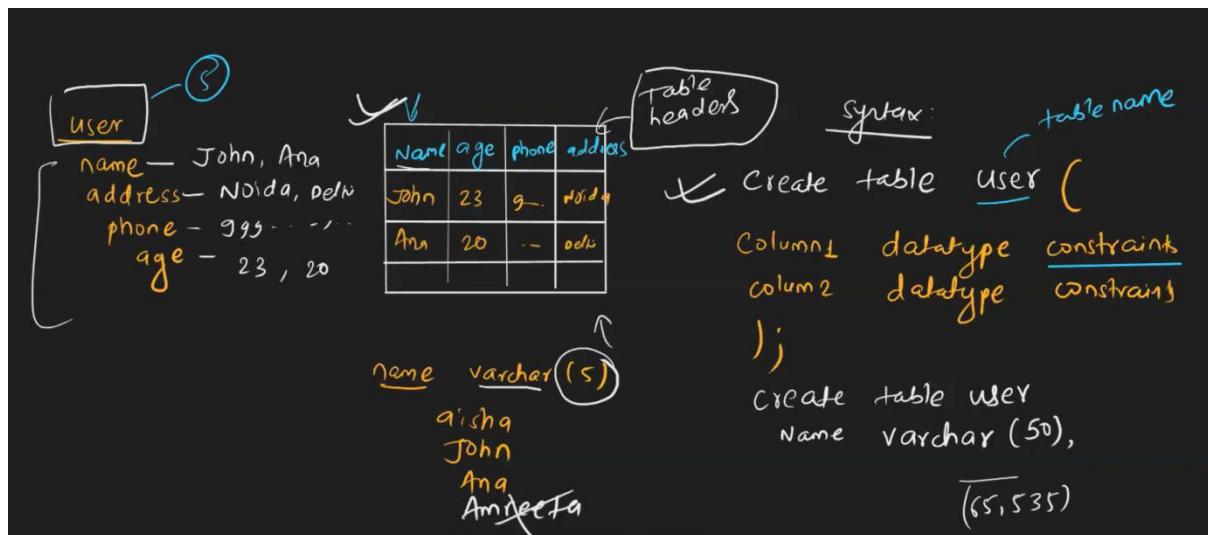
Types of DBMS

- ① RDBMS → data are organized in tabular form
- ② NoSQL DBMS - to handle unstructured data
- ③ Object oriented DBMS - db4o, objectDB
- ④ Graph DBMS - ArangoDB
- ⑤ Document DBMS - RavenDB

# TIME – 43



## Schemas – Structure



# Time – 1.41

To insert

Syntax:

```
Insert into tablename (col1, col2, col3 ... coln)
Values (val1, val2, val3 ... valn),
        (val2, val2, val2 ... valn);
```

```
Insert into users
(Id, username, age, address,
phone,
dateofbirth)
Values (1, "john", 23, "Noida",
"9811111111", '2000-01-01');
```

SQL File 5\*

```
1 • Insert into users( Id, UserName,age, Phone, address, dateOfBirth)
2 values(1, "john", 23, "9999999999", "Noida", '2000-01-01');
3
```

SQL File 5\*

```
1 • Insert into users
2 values(2, "Ana" ,20, "9999999990", "delhi", '2002-01-01'),
3 (3, "Anira", 20, "9090909090", "Delhi", '1998-09-09');
4
```

DATE → 14/02/2024

CONSTRAINTS → it is used for not exist duplicate

Constraints → are rules that we can apply on tables

~~Create table table-name (~~ col datatype constraints );

<u>id</u> int primary key,
<u>email</u> varchar(25) NOT NULL,
<u>phone</u> varchar(10) <u>Unique</u> ,
<u>age</u> int check ( <u>age</u> >= 18),
<u>country</u> varchar default "India",

① NOT NULL → we cannot have empty values.

② Unique → tells that all values must be unique

③ primary key → NOTNULL + unique  
→ there can be only one primary key  
→ uniquely identify each row in a table

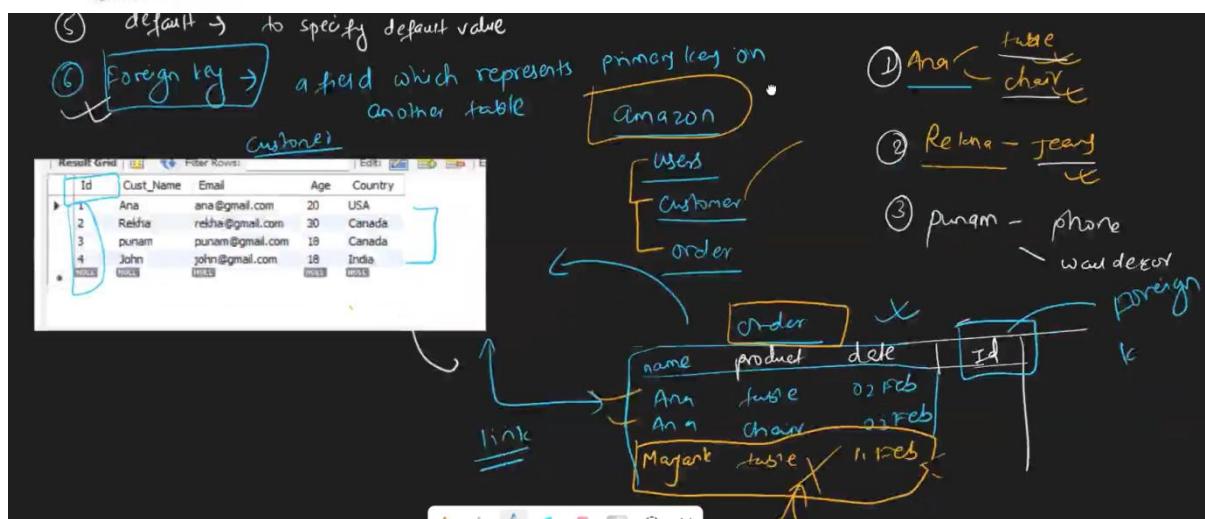
④ check → to validate the value of a column.

⑤ default → to specify default value

```

• • create table customer (
    Id int Primary Key,
    Cust_Name varchar(50) Not Null,
    Email varchar(60) unique,
    Age int check(age>=18),
    Country varchar(50) default "India"
);

```



## Use of FOREIGN KEY

```

1 • • create table orders (
2   OrderId int auto_increment unique,
3   Product varchar(50),
4   Id Int,
5   Foreign Key(Id) references customer(Id)
6 );
7 • Insert into orders
8   values (101, "Table", 1),
9   (102, "Chair", 1); I
0 • select * from orders

```

AND → &&

```

SQL File 5 ×
Limit to 1000 rows
1 • select * from customer where id = 1 && Cust_name = "Ana";
2

```

## NOT EQUAL TO → !=, <>

```
1 • select * from customer where id != 2;
```

IN OPTATOR → (specify multiple values)

BETWEEN → range of data

If you use between operators in name columns, it will work by dictionary order

```
1 • select * from customer where id IN (1, 4, 5); -- specify multiple values  
2 • select * from customer where id between 2 and 5; -- specify range of values
```

## DATE → 15/02/2024

## ACID PROPERTY →

In SQL, ACID typically stands for "Atomic, Consistent, Isolated, and Durable," which

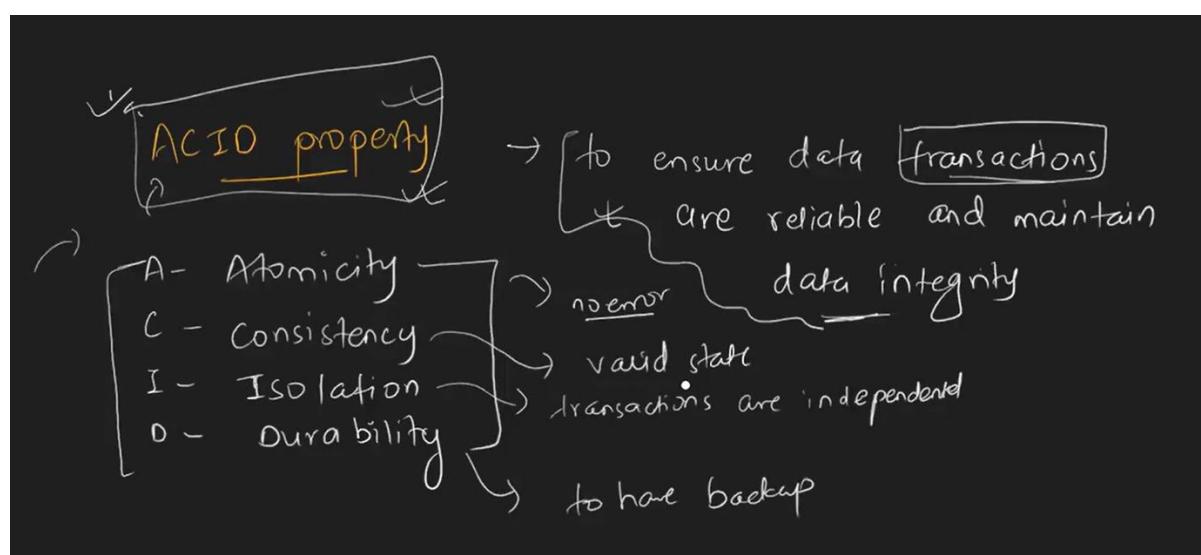
are the four properties that define the behavior of database transactions. These properties are commonly referred to as ACID properties. Here's what each property means:

A → Atomicity

C → Consistency

I → Isolation

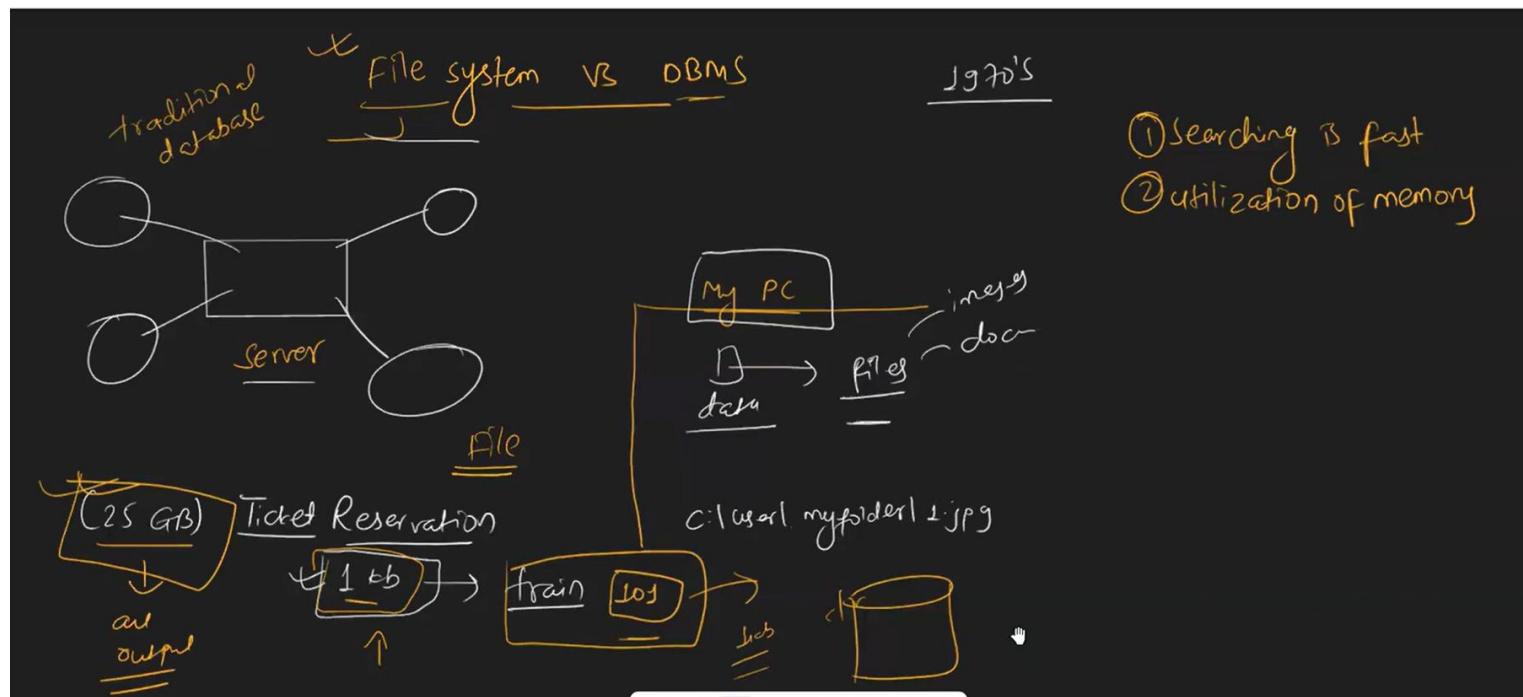
D → Durability



## File system Vs DBMS

**File System →** This system work on hierarchical format and it's for my PC only, if I need to share my data than I can't.

File system → c:/user/my\_folder/jpge



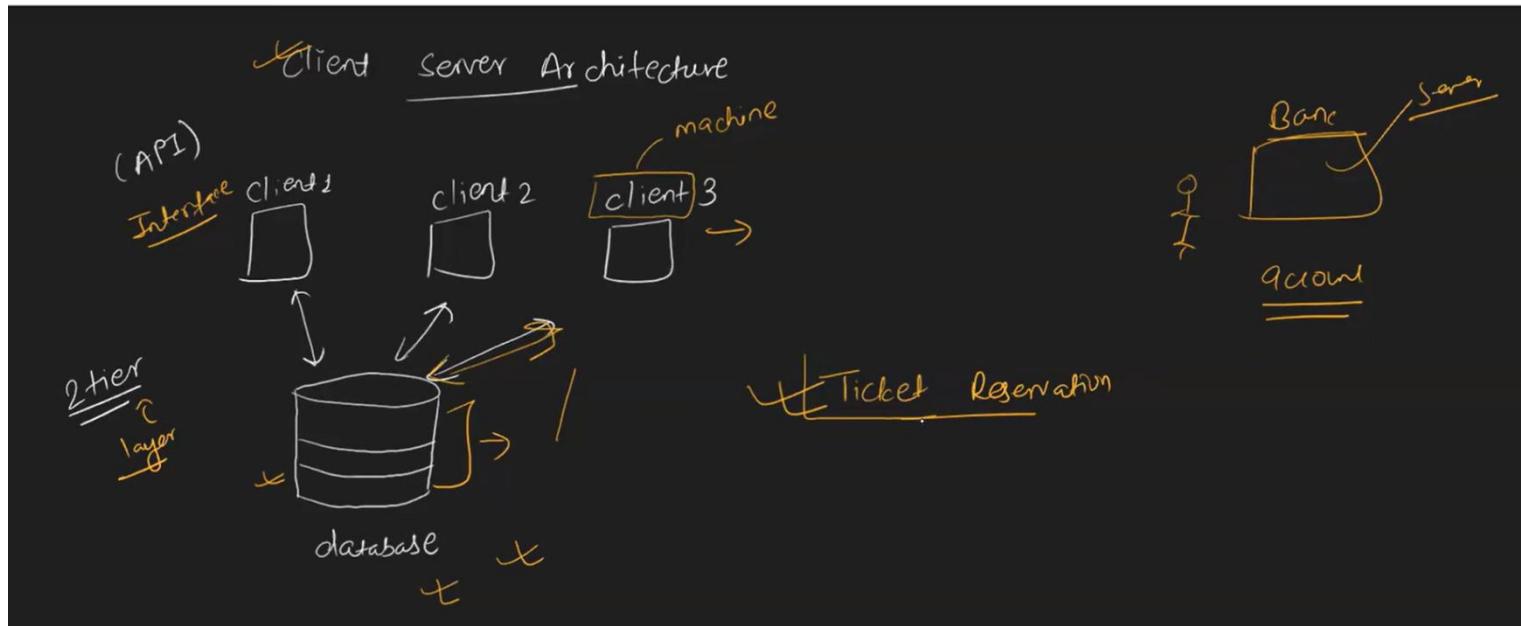
# Advantage of DBMS

## Advantages of DBMS

- ① Searching is fast
- ② Utilization of memory
- ③ Easy to access the data
- ④ Concurrency
- ⑤ Security
- ⑥ Data Redundancy
  - ↓  
Duplication

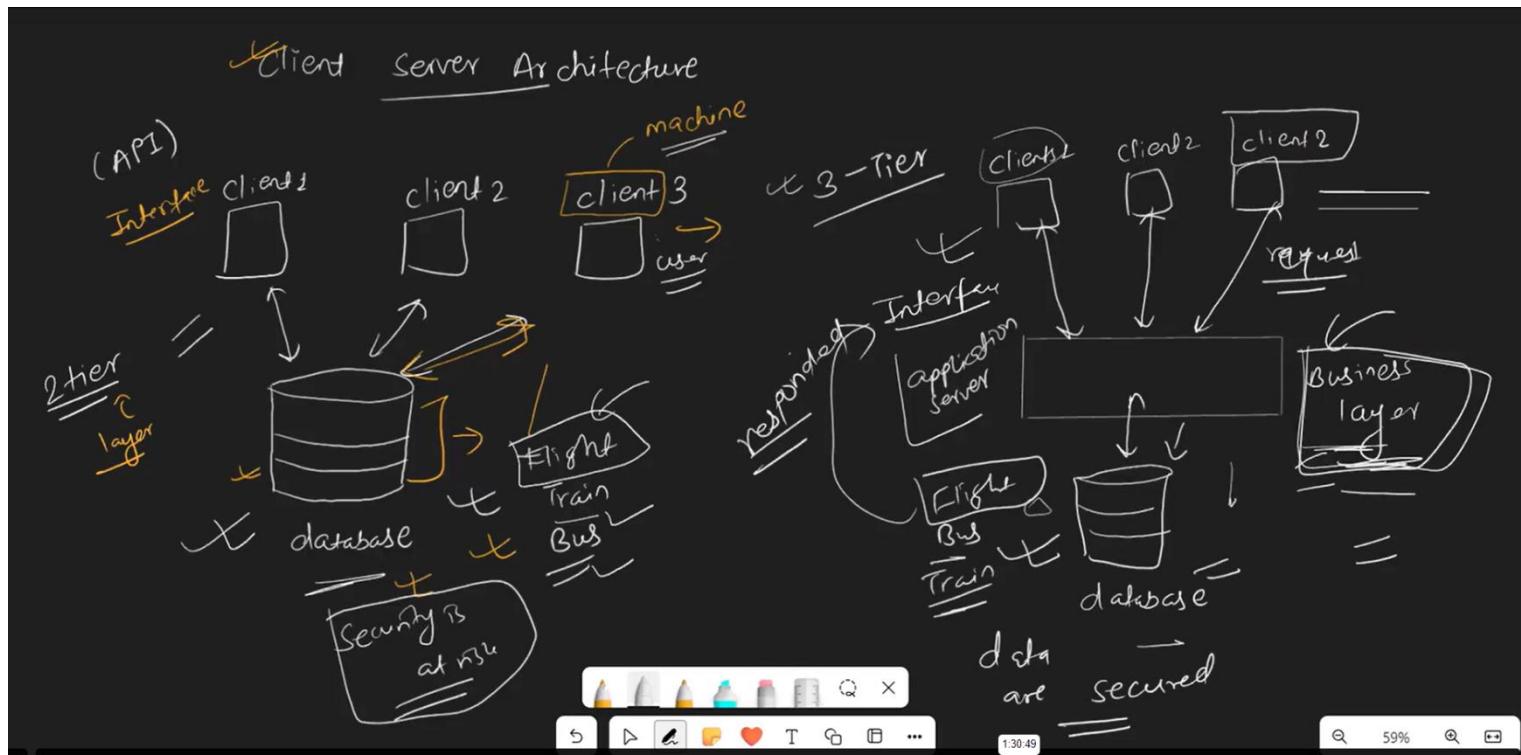
Time → 1.07

# Client Server Architecture →



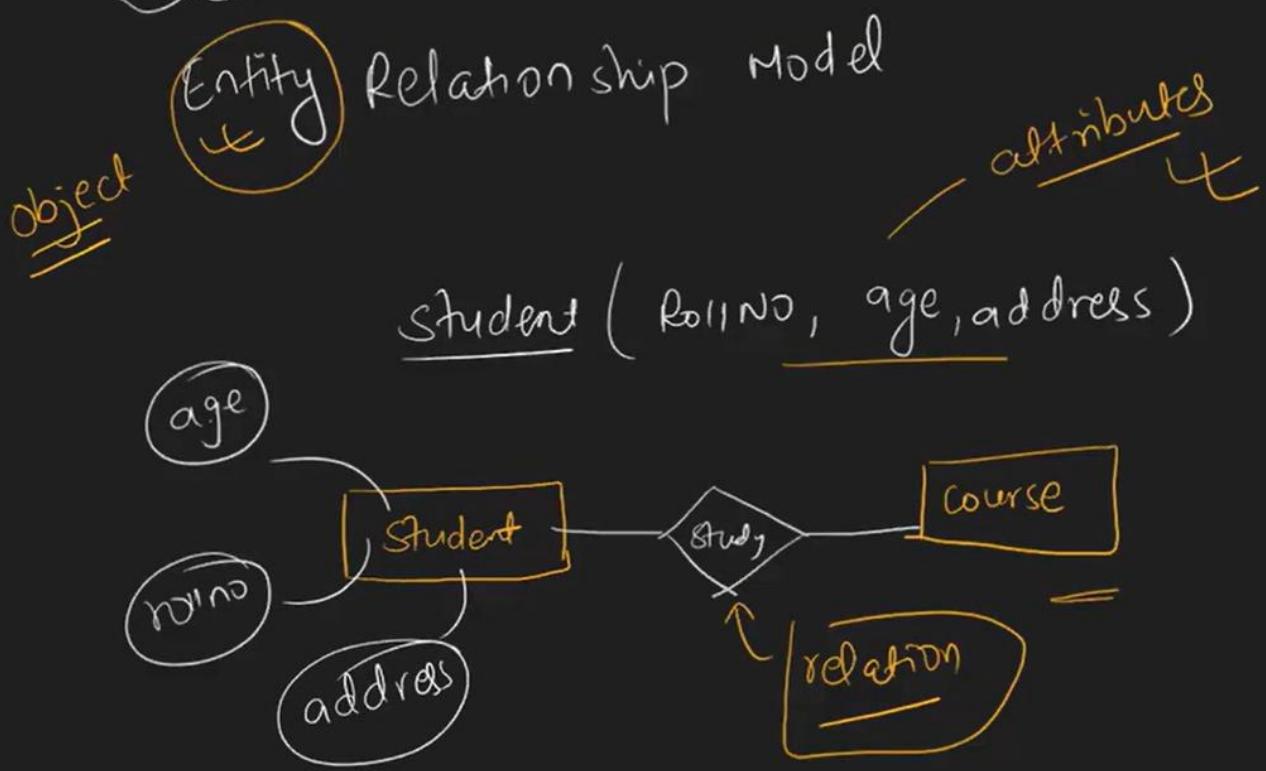
## 2-Tier And 3- Tier

3 Tier is good for now days, it is secure and easy to excess



## ER-Model-- > Entity Relationship model

# ER-Model



## Question →

File Edit View

1. Explain the difference between a database and a DBMS. Provide examples to illustrate your explanation.
2. Define the concepts of entities, attributes, and relationships in the context of Entity-Relationship Modeling. Give examples of each.
3. Write an SQL query to retrieve the names of all students who are enrolled in a course titled "Database Management Systems".
4. Given the following tables: Students (StudentID, Name, Age) and Courses (CourseID, Title, Instructor), write SQL queries to perform the following operations:
  - Insert 10 new students into the Students table.
  - Insert 5 courses in course table.
  - select all of the information of the student from students table.
5. Describe the ACID properties of transactions in DBMS. How do these properties ensure data consistency and integrity?
6. Discuss the advantages and disadvantages of using a relational database management system (RDBMS) compared to a NoSQL database.
7. What are primary keys and foreign keys in a relational database?  
(Define primary keys and foreign keys and explain their role in establishing relationships between tables and enforcing referential integrity.)

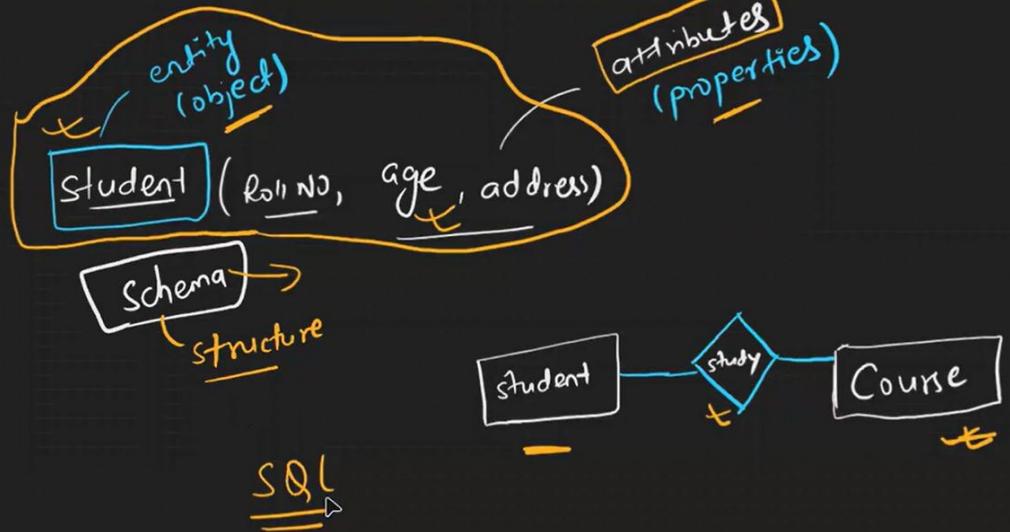
Date- 16/02/2024

## X Entity Relationship Model

↳ a technique for representing the logical structure of databases

an object

X



## Types of attributes

① Single vs multi valued

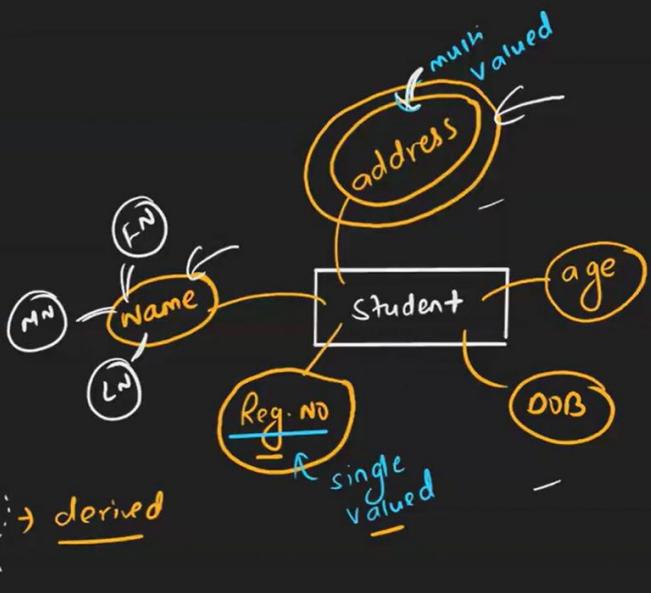
② Simple vs composite

cannot be divided further

③ Stored vs derived

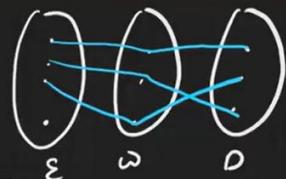
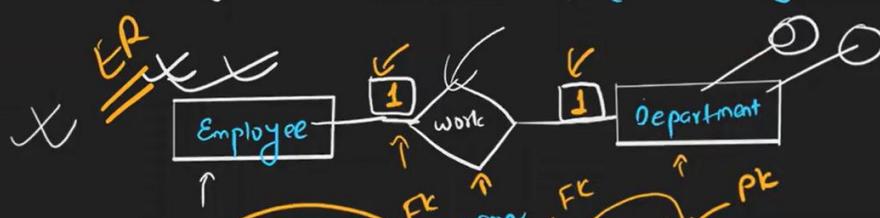
key vs non-key

Unique values



## Types of Relationship (Cardinality)

1-1  
1-M  
M-1  
N-N

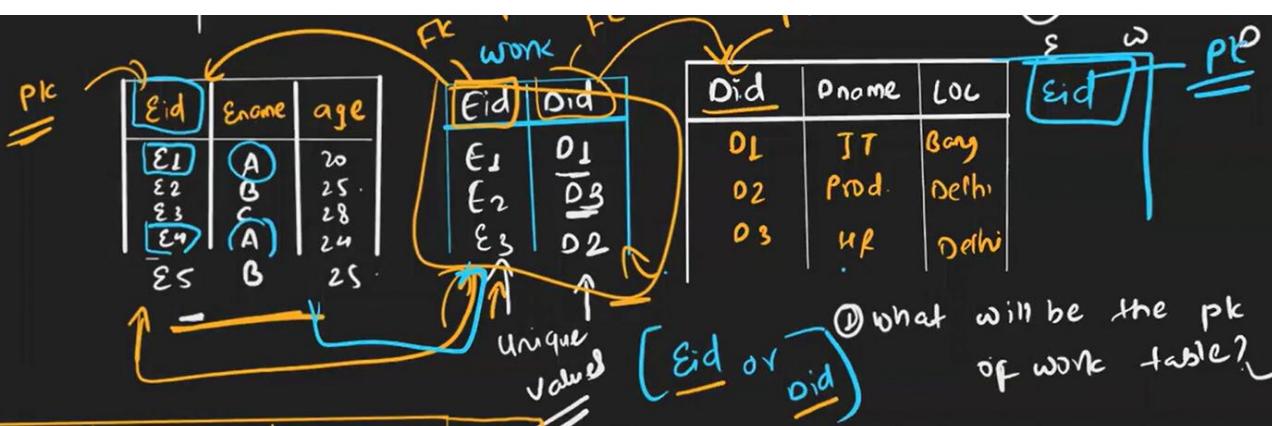


Eid	Ename	age	Did	Dname	Loc
E1	A	20	D1	IT	Bang
E2	B	25	D2	Prod.	Delhi
E3	C	28	D3	HR	Delhi
E4	A	24			
E5	B	25			

Unique values

[Eid or Did]

what will be the pk of work table?

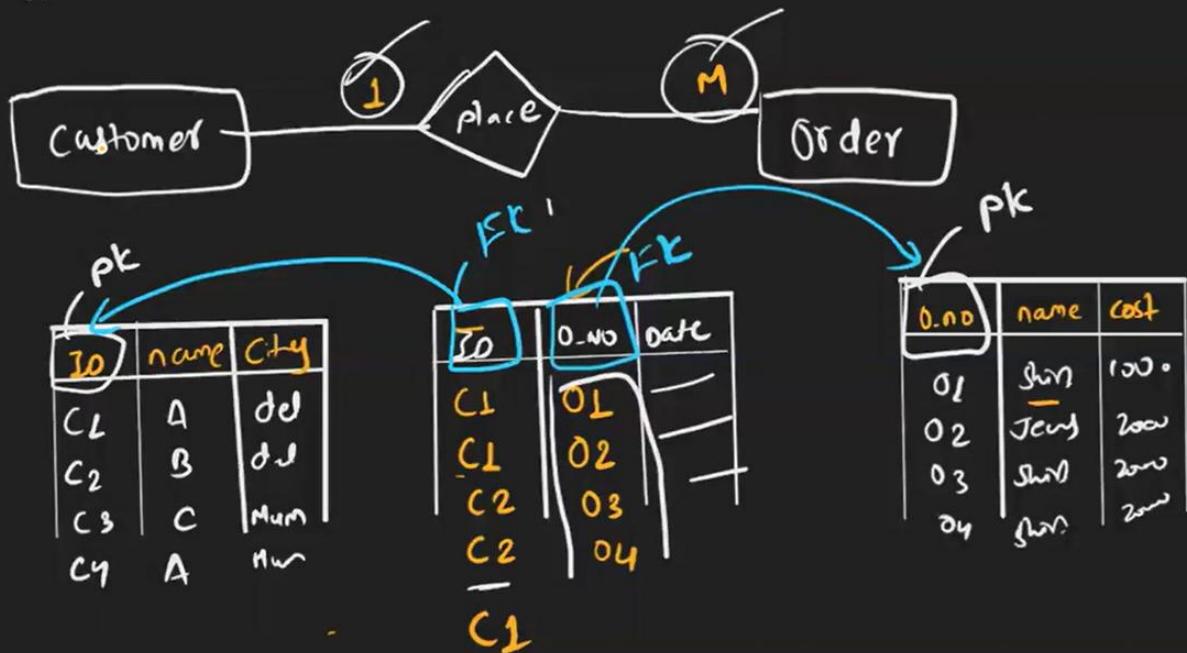


Eid	Ename	age	Did
E1	A	20	D1
E2	B	25	D3
E3	C	28	D2
E4	A	24	-

② Reduction of table

②

One to Many



Teacher

ID	name	age
T1	A	
T2	B	

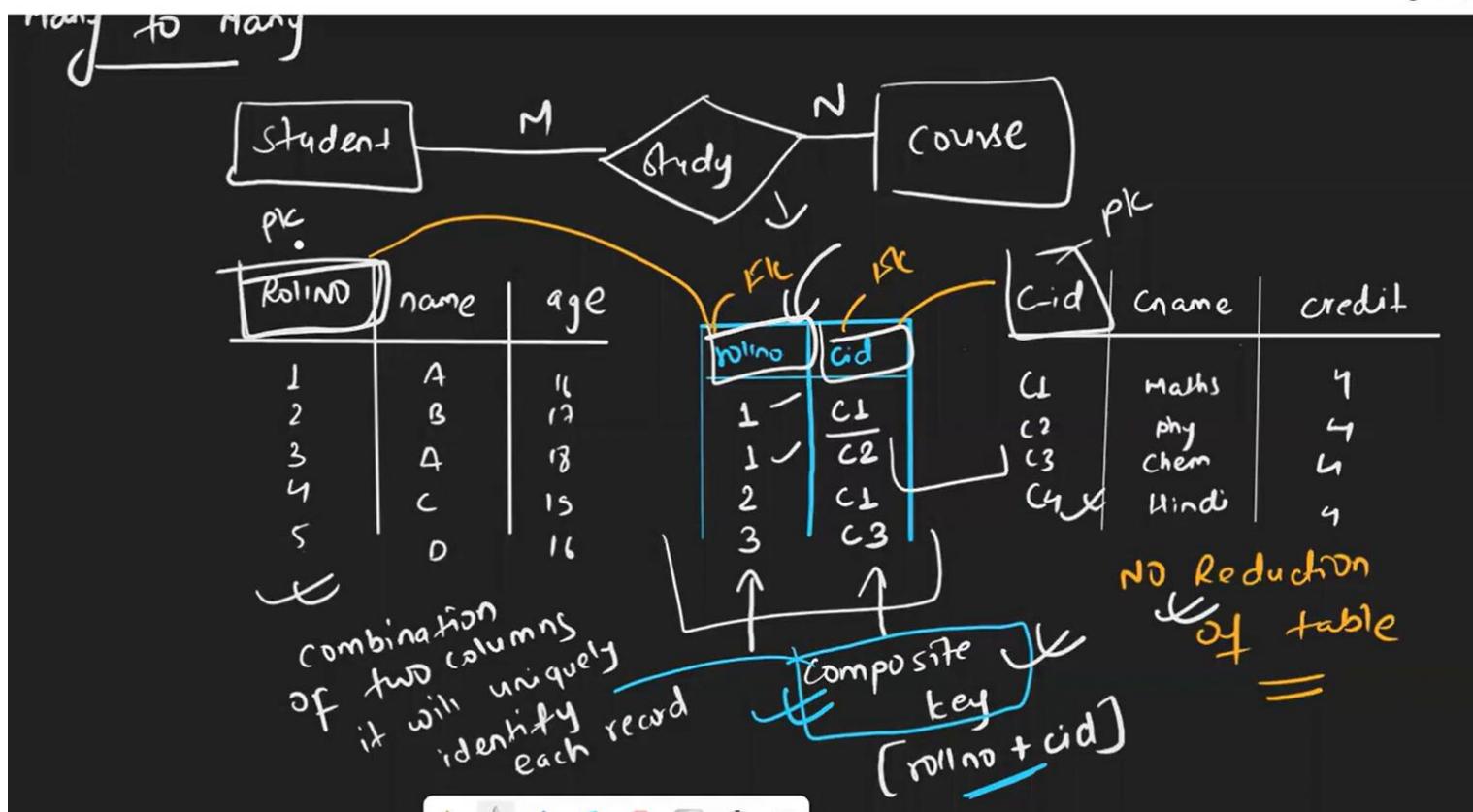
Teaching

ID	SID
T1	S1
T1	S2

Student

SID	Score
S1	X
S2	Y

# MANY TO MANY RELATIONSHIP

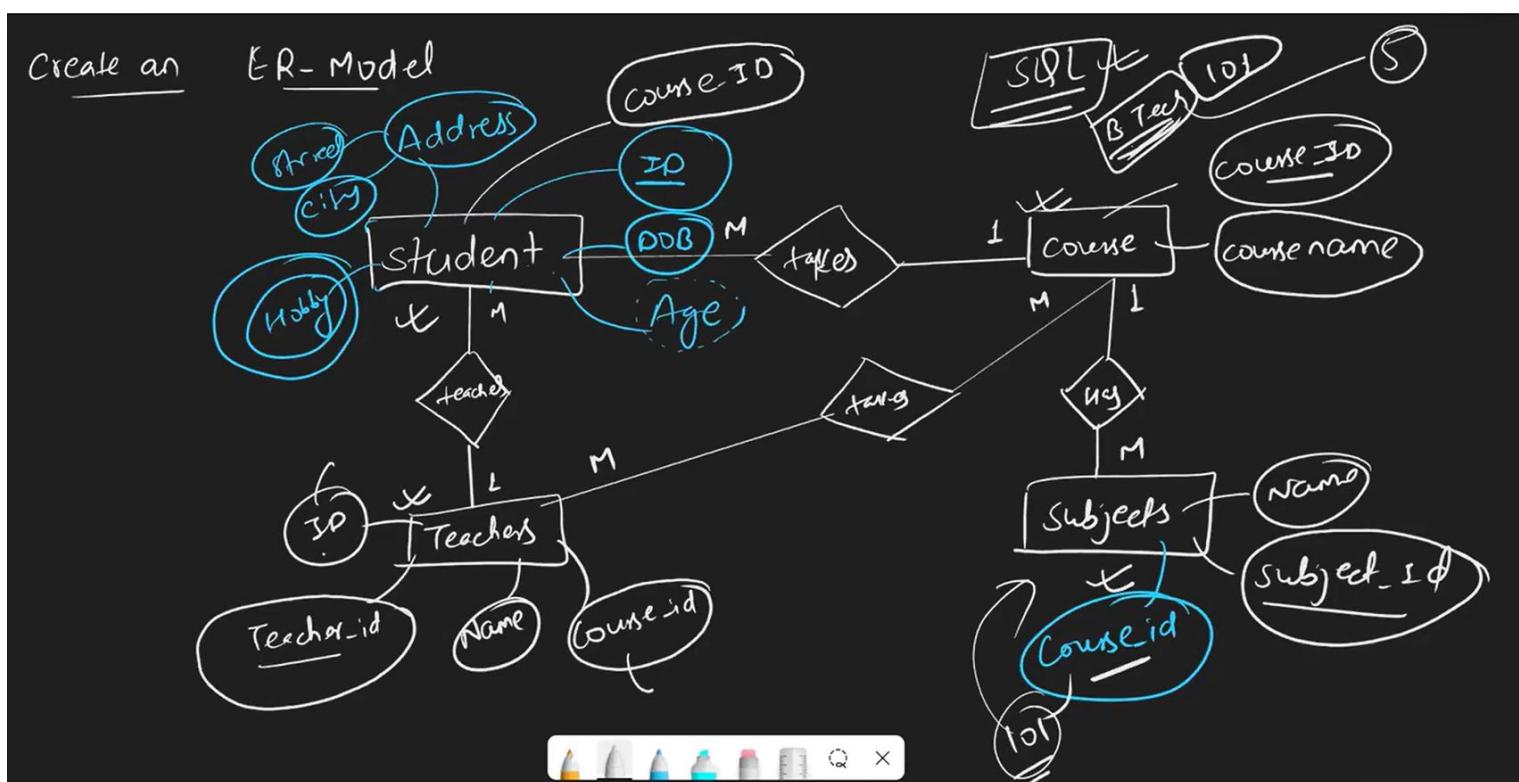
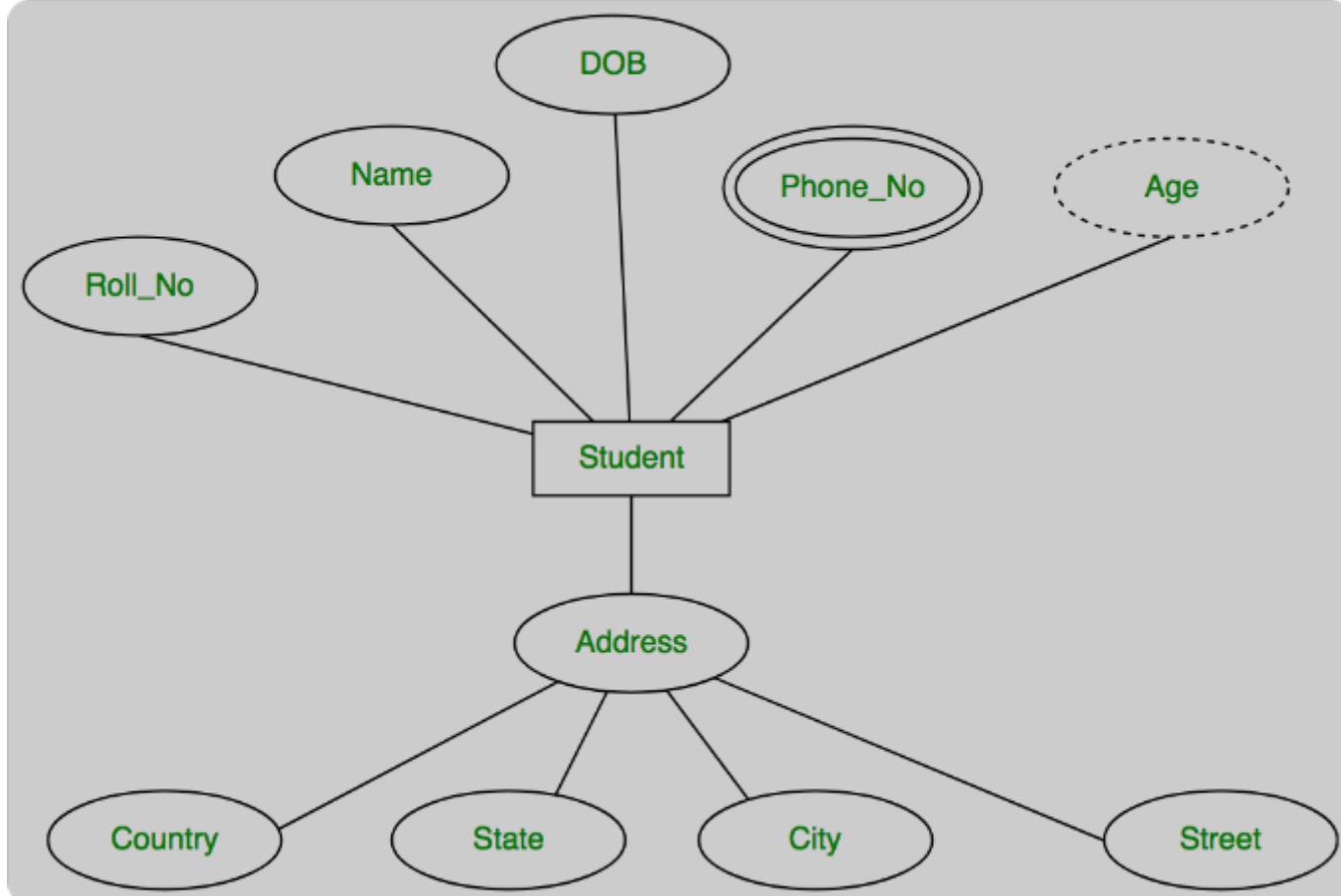


## TODAY LAST TOPIC

Date- 19/02/2024

ER Diagram in DBMS. An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database.

Example of ER model based on STUDENT TABLE



# Home Work : create an ER-Model of library management system



① Create an ER-Model of library Management system



**NORMALIZATION → it's methods to remove the duplicate values in table,**

**Row label duplicate → if any duplicate in row then It's call row label duplicate,**

**Column label duplicate → if any duplicate in column then it's call columns label duplicate**

**Note = if you want to remove row label duplicate you can remove by using primary key**

Normalization

↳ technique to remove OR reduce redundancy from a table (duplicacy)

• Ins

**Row level duplicacy**

**Four records**

**1 lac M**

**So Problem is**

**Deleting sum data**

**Updating data**

**Inserting data**

Normalization

↳ technique to remove OR reduce redundancy from a table (duplicacy)

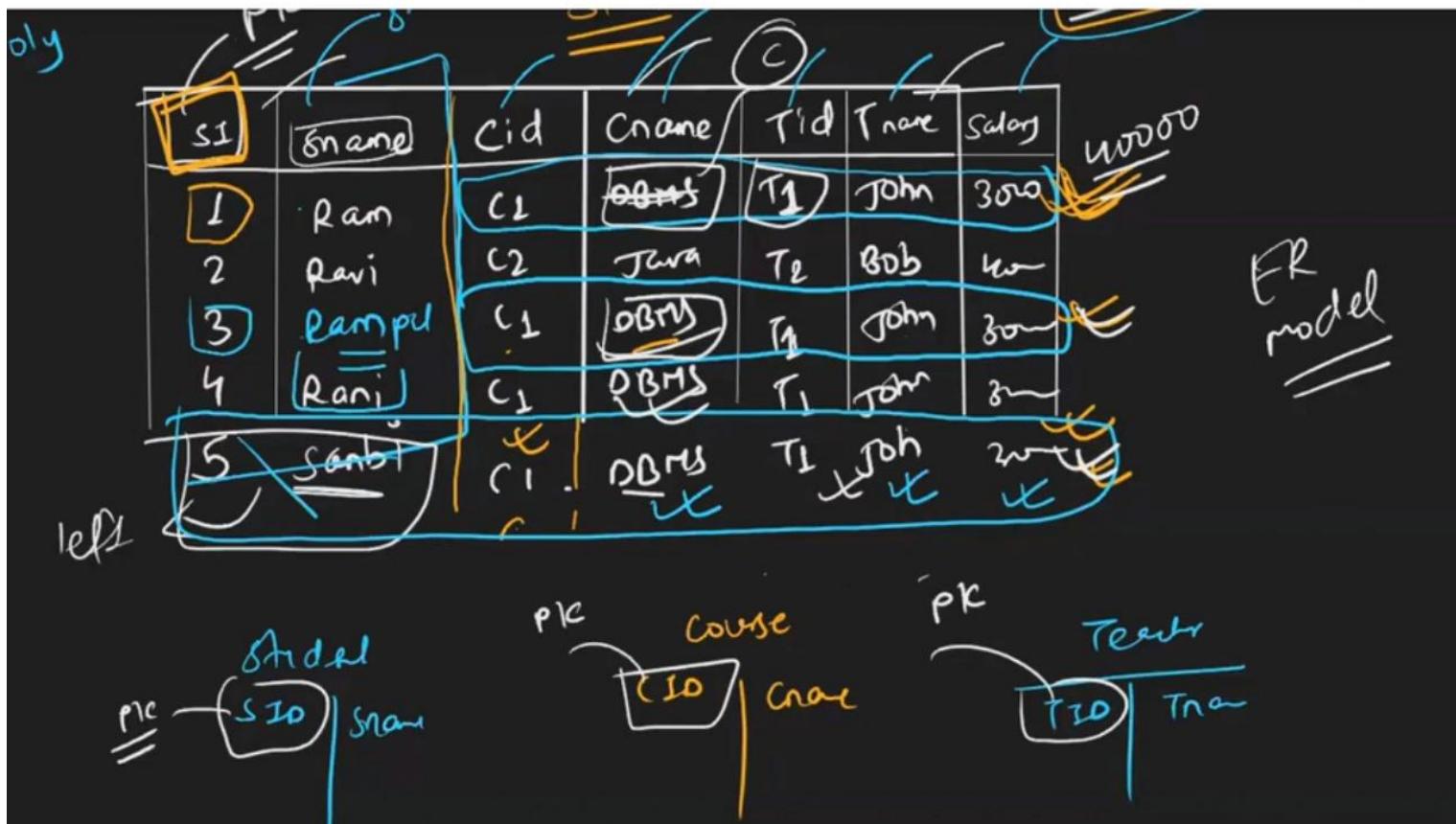
• Ins

**Row level duplicacy**

**Four records**

**1 lac M**

# SOLUTIONS → you have to create different – diff't table with primary key



## Father of DBMS->

Edgar Frank "Ted" Codd (19 August 1923 – 18 April 2003) was an English computer scientist who, while working for IBM, invented the relational model for database management, the theoretical basis for relational databases and relational database management systems.



## Normalization Process→

According to EF Codd (father of dmbs) is table should not contain any multivalued attribute ``

1. Reduce redundancy: By storing data in a structured way, duplication of data is minimized, which saves storage space and reduces the likelihood of inconsistencies.
2. Minimize data anomalies: Normalization helps to prevent insertion, update, and deletion anomalies, ensuring data integrity and consistency.
3. Simplify database design: A normalized database schema typically leads to a more organized and logical database structure, which makes it easier to understand, maintain, and modify.

Normalization is usually carried out through a series of steps, typically categorized into different normal forms (such as First Normal Form, Second Normal Form, Third Normal Form, etc.), each addressing specific types of data redundancies and dependencies.

In practice, normalization involves breaking down larger tables into smaller, more manageable tables and establishing relationships between them using foreign keys. This process helps to ensure that each table represents a single entity or concept and that data is stored without unnecessary duplication or dependency.

## **1NF (first Normal Form)**

## **2NF**

## **3NF**

## **FIREST NORMAL FORM→**

## Normalization

### First Normal Form

→ Table should not contain any multivalued attributes

Student

RollNo	Name	Course
1	Sai	C
2	Harsh	Java
3	Omkar	C.

**COMPOSED PRIMARY KEY** → this is the combination of two columns (by bihalp of columns we create primary key )

### Normal Form

→ Table should not contain any multivalued attributes

Student

RollNo	Name	Course
1	Sai	C/C++
2	Harsh	Java
3	Omkar	C/DBMS

RollNo	Name	Course 1	Course 2
1	Sai	C	C++
2	Harsh	Java	
3	Omkar	C.	DBMS

RollNo	Name	Course
1	Sai	C
1	Sai	C++
2	Harsh	Java
3	Omkar	C
3	Omkar	DBMS

Composite Primary Key → [RollNumber + Course]

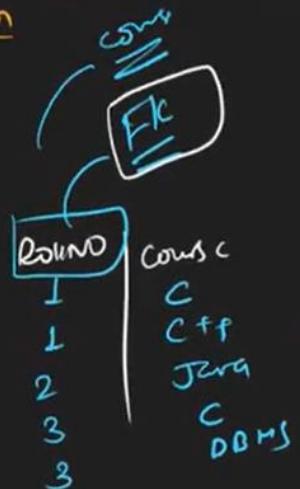
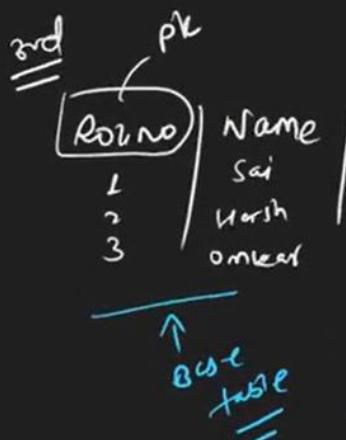
# Three types of solution

2nd		PK	RollNo	Name	Course 1	Course 2				
1	Sai			C		C++	NET	Java	DBMS	
2	Harsh			Java			NET			
3	Dmbar			C		DBMS				

18

<u>RollNo</u>	Name	<u>Course</u>
1	Sai	C
1	Sai	C++
2	Math	Jung
3	Omkar	C
3	Omkar	OBMIS

→ { RollNumber + Course }



Create table — (

Rounds full,

Name      Varchar(10),

Course      Varsh (v)

X Primary key (RollNo, name)

# **2<sup>nd</sup> Normal Form →**

1. Table should be 1<sup>st</sup> normal form
  2. All the non-prime attribute should be fully functional dependent on candidate key

## Second Normal FORM

- ↳ Table should be in 1NF
- ↳ All the non-prime attributes should be fully functional dependent on
  - candidate key

## Second Normal FORM

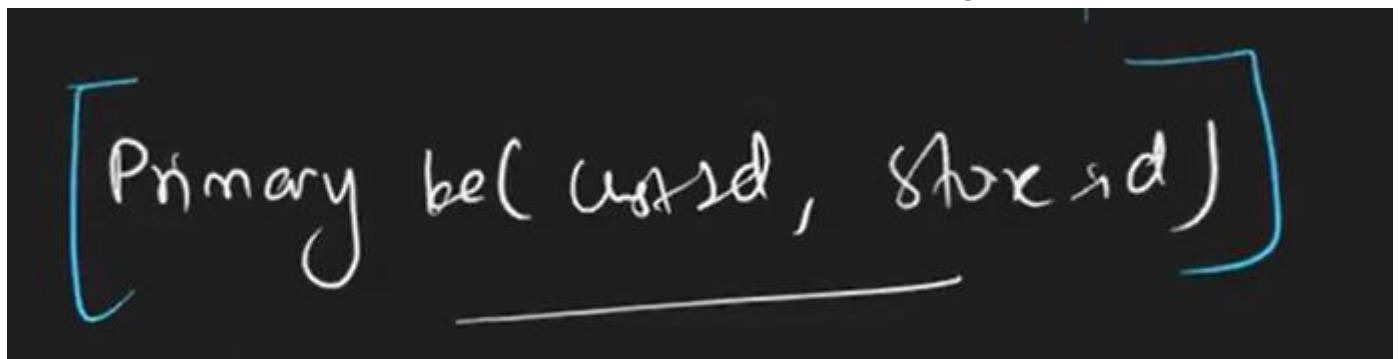
- ↳ Table should be in 1NF
- ↳ All the non-prime attributes should be fully functional dependent on
  - candidate key

Customer		
Cust-ID	Store-id	location
1	2	Delhi
1	3	Mumbai
2	1	Delhi
3	2	Bangalore
4	3	Mumbai

candidate key - custID + storeID  
 prime attr - custID, storeID  
 non prime - location

**Candidate Key → It's a both either primary key & composed key**

In SQL database management systems (DBMS), a candidate key is a set of one or more columns that can uniquely identify a tuple (row) within a table. It means that no two distinct rows in the table can have the same combination of values for the candidate key columns.



**Non-Prime Attribute(gun) → Any attribute that is not part of any candidate key is considered a non-prime attribute.**

Customer		
Cust-ID	Store-id	Location
1	2	Delhi
1	3	Mumbai
2	1	Delhi
3	2	Bangalore
4	3	Mumbai

candidate key — custID + storeID  
prime att — custID, store-ID  
non prime — location

## Second Normal Form

- ↳ table should be in 1NF
- ↳ All the non-prime attributes should be fully functional dependent on candidate key

Customer		
Cust-ID	Store-ID	Location
1	2	Delhi
1	3	Mumbai
2	1	Delhi
3	2	Bangalore
4	3	Mumbai

candidate key - custID + storeID

prime attr - (custID, storeID)

non prime - location

dependent on

can - candidate key

Candidate Key		
Store-ID	PK	candidate key
1		
2		
3		
	X	

location	
Delhi	
Bangalore	
Mumbai	

Customer	
Cust-ID	Store-ID
x	1
x	3
2	1
3	2
4	3

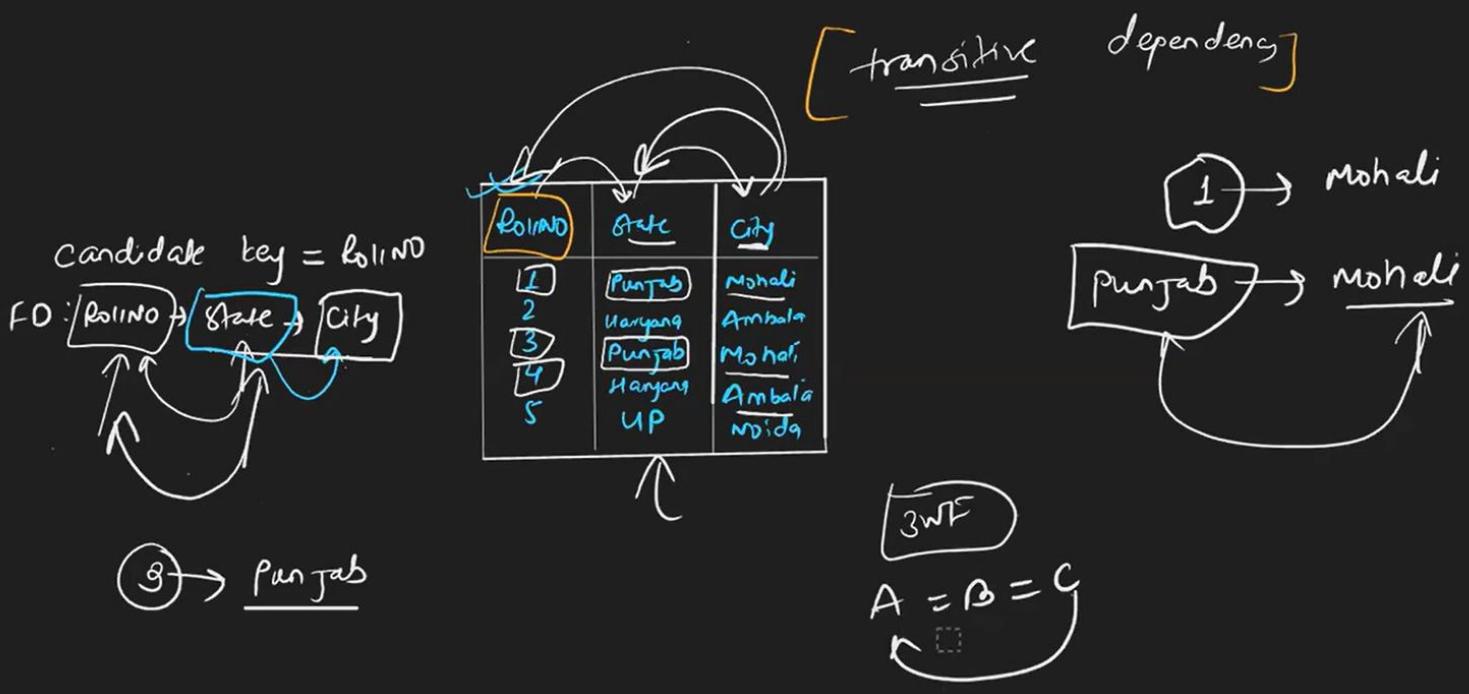
**20/02/2024**

**Third Normal Form →**

**Third Normal Form (3NF)**

A relation is in the third normal form, if there is no transitive dependency for non-prime attributes as well as it is in the second normal form. A relation is in 3NF if at least one of the following conditions holds in every non-trivial function dependency  $X \rightarrow Y$ .

- X is a super key.
- Y is a prime attribute (each element of Y is part of some candidate key).



### Third Normal Form

- ① must be in 2NF
- ② there should be no transitive dependency in table

(S10, (ID, course)) <sup>non-prime att</sup>

PK → Emp-Id

	<u>Ename</u>	<u>dname</u>	<u>Location</u>
1	Alice	HR	New York
2	Bob	Sales	LA
3	Charlie	IT	San Francisco

3NF  
There should not  
be transitive  
dependency

$dname \rightarrow Location \rightarrow Ename$   
 $HR \rightarrow New York \rightarrow$

PK → Emp-Id | Ename

dname | Location

Emp-Id | deptName

PK → Emp-Id

	<u>Ename</u>	<u>dname</u>	<u>Location</u>	<u>dept-id</u>
1	Alice	HR	New York	d1
2	Bob	Sales	LA	d2
3	Charlie	IT	San Francisco	d3
4	Anna	IT	San Francisco	

$dname \rightarrow deptid \rightarrow location$

```

select * from employee;
delete from employee where empid = 7; -- delete a specific record
drop table demo1; -- delete the whole table
truncate table demo; -- delete all the records
    
```

I

```

update employee set email = "steffy@gmail.com" where empid = 3;
update employee set salary = 70000 where empid = 5;

alter table employee drop email; -- delete a particular column
alter table employee add Hobby varchar(50); -- to add a new column in the tabl
    
```

```
alter table employee rename column address to location;  
-- to rename a column  
alter table employee  
modify column salary varchar(20); -- modify the datatypes
```

## Aggregate Function

```
1 -- aggregate function  
2 • select * from employee;  
3 • select max(salary) from employee;  
4 • select min(salary) from employee;  
5 • select sum(salary) from employee;  
6 • select avg(salary) from employee;  
7 • select count(empid) from employee;  
  
-- aggregate function  
select count(empid) from employee;  
select count(*), location from employee  
group by location;  
-- group by clause is used to group rows  
-- that have the same values into summary rows  
-- typically used with aggregate function
```

We can't combined any column with aggregate fn,  
And can't use where clause so that in this condition  
we can use having clause (it same like where clause)

Question :

```
1 • SELECT * FROM orders;  
2 -- find the number of orders placed by each customer  
3 • select count(*), customer_id from orders  
4 group by customer_id;
```

[ ]

Result Grid | Filter Rows: \_\_\_\_\_ | Export: Wrap Cell Content:

count(*)	customer_id
2	101
2	102
1	103

## HAVING CLAUSE

```
1 • SELECT * FROM orders;  
2 -- find the number of orders placed by each customer who has placed more than 1 order  
3 • select count(*), customer_id from orders  
4 group by customer_id  
5 having count(*) > 1; -- we use having to apply condition with aggregate function
```

[ ]

Result Grid | Filter Rows: \_\_\_\_\_ | Export: Wrap Cell Content:

count(*)	customer_id
2	101
2	102

```
1 • SELECT * FROM orders;
2   -- Find customers whose total order amount exceeds a certain value, say $100.
3 • select customer_id, sum(total_amount) from orders
4 group by customer_id
5 having sum(total_amount)>100;
```

I

result Grid	
customer_id	sum(total_amount)
101	150.00
102	120.00

```
1 • select * from employee;
2   -- Like operator : to search for a pattern
3 • select * from employee where name like "r%";| I
4
5   -- starts with r
6   -- % symbolize any number of character, remaining characters
```

```
1 • select * from employee;
2   -- Like operator : to search for a pattern
3 • select * from employee where name like "%a%"; -- ends with an a
4
5   -- starts with r
6   -- % symbolize any number of character, remaining characters
```

```
select * from employee;
-- Like operator : to search for a pattern
select * from employee where name like " Ana";| I

-- starts with r
-- % symbolize any number of character, remaining characters
```

id					
id	name	location	age	salary	Hobby
Ana	Pune	34	30	NULL	

Date → 21/02/2024

## Order by (desc, limit, offset, distinct)

Offset -> it's means leave first record and show after.

```
1 -- order by : its is used for sorting
2 • select salary from employee order by salary; -- for ascending order
3 • select salary from employee order by salary desc; -- for descending order
4
5 -- limit : to retrieve specific number of records ( extracting top records of the table)
6 • select * from employee limit 3 offset 2;
7
8 -- offset : to skip rows from top
```

## SUB QUERY → Query inside in the other query.

Subquery

Nested query, a query inside any other query.

outer query      inner query      result

Query 1      Query 2      Query 3

innermost

Select \* from employee where salary = (select max(salary) from employee);

```
1 • select * from employee where salary =(select max(salary) from employee);
```

select \* from singh where profit =(select profit from singh order by profit desc limit 1);

## X Subquery

EmpId	name	location	age	salary
1	Edwin	Noida	32	100000
2	Reetu	Delhi	23	60000
3	Steffy	Noida	22	90000
4	Rekha	delhi	45	89000
5	Ana	Pune	34	70000
6	ravi	Noida	40	90000

↳ Nested query, a query inside any other query.



2nd  
Second highest

Select \* from employee where salary = 90000

(Select distinct salary from employee order by desc limit 1 offset 1) ;

## JOIN

In SQL, the JOIN operation is used to combine rows from two or more tables based on a related column between them. It allows you to retrieve data from multiple tables simultaneously. JOINS are fundamental to relational databases as they enable you to establish relationships between tables and retrieve meaningful information by correlating data across these tables.

There are several types of JOINs in SQL, including:

1. INNER JOIN: Returns only the rows that have matching values in both tables.
2. LEFT JOIN (or LEFT OUTER JOIN): Returns all rows from the left table (the first table mentioned in the query) and matching rows from the right table.
3. RIGHT JOIN (or RIGHT OUTER JOIN): Returns all rows from the right table and matching rows from the left table.
4. FULL JOIN (or FULL OUTER JOIN): Returns all rows when there is a match in either the left or right table.
5. CROSS JOIN: Returns the Cartesian product of the two tables, i.e., all possible combinations of rows.

## INNER JOIN (natural join) →

## Join

↳ to combine two tables

↳ there should be one common column  
↳ If there is link between two tables  
    dept\_id → FIC

pre employee	
Emp Id	Ename
1	A
2	B
3	C

dept_id	dept_name	Emp Id
101	HR	1
102	IT	2

### ① Inner Join

↳ matching values from two tables



Select \* from employee

Inner Join department

ON employee.EmpId = department.EmpId;

Empid	Ename	depid	deptname	Empid
1	A	101	HR	1
2	B	102	IT	2

## IF YOU WANT TO KNOW ONLY HR DEPT

pre employee	
Emp Id	Ename
1	A
2	B
3	C

dept_id	dept_name	Emp Id
101	HR	1
102	IT	2
101	HR	3

Select employee.ename, department.dept\_name  
Inner Join department  
ON employee.EmpId = department.EmpId;  
where department.dept\_name = "HR";

```
1 • select employee.name, department.deptname from employee inner join department
2   on employee.empid = department.id
3   where department.deptname = "Shipping";
```

**LEFT JOIN** → all data show from left table and  
matching data will show from right table

**NOTE- 1<sup>st</sup> table is left and 2<sup>nd</sup> table is right table  
always**

name, dept.name

### X Join

b) to combine two tables

b) there should be one common column

b) if there is link between two tables

pk employee

EmpId	Ename
1	A
2	B
3	C
4	D

dept_id	dept_name	EmpId
101	HR	1
102	IT	2

### left JOIN

b) matching values of left-table



select employee.ename, department.dept\_name from Employee  
**left join** Department **right**  
ON employee.empid = department.empid;

Ename	deptname
A	HR
B	IT
C	NULL
D	NULL

## RIGHT JOIN -> all data show from right table

name, dept.name

### X Join

b) to combine two tables

b) there should be one common column

b) if there is link between two tables

EmpId	Ename
1	A
2	B
3	C
4	D

dept_id	dept_name	EmpId
101	HR	1
102	IT	2

Ename	deptname
A	HR
B	IT

### Right JOIN

b) matching values of right-table



select employee.ename, department.dept\_name from Employee  
**right join** Department **right**  
ON employee.empid = department.empid;

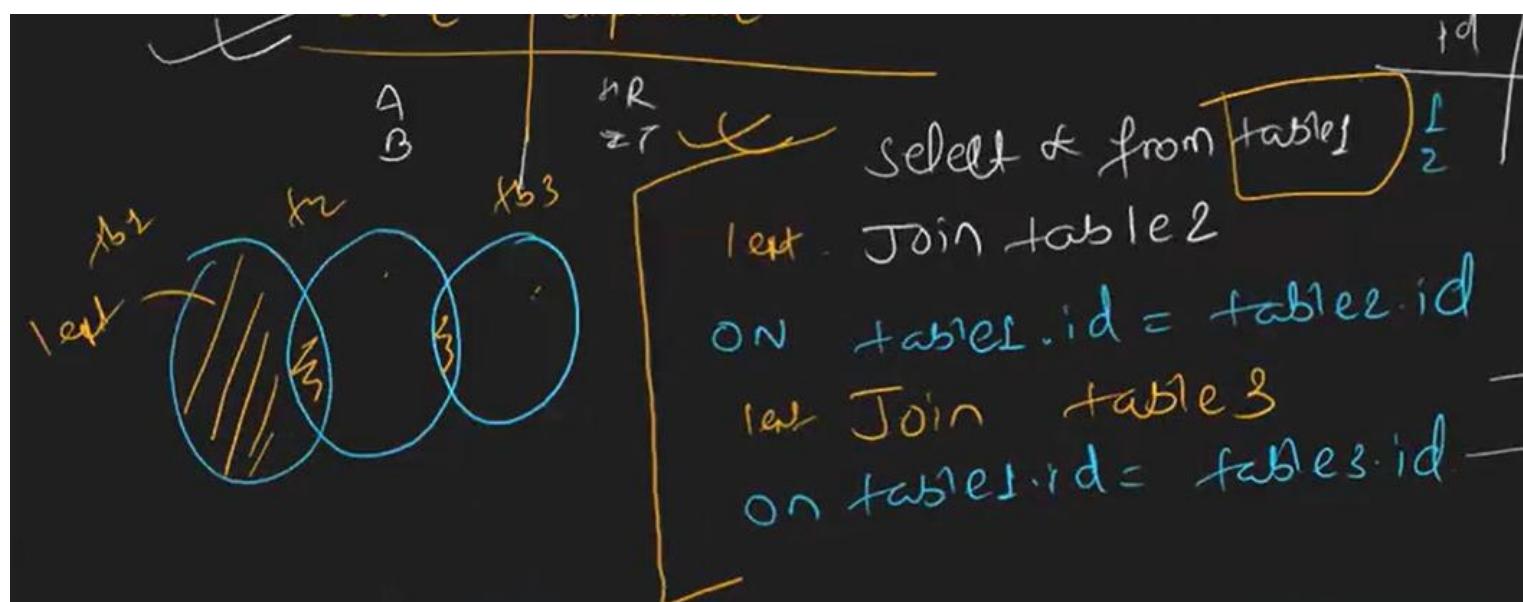
Table 1 - left
id   name
1 A
2 B

Table 2 - right
age   id
10 1
12 2
13 3

id	name	age
1	A	10
2	B	12
3	NULL	13

## More than two table



## Question on Join

### Table Structure 1: Employees and Departments

#### Table: Employees

Columns:

employee\_id (Primary Key)

name

department\_id (Foreign Key referencing Departments table)

salary

#### Table: Departments

Columns:

department\_id (Primary Key)

department\_name

Query 1: Find all employees along with their department names

Query 2: Find the number of employees in each department

Query 3: Find the average salary of employees in each department

Query 4: Find the department with the highest number of employees

Query 5: Find all employees who have not been assigned to any department

**Date → 22/02/2024**

**Stored Procedure → its use for store your query and easy to recall**

Stored procedure

Syntax:

- ① Reuse
- ② Performance
- ③ Maintenance

create procedure " selectdata"  
begin  
↳ select \* from demo where id=3  
end



```
1 • CREATE DEFINER='root'@'localhost' PROCEDURE `createtable`()
2 BEGIN
3   CREATE TABLE demo3 (
4     EmployeeID INT PRIMARY KEY,
5     FirstName VARCHAR(50),
6     LastName VARCHAR(50),
7     Department VARCHAR(50),
8     address varchar(60)
9
10 );
11 END
```

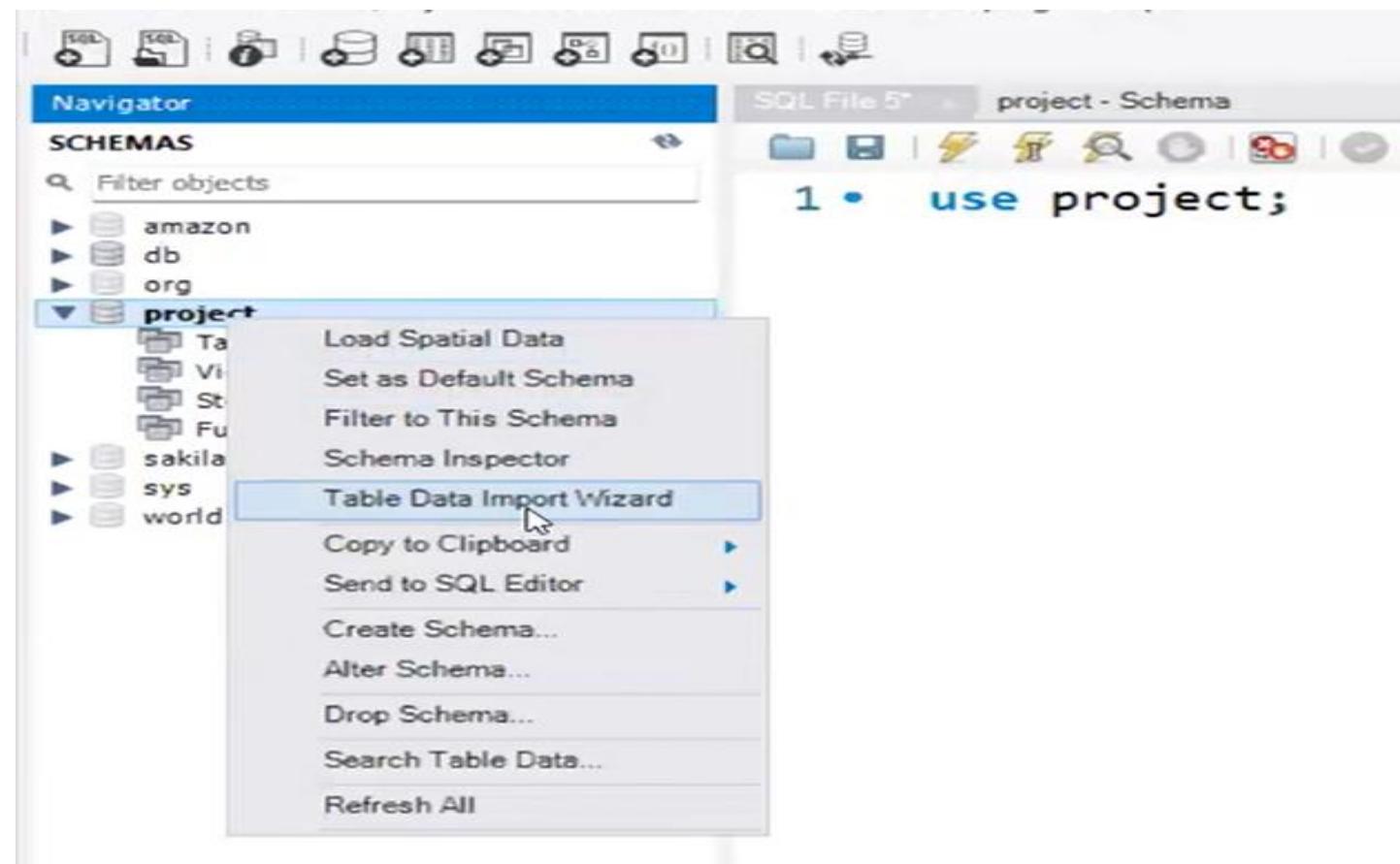
**View Table** → In SQL, a view is a virtual table based on the result set of a SELECT statement. Unlike a base table, which stores data physically, a view does not store data itself. Instead, it is a saved SQL query that can be treated as a table within the database. When you query a view, the underlying SELECT statement is executed, and the result is returned as if you were querying a regular table.

- It is very important if we have 1 lakhs + data and I want to work on certain data then it's very helpful
- **For data security**

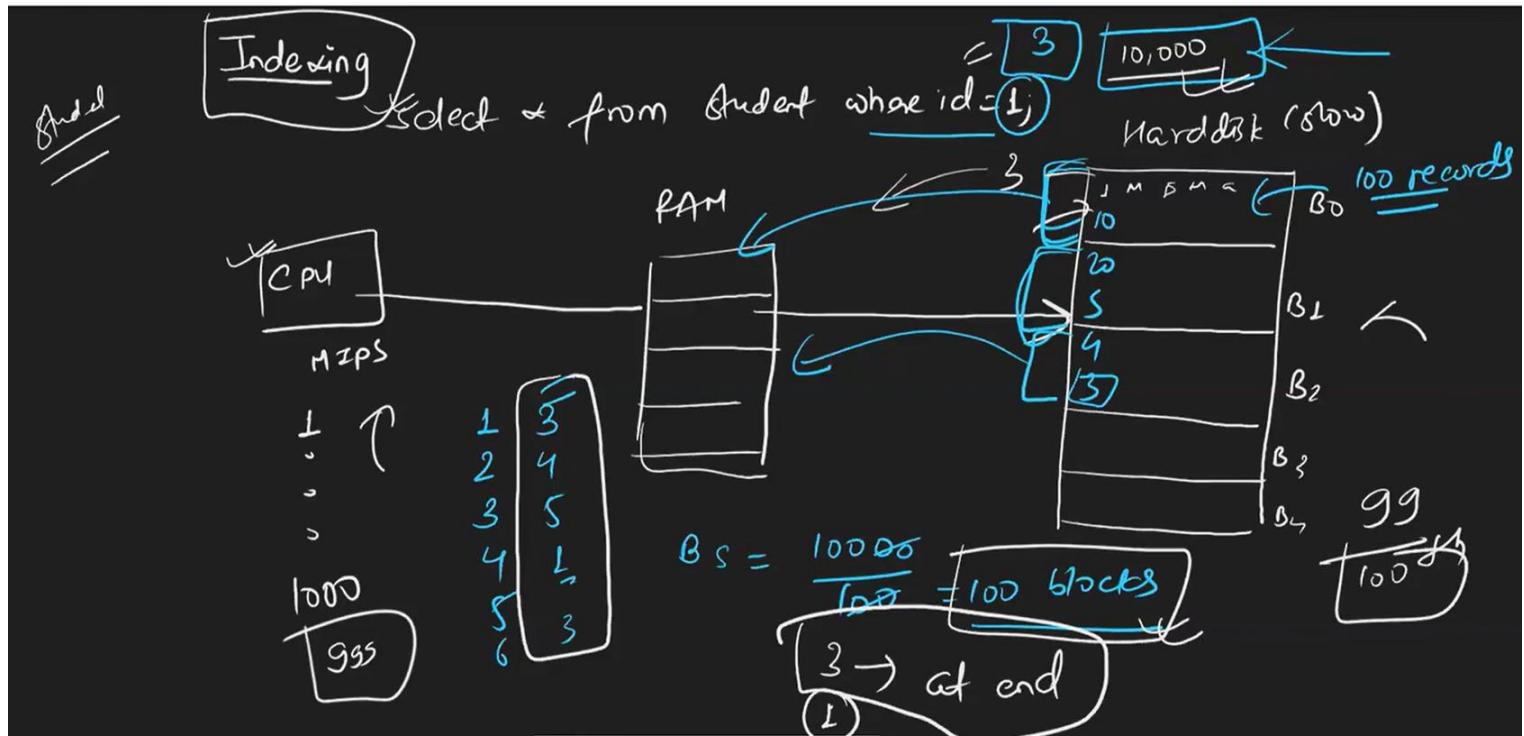
```
1  create view emp as
2  select empid, name, location
3  from employee
4  where empid <5;
5 • select * from employee;
6
7 • select * from emp where empid=1;
8 • update emp set name = "aisha" where empid =1;
```

## IMPORT CSV FILE IN SQL

1. Create Database
2. Right click on database name
3. Click on Table data import wizard
4. Select you file

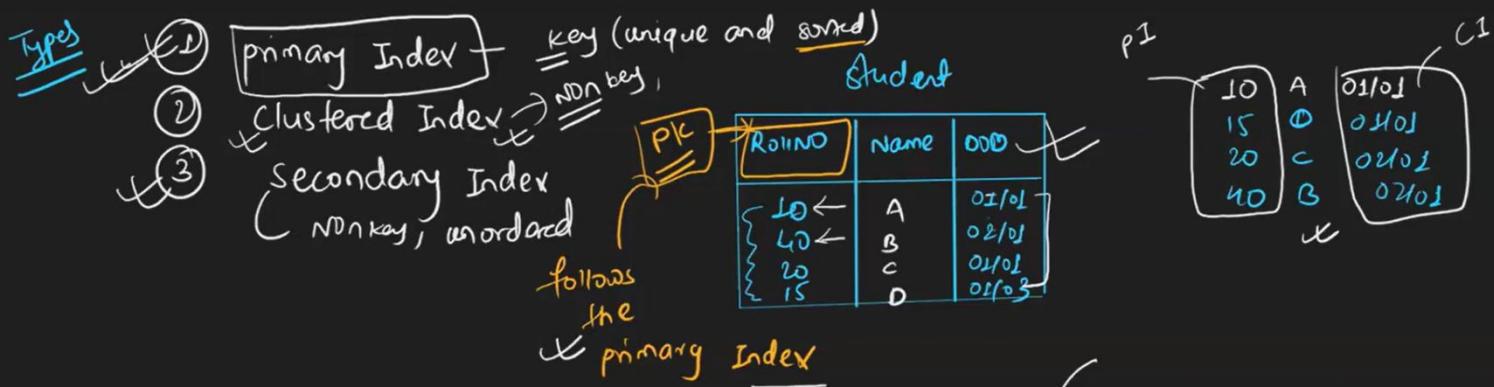


**INDEXING** → In SQL, indexing is the process of creating data structures to improve the speed of data retrieval operations on a database table. An index is a special data structure associated with a table that allows for faster retrieval of rows based on the values of certain columns.



## Three types of INDEX

1. Primary Index
2. Cluster index
3. Secondary Index



↳ select \* from Student where ROLLNO = 15

select \* from Student where

select \* from Student where

dob = '01/01'

name = 'A'

CREATE TABLE Employee1 (

EmployeeID INT primary key,

FirstName VARCHAR(50),

LastName VARCHAR(50),

Department VARCHAR(50),

Salary DECIMAL(10, 2)

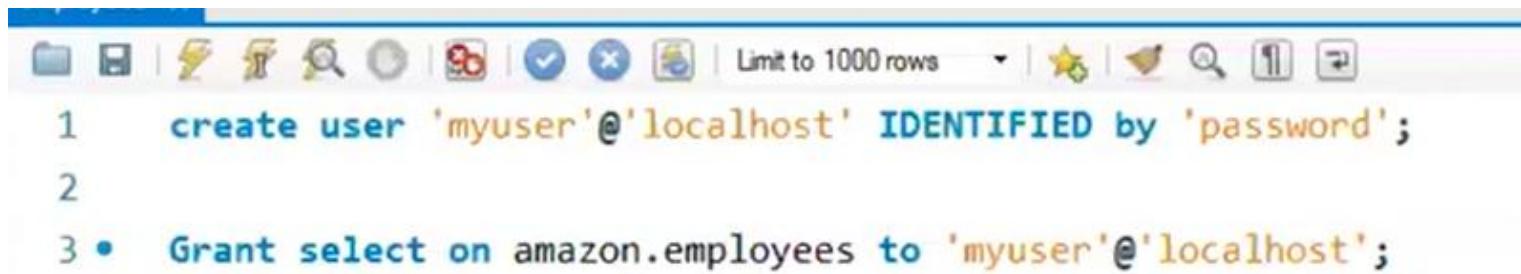
) ;

- create index emp on Employee1 (FirstName, LastName);

Date → 23/02/2024

Create a local host->

Create a table for share to anyone and allow to other person to excess this data, (using operation CRUD)



```
1 create user 'myuser'@'localhost' IDENTIFIED by 'password';
2
3 • Grant select on amazon.employees to 'myuser'@'localhost';
```

Revoke→ Retake you permission (remove your permission on user@localhost)



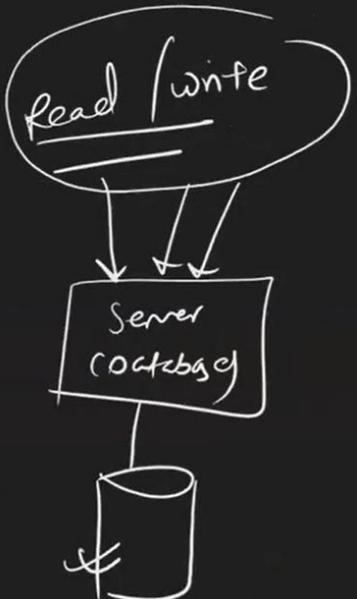
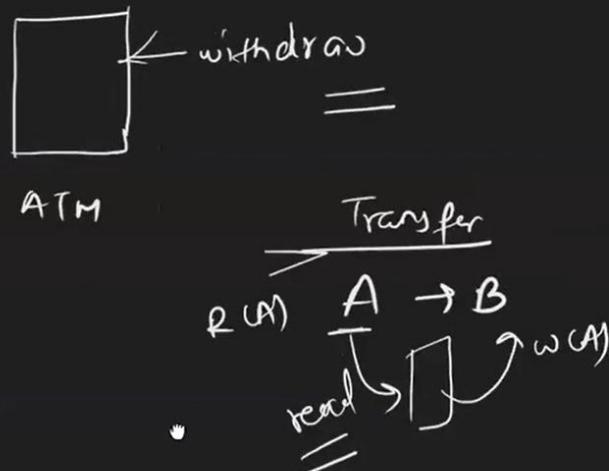
```
1 create user 'myuser'@'localhost' IDENTIFIED by '';
2
3 • Grant select on amazon.employees to 'myuser'@'localhost';
4 • revoke select on amazon.employees from 'myuser'@'localhost';
5
```

# Transactions

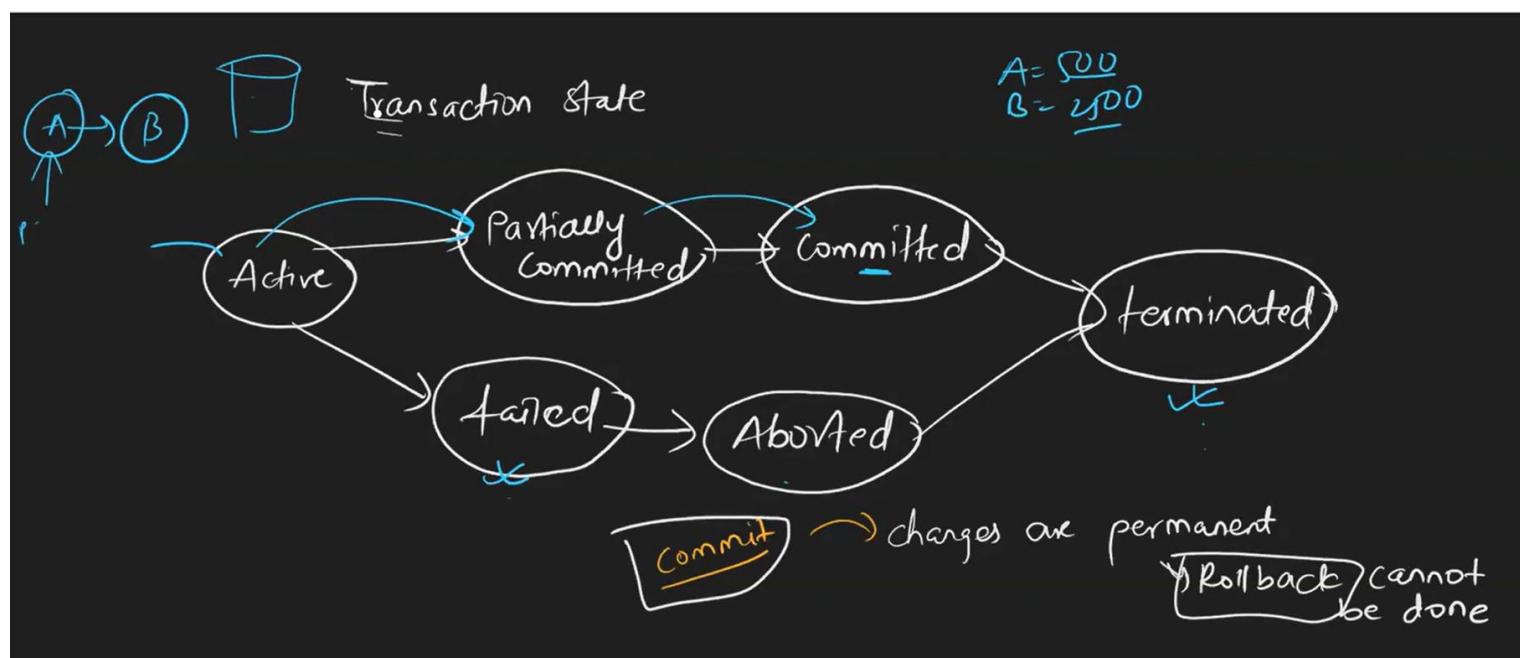
## Transactions

- ↳ generally represents change in database
- ↳

- Transaction
- ① Insert card
  - ② choose language
  - ③ select
  - ④ amount



## State of Transactions



**Set Autocommit=0;**

After run this query your changes will save temporary base , you can roll it back

```
1 select * from employees;
2 • update employees set first_name = "aisha" where employee_id = 1;
3 • update employees set first_name = "Rekha" where employee_id = 2; I
4 • rollback;
```

Result Grid							
employee_id	first_name	last_name	email	hire_date	department_id	salary	
1	aisha	Doe	john.doe@example.com	2023-01-15	1	50000.00	
2	Rekha	Smith	jane.smith@example.com	2022-05-20	2	60000.00	
3	Michael	Johnson	michael.johnson@example.com	2023-09-10	1	55000.00	
4	Emily	Williams	emily.williams@example.com	2024-02-05	2	62000.00	
HULL	HULL	HULL	HULL	HULL	HULL	HULL	

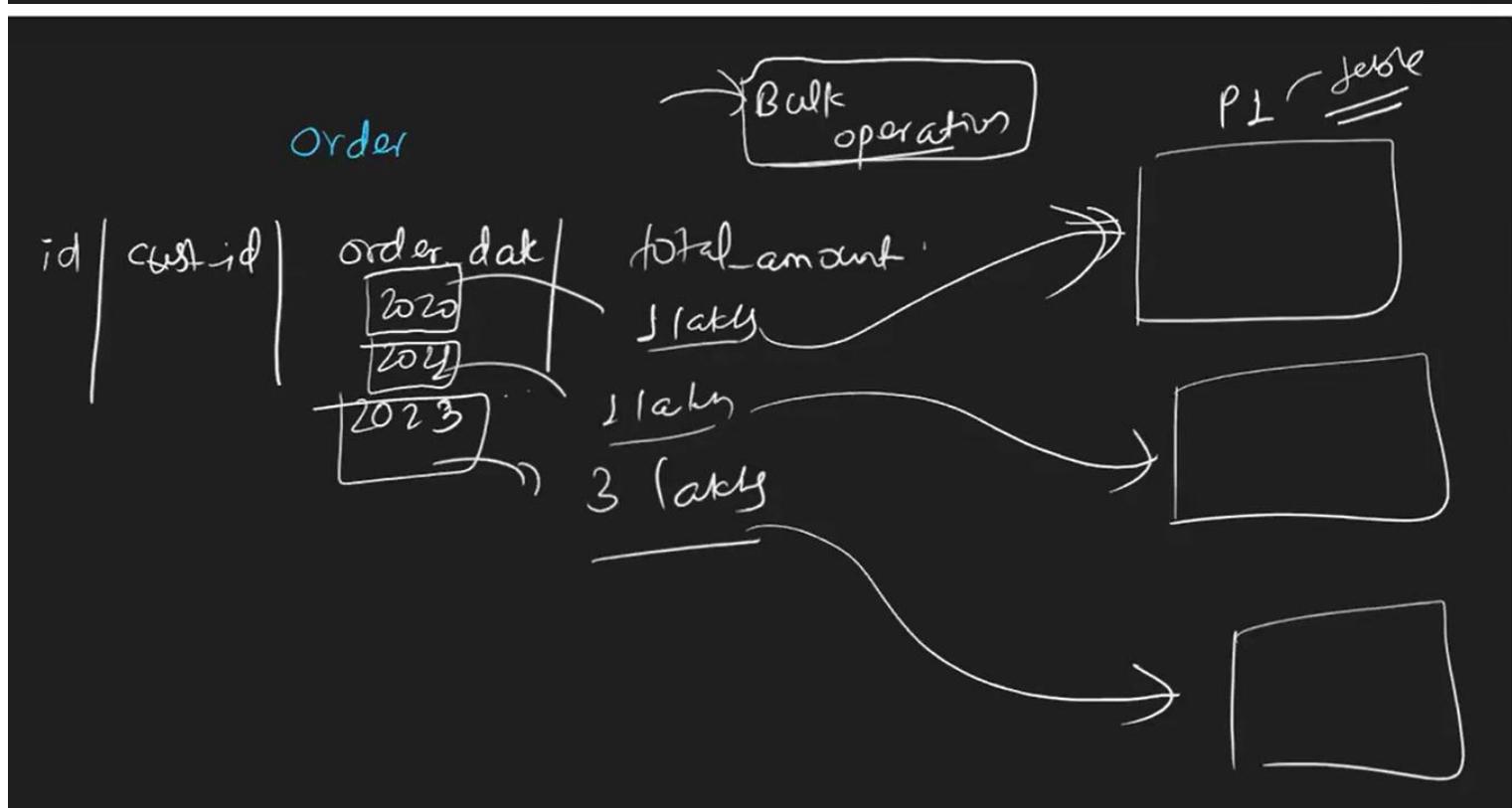
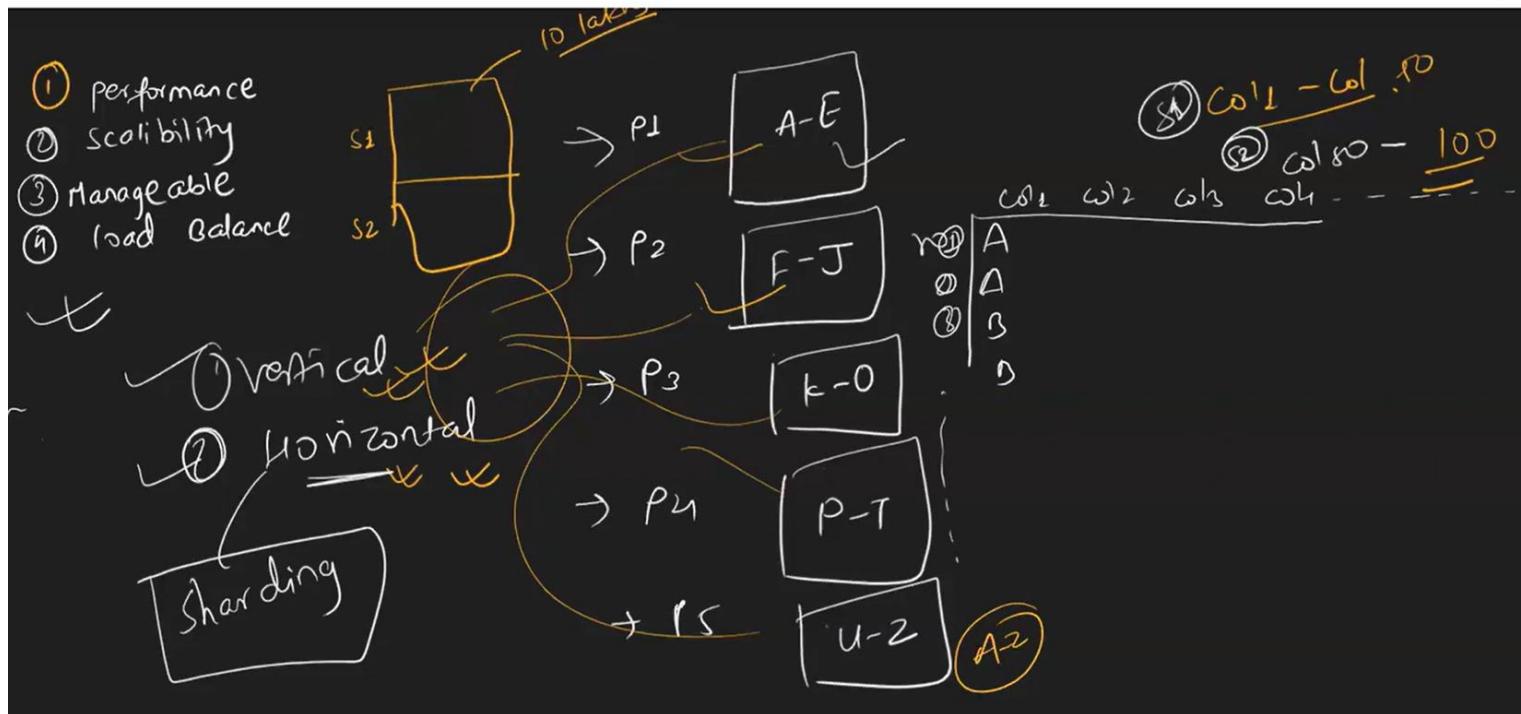
**Rollback → if you apply this query then your changes will be rolled back.**

**Commit → it is for save your changes, after applying this query you can undo the changes.**

```
1 select * from employees;
2
3 • update employees set first_name = "aisha" where employee_id = 1;
4 • commit;
5 • update employees set first_name = "Rekha" where employee_id = 2;
6 • rollback;
7
8 • savepoint my_savepoint1;
9
10 • update employees set first_name = "Rekha" where employee_id = 2;
11 • savepoint My_savepoint2;
12 • delete from employees where employee_id = 4;
13
14 • rollback to My_savepoint2;
```

# Partitioning (dividing)

Divide the table of small part



# Project→

<https://docs.google.com/document/d/1aQisqPcS0QYnBAOZ5xsk9tsVQ27N9oCqUzUXsoMiQhQ/edit?pli=1#heading=h.qdk56qu7xwsy>

## Ask Academic Doubt

BACK

Subject  
submission of master class DBMS project

Priority

Query Type



Upload Attachment

SUBMIT QUERY

Last date- 8 March 24