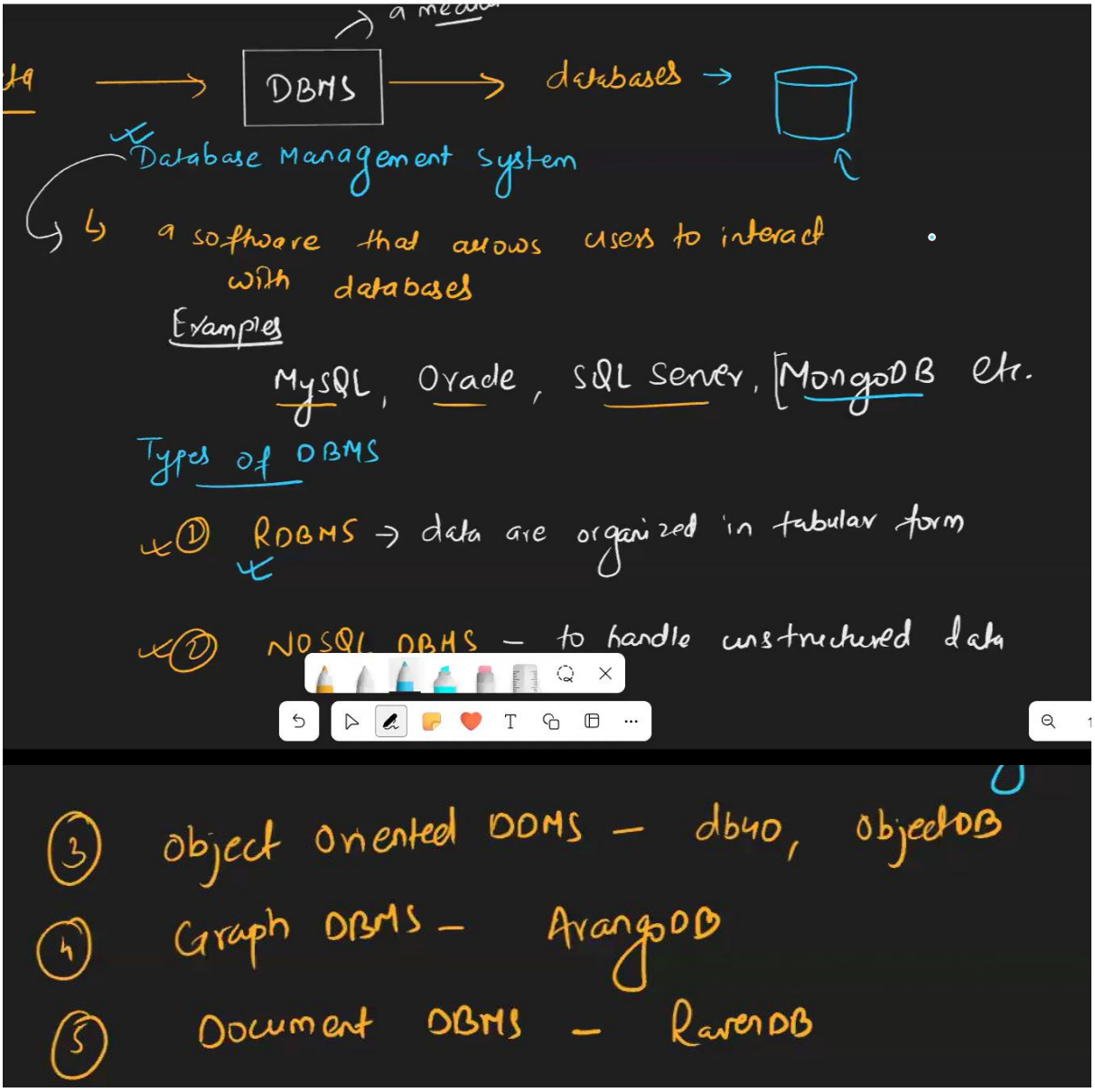
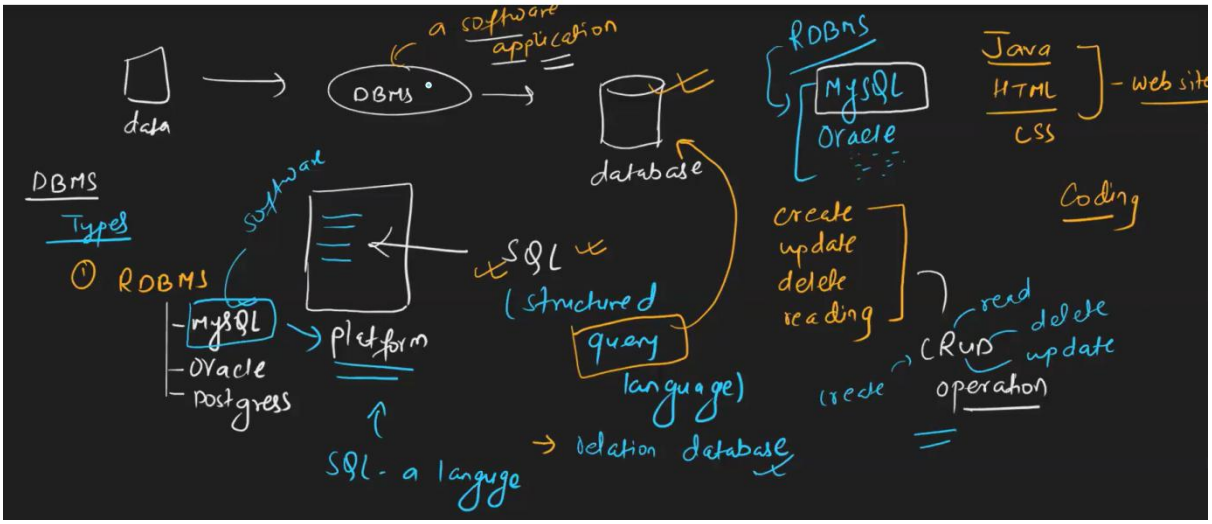
SQL & DBMS

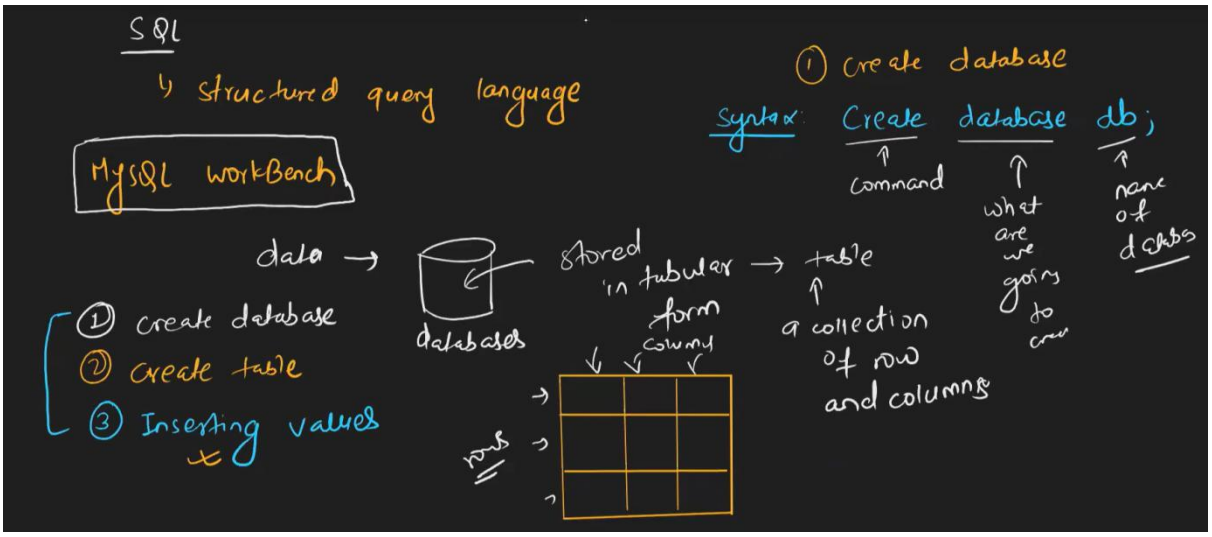
DATE- -> 13/02/2024

[DBMS Notes - Google Docs]

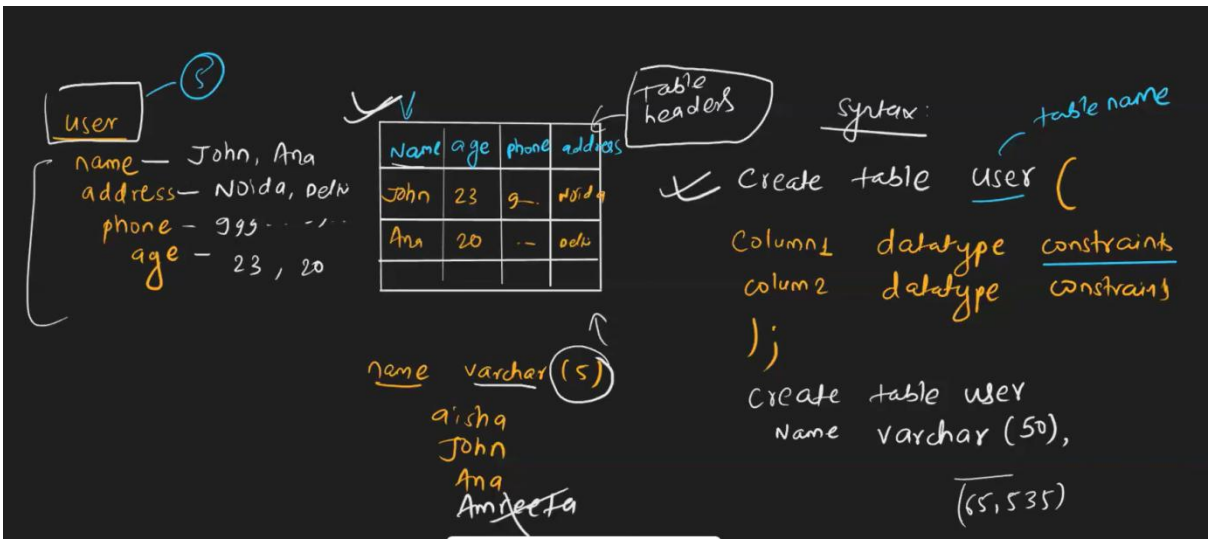


TIME – 43

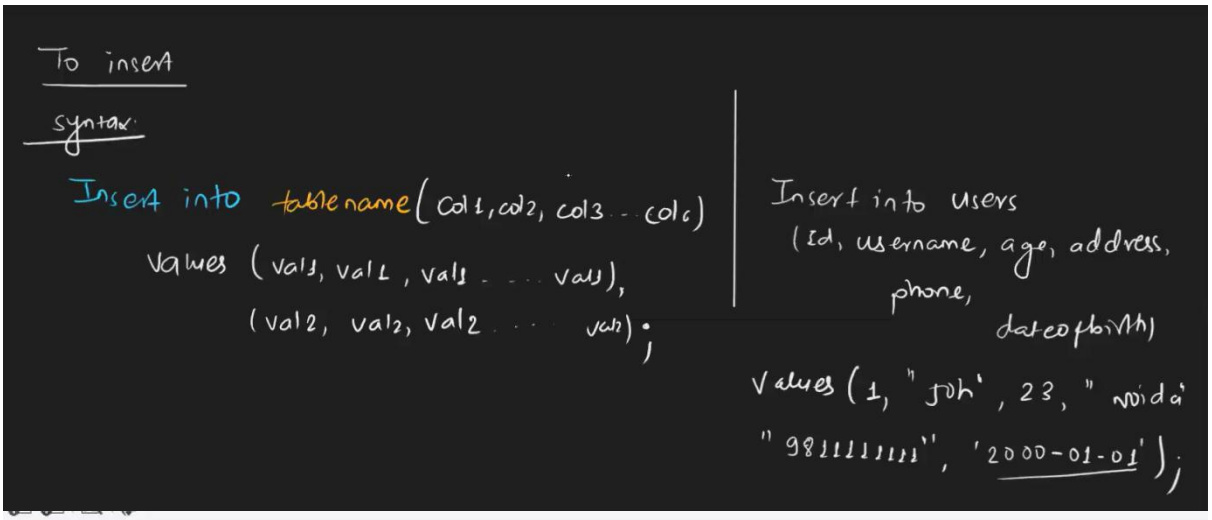


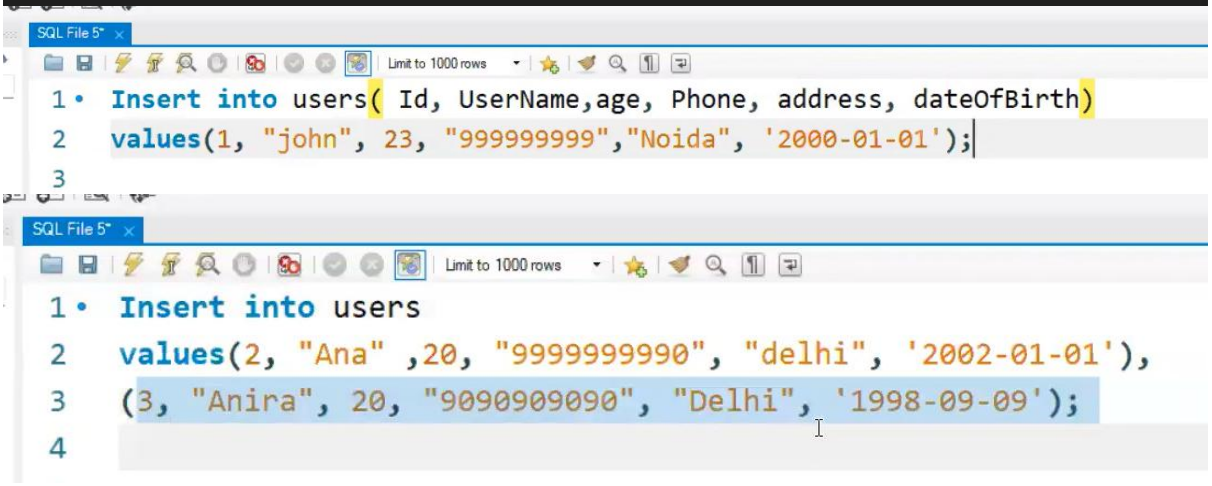


Schemas – Structure



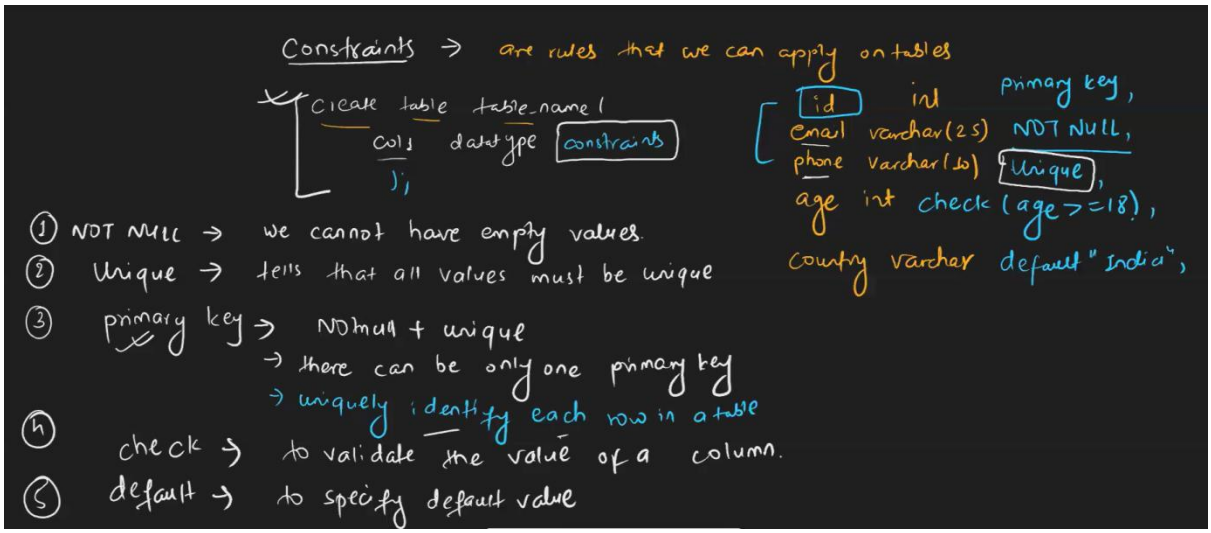
Time – 1.41



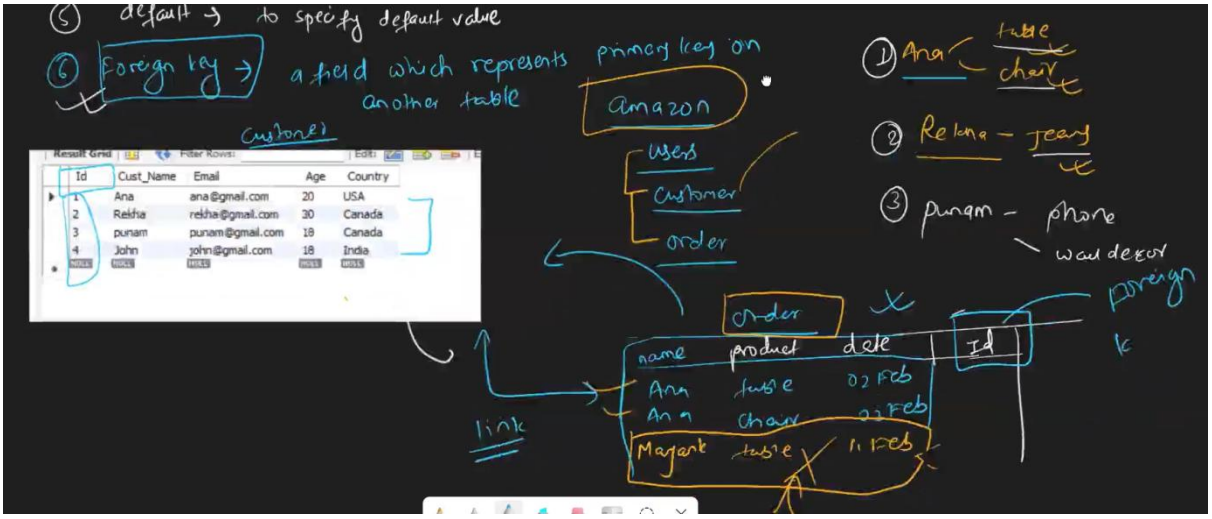


DATE → 14/02/2024

CONSTRAINTS → it is use for not exist duplicate values



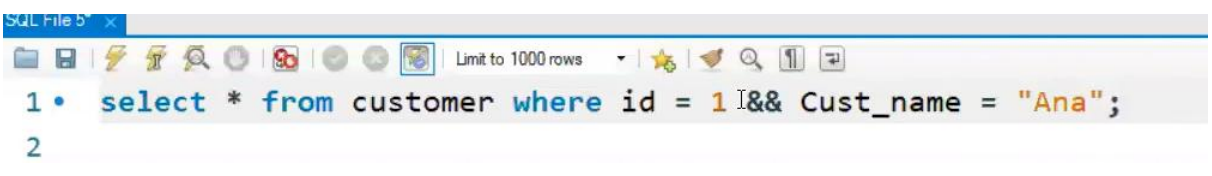
```sql
create table customer (
Id int Primary Key,
Cust_Name varchar(50) Not Null,
Email varchar(60) unique,
Age int check(age>=18),
Country varchar(50) default "India"
);
```



Use of FRONER KEY

```sql
create table orders (
OrderId int auto_increment unique,
Product varchar(50),
Id Int,
Foreign Key(Id) references customer(Id)
);
Insert into orders
values (101, "Table", 1),
(102, "Chair", 1);
select * from orders
```

AND → &&



NOT EQUAL TO → !=, <>

```
Limit to 1000 rows
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

```
SQL File 5*
Limit to 1000 rows
1 • select * from customer where id IN (1, 4, 5); -- specify multiple vlaues
2 • select * from customer where id between 2 and 5; -- specify range of values
```

DATE → 15/02/2024

ACID PROPERTY →

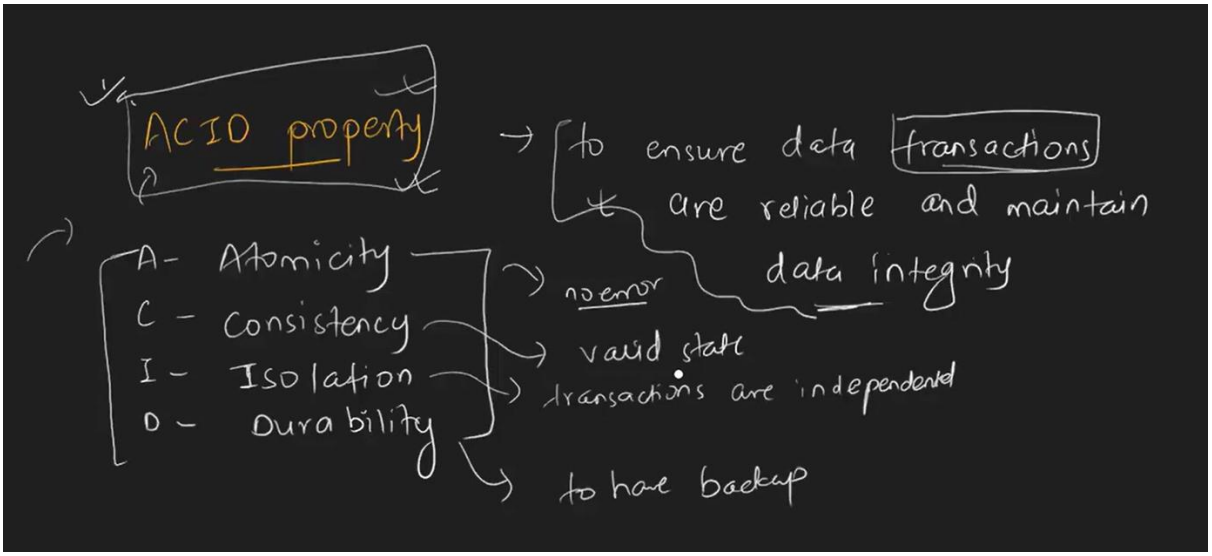
In SQL, AICO typically stands for "Atomic, Consistent, Isolated, and Durable," which are the four properties that define the behavior of database transactions. These properties are commonly referred to as ACID properties. Here's what each property means:

A→ **Atomicity**

C → Consistency

I→ Isolation

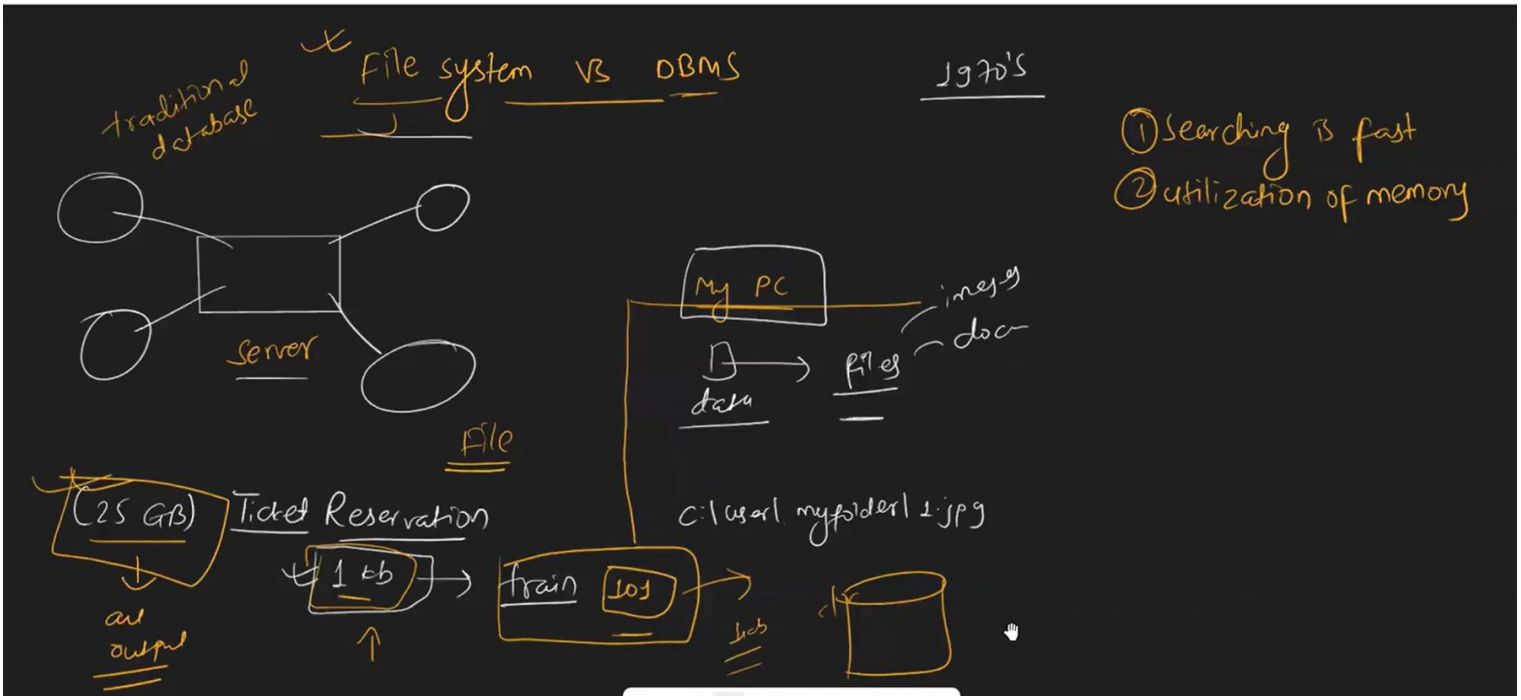
D → Durability



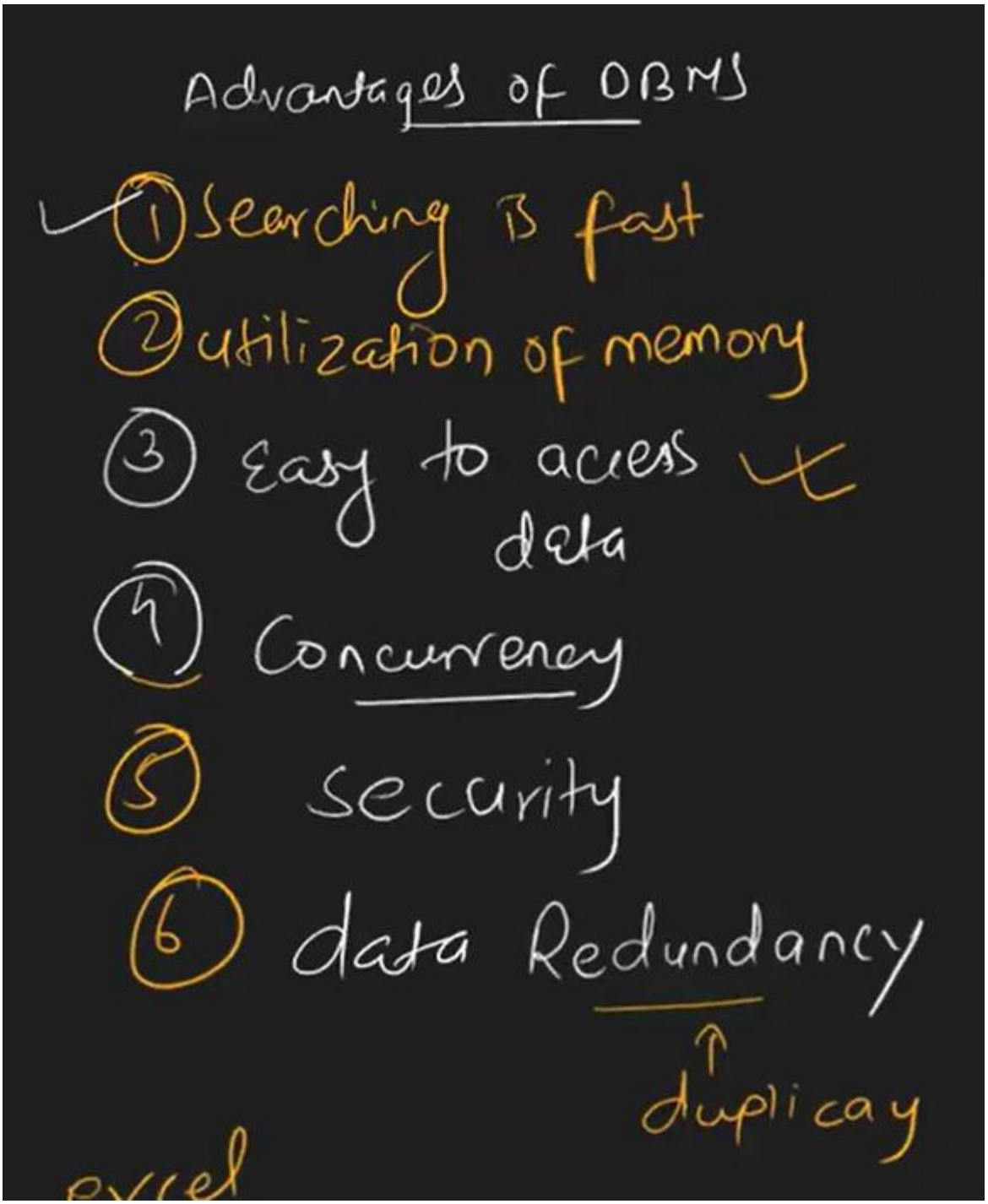
File system Vs DBMS

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

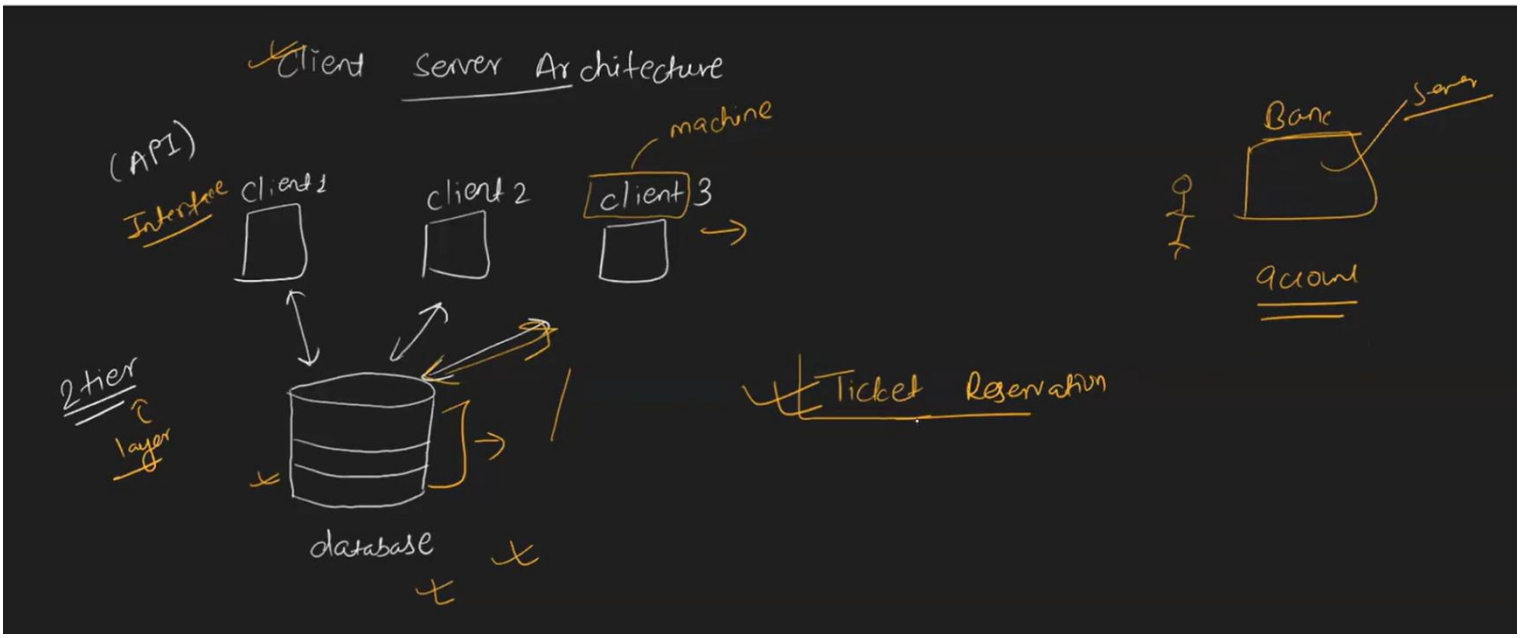


Advantage of DBMS



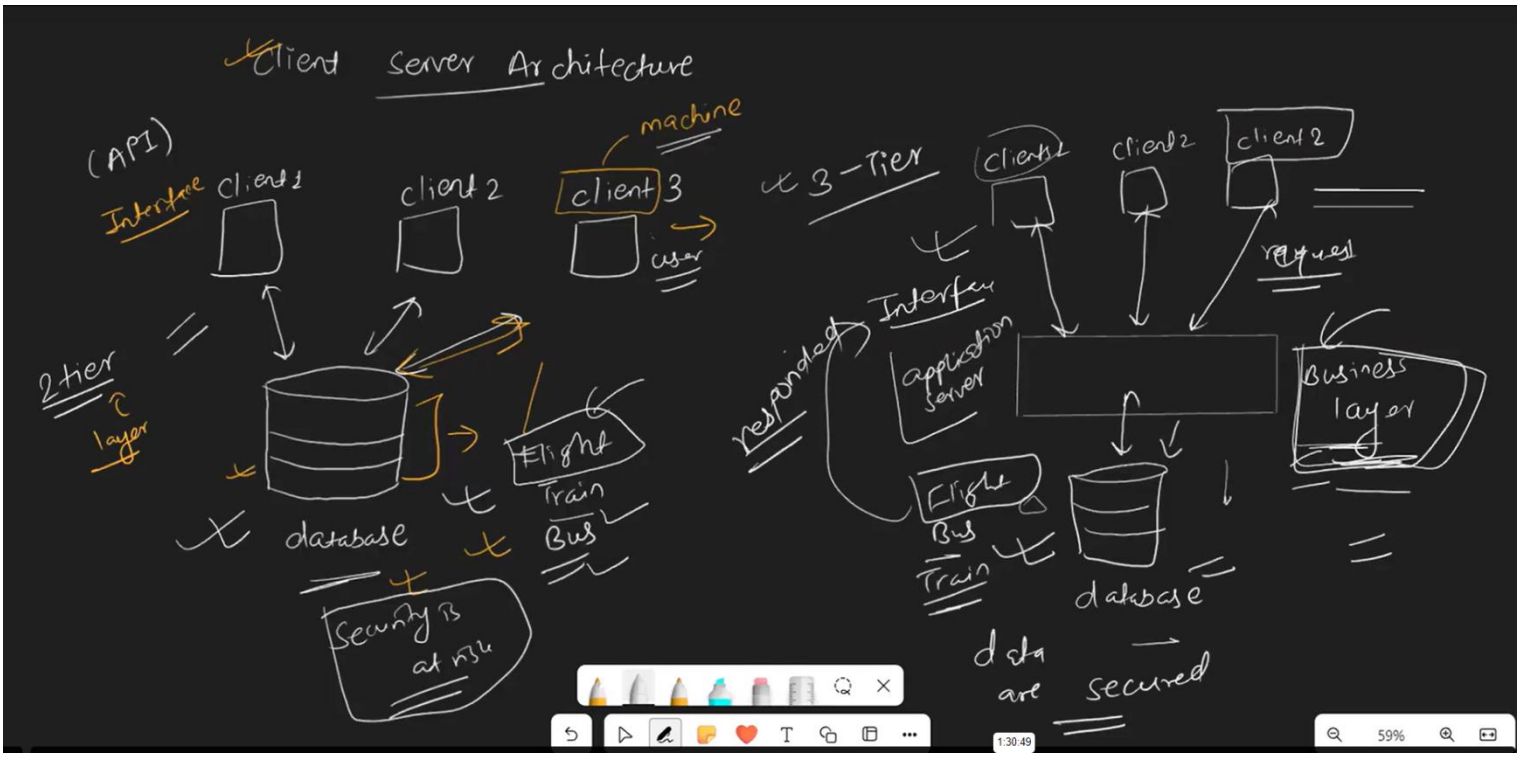
Time→ 1.07

Clint Server Architecture →

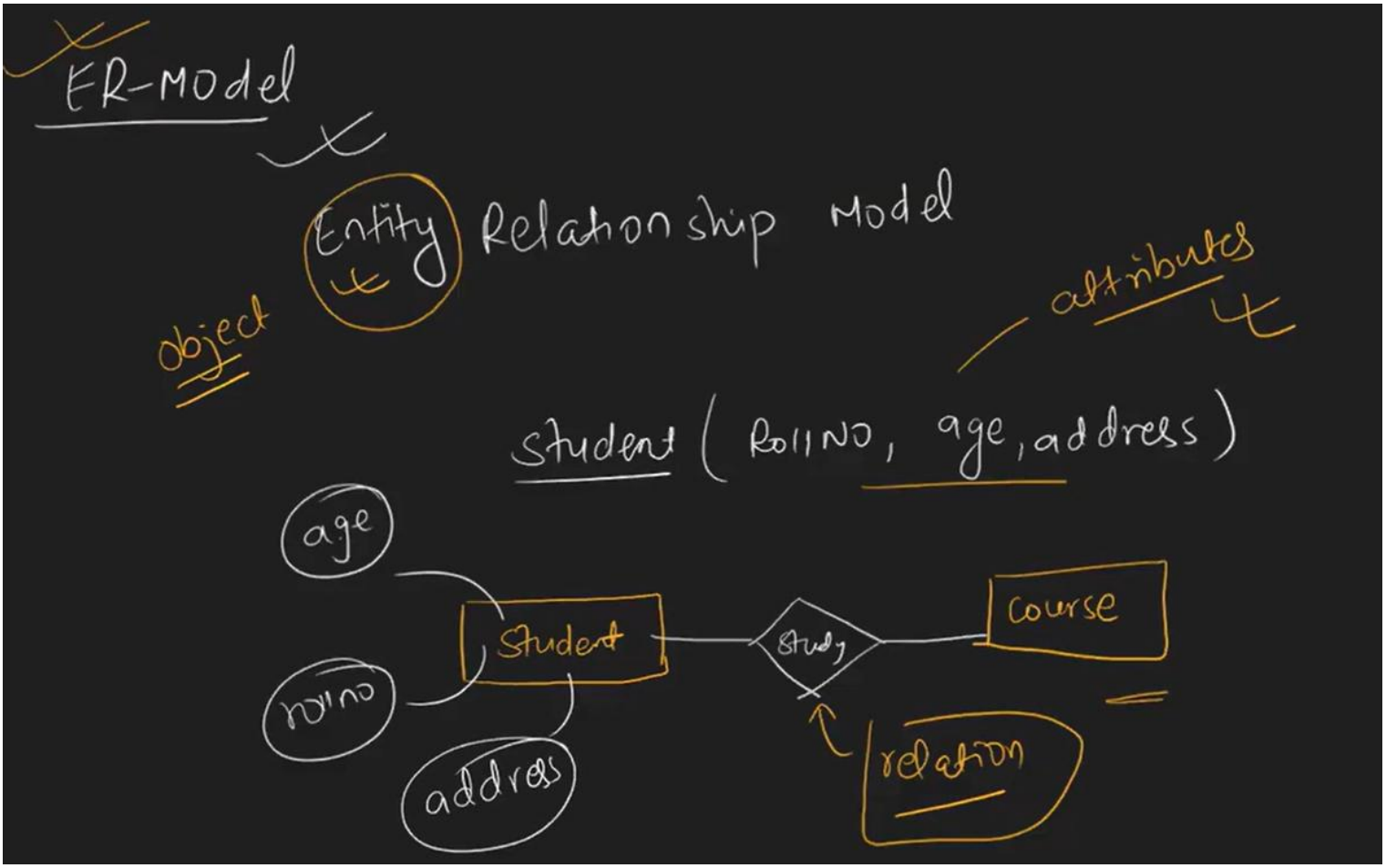


2-Tier And 3- Tier

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



ER-Model-- > Entity Relationship model



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?

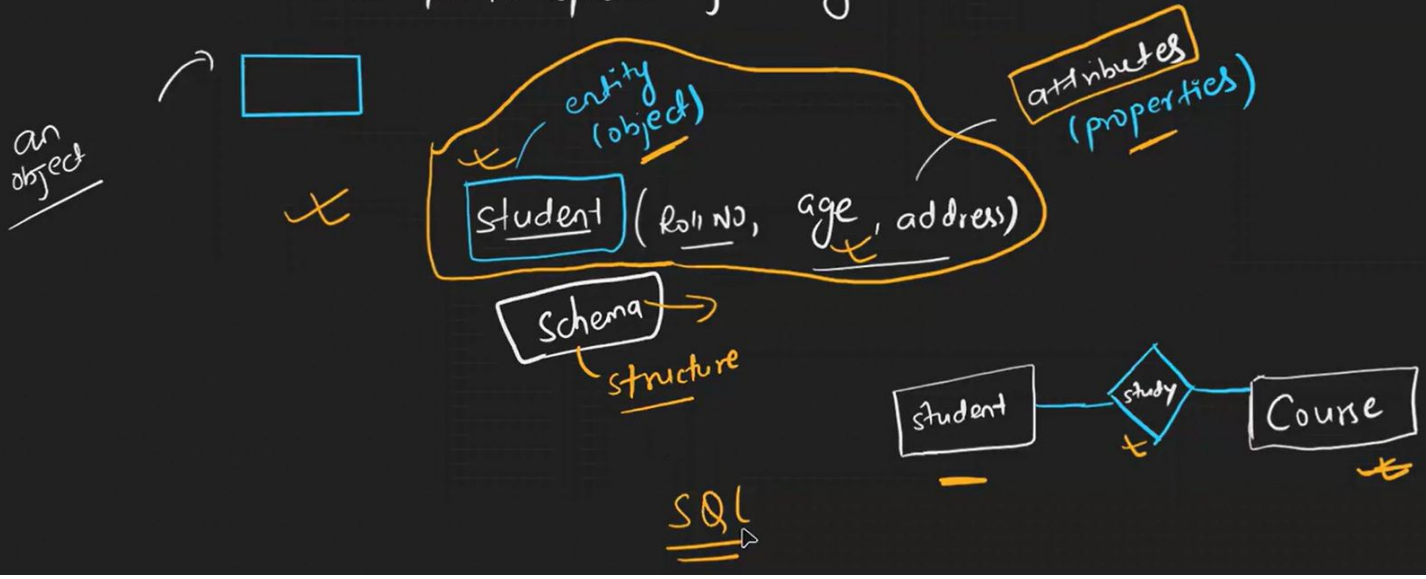
6. Discuss the advantages and disadvantages of using a relational database management system (RDBMS) compared to a NoSQL database.

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

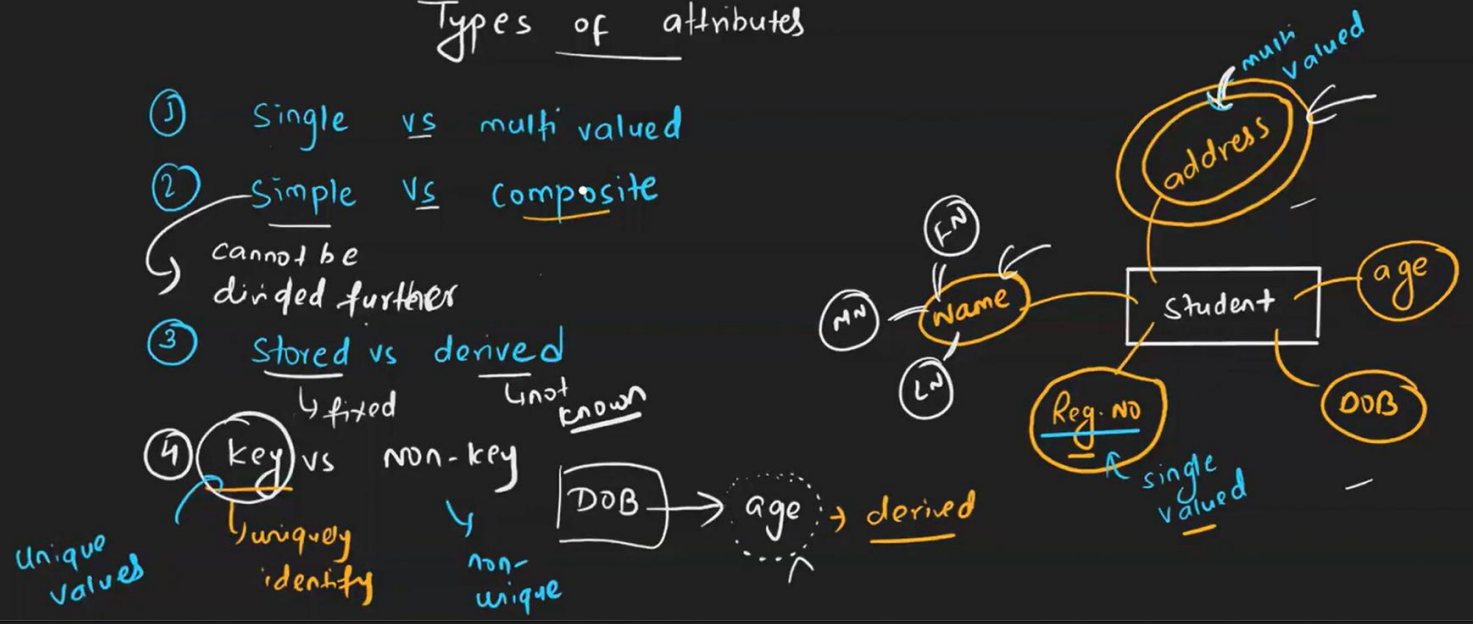
Entity Relationship Model

↳ a technique for representing the logical structure of databases



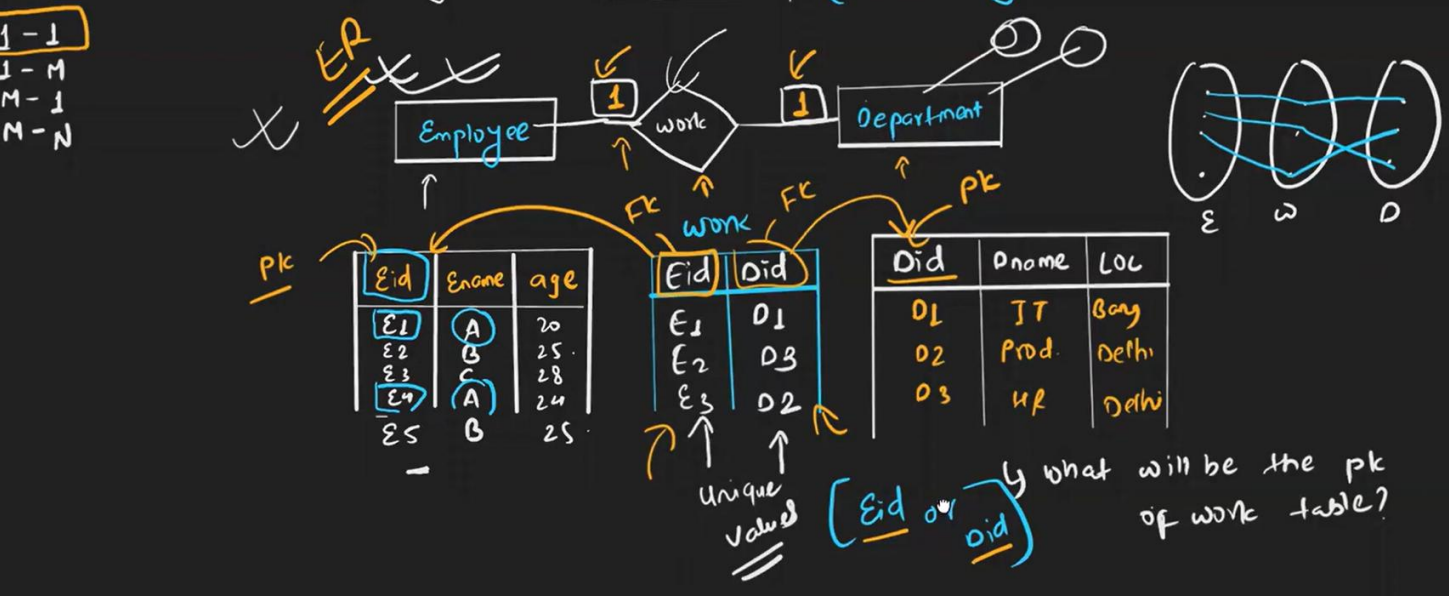
Types of attributes

1. Single vs multi valued
2. Simple vs composite
   - cannot be divided further
3. Stored vs derived
   - ↳ fixed
   - ↳ not known
4. key vs non-key
   - ↳ Unique values
   - ↳ uniquely identify
   - ↳ non-unique

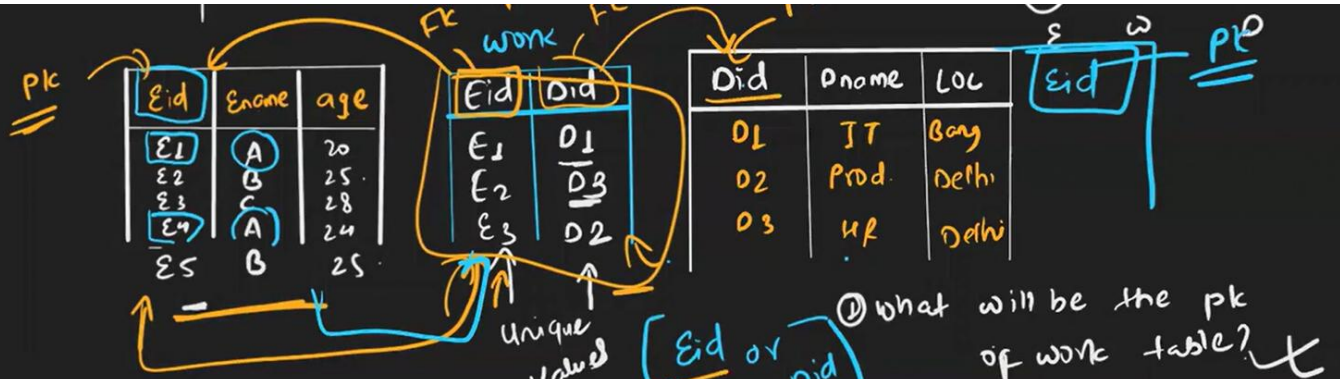


Types of Relationship (cardinality)

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



↳ 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	—

fk

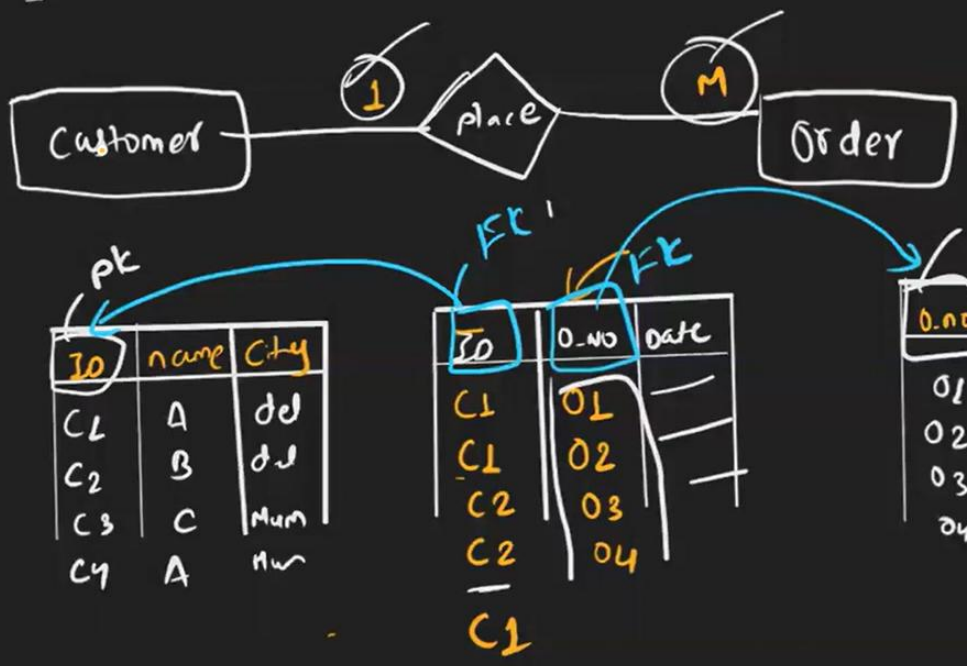
← NULL

② What will be the pk of work table?

② Reduction of table

(2)

One to many



Teacher

ID	name	age
T1	A	
T2	B	

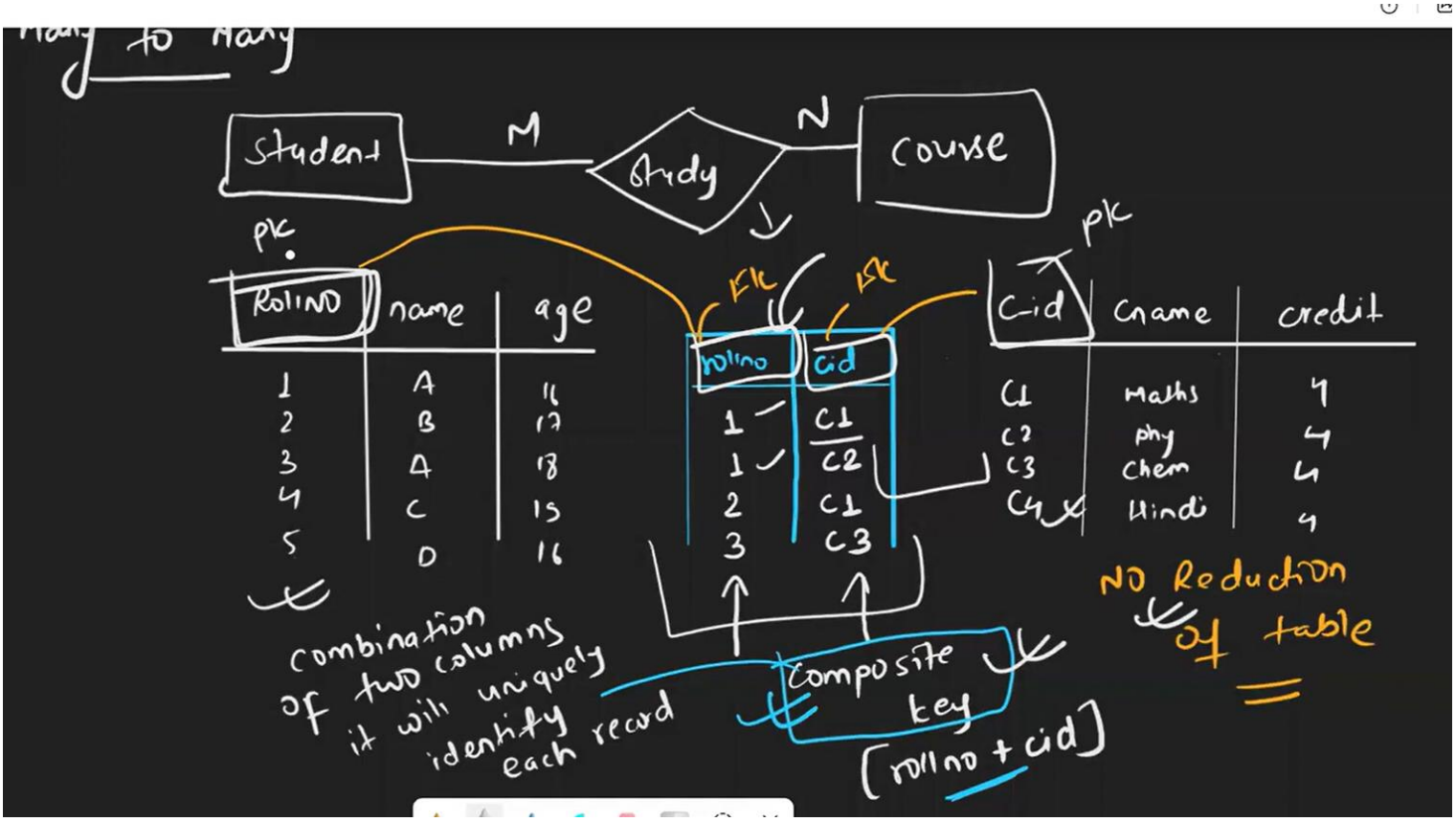
Teaching

ID	SID
T1	S1
T1	S2

Student

SID	Sname
S1	X
S2	Y

MANY TO MANY RELATIONSHIP

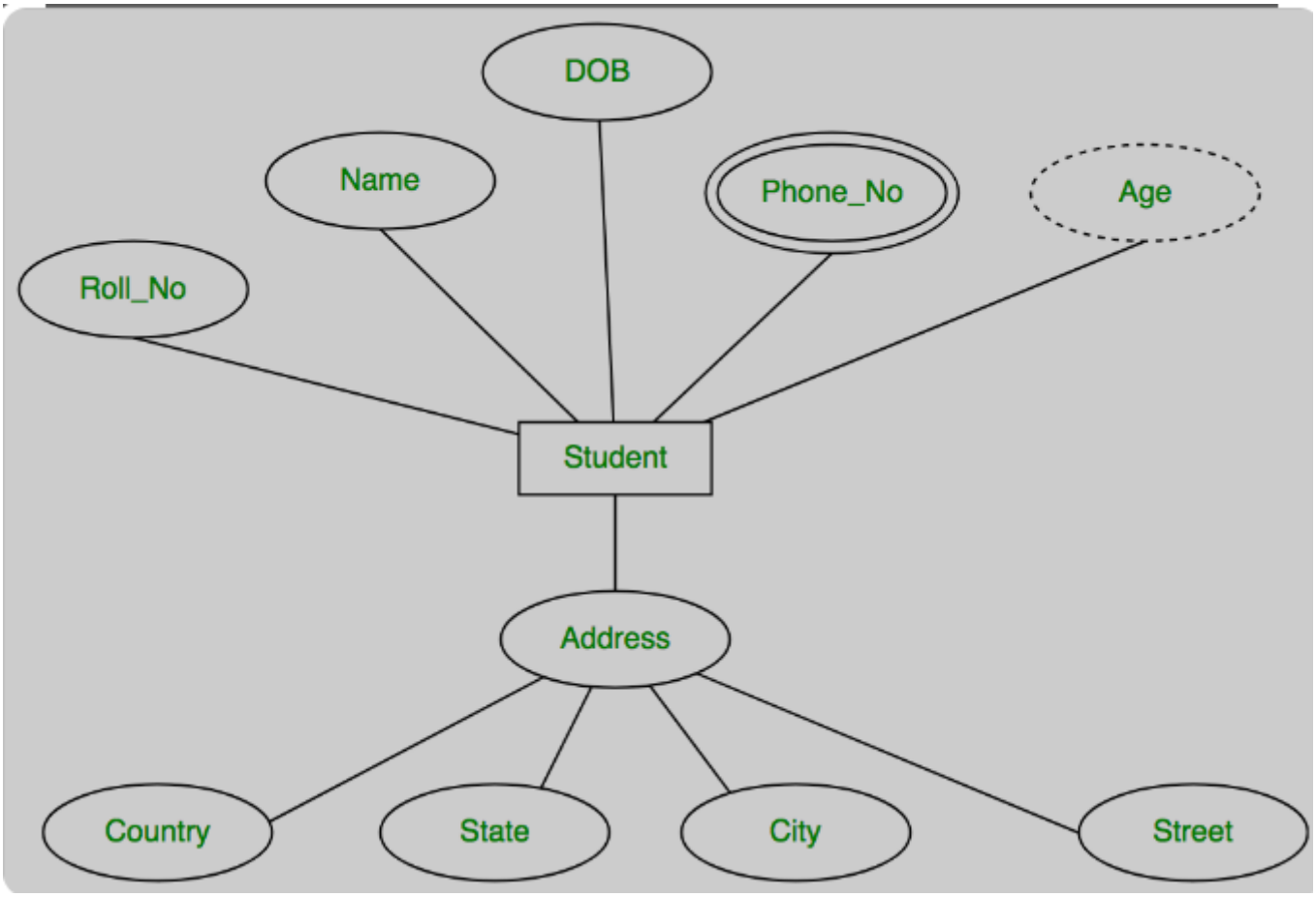


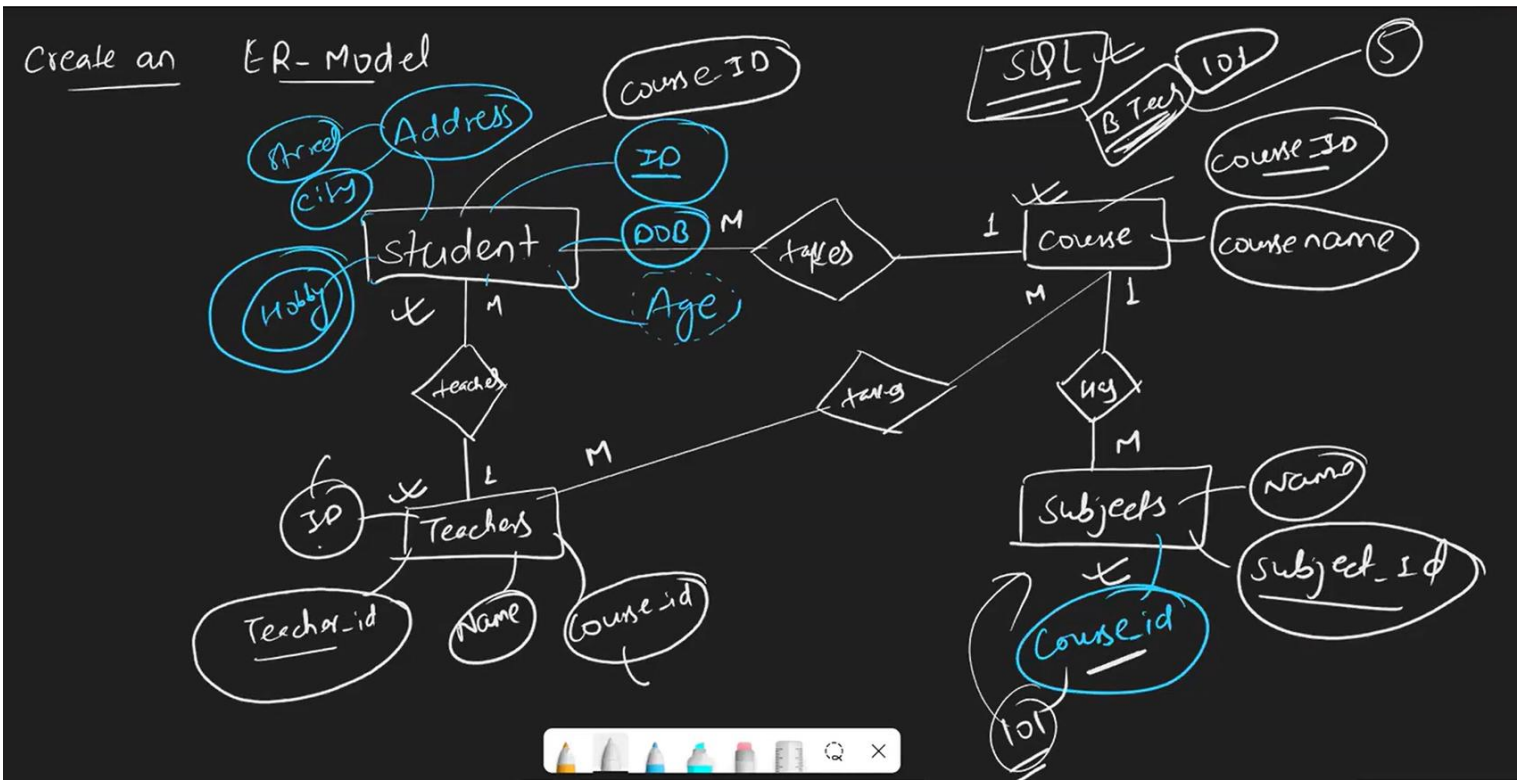
TODAY LAST TOPIC

Date- 19/02/2024

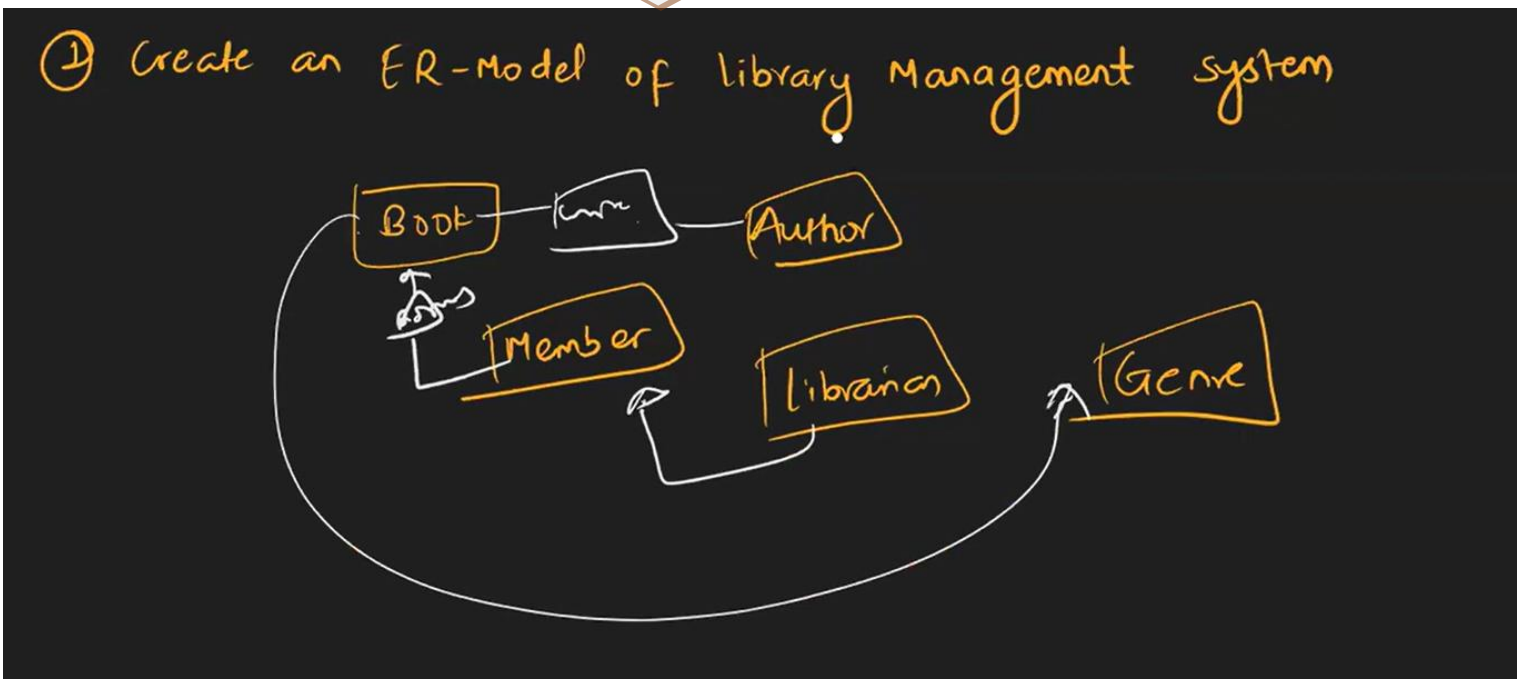
ER Diagram in DBMS. An Entity–relationship model (ER model) describes the structure of a database with the help of a diagram, which is known as Entity Relationship Diagram (ER Diagram). An ER model is a design or blueprint of a database that can later be implemented as a database.

Example of ER model based on STUDENT TABLE


DOB
Name
Phone_No
Age
Roll_No
Student
Address
Country
State
City
Street


Create an ER-Model
Course_ID
Address
Street
City
ID
Student
DOB
Hobby
Age
M
N
Teaches
Takes
1
Course
Course_ID
Course name
M
1
Has
M
Subjects
Name
Subject_Id
Course_id
101
SQL
B Tech
101
5
ID
Teachers
Teacher_id
Name
Course_id

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

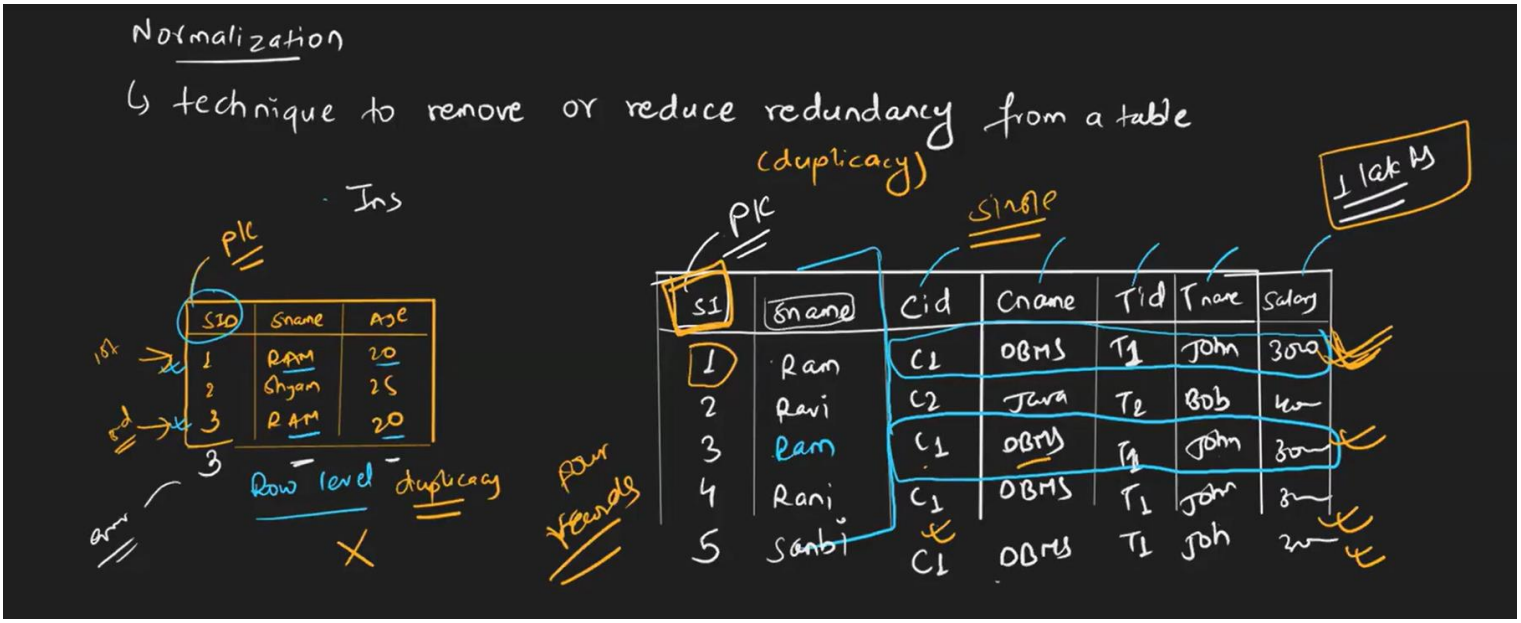


NORMALIZATION → it's methods to remove the duplicate values in table,

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

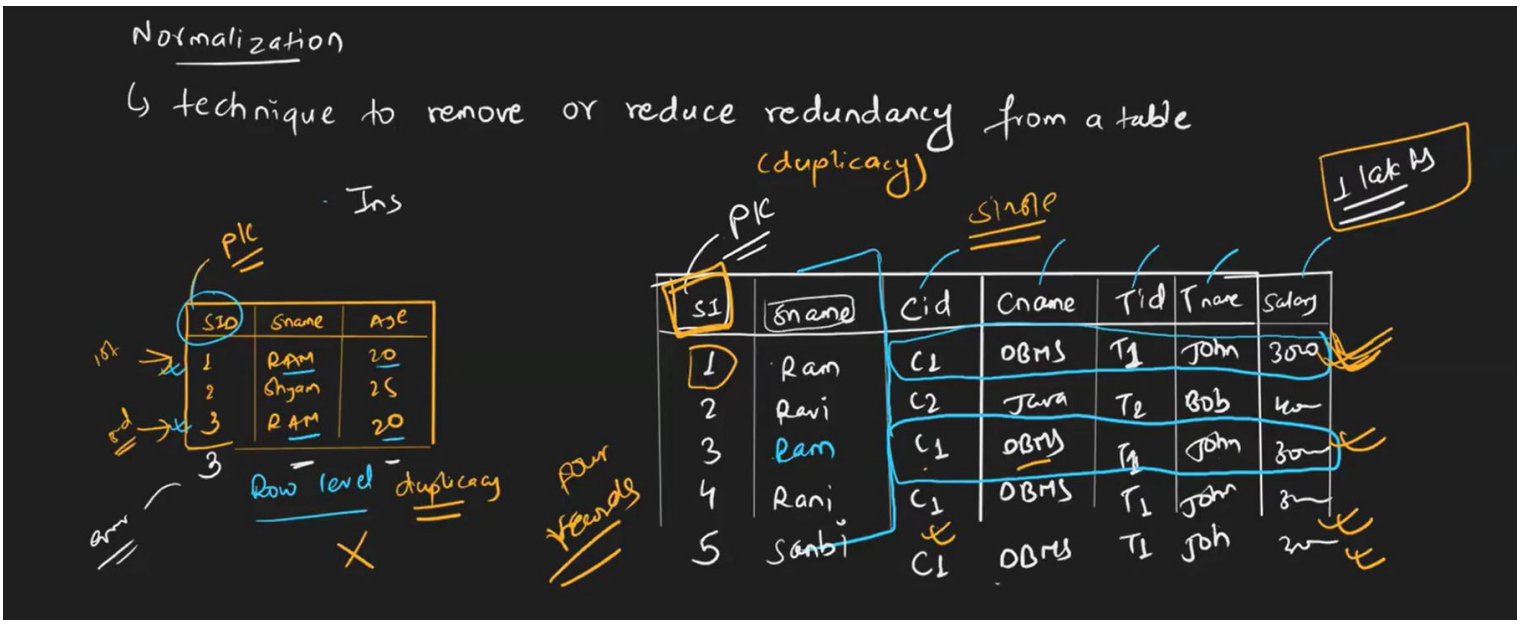


So Problem is

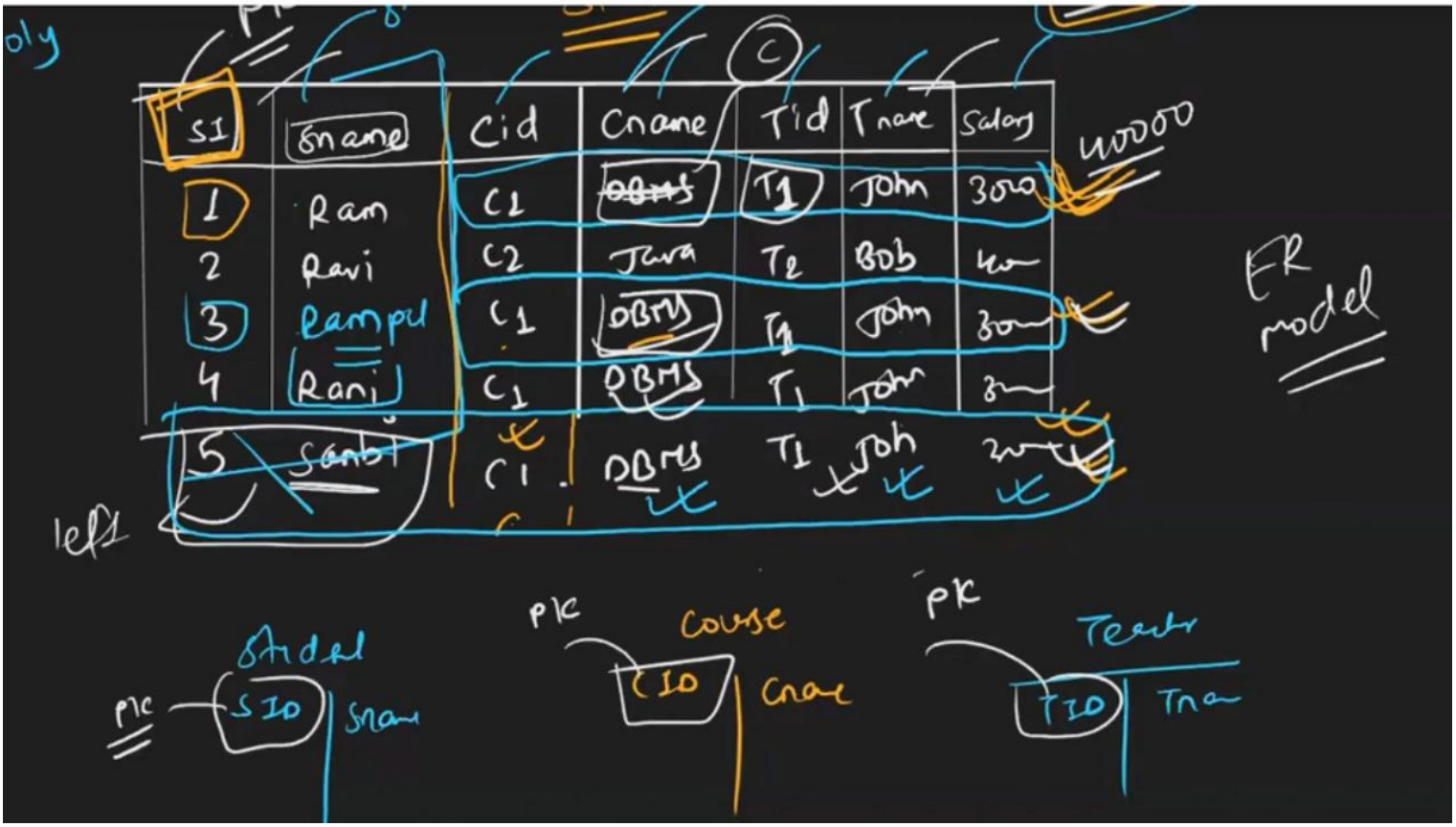
Deleting sum data

Updating data

Inserting data



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 multivalied 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.

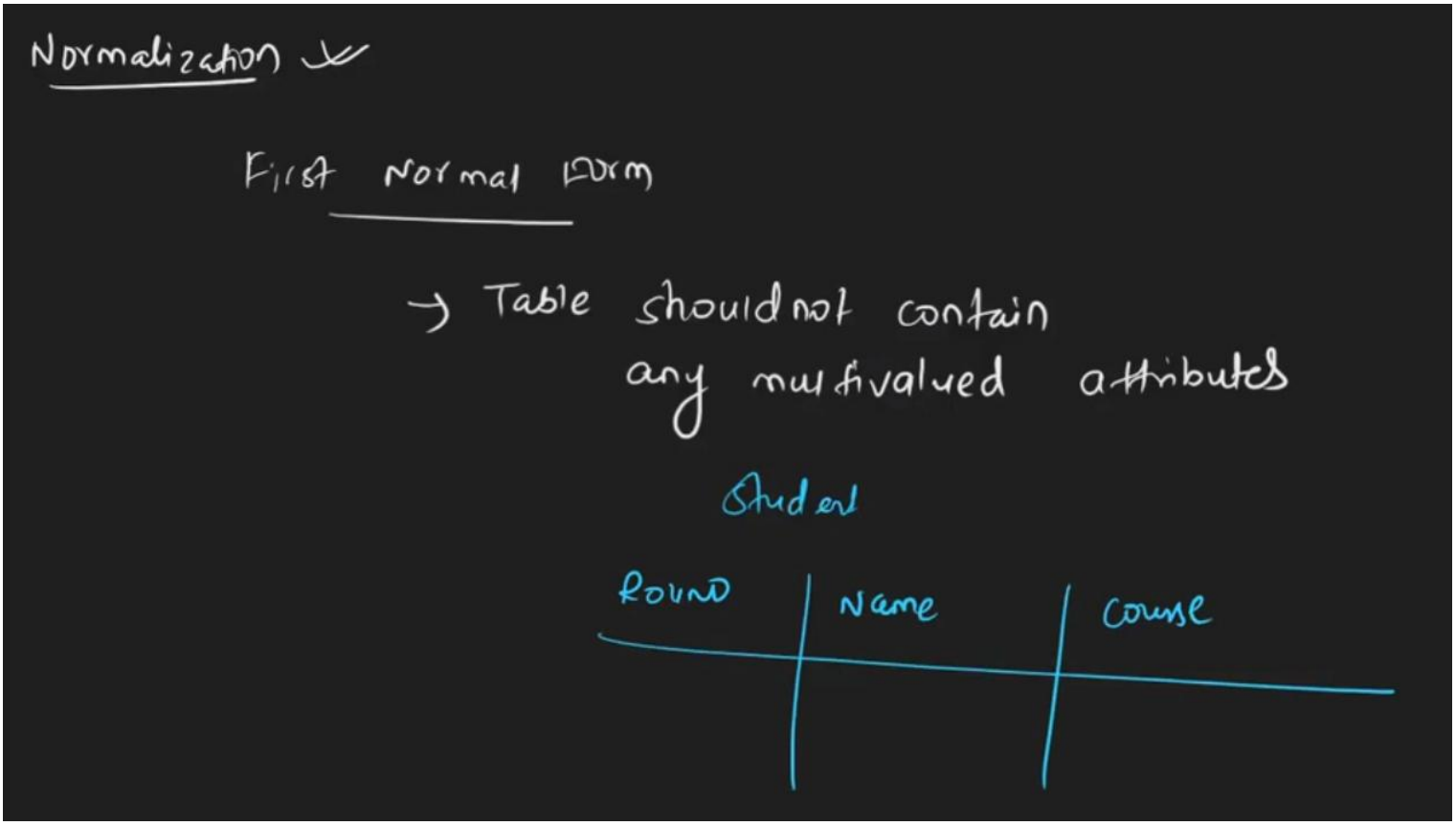
In practice, normalization involves breaking down larger tables into smaller, more manageable tables and establishing relationships between them using foreign keys. This process helps to ensure that each table represents a single entity or concept and that data is stored without unnecessary duplication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

1NF (first Normal Form)

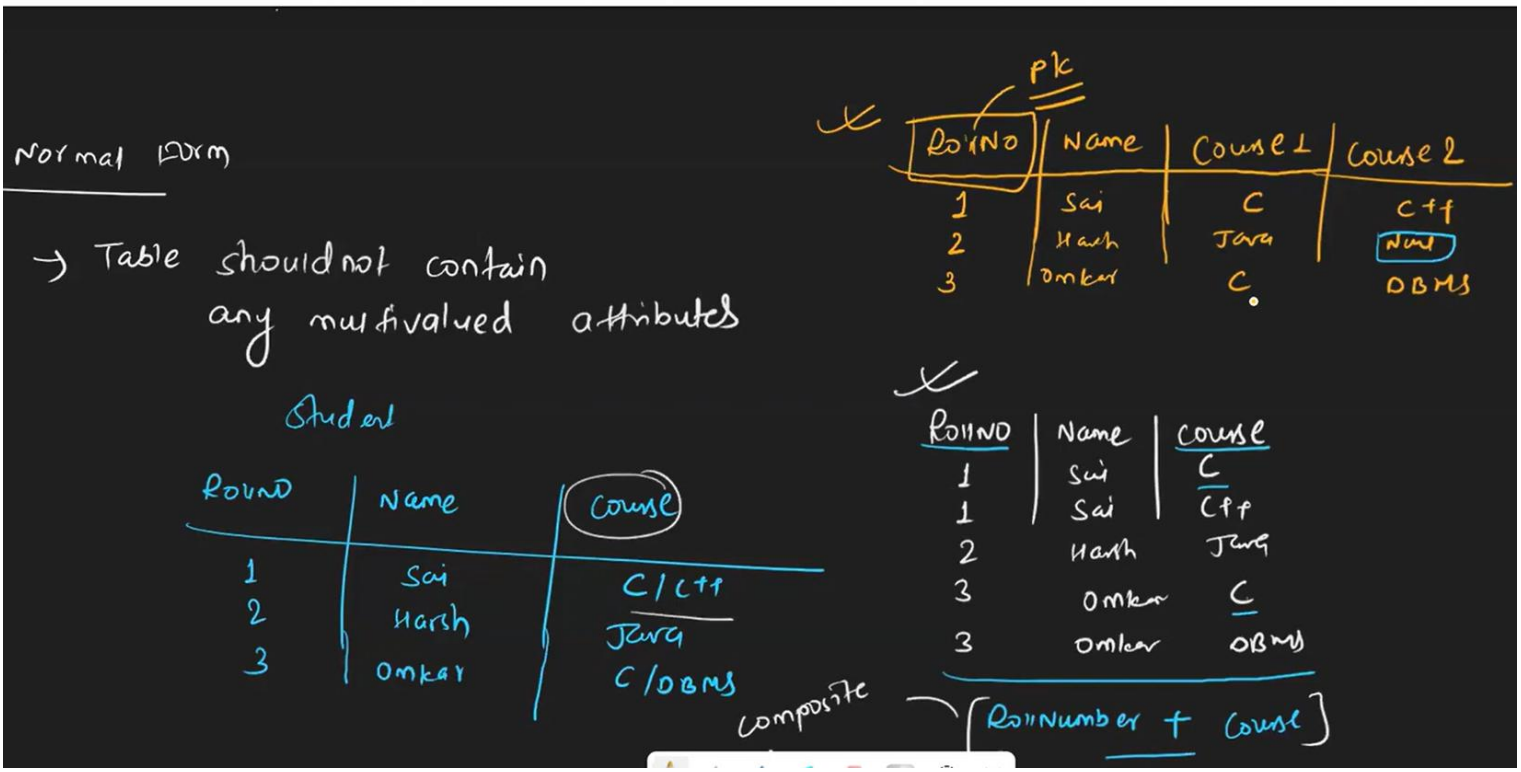
2NF

3NF

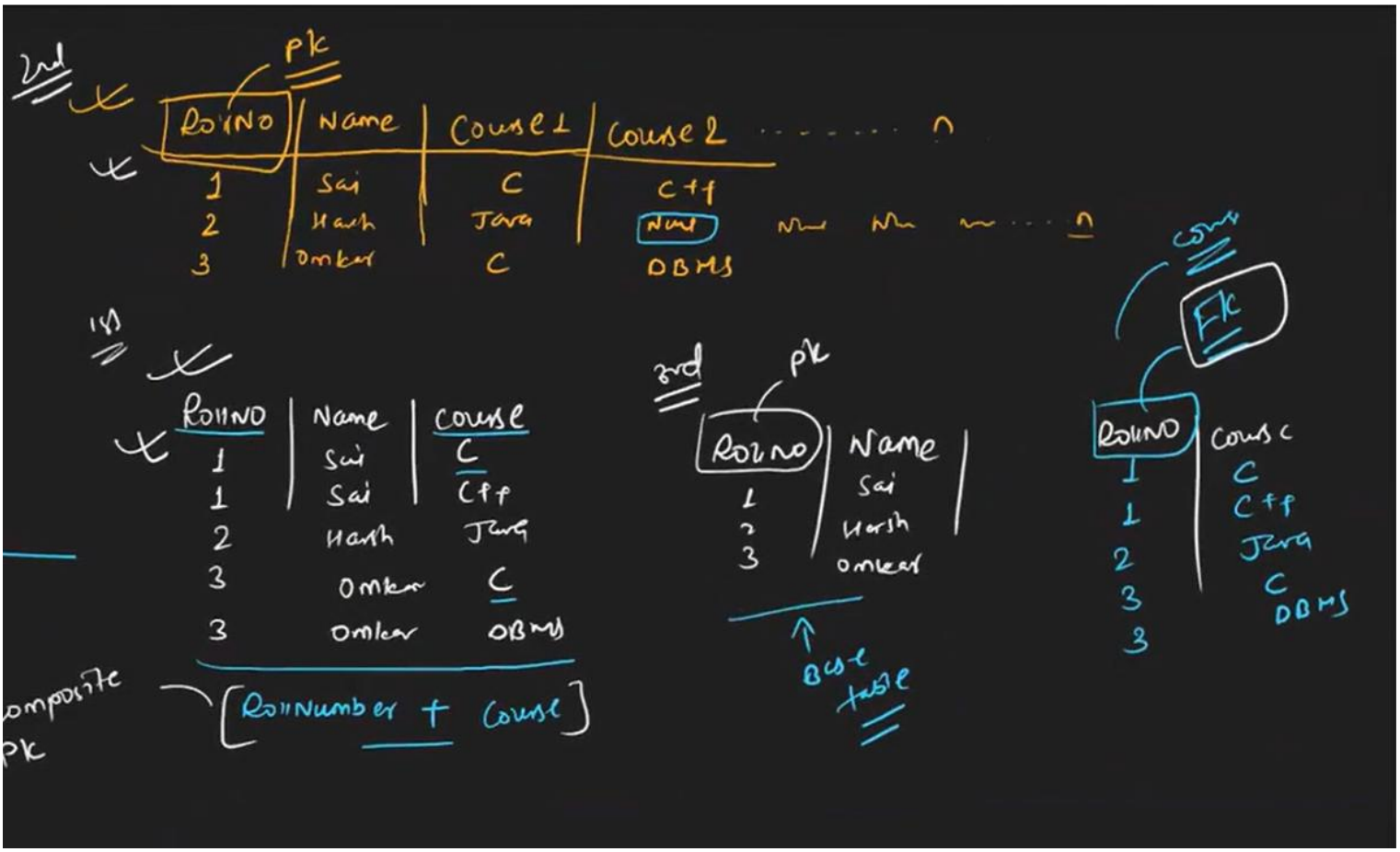
FIREST NORMAL FORM→

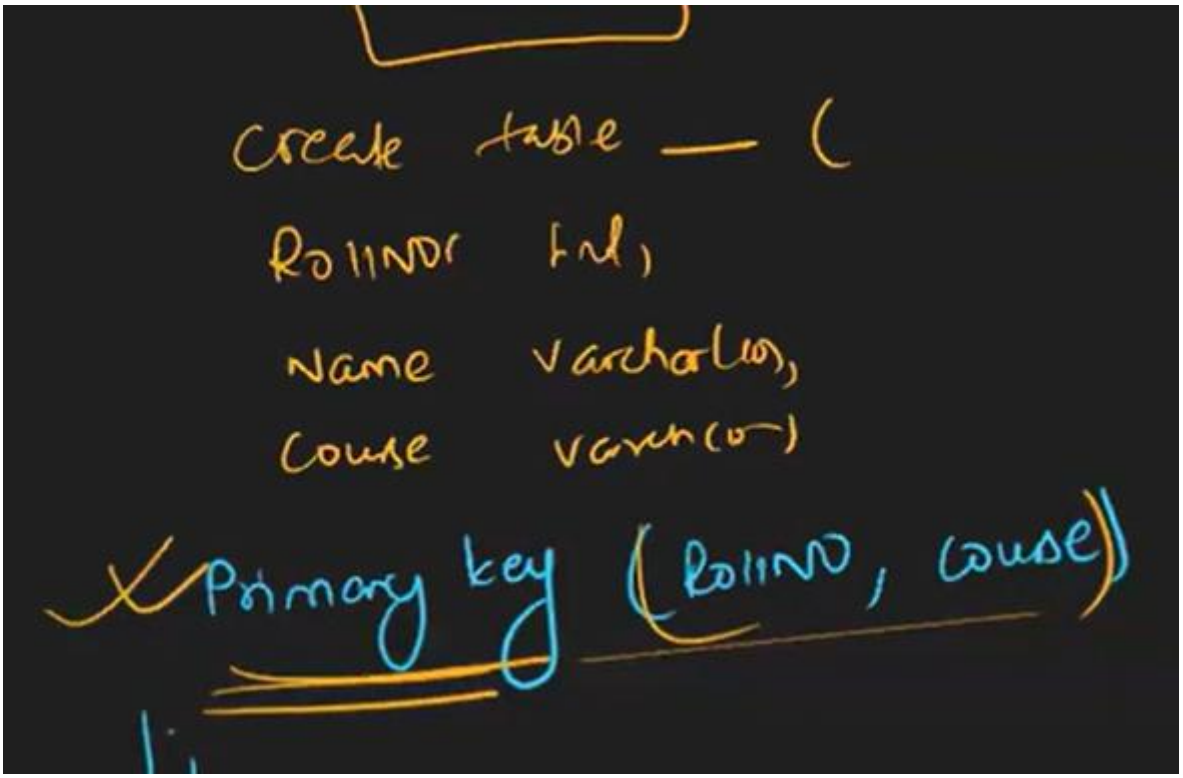


COMPOSED PRIMARY KEY → this is the combination of two columns (by bihalp of columns we create primary key)



Three types of solution



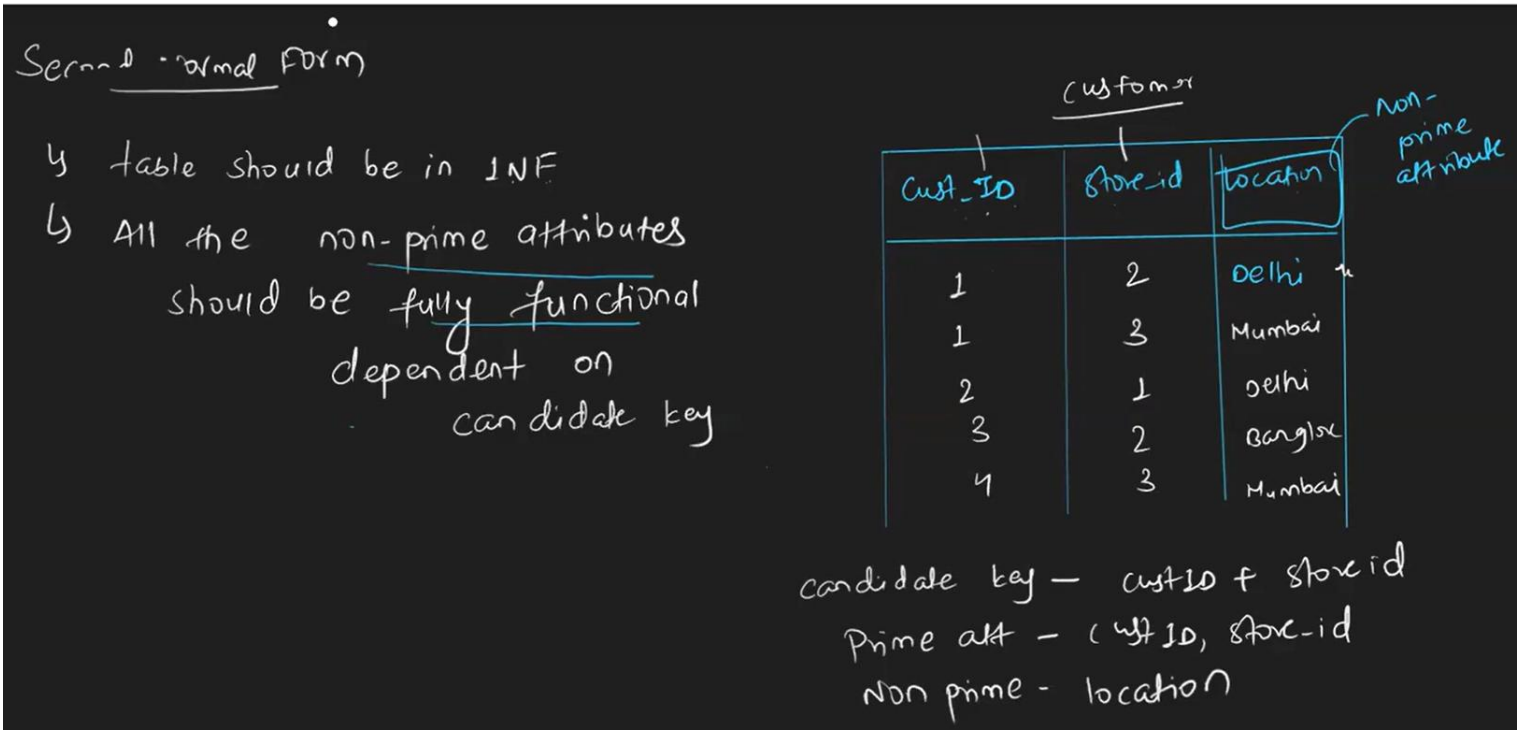


2nd Normal Form→

1. Table should be 1st normal form
2. All the nun-prime attribute should be fully functional dependent on candidate key

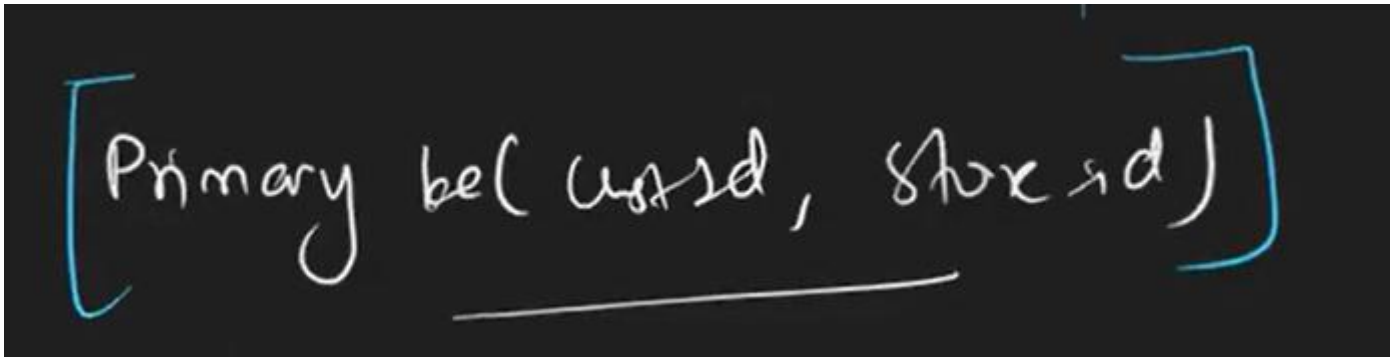
Second Normal Form

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

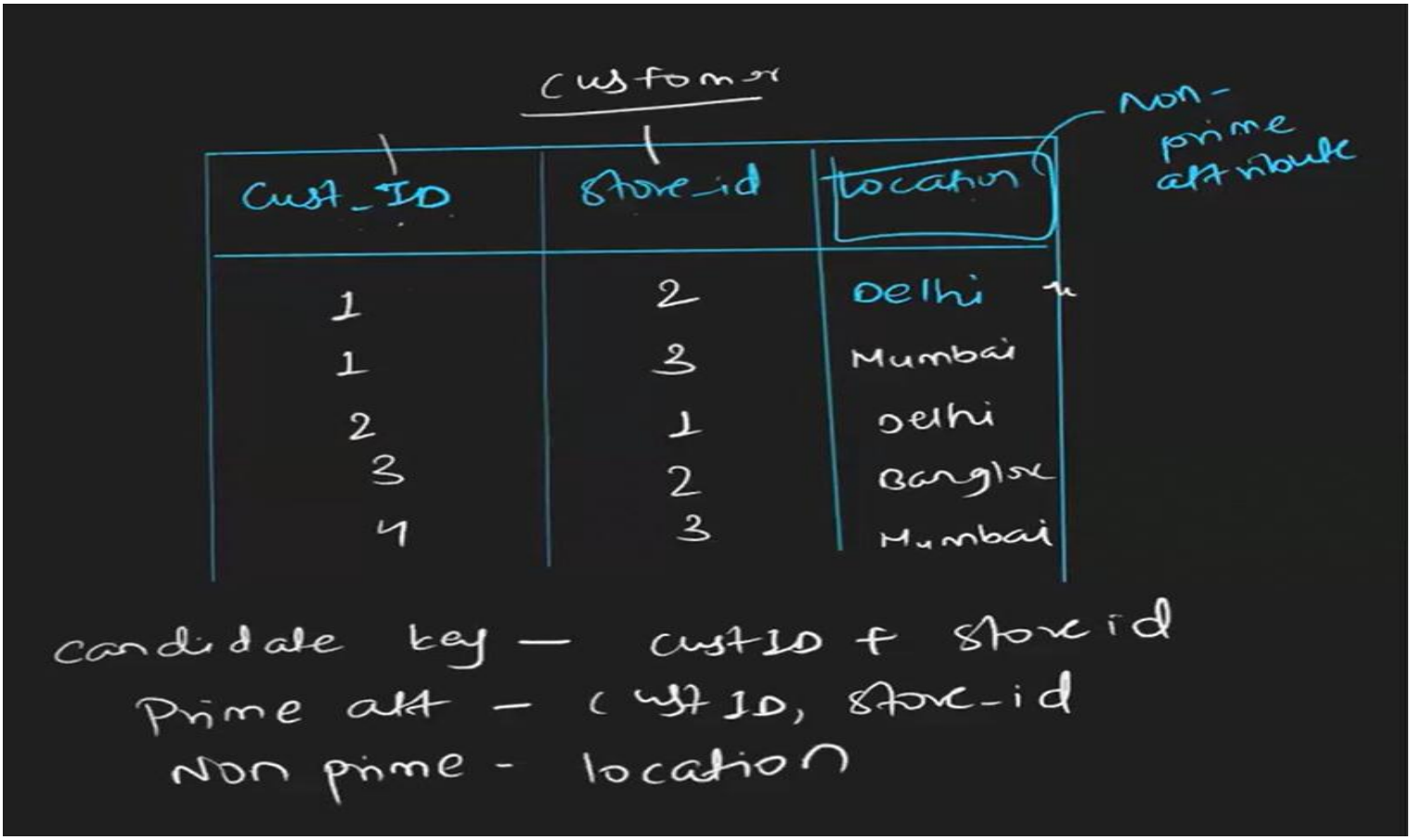


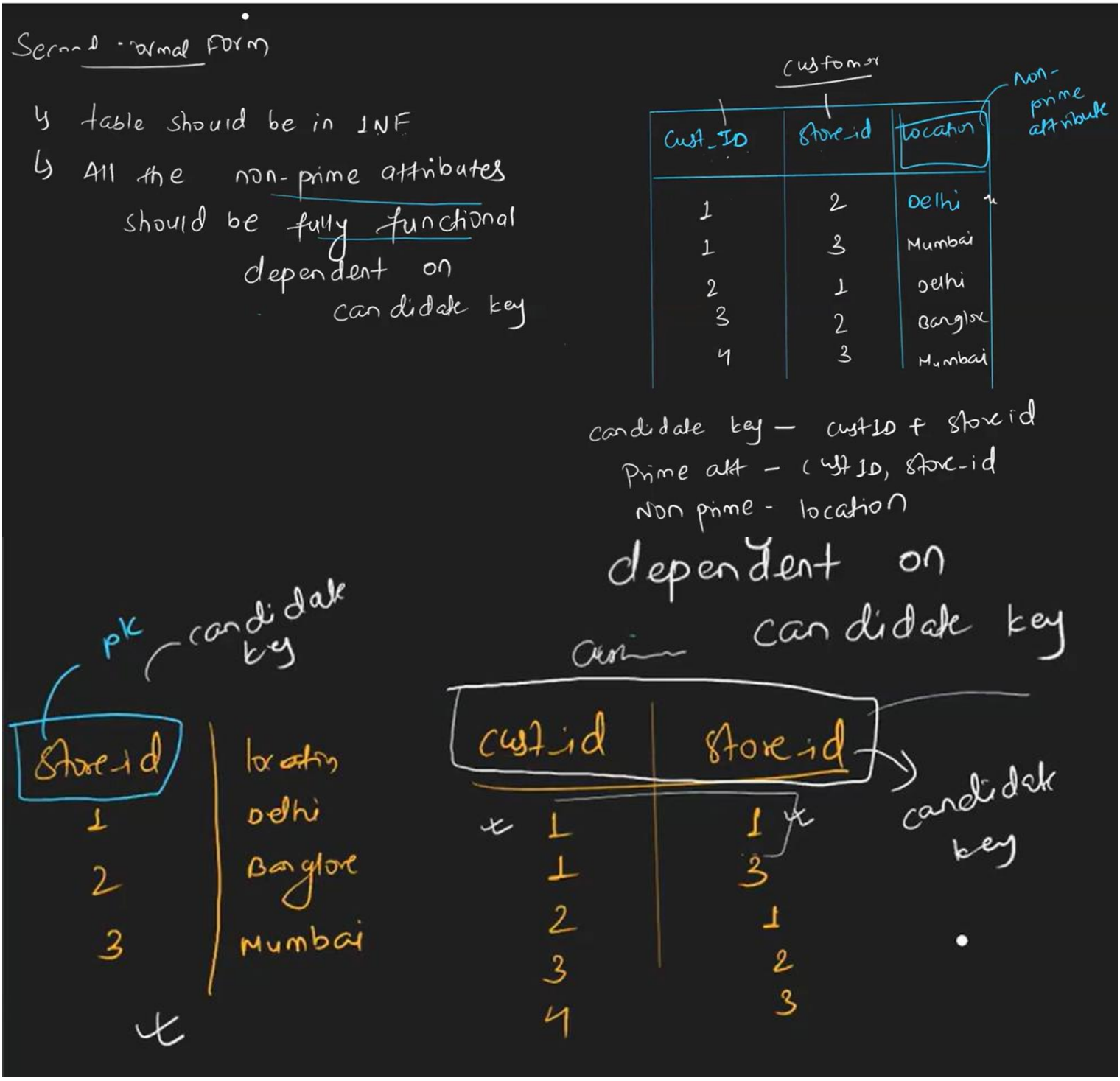
Candidate Key→ It's a both either primary key & composed key

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.





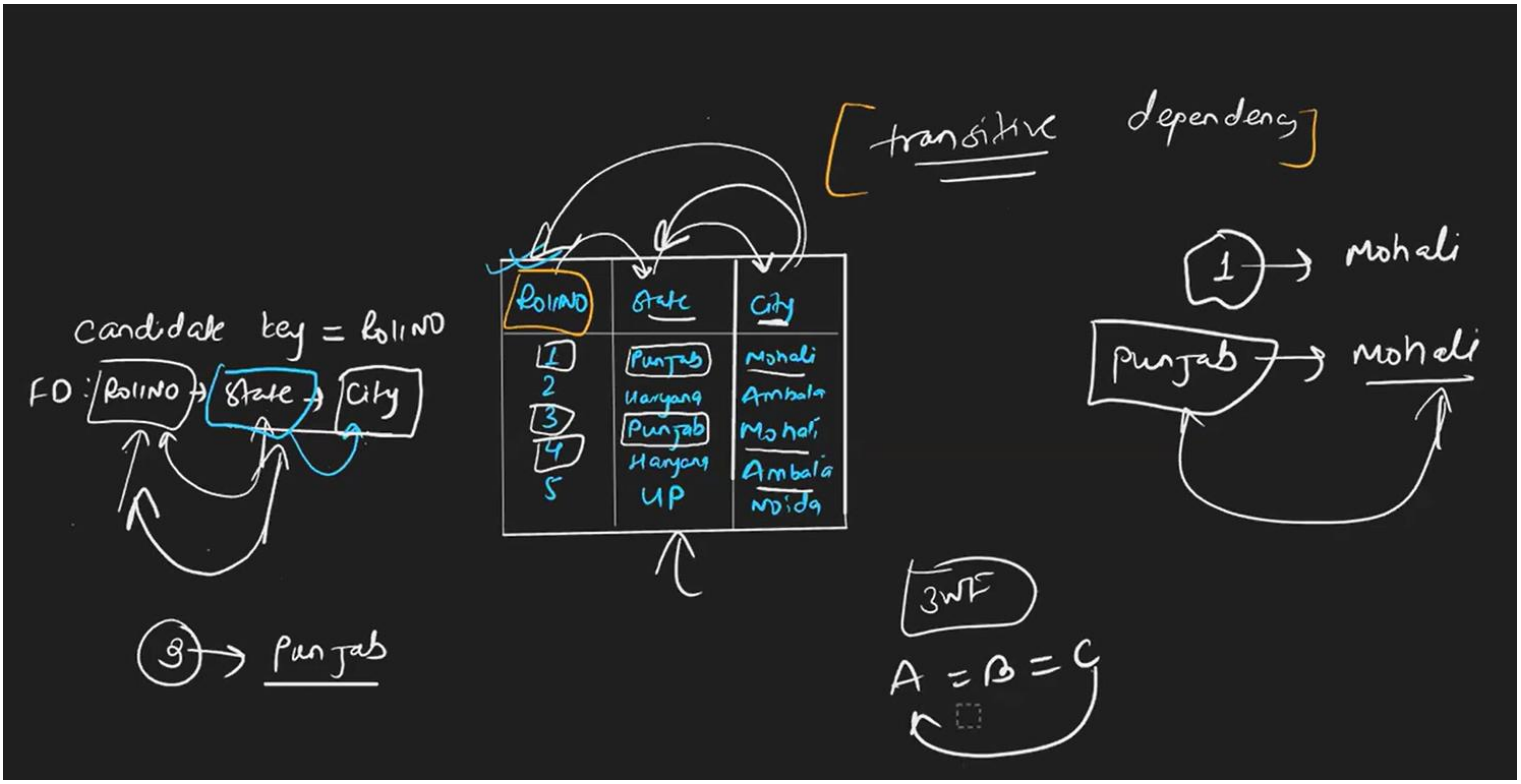
20/02/2024

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 –> Y.

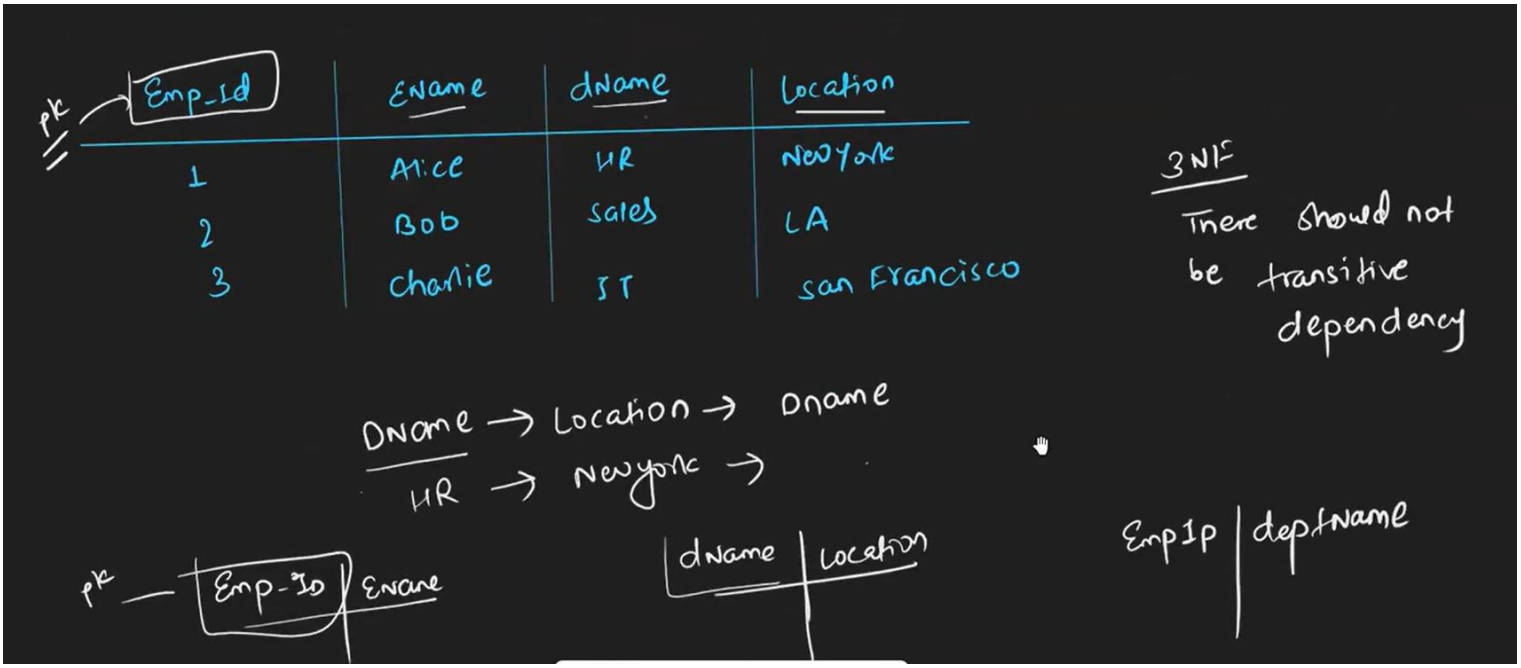
- X is a super key.
- Y is a prime attribute (each element of Y is part of some candidate key).

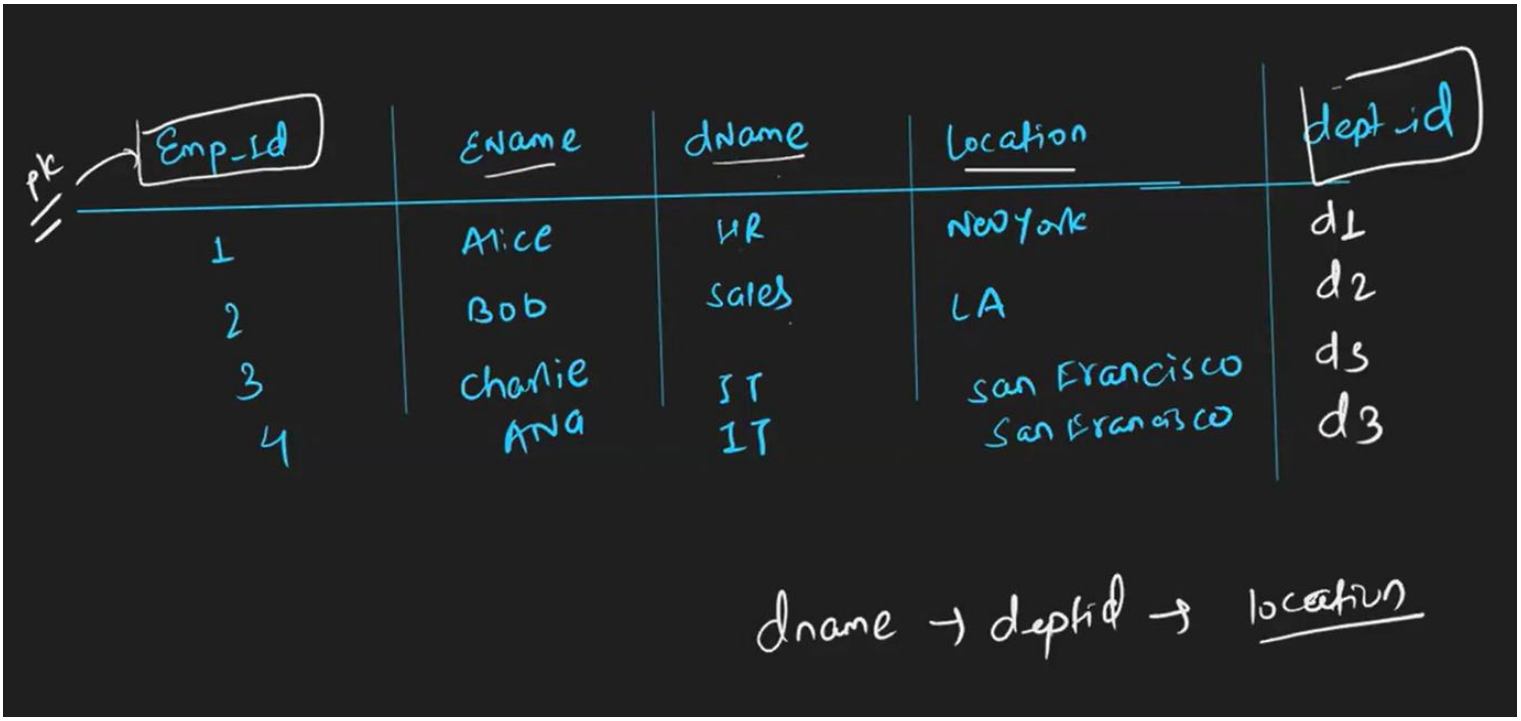


Third Normal Form

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

(SID, (ID, Course)) non-prime attr






```
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
```

```
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
```

```sql
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

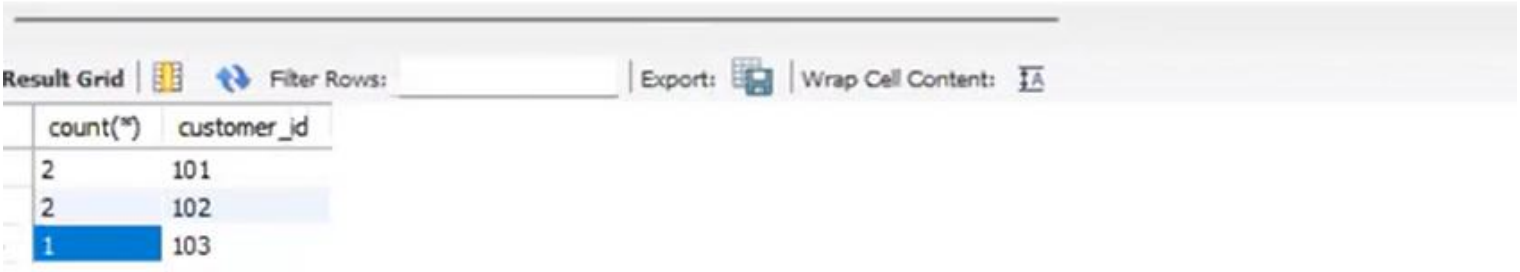
```sql
1    -- aggregate function
2 •  select * from employee;
3 •  select  max(salary) from employee;
4 •  select min(salary) from employee;
5 •  select sum(salary) from employee;
6 •  select avg(salary) from employee;
7 •  select count(empid) from employee;

-- 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 :

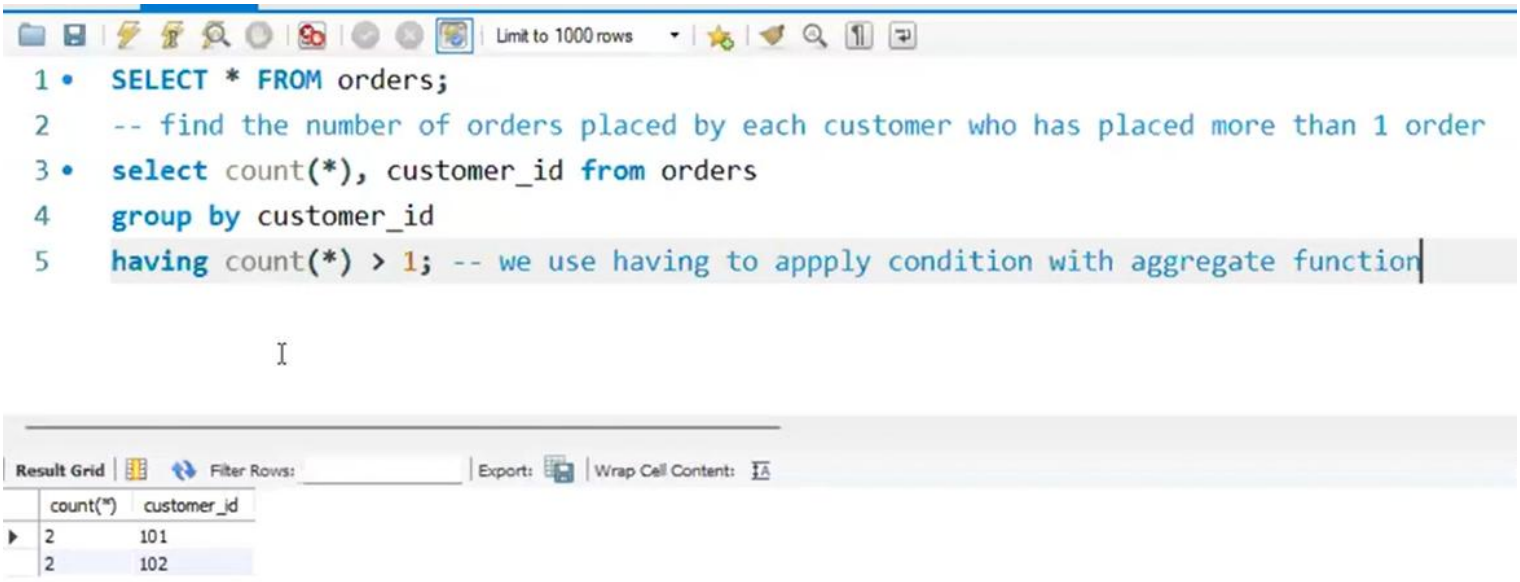
```sql
SELECT * FROM orders;
-- find the number of orders placed by each customer
select count(*), customer_id from orders
group by customer_id;
```



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

count(*)	customer_id
2	101
2	102
1	103

HAVING CLAUSE



Limit to 1000 rows

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


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

count(*)	customer_id
2	101
2	102

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

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

customer_id	sum(total_amount)
101	150.00
102	120.00

```sql
select * from employee;
-- Like operator : to search for a pattern
select * from employee where name like "r%";

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

```sql
select * from employee;
-- Like operator : to search for a pattern
select * from employee where name like "%a%"; -- ends with an a

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

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

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

Filter Rows: | Export: | Wrap Cell Content:

id	name	location	age	salary	Hobby
	Ana	Pune	34	34	NULL

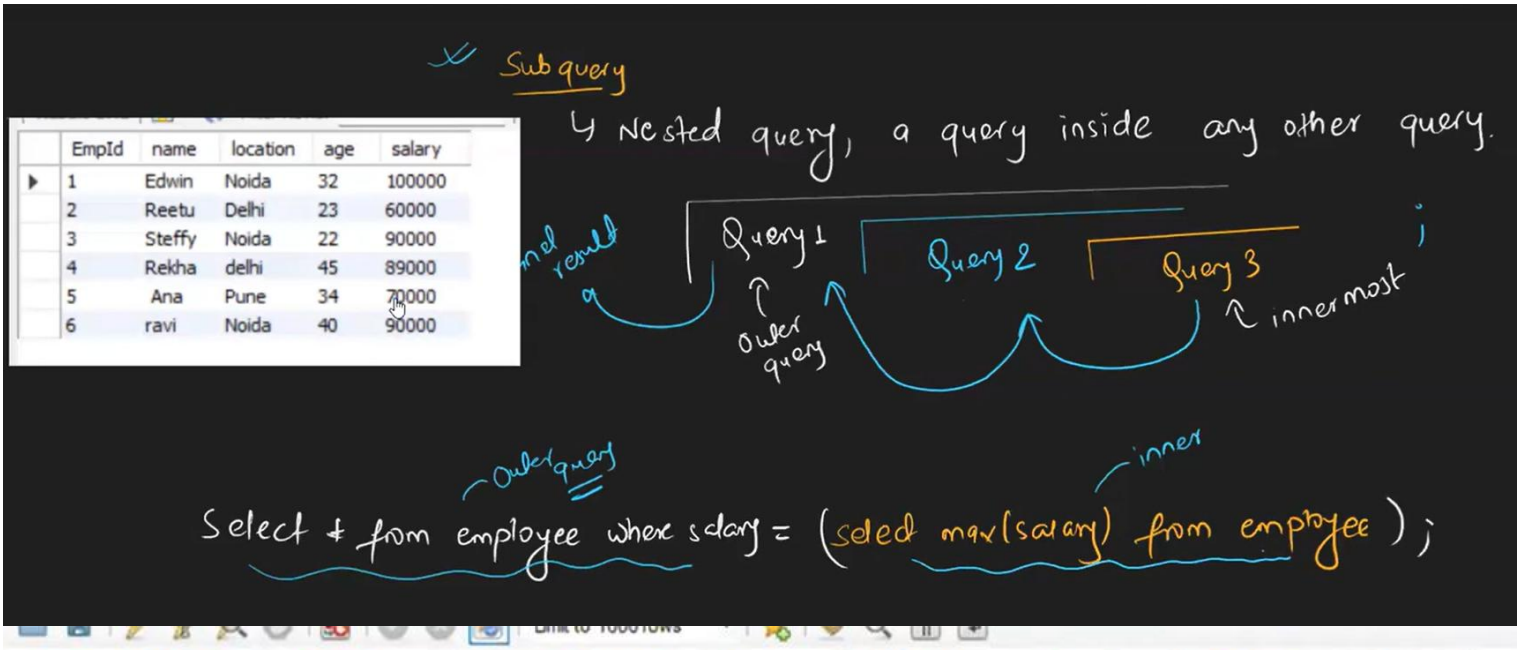
Date→ 21/02/2024

Order by (desc, limit,offset, distinct)

Offset-> it's mins 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
```

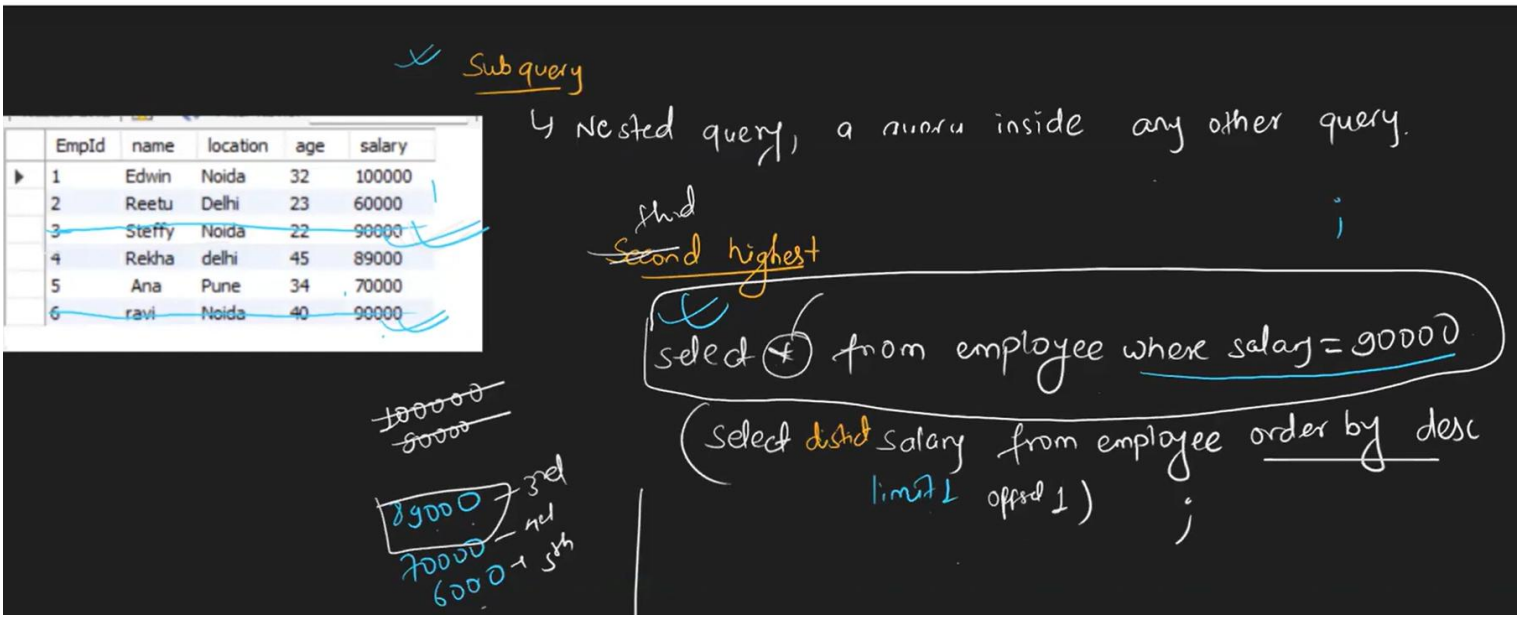
SUB QUERY → Query inside in the other query.



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

```
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);



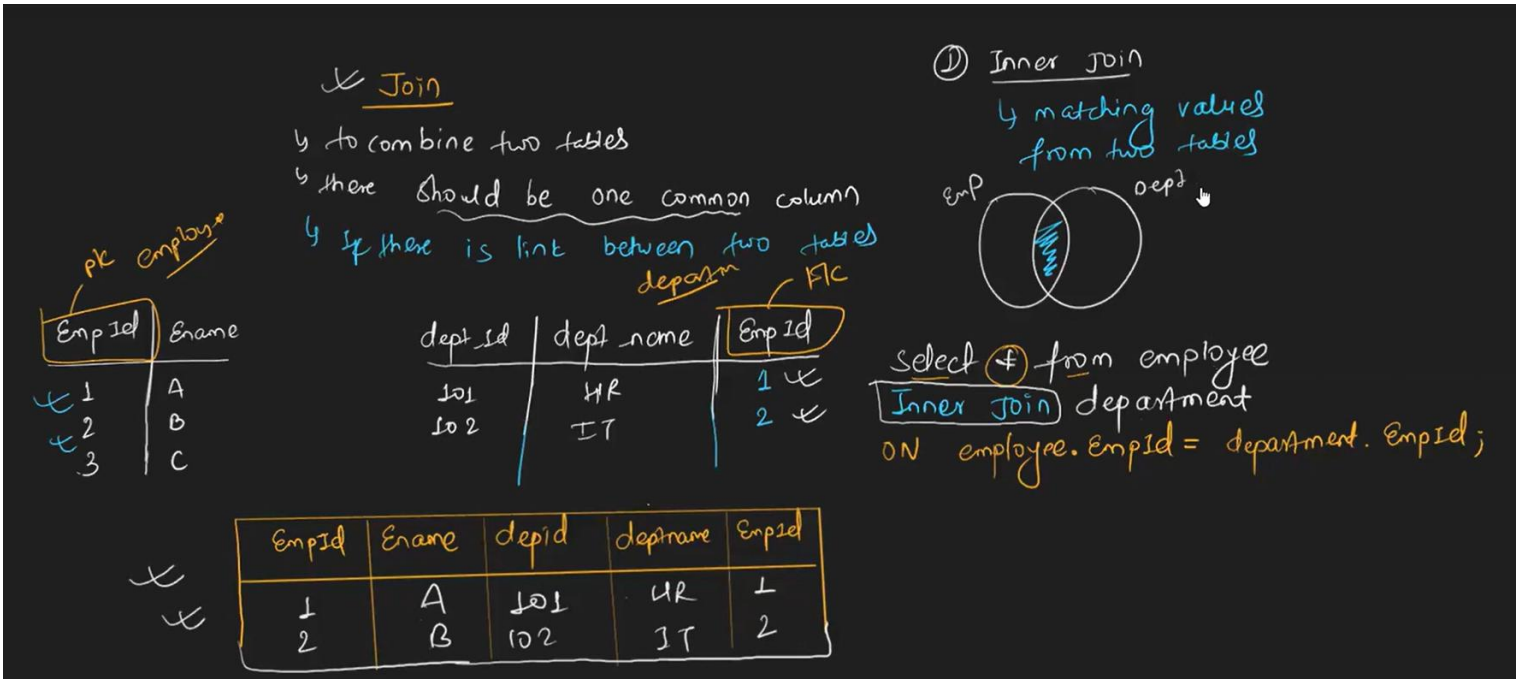
JOIN

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.

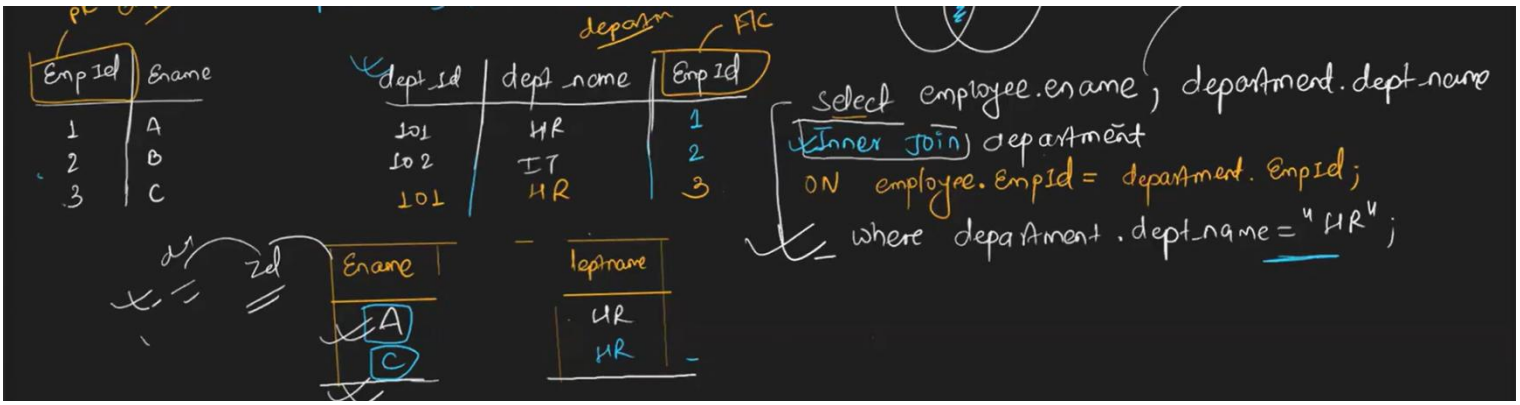
There are several types of JOINs in SQL, including:

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) →



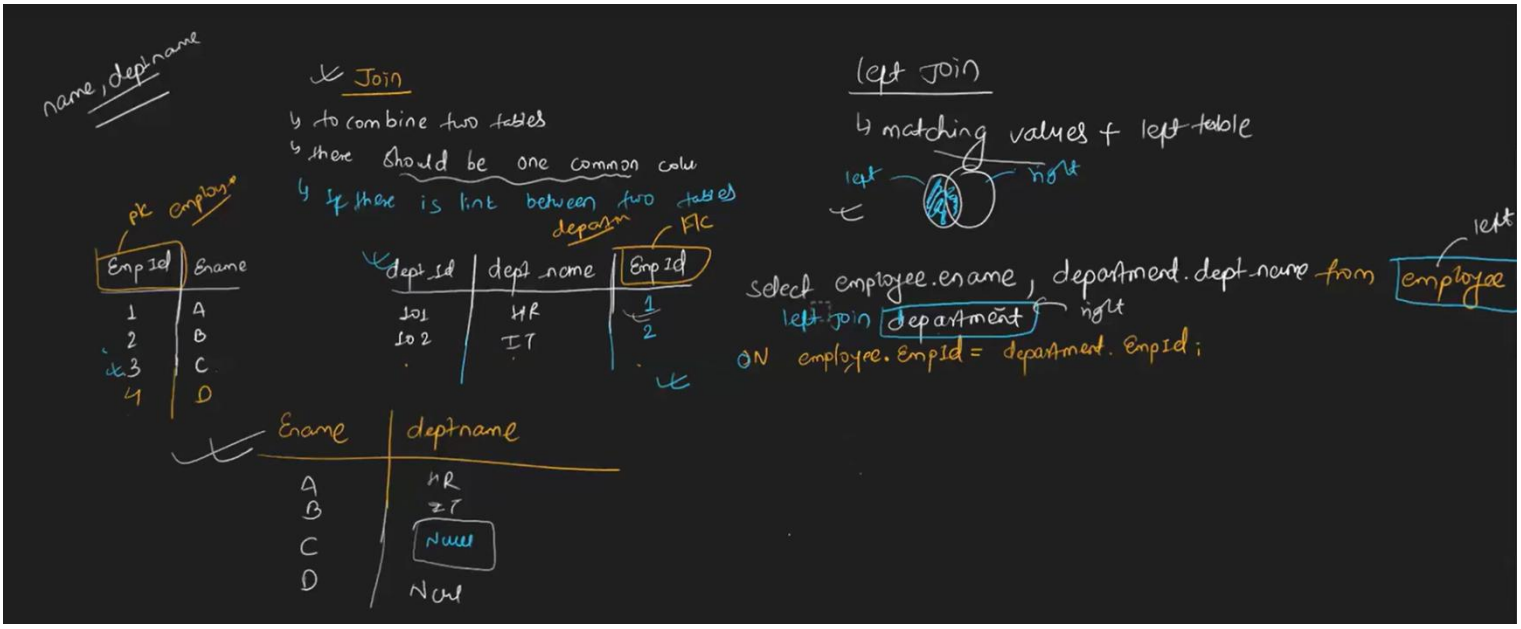
IF YOU WANT TO KNOW ONLY HR DEPT



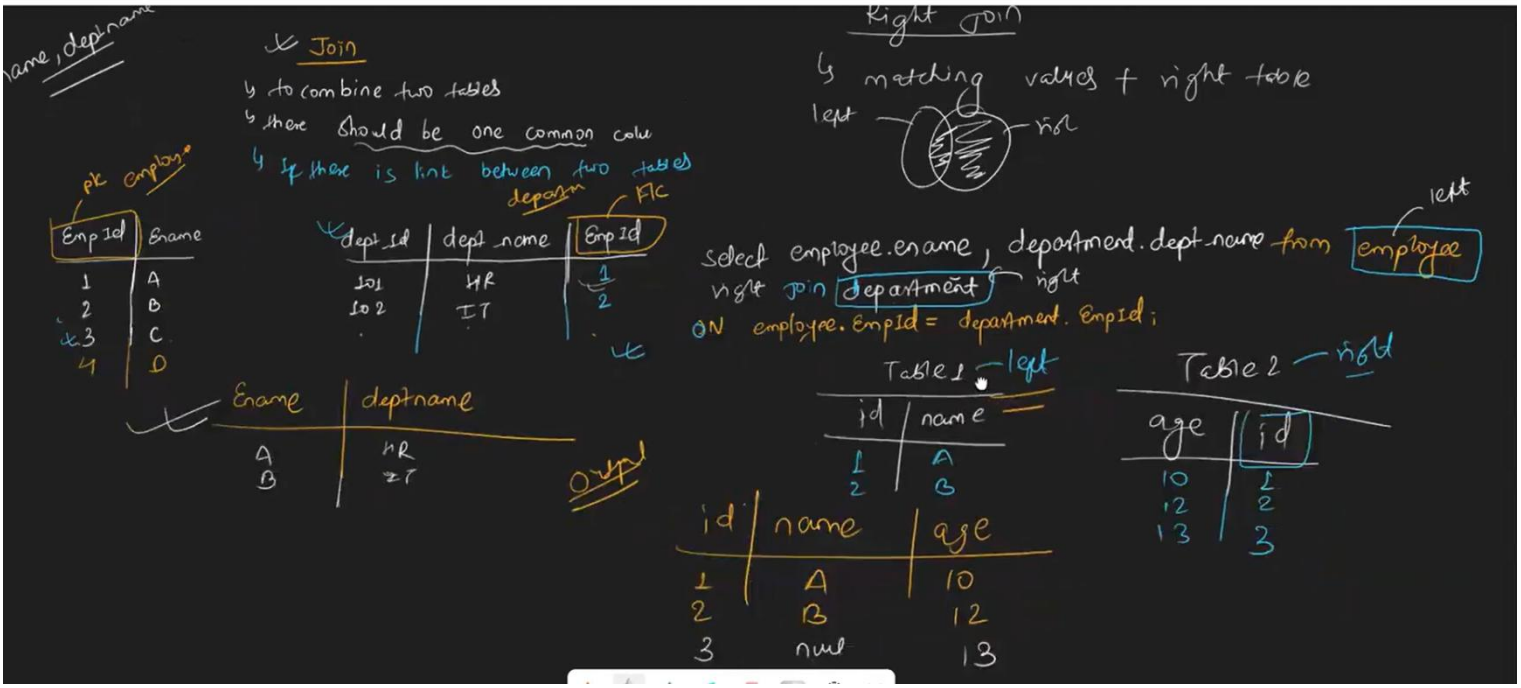

```
1 • select employee.name, department.deptname from employee inner join department
2   on employee.empid = department.id
3   where department.deptname = "Shipping";
```

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

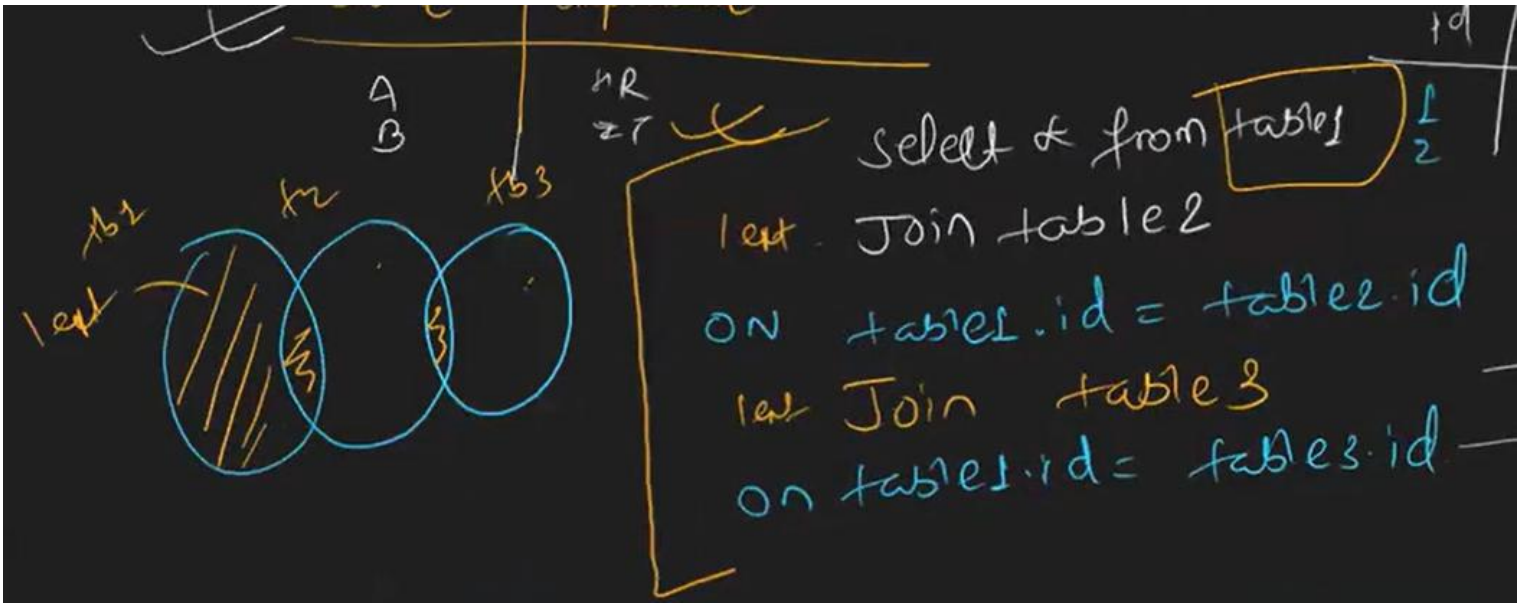
NOTE- 1st table is left and 2nd table is right table always



RIGHT JOIN -> all data show from right table



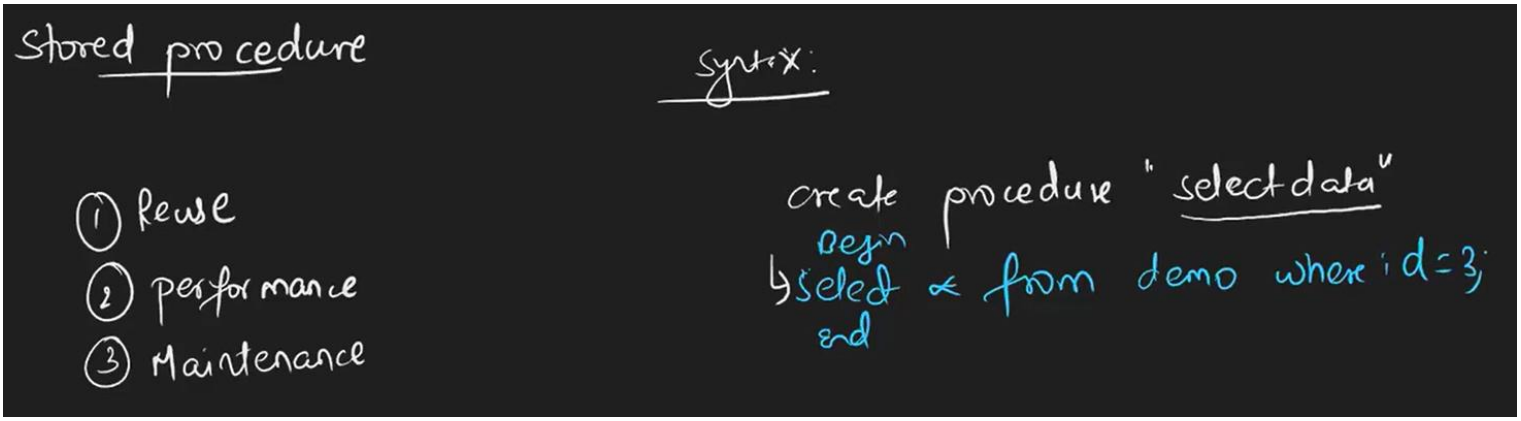
More then two table

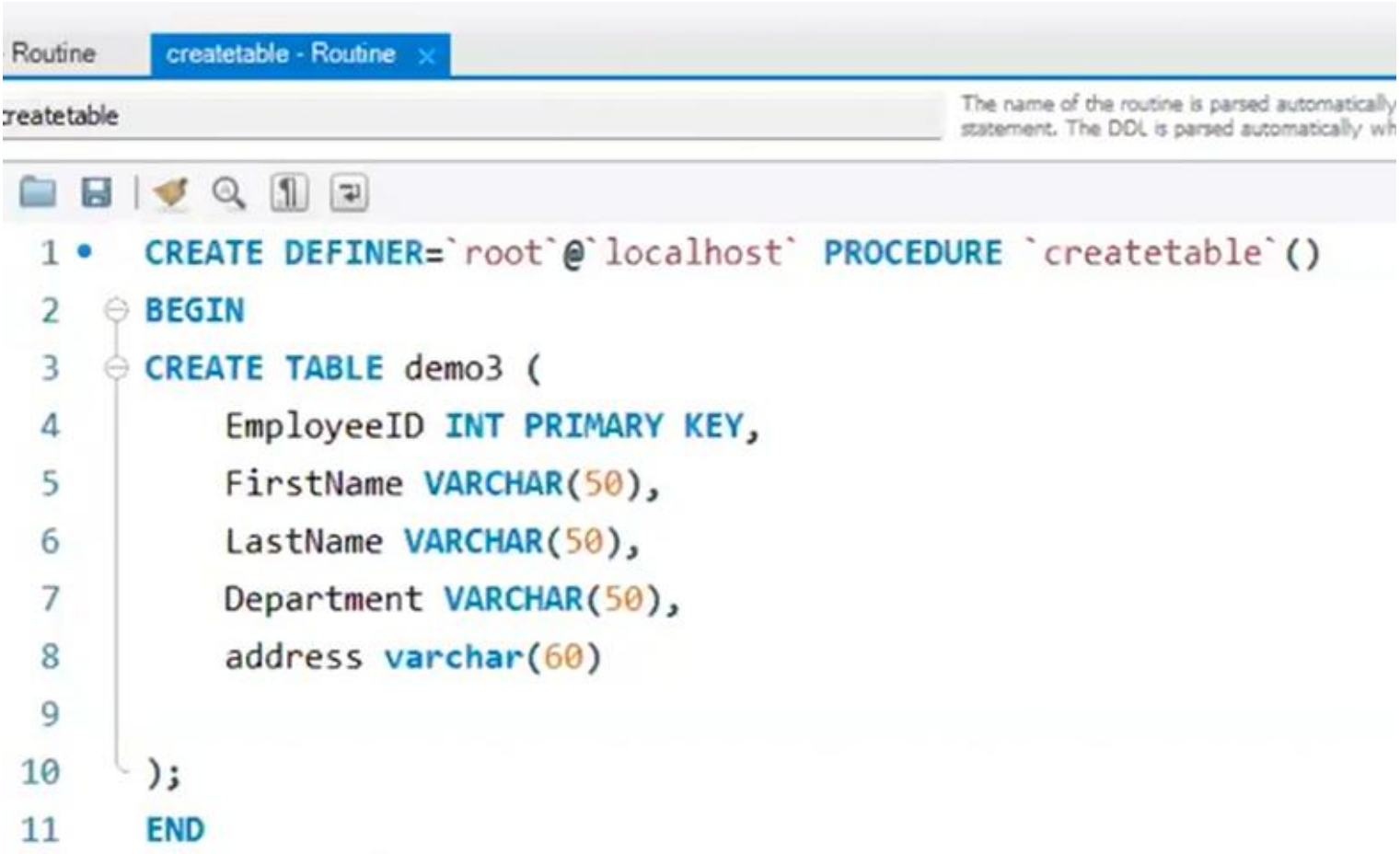


```
Question on Join

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





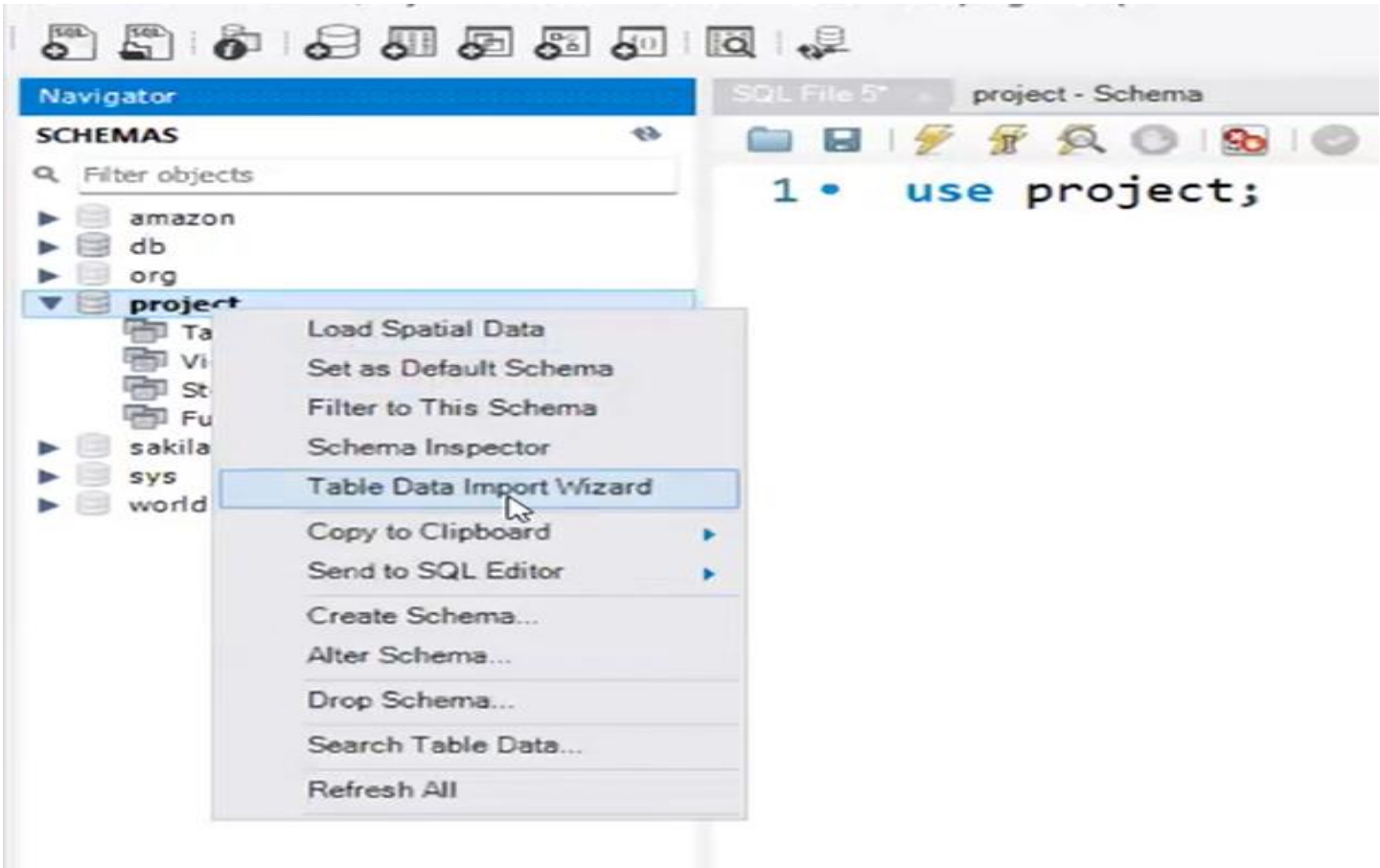
View Table → In SQL, a view is a virtual table based on the result set of a SELECT statement. Unlike a base table, which stores data physically, a view does not store data itself. Instead, it is a saved SQL query that can be treated as a table within the database. When you query a view, the underlying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

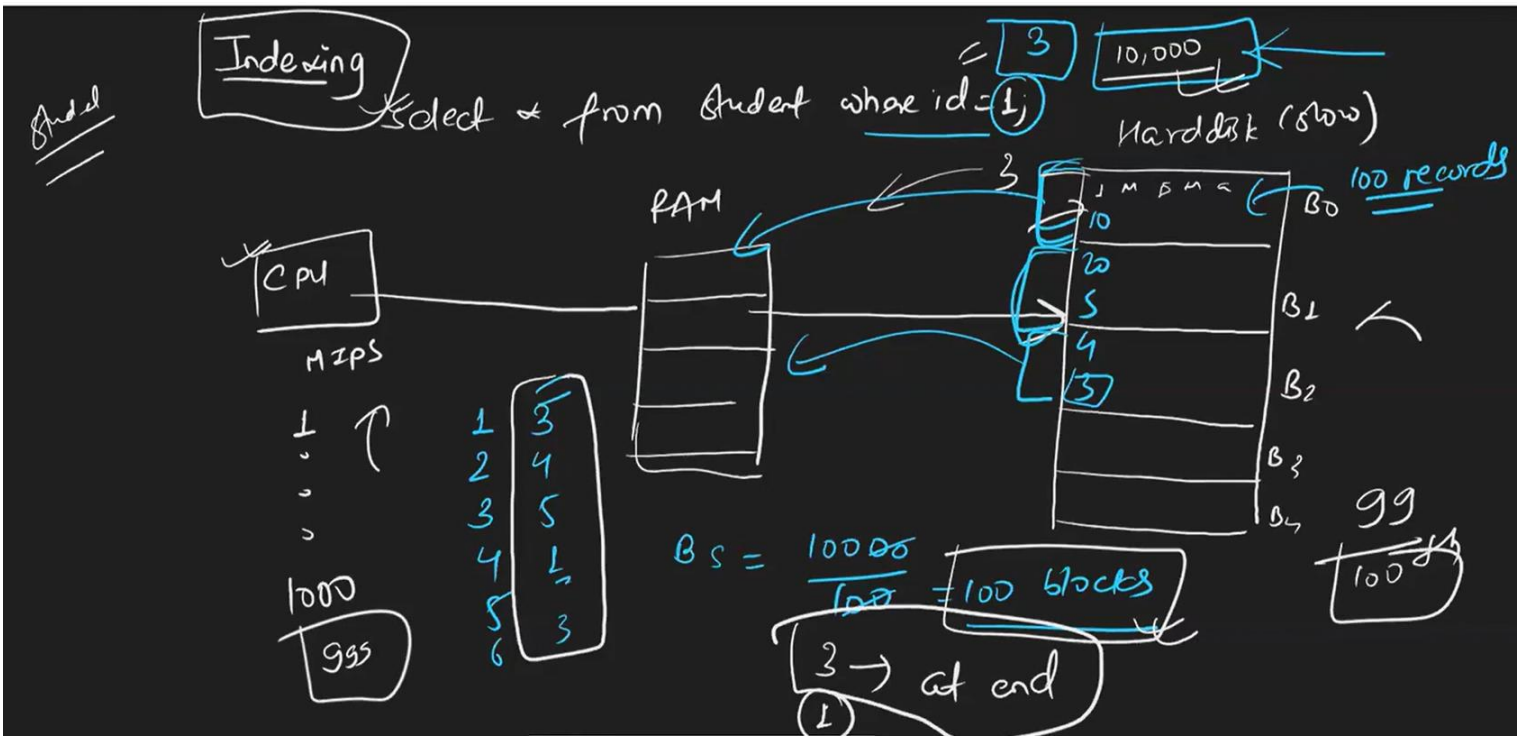
```sql
create view emp as
select empid, name, location
from employee
where empid <5;
select * from employee;

select * from emp where empid=1;
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

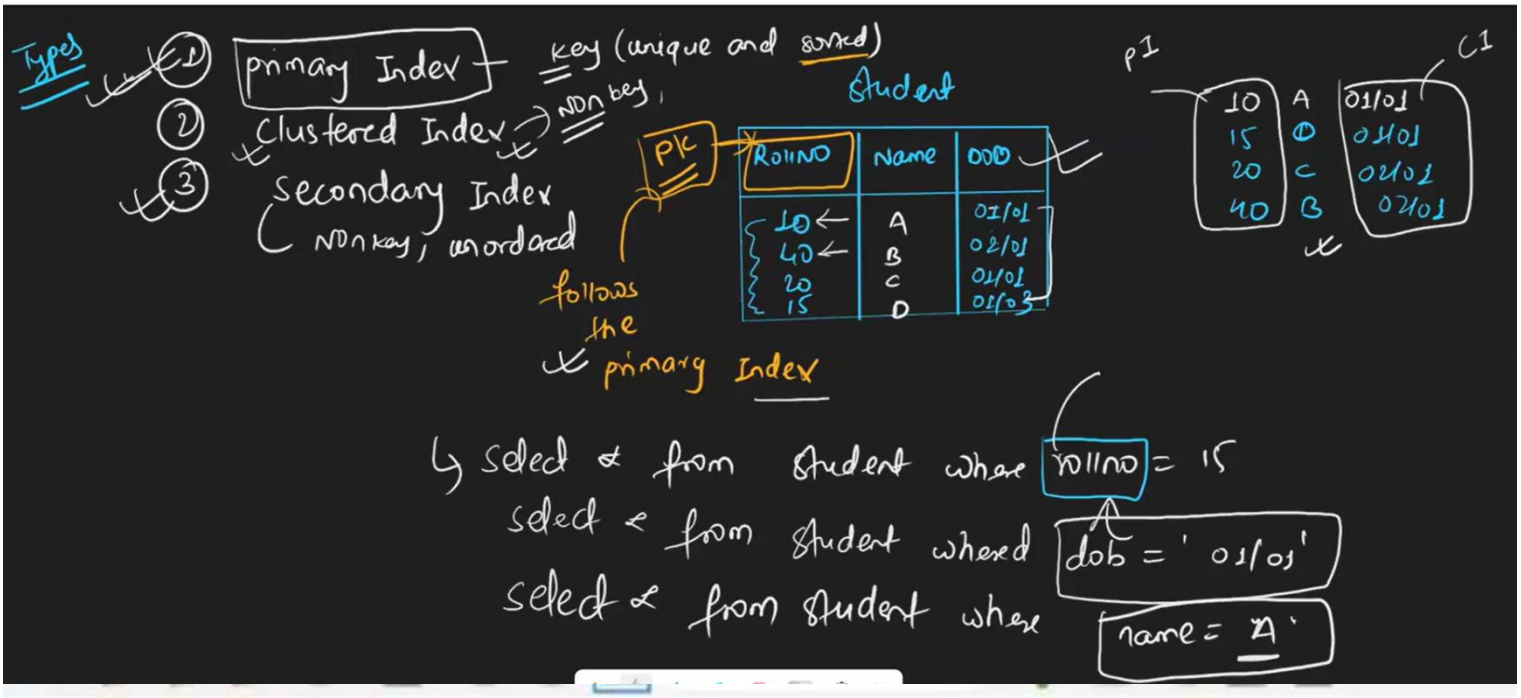


INDEXING → In SQL, indexing is the process of creating data structures to improve the speed of data retrieval operations on a database table. An index is a special data structure associated with a table that allows for faster retrieval of rows based on the values of certain columns.



Three types of INDEX

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



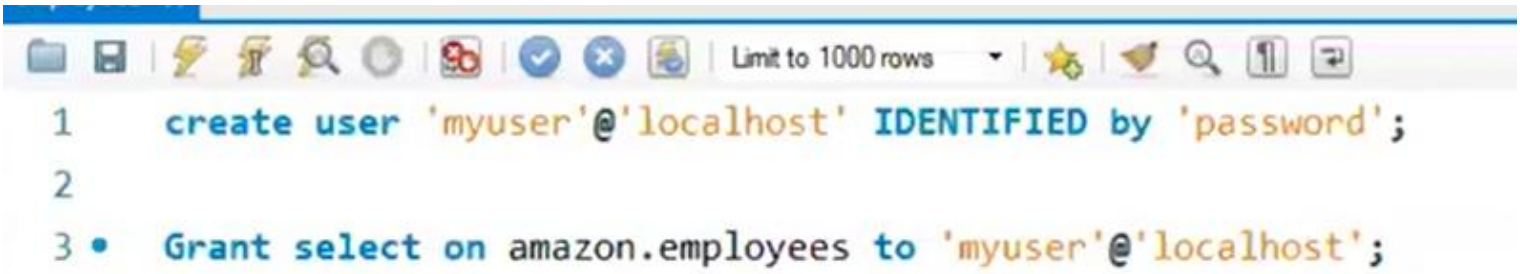

```sql
CREATE TABLE Employee1 (
    EmployeeID INT primary key,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50),
    Salary DECIMAL(10, 2)

);
```

```sql
create index emp on Employee1 (FirstName, LastName);
```

Date → 23/02/2024

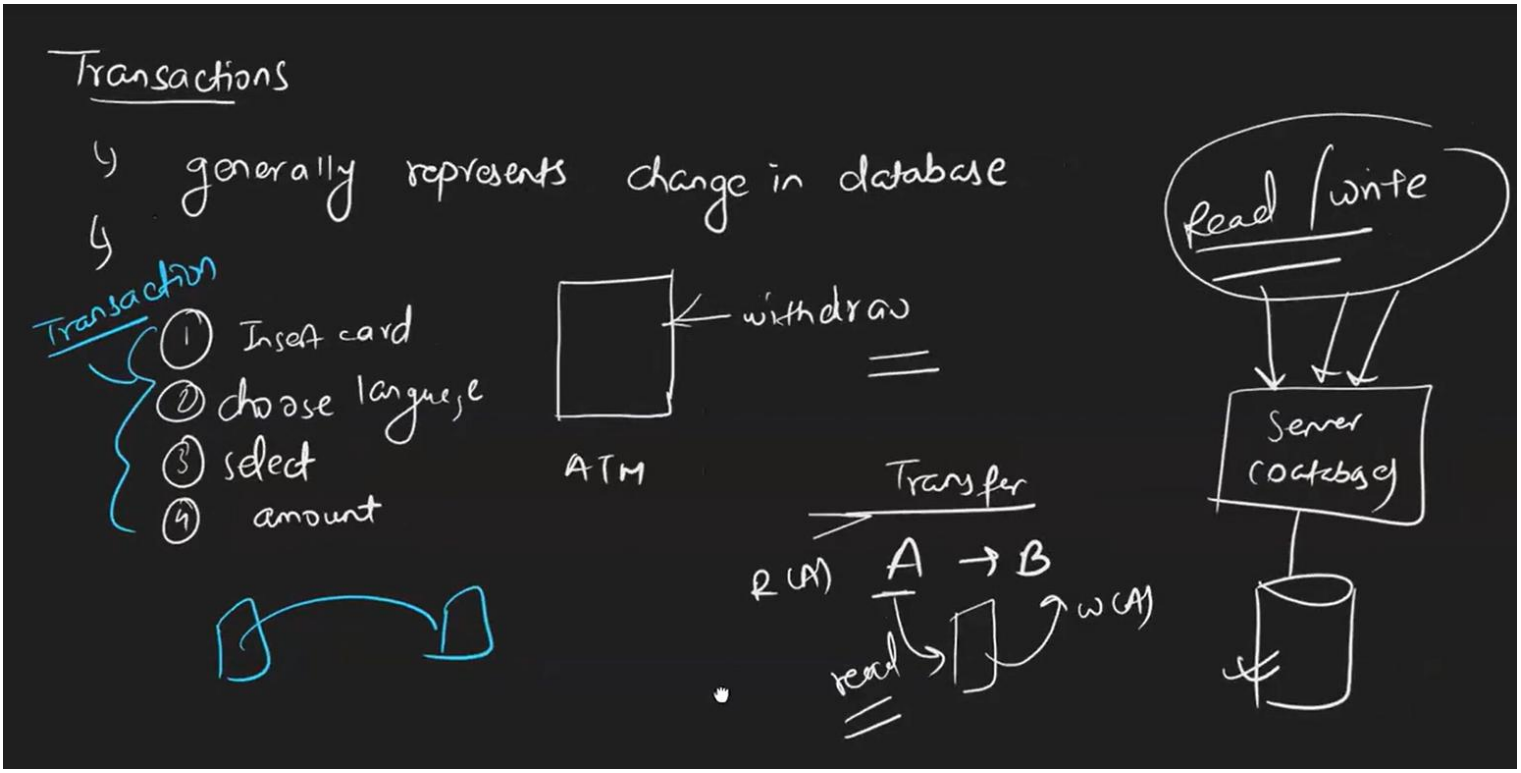
Create a local host->

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

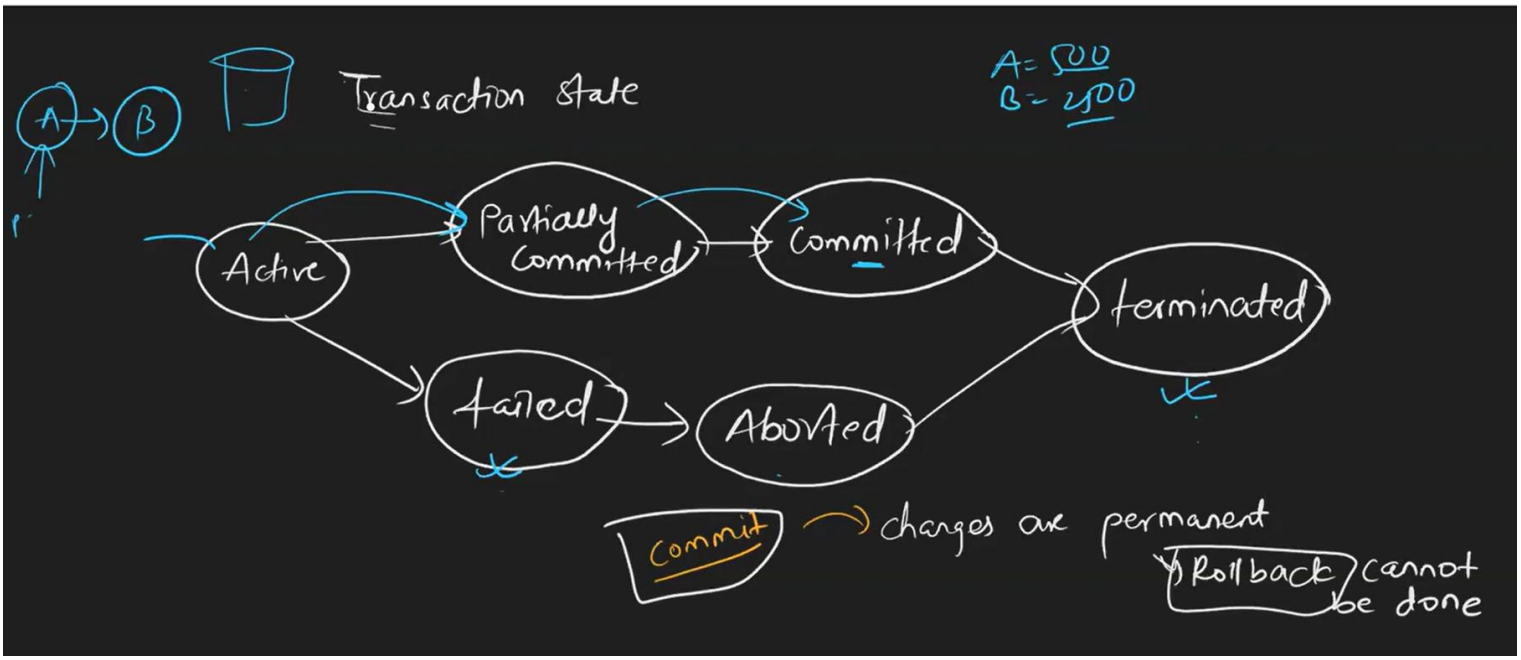


Revoke→ Retake you permission (remove your permission on user@localhost)

Transactions



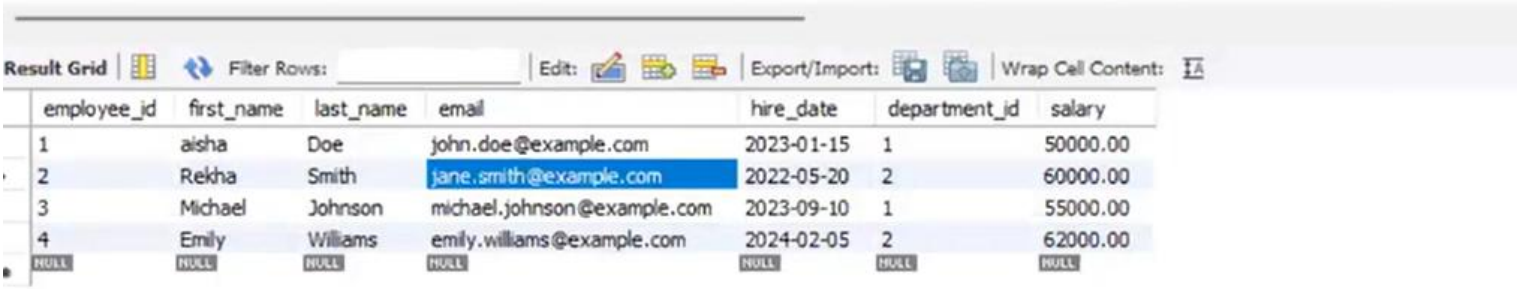
State of Transactions



Set Autocommit=0;

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

```sql
select * from employees;
update employees  set first_name = "aisha" where employee_id = 1;
update employees  set first_name = "Rekha" where employee_id = 2;
rollback;
```



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

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
NULL	NULL	NULL	NULL	NULL	NULL	NULL

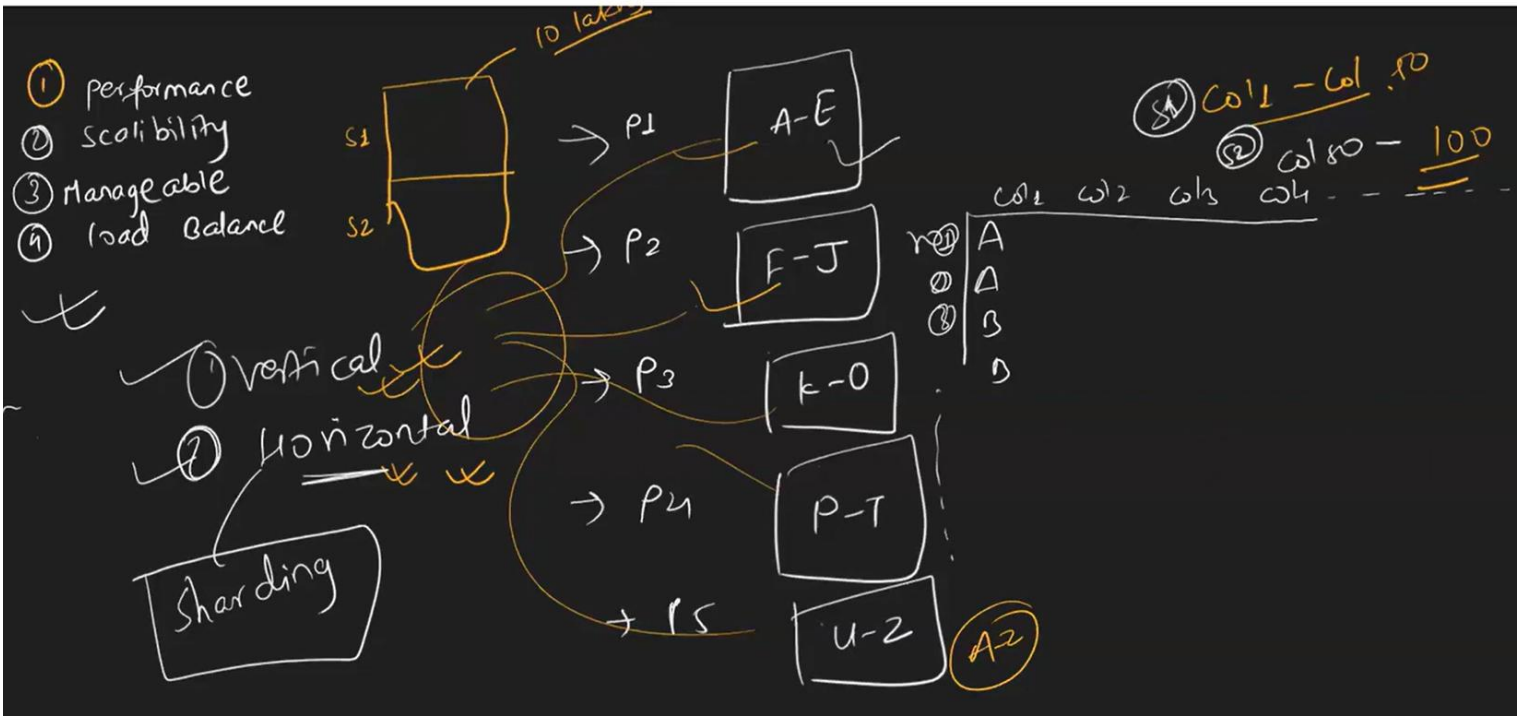
Rollback → if you apply this query then you change will rollback.

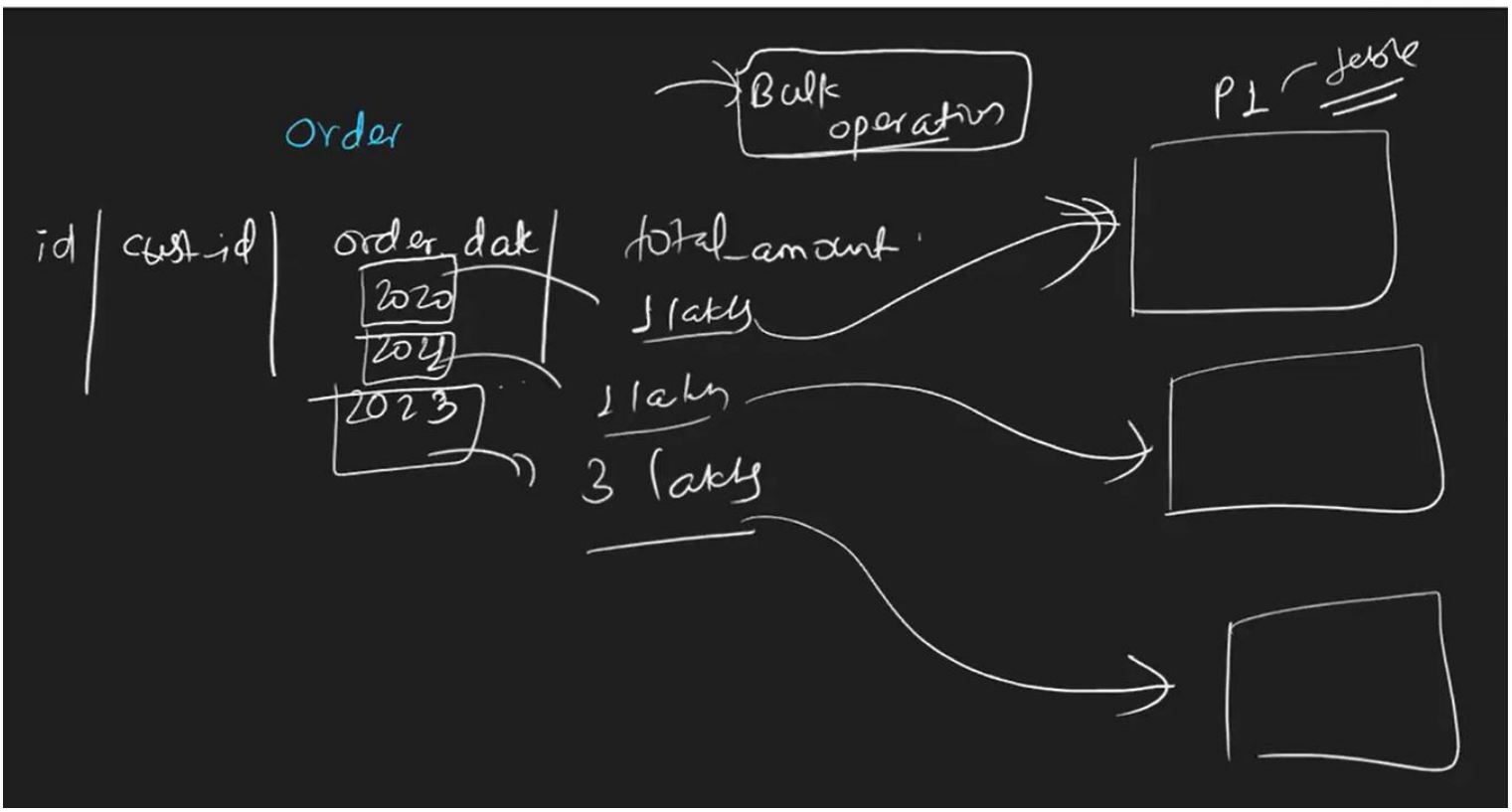
Commit→ it is for save your changes, after apply this query you can undo the changes.

```sql
select * from employees;

update employees  set first_name = "aisha" where employee_id = 1;
commit;
update employees  set first_name = "Rekha" where employee_id = 2;
rollback;

savepoint my_savepoint1;

update employees  set first_name = "Rekha" where employee_id = 2;
savepoint My_savepoint2;

delete from employees where employee_id =4;

rollback to My_savepoint2;
```

Partitioning (dividing)

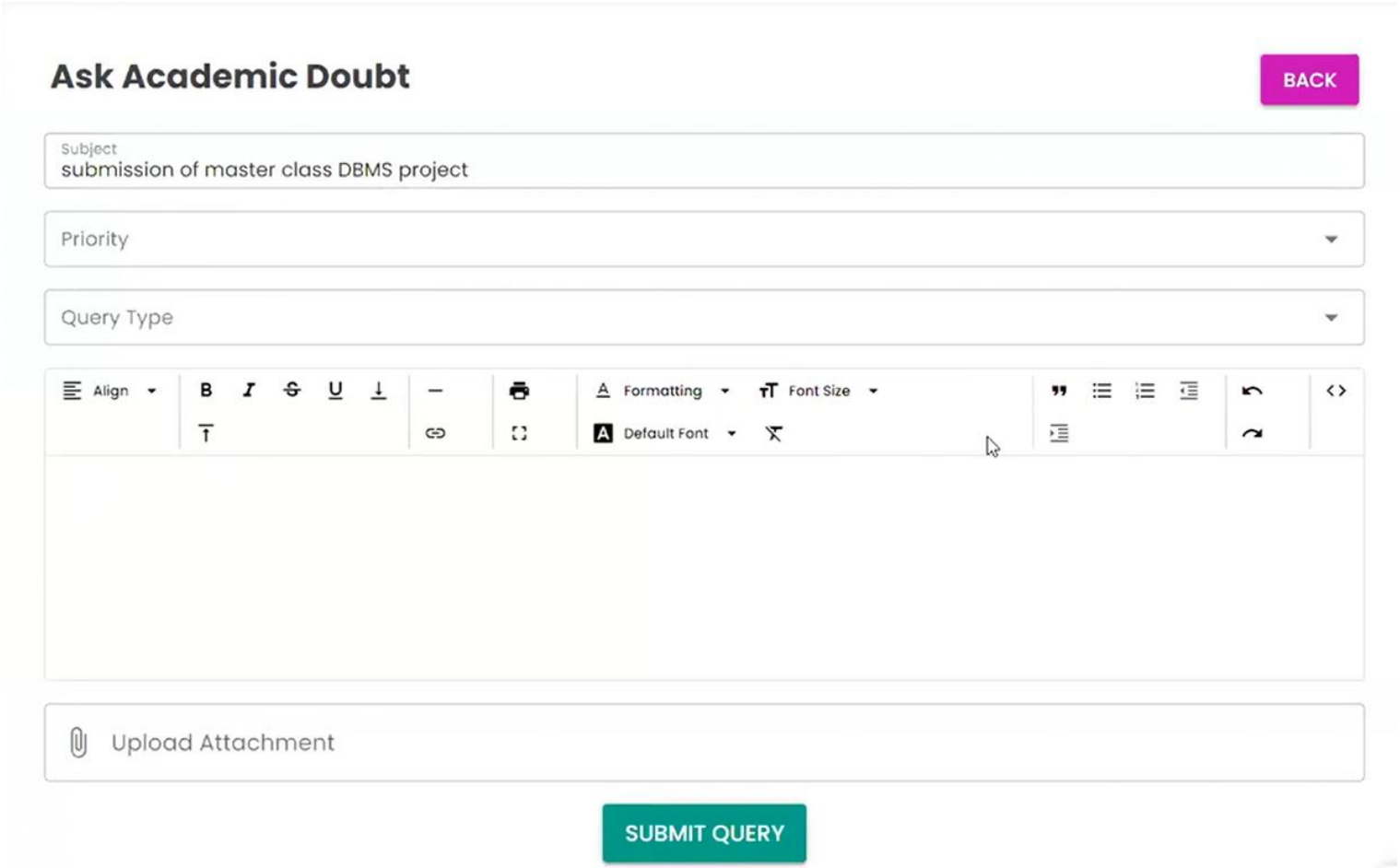
Divide the table of small part





Project→

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



Last date- 8 March 24