

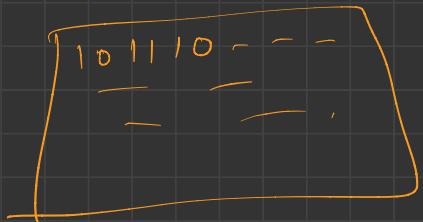

Lec-1

DBMS → ???

⇒ Data ?? ⇒ Bits / Bytes →
≡ ↳ collection of raw bytes

image → Collection of bytes.

integers → 11111111



memory.
in DD

BMI	Weight	Height (inches)
21.16	112	61
21.79	135	66
28.29	213	73

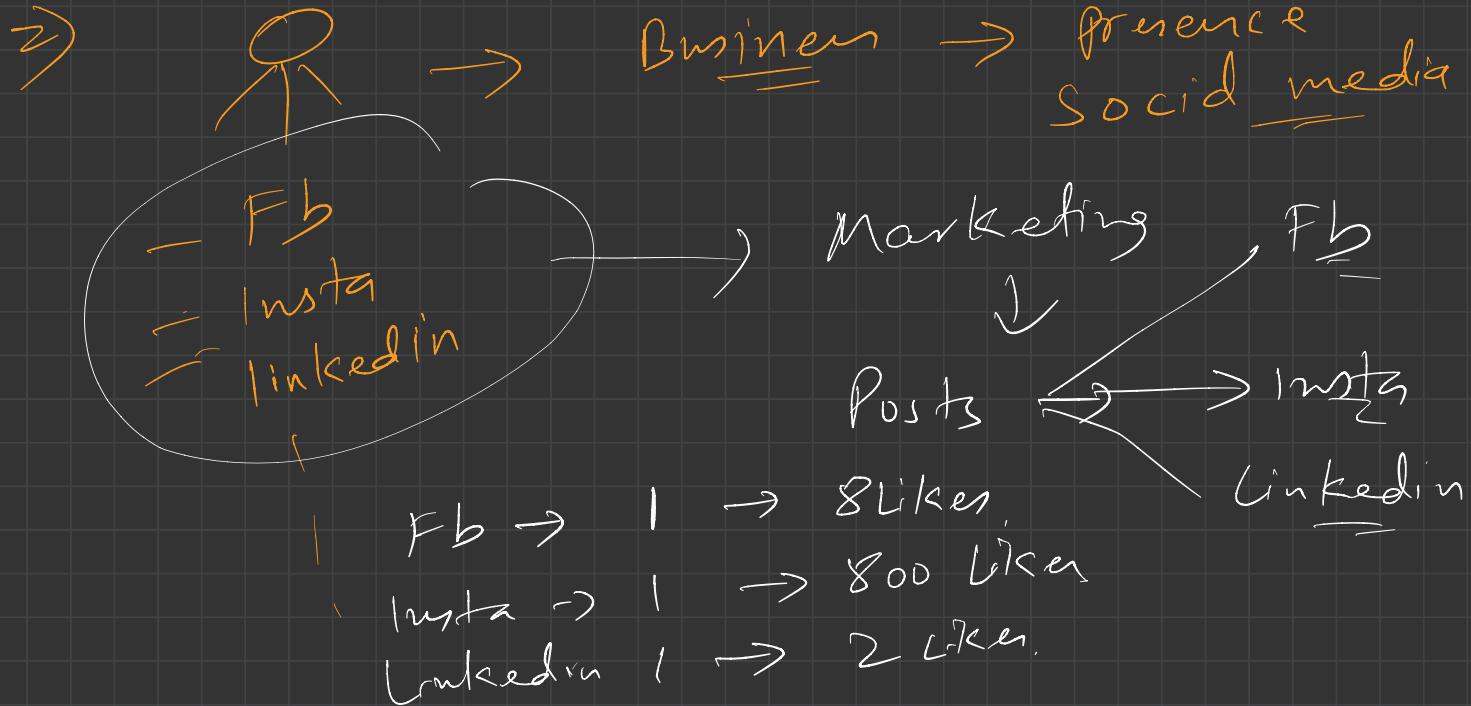
↓
Process / Interpret

↓
meaning (

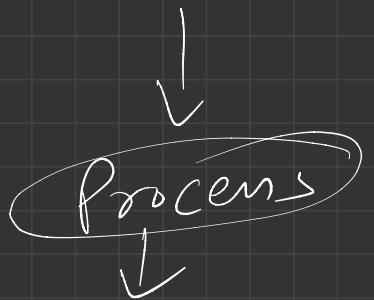
↓
Information

① Data

② Information



2) Data \rightarrow Likes Count \rightarrow 4 bytes integer



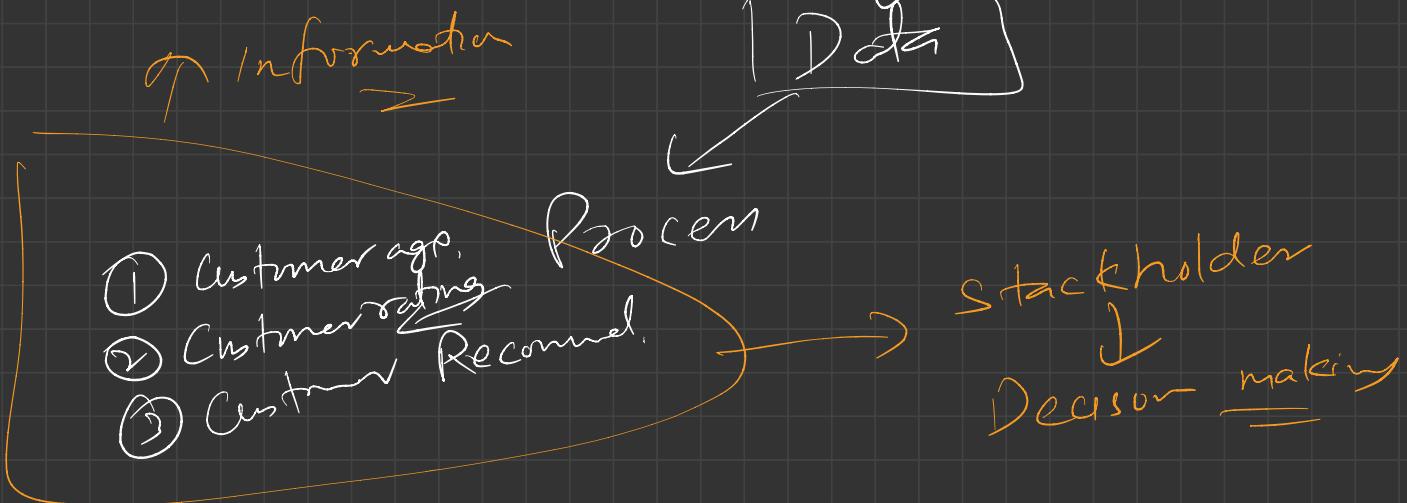
Conclusion \rightarrow Insta \rightarrow max. likes /

Decision making

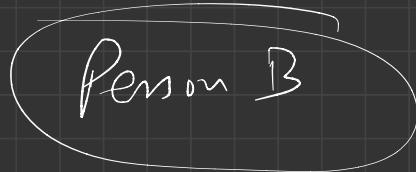
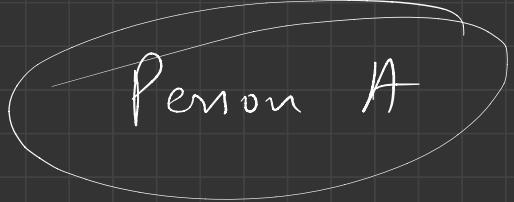
I should focus on Insta.

→ Amazon → Product → feed back

→ Feed back → tent → (by yes)



→ Data is the new oil.



→ 10 Dishes

— Feedback X

→ Data X

10 Dishes

FB ✓

Data collect ✓

→ Database →

→ DBMS →

School students

①

②

③

④

2001

2002

① DB → (store)

② set of programs
→ access

→ add

→ update

→ Delete

addition method

→ File system ↗

① earlier ↗

→ Bank → Saving account

- debit / credit
- new a/c
- Balance find.
- gen. monthly statement

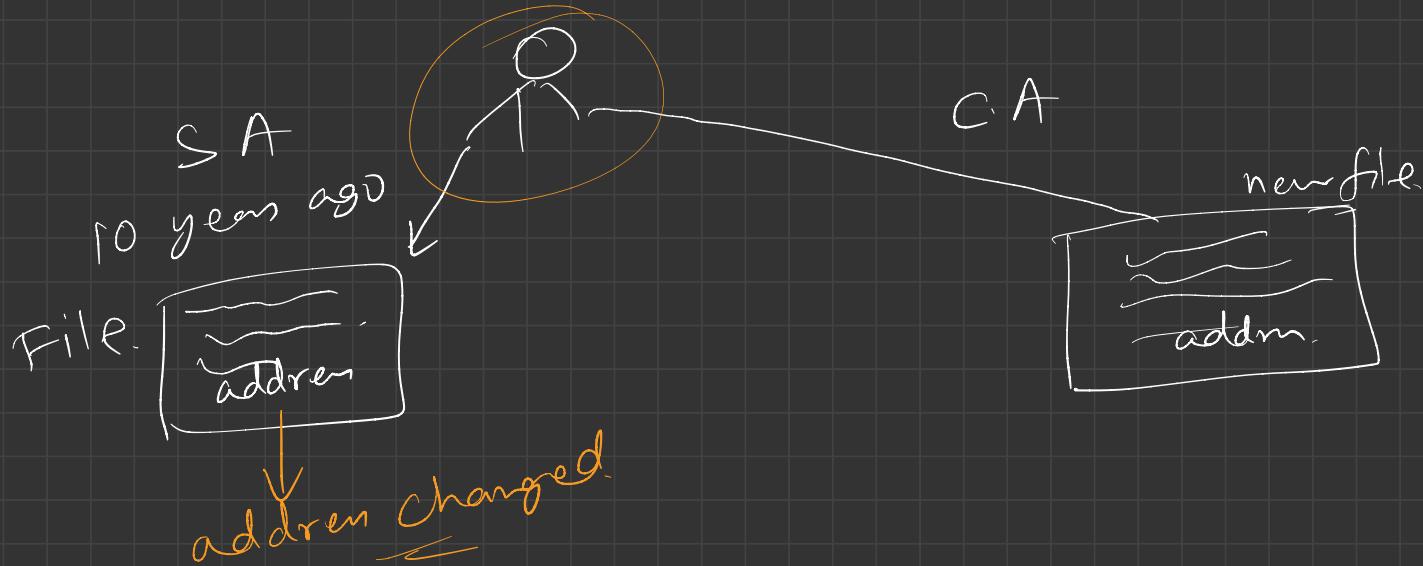
→ Bank → after 10 years

→ Current a/c

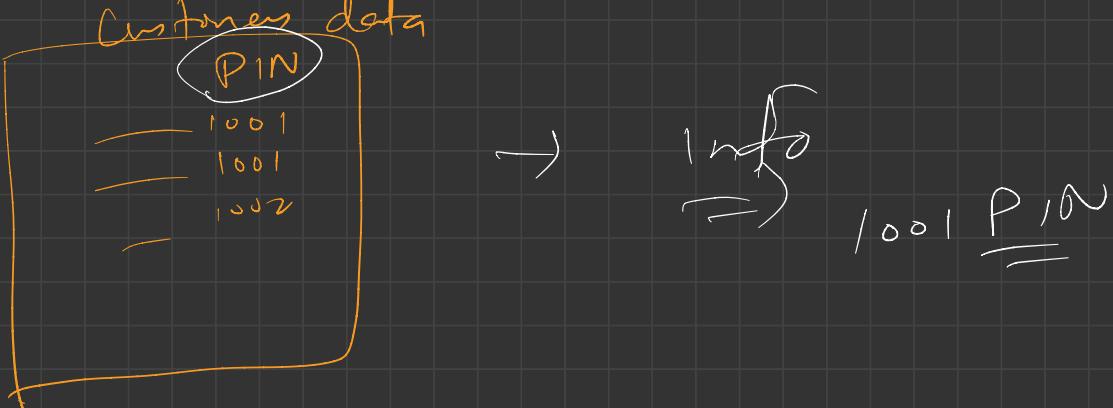
— Interest ↗

①

Data Redundancy & inconsistency



(D) Difficulty in Data access





Lec-2

* Abs**t**raction → DBMS
① → car driving

② Tally - Business soft

⇒ DBMS

z

Main 1 ⇒

Amazon

Customer

- Name

- Add

- Ph no.

- Billing | delivery

- Age

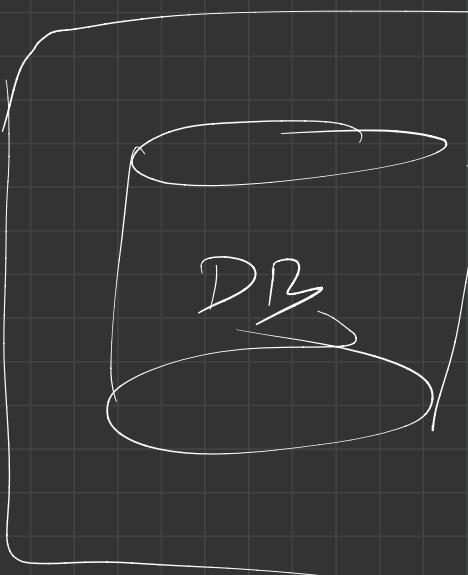
- CC

- DL

- VP

- Product bought

-



PBMS

Logistics

3

Customer
service
center

Logistics → view schema.

[Name | add.] ph no

Custom
service.
dept

→ [Name | address | ph no | Product
bought]

* Conceptual Level

Physical Level

Name, phno., addres, Batchn,
Name, ph no. addm, Bch
;

file.

Course →

Course ID | Name | Duration

Logical Level

Student

Name
Ph no.
address.
Batch no.

Student

	Name	Ph no.	addr	Batch
①	Xyz	8	m	2019
②	Lakshy	a	m	m
3				Course ID

* Instance of DB

Student
⇒

	Name	ph	address	Batch	Stn-ID
①	XYZ	XX	XX	XX	01
②	ABC	XX	XX	XX	02
③	~	~	~	~	03

12:00 AM → DB ?? 2 rows, 2 data points

Next 12:00 AM → 3 students.

DB Schema (logical schema)

① attributes of table.

② consistency constraints. (Primary key)

③ Relationships.

* Logical Level

* Data Model.

* DBMS → we → Language.

→ Schema define → Student → Stu-ID, Name,
add, ph. --



DDL

Insert, (Delete), update,
Retrieval.

Stu-ID	Name	add,	Ph
1			
2			
3			

* Data Mani. Langnge

DDL \rightarrow Create table students (stu-ID int,
Name varchar(50),
add varchar(90),
ph. int
);

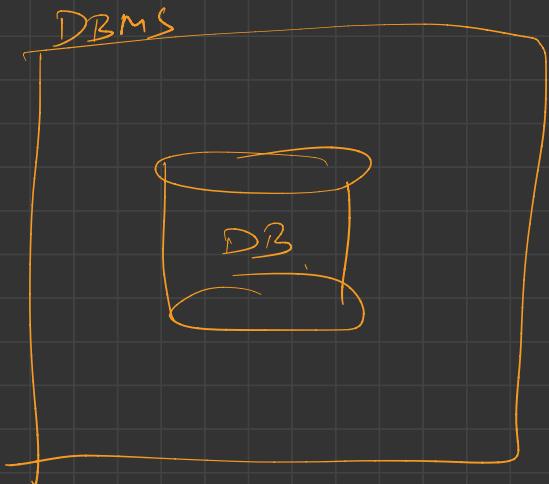
DML \rightarrow select * from students;

* Now app access DB?

App → JavaScript, C/C++

function lastOrderDetails{

JDBC | ODBC
↓ ↓
Java C/C++



}

Interface



CandidateDAOImpl.java

Source master e9b8380 Full commit

```
8 import org.slf4j.Logger;
9 import org.slf4j.LoggerFactory;
10 import org.springframework.dao.DataAccessException;
11 import org.springframework.dao.DataIntegrityViolationException;
12 import org.springframework.dao.IncorrectResultSizeDataAccessException;
13 import org.springframework.jdbc.core.BeanPropertyRowMapper;
14 import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
15 import org.springframework.jdbc.core.namedparam.SqlParameterSource;
16 import org.springframework.jdbc.support.GeneratedKeyHolder;
17 import org.springframework.jdbc.support.KeyHolder;
18
19 import java.util.List;
20
21 public class CandidateDAOImpl extends AbstractDAO implements CandidateDAO {
22     private static final Logger logger = LoggerFactory.getLogger(CandidateDAOImpl.class);
23
24     private static final String SQL_TALENT_PARTNER_ANALYTICS = "SELECT TALENT_PARTNER, count(*) as total, sum(case when status = 'SELECTED' then 1 else 0 end)
25     public Integer add(CandidateQO candidateQO) {
26         try {
27             MapSqlParameterSource namedParameters = new MapSqlParameterSource();
28             namedParameters.addValue("name", candidateQO.getName());
29             namedParameters.addValue("phone", candidateQO.getPhone());
30             namedParameters.addValue("emailId", candidateQO.getEmailId());
31             namedParameters.addValue("driveId", candidateQO.getDriveId());
32             namedParameters.addValue("resume", candidateQO.getResumePath());
33             namedParameters.addValue("status", candidateQO.getStatus());
34             namedParameters.addValue("createduserId", candidateQO.getCreatedUserId());
35             KeyHolder keyHolder = new GeneratedKeyHolder();
36             int affectedRowCount = this.namedParameterJdbcTemplate.update(
37                 "INSERT INTO CANDIDATE "
38                     + "(NAME, PHONE, EMAIL_ID, RESUME_PATH, DRIVE_ID, STATUS, CREATED_USER_ID) " + "VALUES "
39                     + "(:name, :phone, :emailId, :resume, :driveId, :status, :createduserId)",
40                     namedParameters, keyHolder);
41             if (affectedRowCount == 1) {
42                 return keyHolder.getKey().intValue();
43             }
44         } catch (Exception e) {
45             logger.error("Error occurred while adding candidate: " + e.getMessage());
46             throw e;
47         }
48     }
49 }
```

Pull requests Check out ...

Last updated
2020-07-26File type
Java

Lines 144 Size 7.03 KB

0 builds

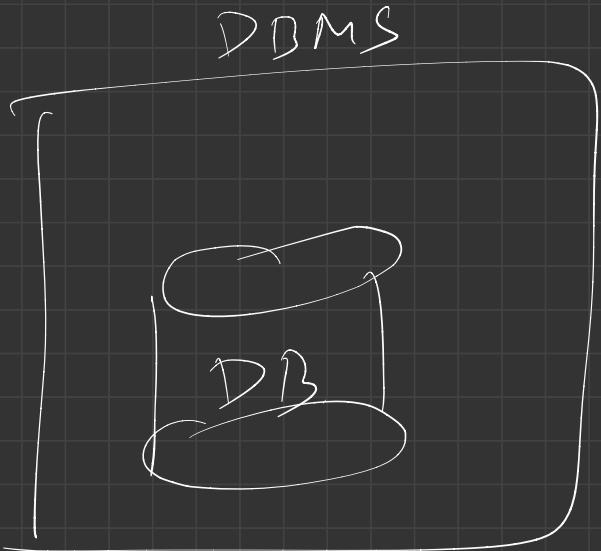


It looks like you haven't configured a build tool yet. You can use Bitbucket Pipelines to build, test and deploy your code.

Your existing plan already includes build minutes.

Set up a pipeline

Give feedback



Control → - data ??
— Data access API's

A simple stick figure stands next to handwritten text. The text reads "Control → - data ??" followed by a horizontal line and "- Data access API's". The entire text is underlined.

* DBMS Application architecture

client machine
=

① T1 archi ⇒

e.g. (1) sql. learn.

end-user → PC

DBMS Server → PC

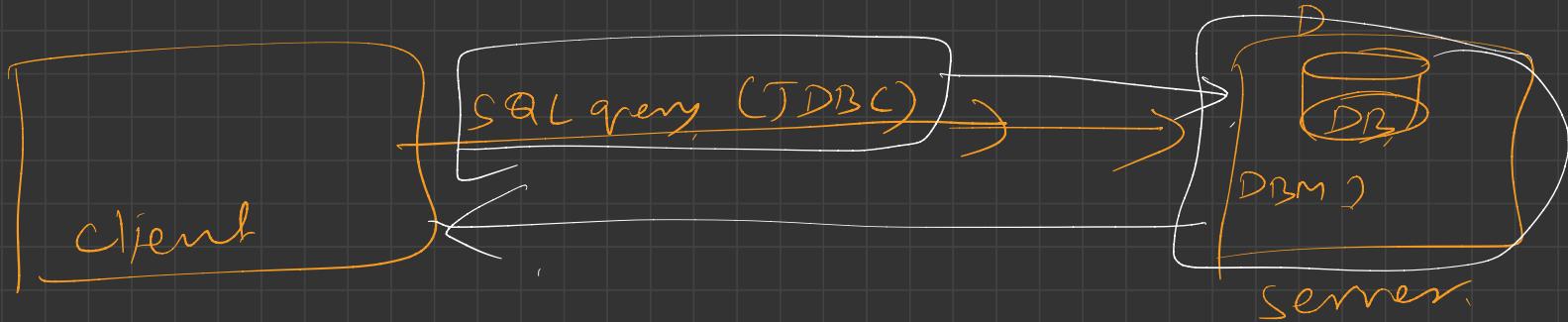
↓
 $\text{DB} \rightarrow \text{PC}$ (files)

Server machine
=



②

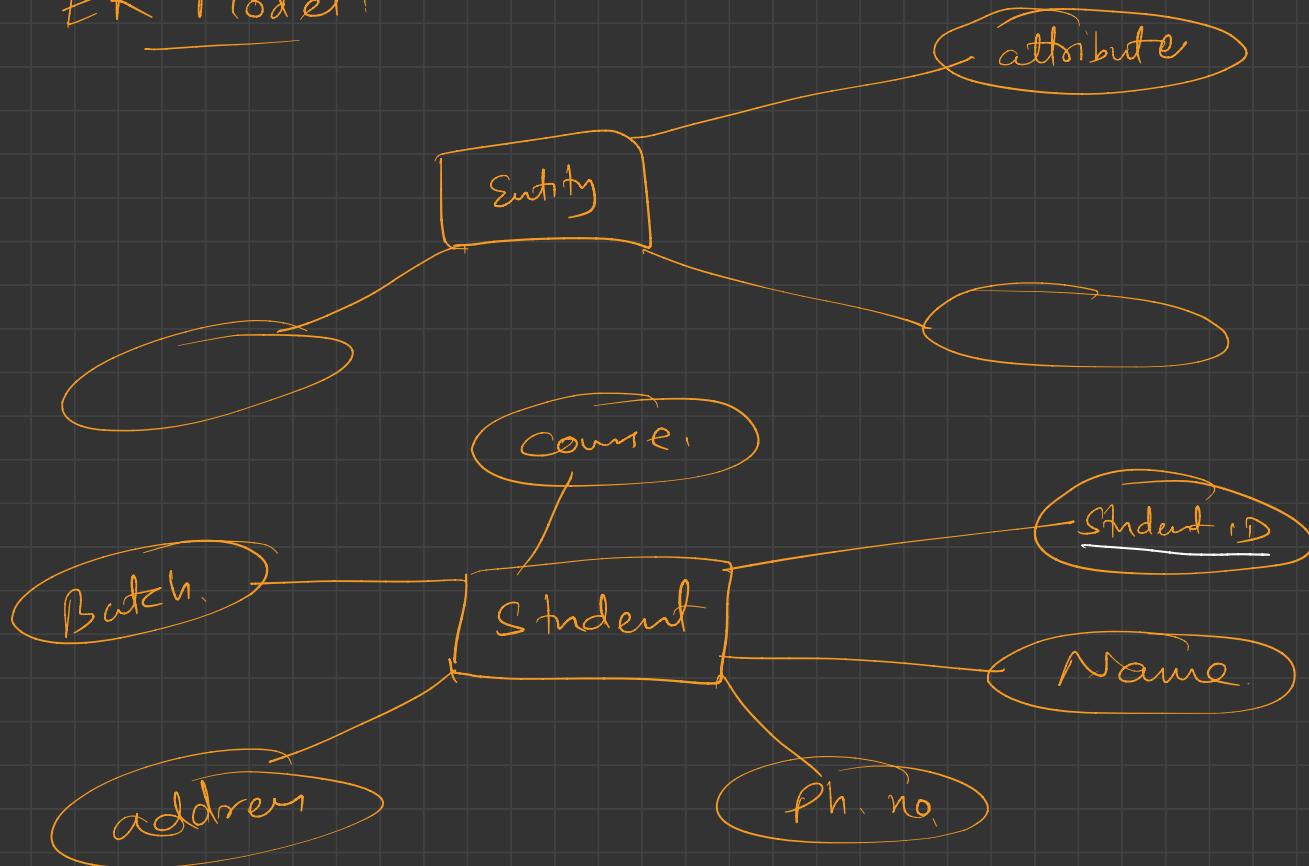
T2 >

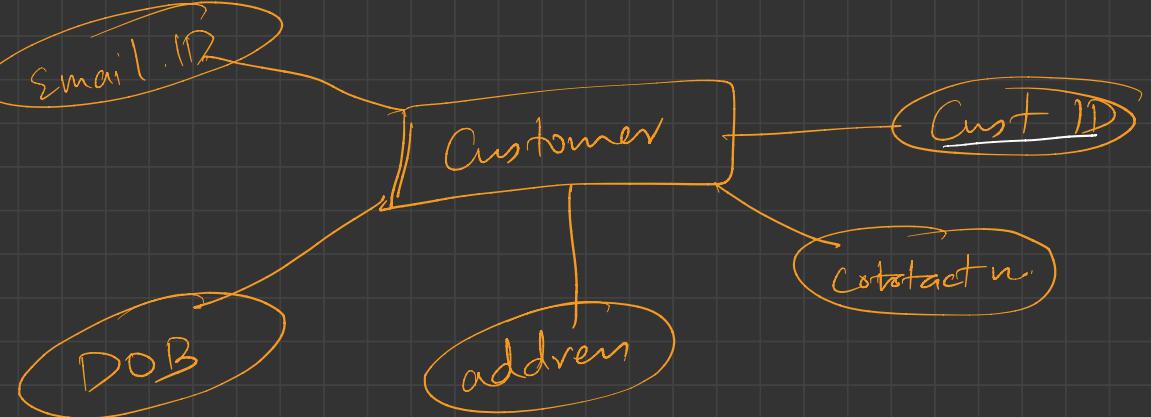




Lec - 3

ER Model:





→ Unique attribute → Primary Key

Entity Set →



Attributes

Relationships :-

Banking.

Customer , Loan , Account.

① Customer borrow Loan.

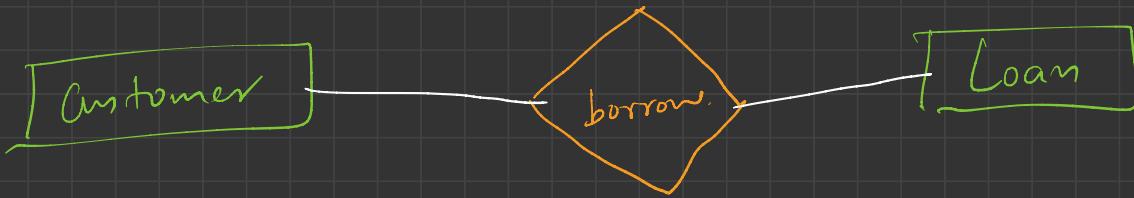
② Customer deposit Account.

→ Customer places Order.

→ Professor takes Course.

→ Student enroll Course.

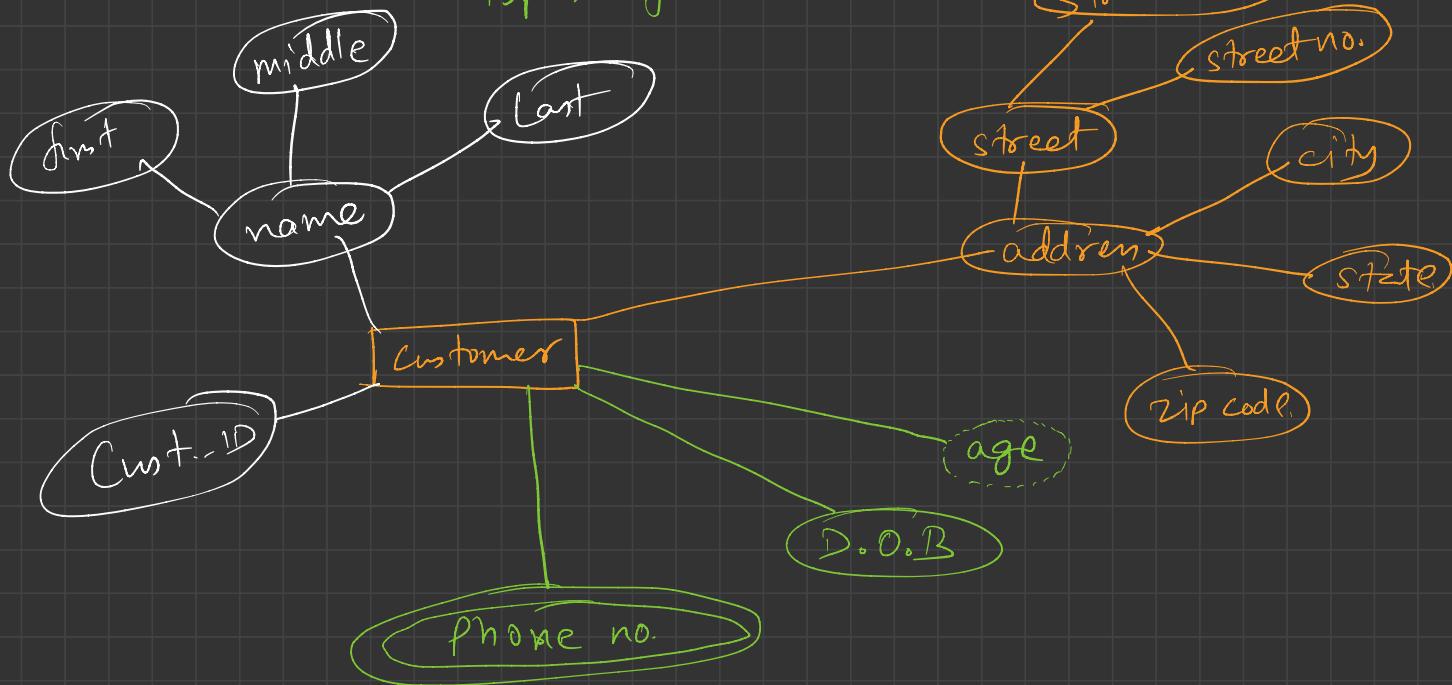
→ Citizen has Vehicle.



* Attributes



Types of attributes



* NULL value

Relationships 1 — association among 2 or more entities



Strong Relationship

Weak Relationship

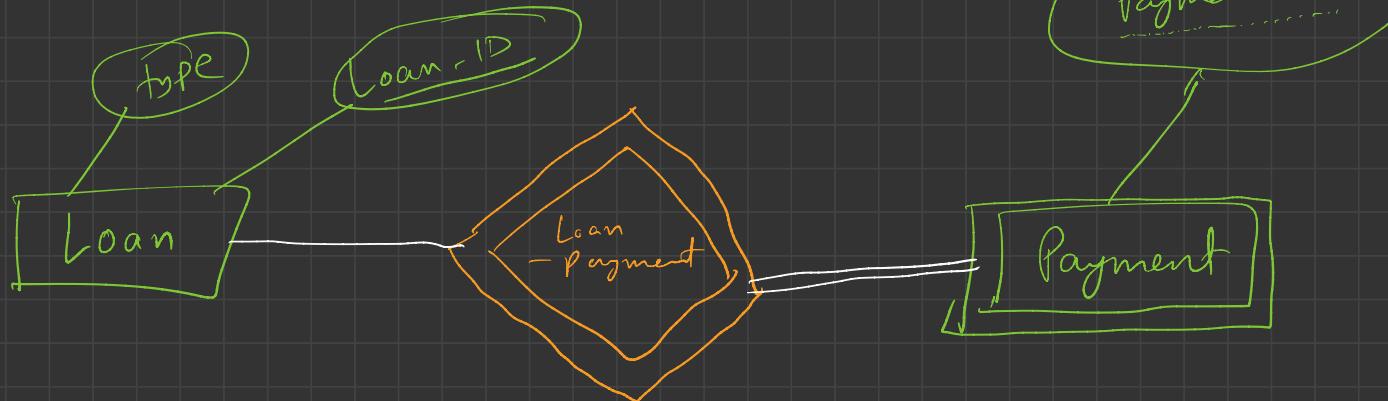
University system



strong entity

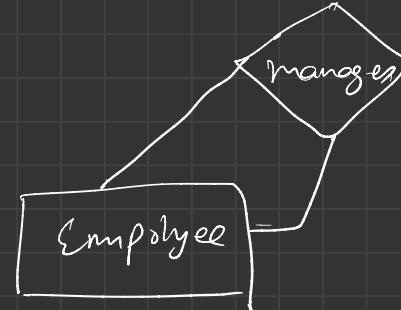
Weak Entity

weak Entity & weak relation



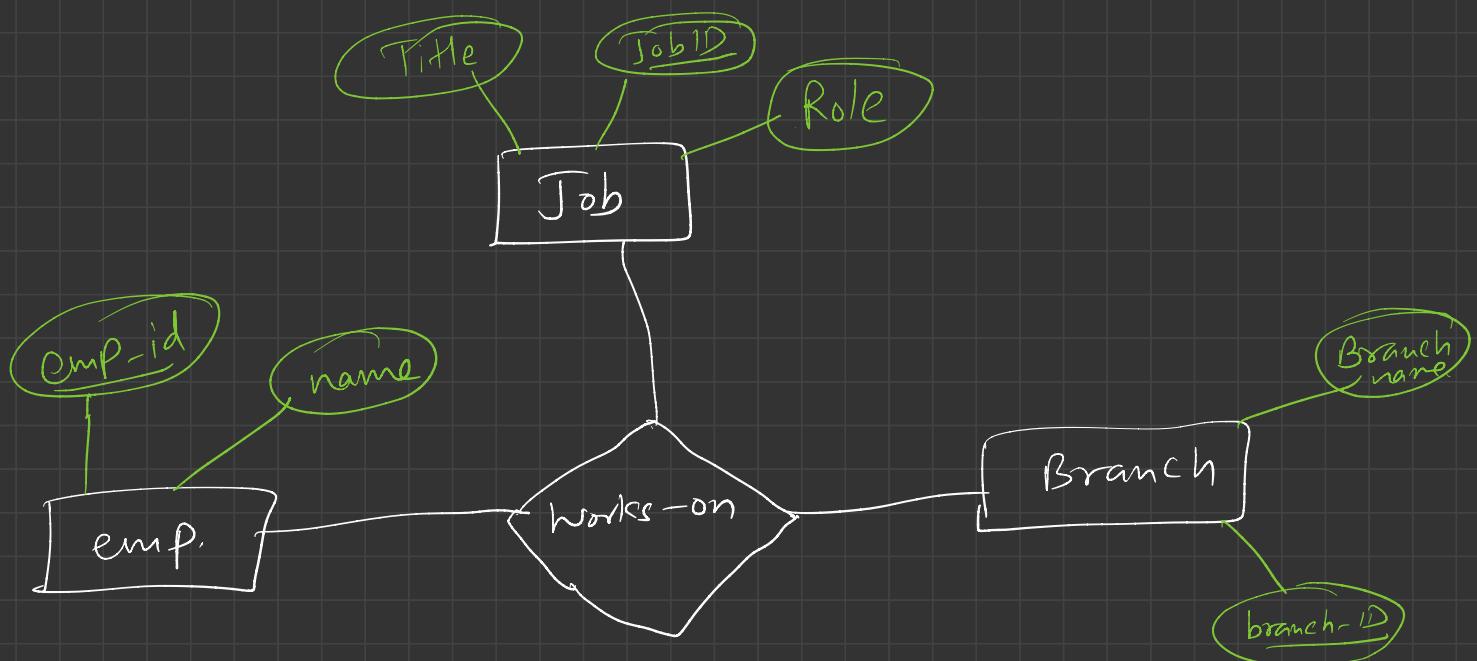
Degree of relⁿ ↗

① Unary ↗



② Binary relⁿ ↗

③ Ternary relⁿ ↗

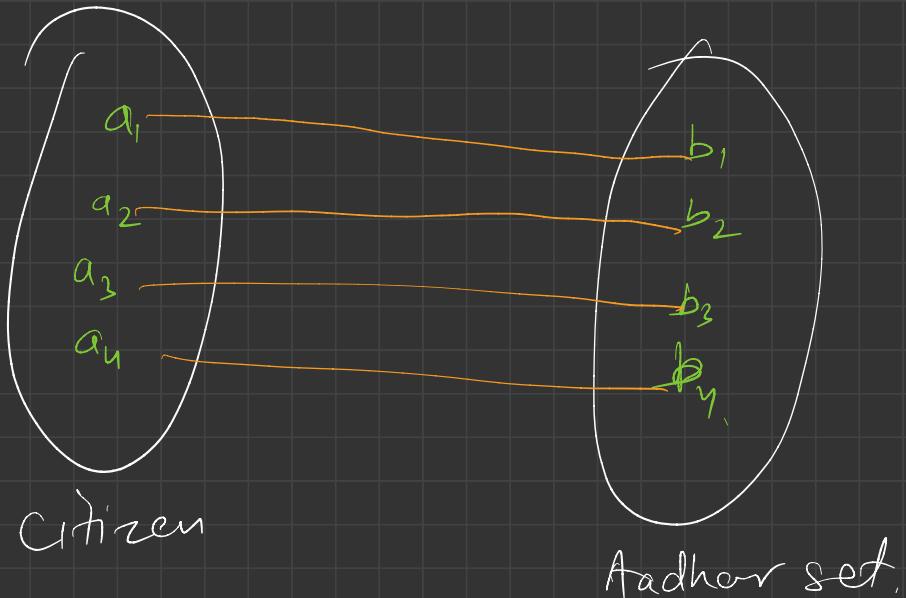


Ternary Relⁿ

→ Relationship constraints

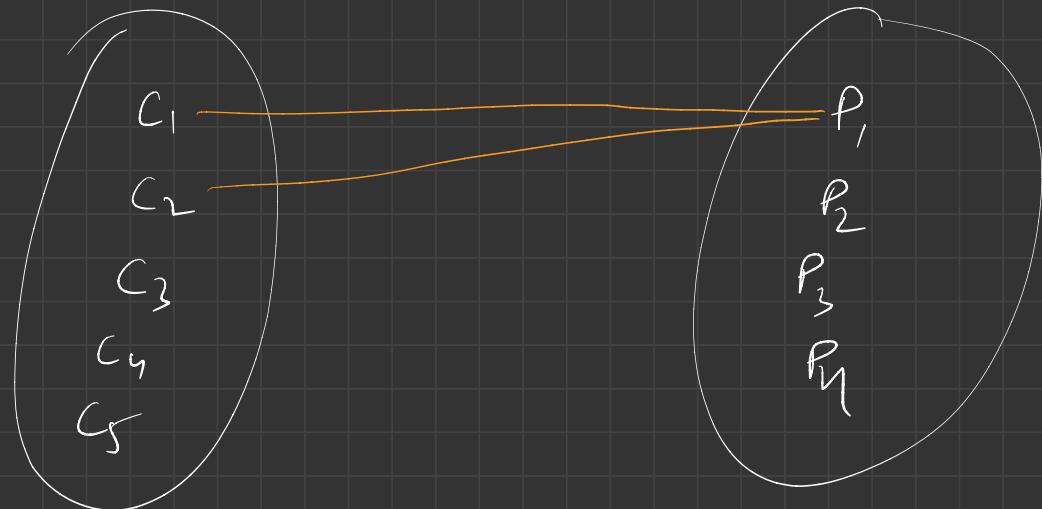
① Mapping cardinality

① 1 : 1



(3)

N : 1

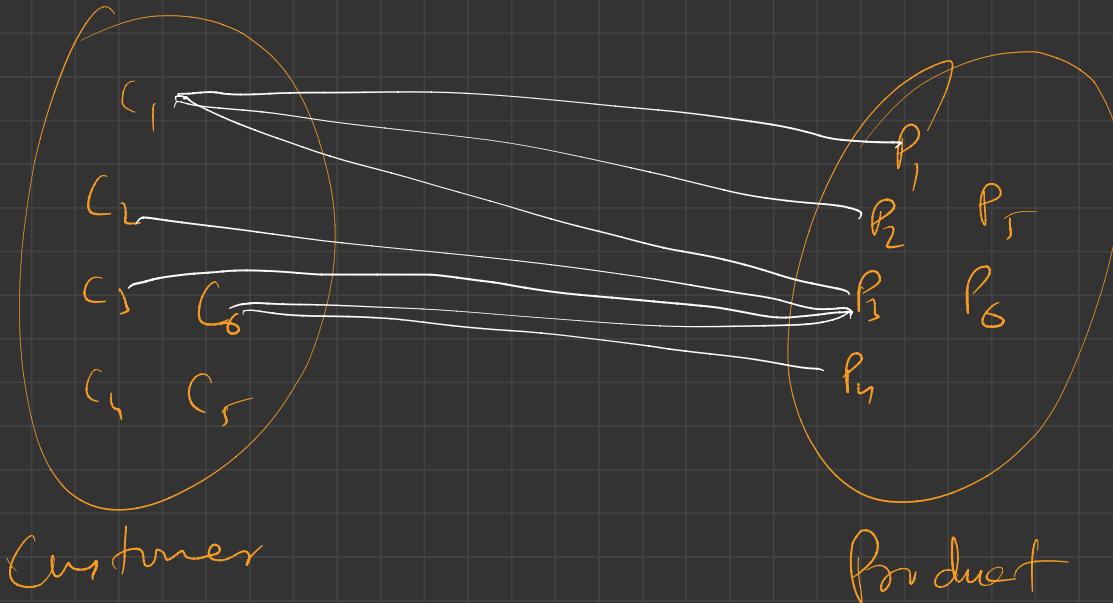


Course

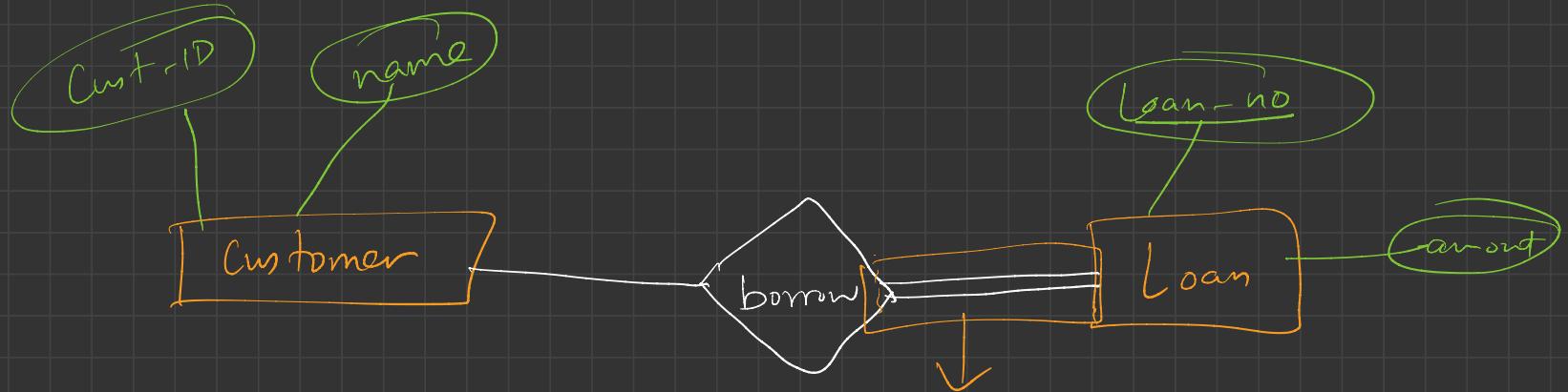
Professor



N : M



* Participation Constraint



total participation.

Cust → Loan → Partial Participation.

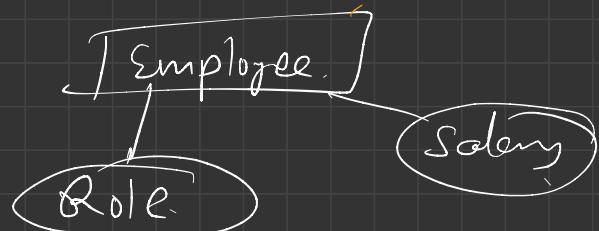
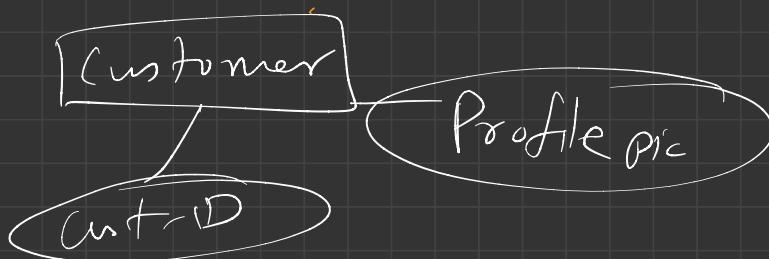
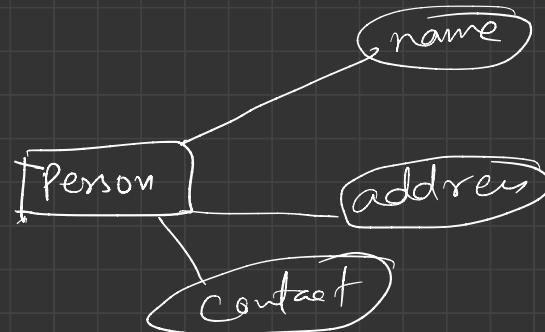
Loan → customer → Total "

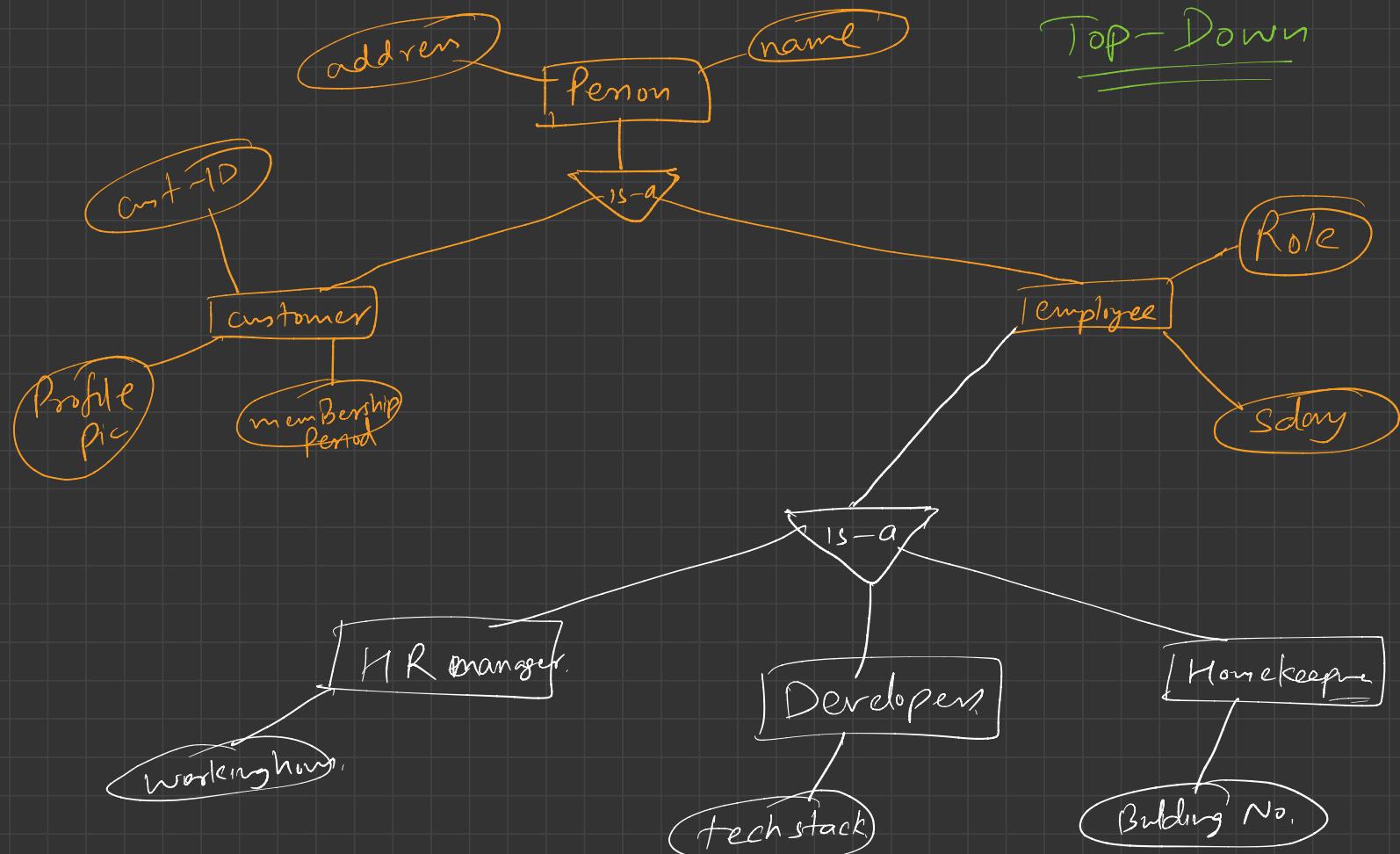


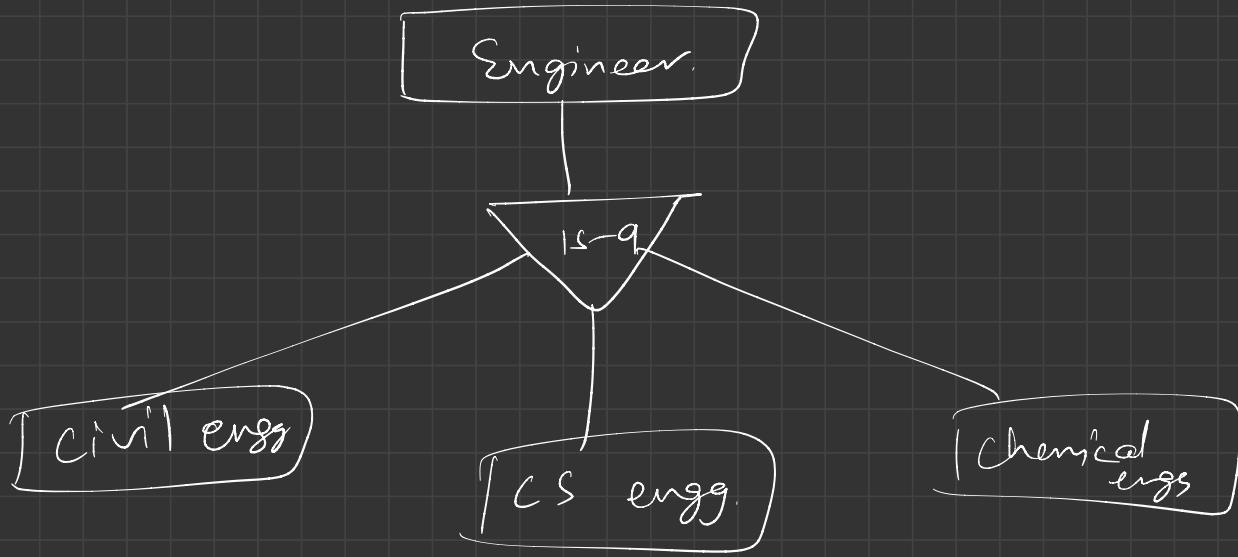
Lec-4

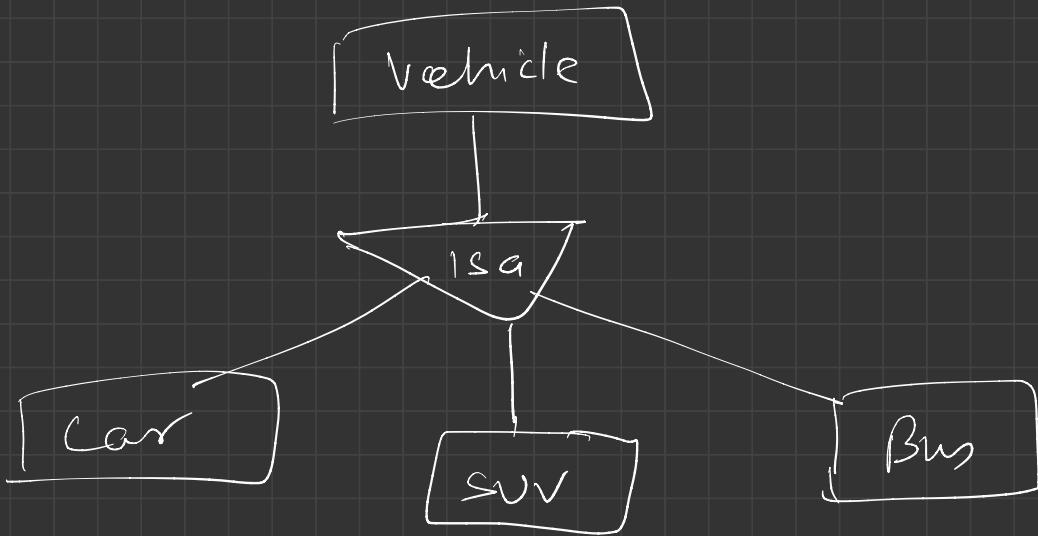
Extended ER Features

① Specialization

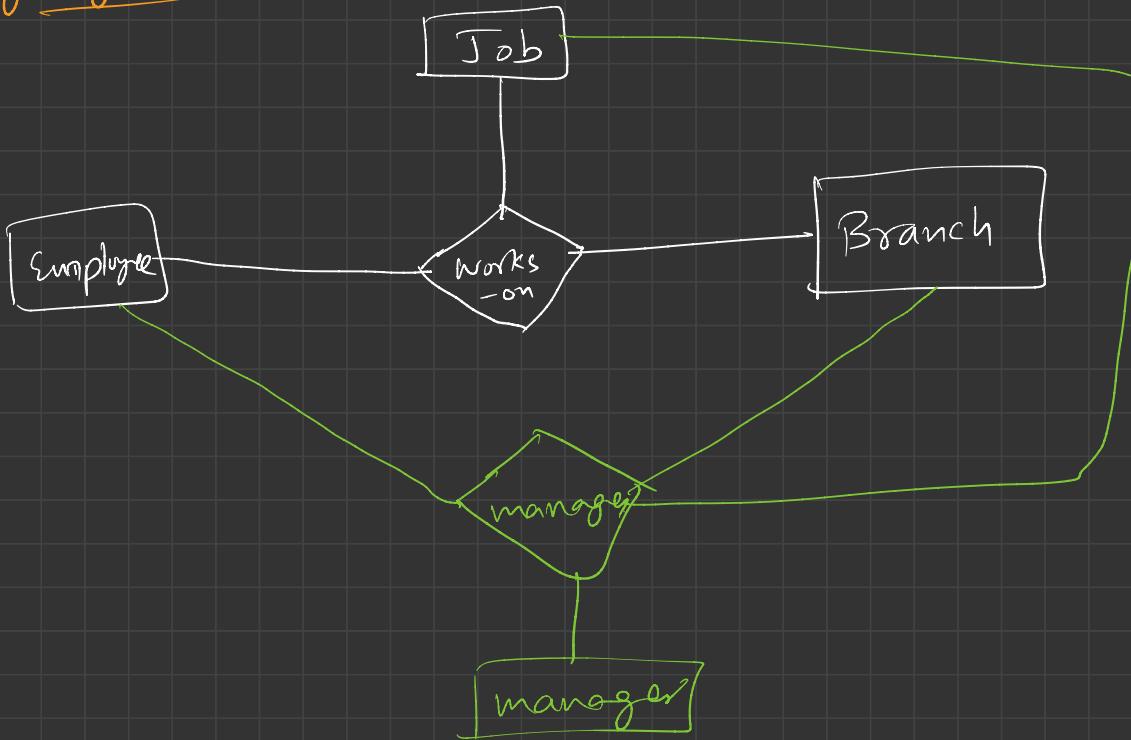


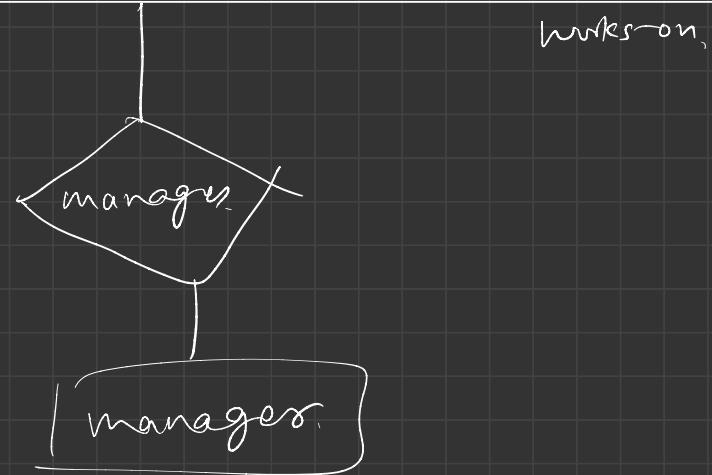
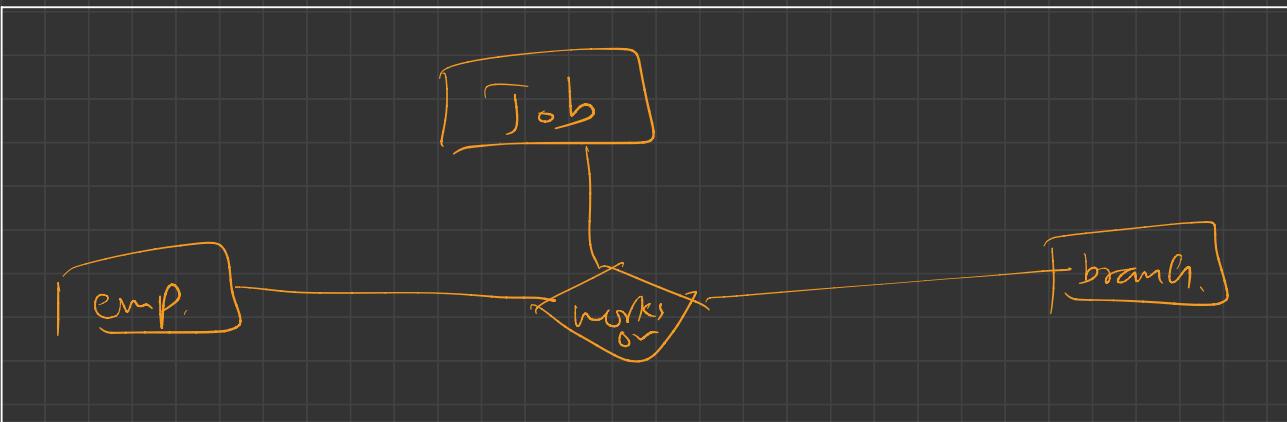




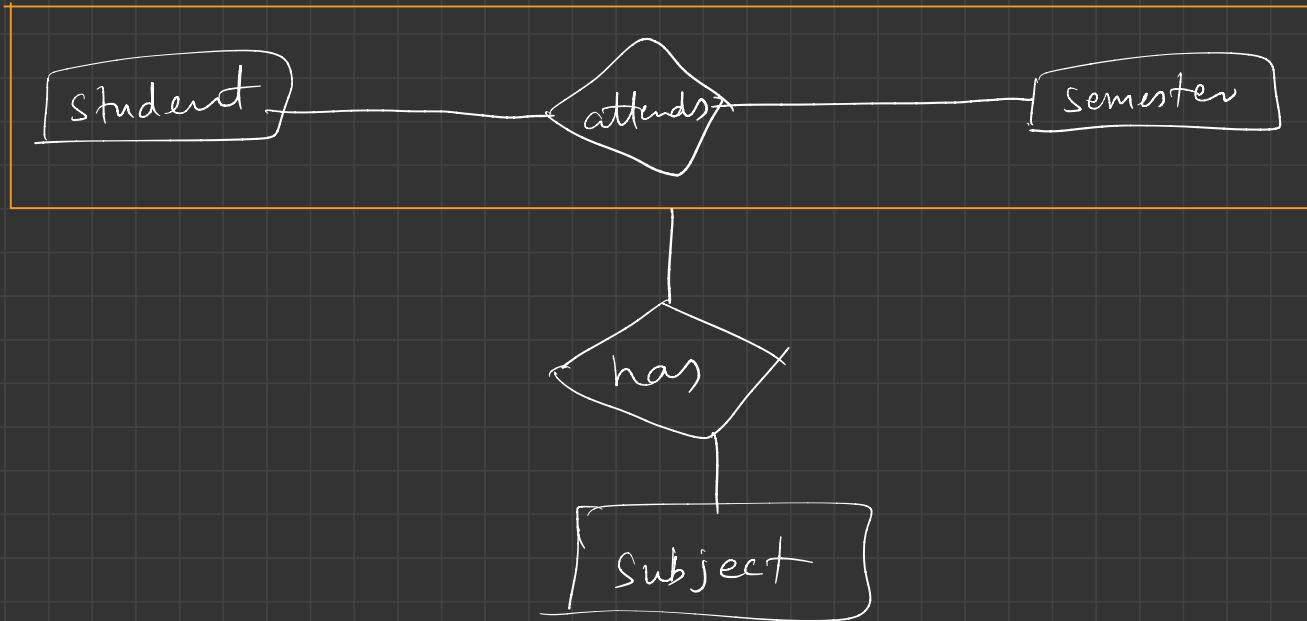


* Aggregation





eg.





Lec-5

* Steps to make ER diagram -

- ① identify Entity sets.
- ② identify attributes & their types,
- ③ " Rel'n & constraints
 - Mapping
 - Participation

* ER - Model of Banking System

① Banking system — Branches, (name)

② Bank → customers.

③ Customers — accounts, , & take loan.

④ Customer associated with some banker.

⑤ Bank has employees.

⑥ Accounts —
 |
 Saving a/c
 |
 current a/c.

⑦ Loan originated by branch

|
 Loan ≥ 1 customers.
→ payment schedules

① Entity sets

① Branch ② Customer ③ Employee

④ Saving a/c ⑤ Current a/c

⑥ Loan ⑦ Payment (loan) (weak entity)

② Attributes:-

① branch → name, city, assets, liabilities

② Customer → cust-id, name, address, contact no.
 DOB, age,
 ↓
 derived.
 ↓
 Composite.
 ↓
 multivalued.

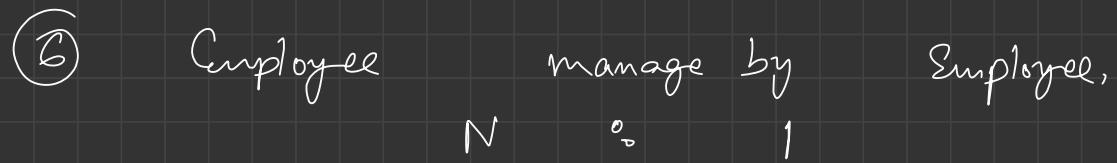
- ③ Employee → emp-id, name, contact no., dependent name,
years of service, start-date
↓
derived attr. ↓
single valued. multivalued.
- ④ Saving account → acc-number, balance, interest-rate
, daily withdrawal limit.
- ⑤ Current a/c → acc-number, balance, per transaction charges,
overdraft-amount.
- ⑥ Generalized Entity "Account" → acc-no., balance
- ⑦ Loan → loan-number, amount
- ⑧ Weak Entity payment → Payment no., date, amount.

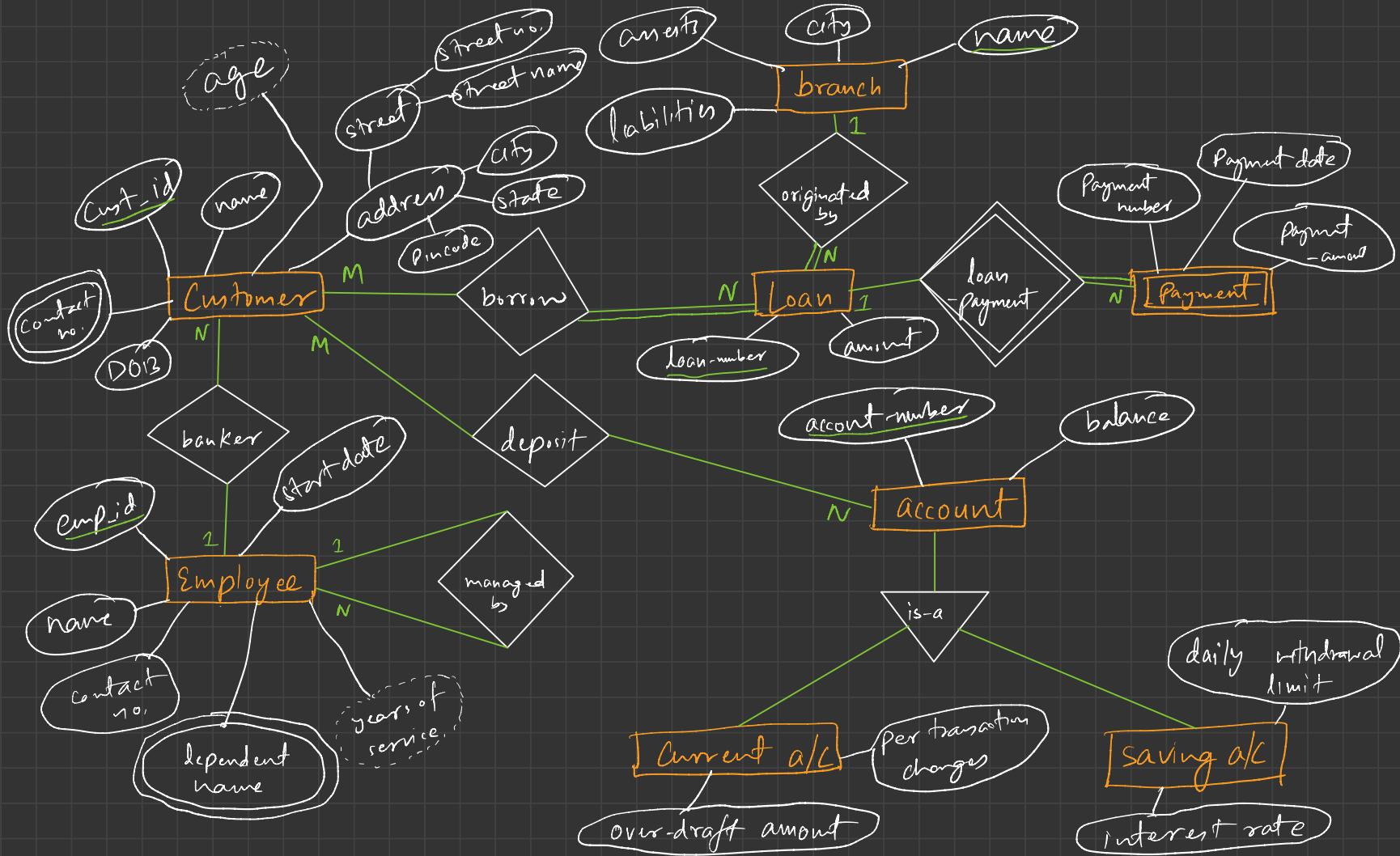
③ Relⁿ & Constraints

① Customer borrow loans,
M : N
==

② Loan originated by branch,
N : 1
==

③ Loan loan-payment Payment,
1 : N
==





N.W

- ① Online delivery system.
- ② University



Lec- 6

Facebook - DB formulate using ER model

* ER diagram

① Features. & we can.

① profile → user-profiles → friends.

② user can post

③ Post → Contains → text-content, images, videos.

④ Post → like, comment.

① Identify entity sets.

① user-profile

② user-post

③ post-comment

④ post-like

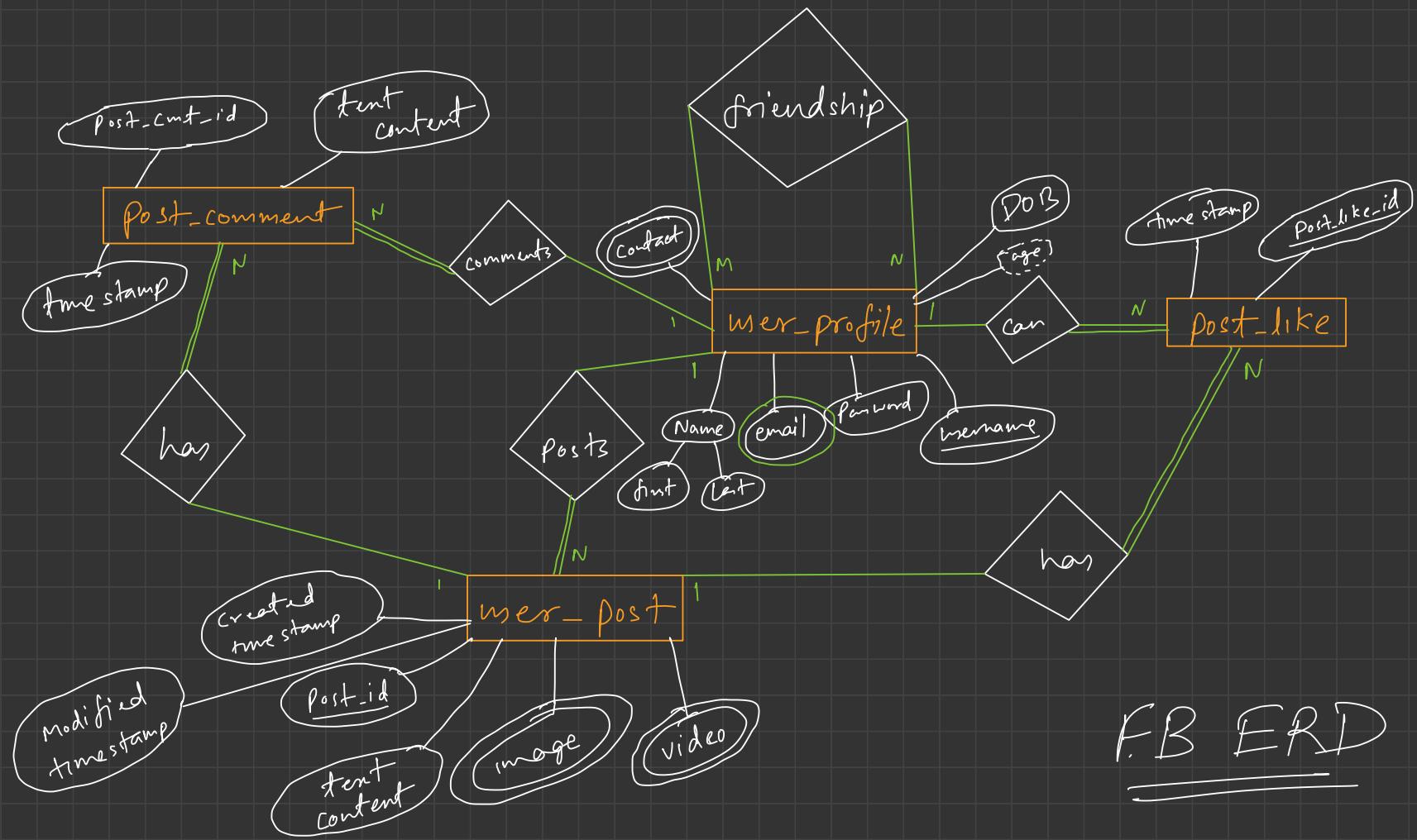
⑤ Attributes + types

① user-profile → Name, username, email, password, contactno.
Composite, DOB, age. multivalued, multivalued,
derived.

② user-post → post-id, textcontent, image, video, multivalued, multivalued,
created timestamp, modified timestamp.

(3) Post-Comment → post-comment_id, textContent, timestamp

(4) put-like → putlike_id, timestamp





Lec - 7

Relational Model :-

⇒ Tables →

Customer →
— Cust-ID
— Name
— address
— Contact

Table → Relation →

	Cust-ID	Name	addrn	Contact
①	1	Lakshay	---	888--
②	2	Raj	--~	--~

⇒ Degree of table → No. of attributes.

Cardinality → Total no. of tuple.

⇒ DB design ↗

① ER Model → ER diagram.

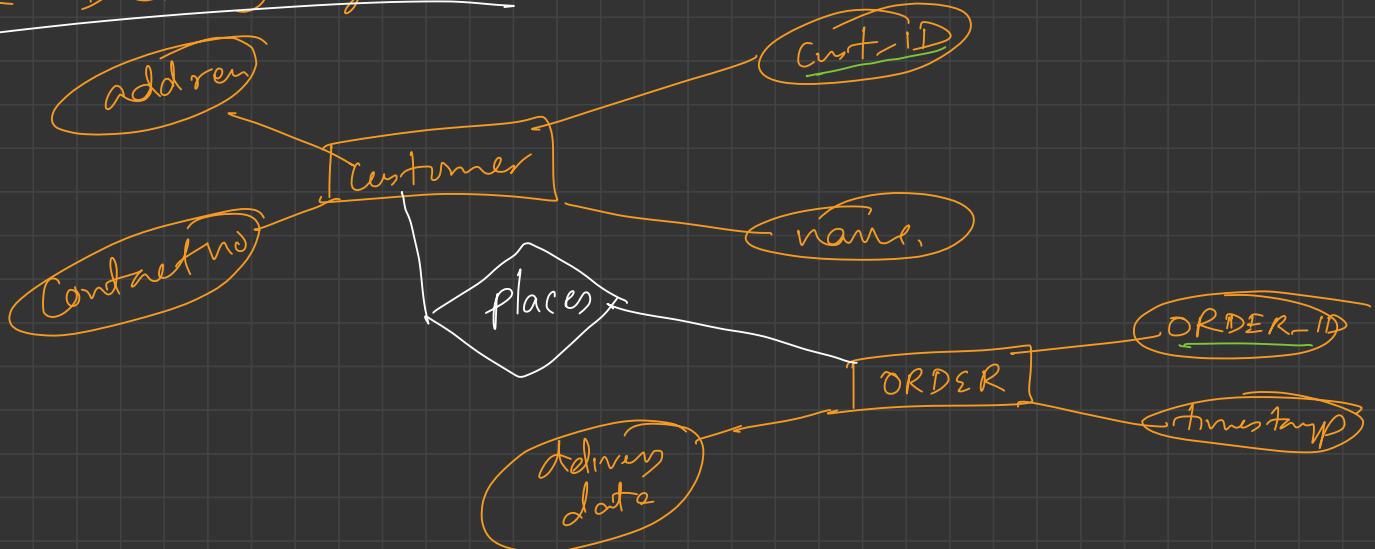
② Relational Model ↗

③ RDBMS → Software DBMS

Software implementation of R.M

MySQL, MS access, Oracle etc.

Online Delivery system



① Customer (CUST-ID, name, address, contactno)

② ORDER (order-ID, timestamp, delivery-date)

→ Key → Foreign Key

2) Keys → Primary Key → each data point
uniquely identifies

Cust-ID	Name	Contact	Address	Email

- ① Super key → {Name, Contact}, {Name, Email},
{Cust-ID, Email}, {Cust-ID}, {Null},
{Cust-ID, Name, Contact, Email}
- ↙ {Raj, 888}
{Raj, 811}

② C.K. \rightarrow {Contact}, {Cust-ID}, {Cust-ID, contact},
{Cust-ID, email}.

③ P.K. \rightarrow {Cust-ID} \rightarrow P.K.

④ A.K. \rightarrow {C.K.} - P.K. \rightarrow

④ Foreign Key L -

Customer (cust-ID, name, address, contact no)

Order (order-ID, timestamp, delivery date, cust-ID) \rightarrow F.K

Customer Referenced Rel'n / Parent table.

Cust-ID	Name	addrm	Contact
1	Kaz	- ~	--
2	To	--	--
3	dada	--	--
4	--	- --	--

ORDER Child table.

Order-ID	Timestamp	delivery date	Cust-ID
21	- --	- -	1
22	- -	- -	2
23	- -	- . .	3

/ Referencing rel'n.

X Surrogate key →

Table of School A

<u>reg.no</u>	name
101	Ram
102	Monika
103	Tata.

Merged.
→

<u>Sur-no</u>	reg-no	name
1	101	Ram
2	102	Monika
3	103	Tata
4	AB101	Tommy
5	AB102	Nota.

table of school B

<u>reg-no</u>	name
AB101	Tammy
AB102	Nota.

CRUD

Rollno	Name	Contact.	Age.
1	A	---	
2	B	---	
3	C	- - -	
4	812	ABC.	

Create

* Referential constraint

Customer :-

↓
Parent reln
Referenced table.

Cust-ID	Name	add	Contact no.
1	Kaz	- -	- -
2	Jo	- -	- -
3	Dada	- -	- -

ORDER 1-

↳ Child Reln/
Referencing table

order-ID	Timestamp	delivery date	Cust-ID → F.K.
21	- - -	- -	1
22	- - -	- -	2
23	- - -	- -	Null,

① Insert constraint → value can't be inserted in child table if the value is not lying in parent table.

② Delete constraint → Value can't delete from parent table if the value is lying in child table.

⇒ ON Delete Cascade ↴

Can we delete value from parent table if the value is lying in the child table w/o violating referential constraint?

→ delete value from parent table → delete corresponding entry from child table too.

create table Order (order-ID int P.K, --, --, cust-ID int
referencing Customer
on delete cascade);

⇒ Can F.K have Null value ?

ON Delete Null ↴

delete value from parent table → put corresponding F.K value Null.

* Key Constraints -

- ① Not Null → By default a attribute / column can be Null.
→ enforce a column to not accept null.

```
Create Table Customer (
    ID int Not Null,
    Name varchar(50) Not Null,
    Age int,
);
```

② Unique constraint =>

↳ ensure all values in col. are diff.

- Both unique & P.K constraint provide uniqueness.
- you may have many unique constraints per table
But only one P.K constraints per table,

```
Create Table Customer (
    ID int Not Null,
    Name varchar(50) Not Null,
    Unique (ID)
);
```

③ Default constant →

↳ set default value of col.

```
create table customer (
    :
    prime_status int DEFAULT 0,
    :
    :
);
```

④

Check :- limit value Range (Domain)

```
(  
:
```

```
    CHECK (age >= 18)  
)
```

(8)

Primary key constraint

- Uniquely identify each tuple.
- P.K \neq Null.
- Relatum only 1 P.K.

(
|
{

PRIMARY KEY (ID)
;)

⑥ Foreign Key Constant
— Keeps rel'n b/w 2 table.

Create table Order (

PRIMARY KEY (order_ID)

FOREIGN KEY (Cust-ID) Referencing
Customer (Cust-ID)

) ;



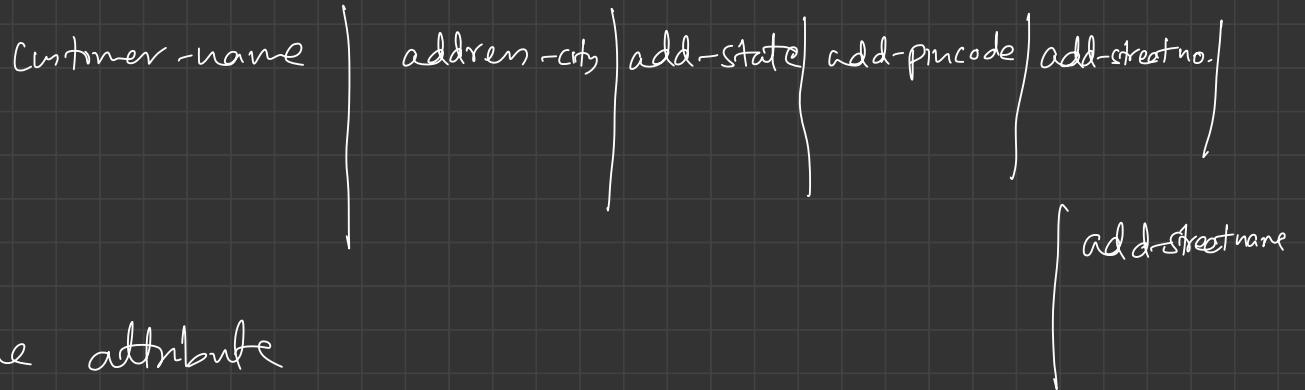
Lec - 8

* Transformation from ER model to Relational model.

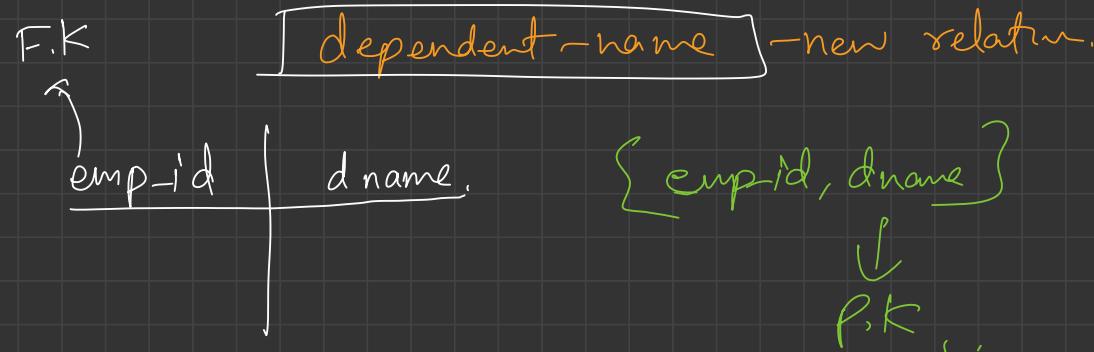
Loan → { Loan-number } amount

weak Entity Payment
P.K
⇒
Payment no. | Payment-date | Payment-amount
P.K
=

Customer Table



* Multi-value attribute



* Generalization

Method |
=====

* Aggregation →

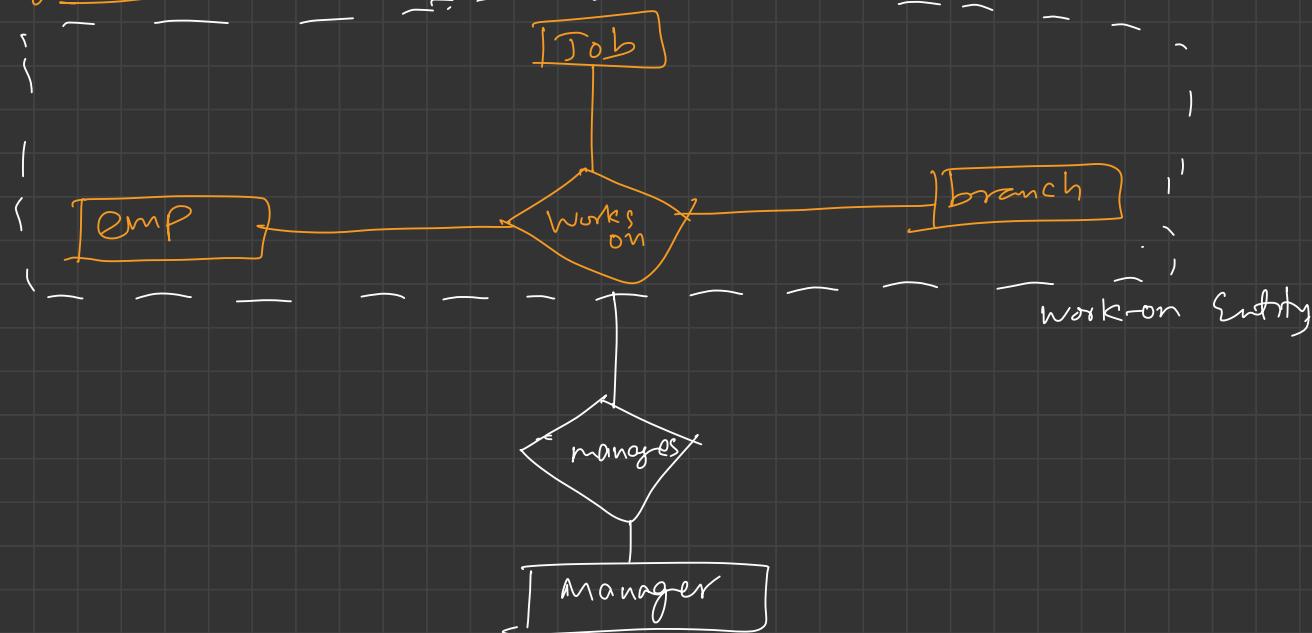
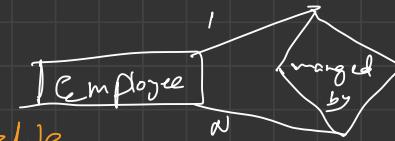


Table ` manages (mgr_id, emp_id, job_id, branch_id)

P.K.

* Unary Relationship :-

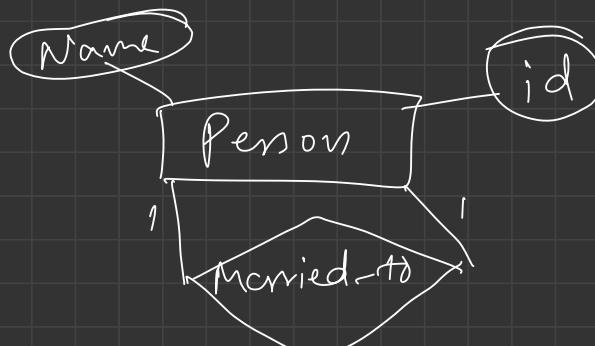
we will add another attribute in Employee table,
which will be F.K.



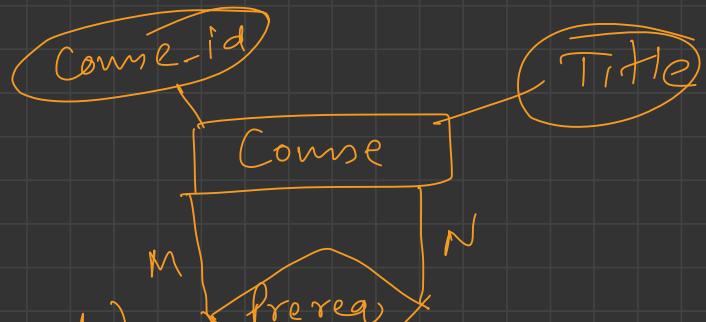
Emp_id	name	joining date	Emp_mgr_id
201	---	---	205
202	--	--	205
205	--	--	Null.

* 1 : 1 →

Person (id, Name, spouse_id)
{ F.K }



* $M : N \Rightarrow$



① Course(id , Title)

② Prereq(id , prereq , course_id)

\downarrow \downarrow
{Fk} {Fk}

P.K

* FB Relational Model

- (1) user-profile (username, name-first, name-last, password, DOB)
 - (2) user-profile-email (username {F.K}, email)
 - (3) user-profile-contact (username {F.K}, contact-number)
 - (4) friendship (profile-req {F.K}, profile-accept {F.K}) → Compound key
 - (5) post-like (post-like-id, timestamp, post-id {F.K}, username {F.K})
 - (6) user-post (post-id, created-timestamp, modified-timestamp, text-content, username {F.K})
 - (7) user-post-image (post-id {F.K}, image-url)
 - (8) user-post-video (post-id {F.K}, video-url)
 - (9) post-comment (post-comment-id, text-content, timestamp, post-id, username {F.K})



Lec-9

* SQL in One video

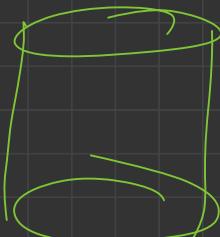
SQL \Rightarrow structured Query language

RDBMS \Rightarrow MySQL, Oracle, MS access etc.

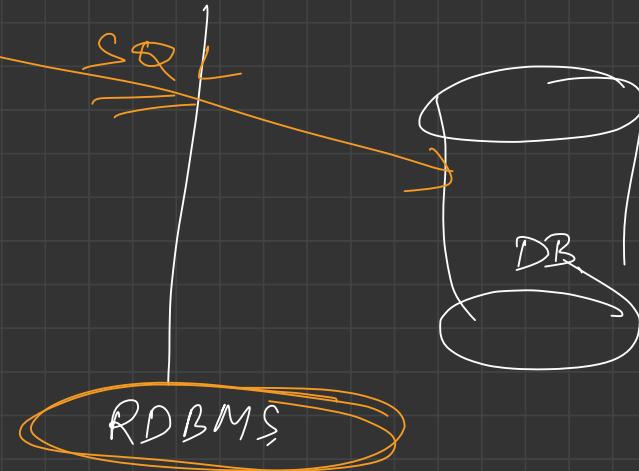
CRUD

SQL queries.

API RDBMS



CRUD → Create
Read
Update
Delete.



RDBMS → ① MySQL → Opensource RDBMS
② MSSQL server → Meta, Adobe etc.
③ Oracle
④ IBM,

SQL

- ① Query language.
- ② Way to access data

MySQL

- ① MySQL itself a RDBMS
- ② CRUD done on it using SQL.

⇒ Install → ① MySQL Community Server
② MySQL Workbench

2) Workbench → DB Create
Table

* SQL Data types

```
Create table student(  
    id INT PRIMARY KEY,  
    name Varchar(255)
```

variable datatype

)'

VARCHAR(255) →

[R | A | M]

CHAR(255) →

[R | A | M]

①

255

Aned-
dotum

255
f

SIGNED, unsigned

TINYINT $\Rightarrow (-128 \text{ to } 127)$

UNSIGNED TINYINT $\Rightarrow (0 \rightarrow 255)$

Create TABLE table_name (

Col1 INT,

Col2 INT UNSIGNED

)

→ Advanced DT ↴

① JSON → JS Object Notation

Curly - {
 ;
 col1 JSON,
 ;
 }

SQL Types of Commands

① DQL

\Rightarrow DQL \Rightarrow ① DB
② table

user defined table

Student

Dual table

\hookrightarrow without using FROM

SELECT

wildcards

\rightsquigarrow $'\text{a}^*\text{pa}^*'$ \rightsquigarrow $'\underline{abc}\underline{padb}'$
 $'\underline{pab}\underline{c}'$
 $'ad\text{pa}'$

$\backslash \text{-pa-}'$ apab
 bpa c

Sorting

Select * from worker ORDER BY salary;

ASC (Default)

DESC (↓)

* Distinct values

— Department (distinct)

HR
HR
Admin
Admin.
Account

→
HR
Admin
Account

* Data Grouping

→ find no. of employees working in different dept

→ HR → ?

Accout → ?

Admin → ?

→ Grouping → Aggregation
(cont)

→ GROUP BY → Aggregation of ⁿ
COUNT, SUM
AVG, MIN, MAX.

Workers

Name	Department
Monika	HR
Nur.	Admin
C	
D	HR
E	Admin
F	Admin
G	Accnt
H	Accnt
I	Admin

→

Name	Department
Monika	HR
C	HR
;	Admin
(Admin
)	Admin
,	"
	Accnt
	Accnt

Q Find avg. salary per department?

* HAVING

=====

→ Select where to filter

"GROUP BY", HAVING

→ Q department, count, HAVING More than 2 workers?

* WHERE VS HAVING

=====
=====

* DDL

Constraints

① Primary Key Constraint

- Not Null
- Unique
- Only one P.K

Good practice

→ P.K int

② Foreign Key

↪ FK refers P.K of other table.

Workbench

③ Unique

```
Create table customer (
    ;
    Name VARCHAR(255) UNIQUE,
)
;
```

④ CHECK

```
Create table account (
    ;
    CONSTRAINT acc_balance_chk CHECK (
        balance >= 0),
)
;
```

⑨ DEFAULT

```
Create table account (
    balance INT NOT NULL Default 0,
);
```

* DML Data Modification Language

A Referential Constant

① INSERT

ON Delete Cascade

) ON Delete Set Null;

Replace :-

- ① Data already present, Replace.
- ② Data not present, INSERT

Replace VS Update

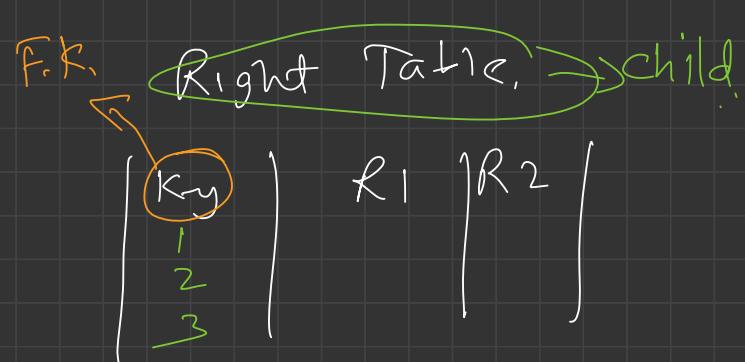
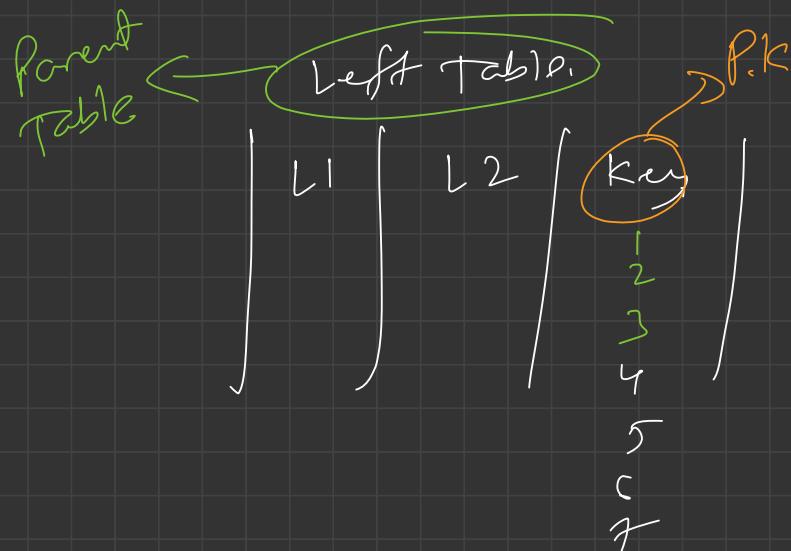
→ if row is not present, Replace will add
a new row while UPDATE will do
Nothing

#

JOINS

2) F.K \rightarrow Relations

\rightarrow we \rightarrow Data $\xrightarrow{\text{fetch}}$ JOINS



① INNER JOIN

Resultant table

key	L1	L2	R1	R2
1				
2				
3				

- Join → To apply joins, there should be a common attribute.

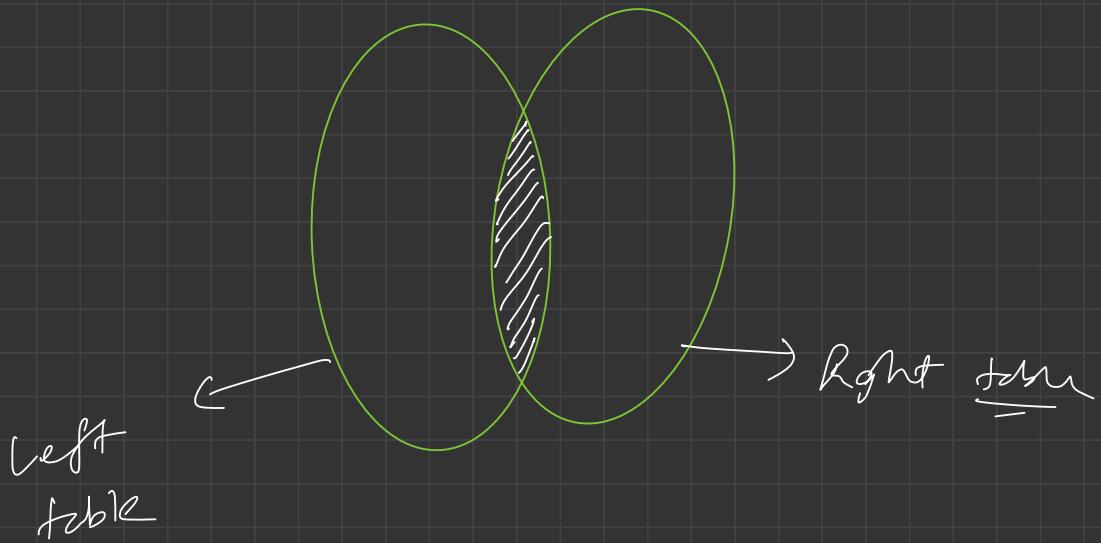
Syntax!

Select C.* , O.* FROM Customer AS C INNER JOIN Orders AS O
ON C.id = O.cust-id;

C.id, C.name

O.product

aliasing



INNER JOIN

⑦ Left Join

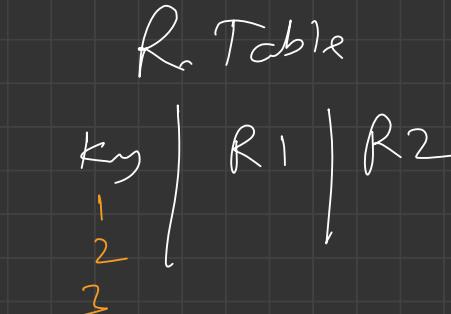
Left Table

L1	L2	key
1		1
2		2
3		3

4
5
6

R Table

key	R1	R2
1		
2		



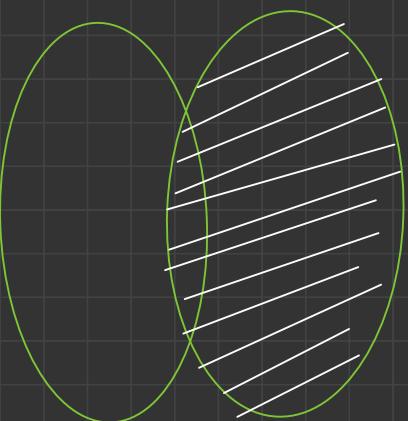
Resultant

key	L1	L2	R1	R2
1				
2				
3				
4			Null	Null
5			Null	Null
6			Null	Null

```
select c.* , o.* FROM Customer AS C LEFT JOIN  
Orders AS O  
ON C.id = O.cust_id;
```

③ Right Join

— Intended in knowing only the right side info



Left side

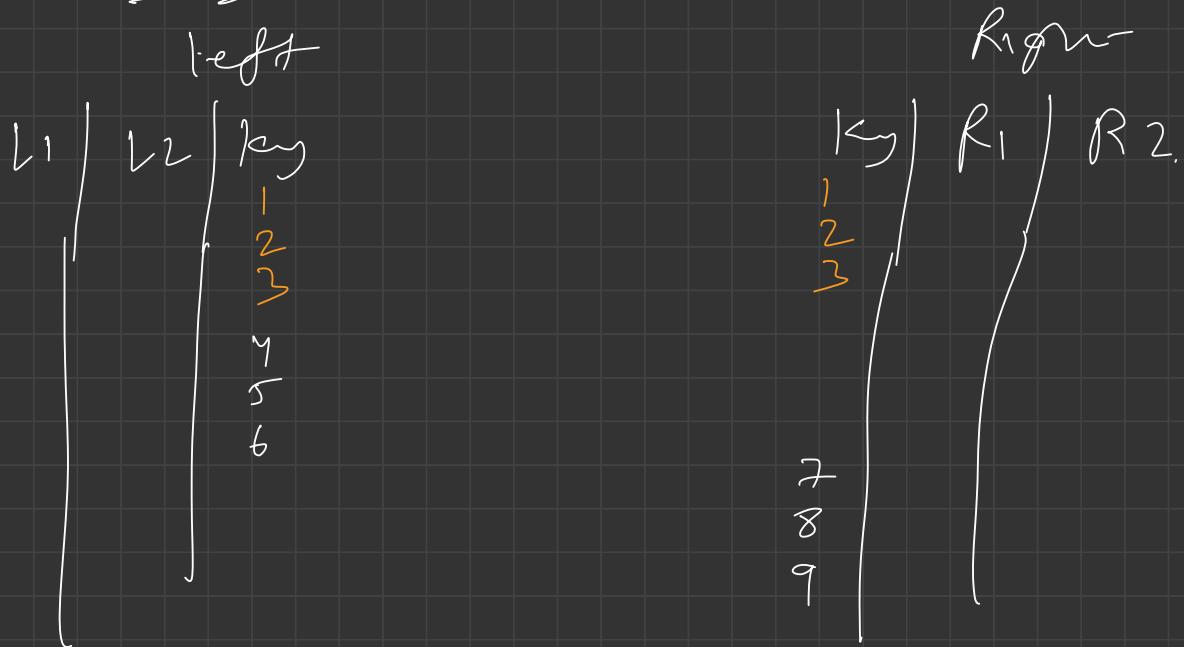
$$L_1 \quad | \quad L_2 \quad | \quad k_1 \\ k_2 \\ k_3$$

R. Feb 20

Rent font

A diagram illustrating a stack of five horizontal lines, labeled L₁ through L₅ from top to bottom. To the left of the stack, vertical labels k₁, k₂, k₃, k₄, and k₅ are positioned such that they align with the first, second, third, fourth, and fifth lines respectively. To the right of the stack, two groups of three lines each are shown, representing regions R₁ and R₂. The first group of three lines, corresponding to the first four lines of the stack, is associated with label R₁. The second group of three lines, corresponding to the last four lines of the stack, is associated with label R₂.

④ Full Join



MySQL Full Join X keyword X.

Emulated
=

LeftJoin \cup Right Joins

Syntax

= select * from leftTable as l LEFT JOIN rightTable as r
ON l.key = r.key.

UNION

select * from leftJoin as l RIGHT JOIN rightTable as r
ON l.key = r.key;


 Cross \otimes dim \rightarrow Content product
 $\Rightarrow \tau_L$ rows τ_R columns
 Rentat $\rightarrow 5 \times 10 = 50$ rows

e.g.

L_1	L_2
1	A
2	B

R_1	R_2
3	C
4	D

2) Rentat

\rightarrow Industrial use X.

L_1	L_2	R_1	R_2
1	A	3	C
2	A	4	D
2	B	3	C
1	B	4	D

Self Join

- emulate.

- 'INNER JOIN'

- Alias 'AS'

e.g. select
e1.id, e2.id, e2.name

FROM
employee AS e1

INNER JOIN

employee AS e2

ON e1.id = e2.id;



→ Work bench senior

* Project

	id	empID	name	startdate	clientID
▶	1	1	A	2021-04-21	3
	2	2	B	2021-03-12	1
	3	3	C	2021-01-16	5
	4	3	D	2021-04-27	2
	5	5	E	2021-05-01	4
	HULL	HULL	HULL	HULL	HULL

* EMPLOYEE

* C-LINEN

Q Can we use JOIN w/o using JOIN keyword?

→ Yes

Syntax

Select * FROM leftTable, RightTable WHERE
leftTable.id
= RightTable.id;

* Set Operations

{ 1, 2, 3, 4, 5, 6, 7 }

Table 1

	col 1	col 2
A	1	
B	1	
C	2	

Table 2

	col 1	col 2
A	1	
B		2
D		3

- (1) Union
- (2) Intersection
- (3) Minus

→ UNION
=

$$\overline{T_1 \cup T_2}$$
$$T_1 \cup T_2 \rightarrow$$

	col 1	col 2
A	1	
B		1
C	2	
D		2
		3

→ Rows.

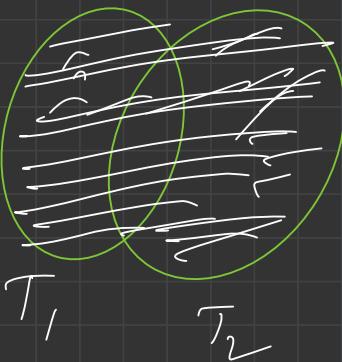
Syntax: →

Select * FROM Table1

UNION

Select * FROM Table2;

Set operatⁿ
→ Combine
two or more
Select statemt.



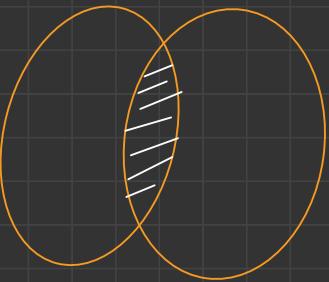
JOIN vs UNION

Full JOIN :-

Col 1	Col 2	Col 1	Col 2
A	1	A	1
B	1	B	2
C	2	null	num.
null	null	D	3

②

INTERSECT



Select * from T_1

INTERSECT ~~X~~

Select * from T_2 ; \Rightarrow Emulate

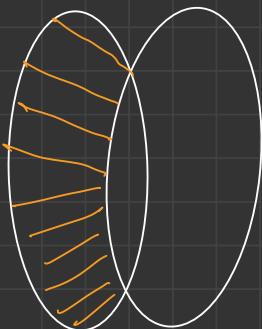
Select DISTINCT id from T_1

INNER JOIN T_2

using (id);

③ MINUS

$T_1 - T_2$



Syntax -> Select id FROM T_1
LEFT JOIN T_2 using(id)
WHERE $T_2.id$ IS NULL;

Example

A B C

Dept 1

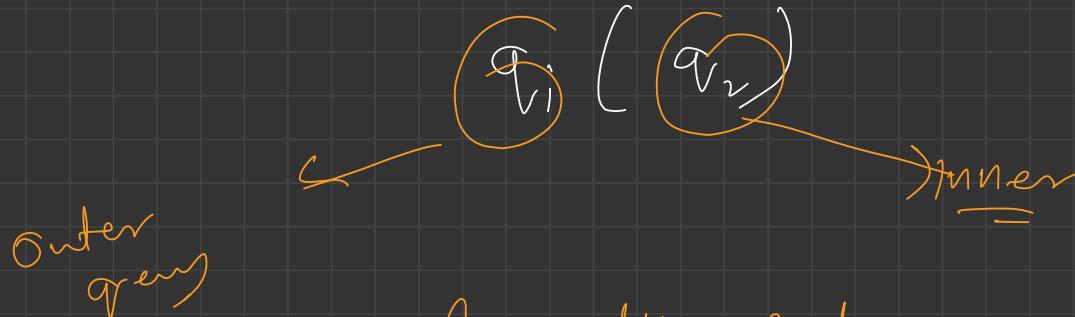
	empid	name	role
▶	1	A	engineer
	2	B	salesman
	3	C	manager
	4	D	salesman
	5	E	engineer
	HULL	HULL	HULL

Dept 2.

	empid	name	role
▶	3	C	manager
	6	F	marketing
	7	G	salesman
	HULL	HULL	HULL

Sub Queries

— alternative to joins



Generally, Outer query depends on
Inner query

e.g. Select * from table where id IN (select id
from table
where name = 'Lat'))

→ Work bench senior

* Project

	id	empID	name	startdate	clientID
▶	1	1	A	2021-04-21	3
	2	2	B	2021-03-12	1
	3	3	C	2021-01-16	5
	4	3	D	2021-04-27	2
	5	5	E	2021-05-01	4
	HULL	HULL	HULL	HULL	HULL

* EMPLOYEE

* CLIENT

* Correlated Subquery

Outer (Inner),

— inner query that refers the outer query

```
-- Correlated subquery  
-- find 3rd oldest employee  
SELECT *  
FROM Employee e1  
WHERE 3 = (  
    SELECT COUNT(e2.age)  
    FROM Employee e2  
    WHERE e2.age >= e1.age  
);
```

for each $e1.age$ inner → completely one time

① $e1.age \geq 32$

inner

② (2)

② $e1.age = 44$
 $\Rightarrow \emptyset$

③ 22
 $\Rightarrow 4$

32.
44.
22
31.
21

① $e1.age$.
31
 \Rightarrow ②

SQL Views

⇒ MySQL → views

Customer ⇒ id | name | age | address |

view → name | age → alias
— custom view

2)

SQL

||

2)

SQL

Query

~~AFS~~

Interview specific

ON SQL

Home work

1

2

Note making

Practice



Lec-10

Practice SQL Questions

→ DB

→ Tables / Relation

→ Data Relationships

→ Questions.

→ SQL Query

ORG DB

• WORKER TABLE

	WORKER_ID	FIRST_NAME	LAST_NAME	SALARY	JOINING_DATE	DEPARTMENT
►	1	Monika	Arora	100000	2014-02-20 09:00:00	HR
	2	Niharika	Verma	80000	2014-06-11 09:00:00	Admin
	3	Vishal	Singhal	300000	2014-02-20 09:00:00	HR
	4	Amitabh	Singh	500000	2014-02-20 09:00:00	Admin
	5	Vivek	Bhati	500000	2014-06-11 09:00:00	Admin
	6	Vipul	Diwan	200000	2014-06-11 09:00:00	Account
	7	Satish	Kumar	75000	2014-01-20 09:00:00	Account
	8	Geetika	Chauhan	90000	2014-04-11 09:00:00	Admin
	HULL	HULL	HULL	HULL	HULL	HULL

• BONUS TABLE

	WORKER_REF_ID	BONUS_AMOUNT	BONUS_DATE
►	1	5000	2016-02-20 00:00:00
	2	3000	2016-06-11 00:00:00
	3	4000	2016-02-20 00:00:00
	1	4500	2016-02-20 00:00:00
	2	3500	2016-06-11 00:00:00

• TITLE TABLE

	WORKER_REF_ID	WORKER_TITLE	AFFECTED_FROM
►	1	Manager	2016-02-20 00:00:00
	2	Executive	2016-06-11 00:00:00
	8	Executive	2016-06-11 00:00:00
	5	Manager	2016-06-11 00:00:00
	4	Asst. Manager	2016-06-11 00:00:00
	7	Executive	2016-06-11 00:00:00
	6	Lead	2016-06-11 00:00:00
	3	Lead	2016-06-11 00:00:00

Q Remove all the reversed number pairs from given table.

A	B
1	2
2	4
2	1
3	2
4	2
5	6
6	5
7	8

1, 2 \leftrightarrow 2, 1

2, 3 \leftrightarrow 3, 2

O/P

A	B
1	2
2	4
3	2
5	6
7	8

Sol 1

①

JOINS

②

CO-related

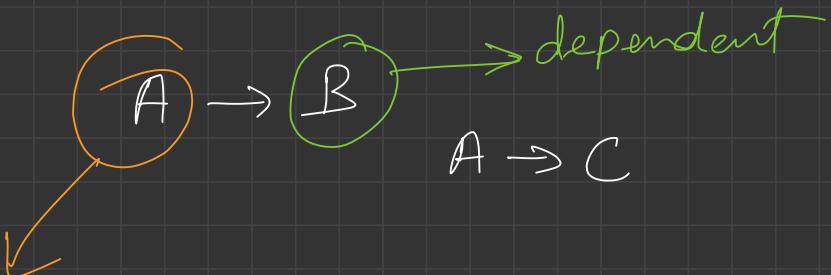


Lec-11

Normalization ?
====

Functional dependency

Table



A	B	C	D
1	2	1	5
2	4	5	6

Determinant

eg.

Emp

Emp-id

name

dept.

address

F.D: \rightarrow

Emp-id \rightarrow Emp-name.

Emp-id \rightarrow dept

① Trivial F.D \leftarrow

$A \rightarrow B$, B is a subset of A .

eg. $\{Emp-ID, Name\} \rightarrow Emp-ID$

eg. $A \rightarrow A$, $A \subseteq A$

$B \rightarrow B$ $B \subseteq B$

② Non-Translational F.D) \Rightarrow

$\{ \text{EMP_ID, name} \} \rightarrow \{ \text{Emp_addr} \}$

A

\rightarrow

B

Q. $B \subseteq A$? Ans No,

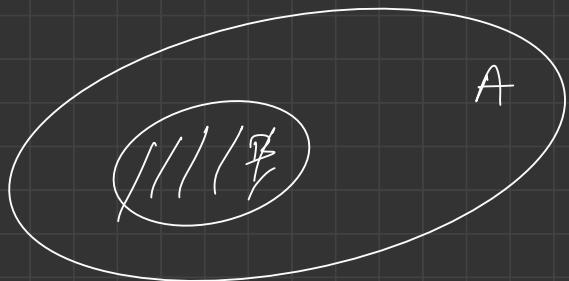
\Rightarrow This is N.T.F.D

$\Rightarrow A \rightarrow B, B \not\subseteq A, A \cap B = \text{NULL}$.

Emp-ID \rightarrow Emp-dept

Venn diag

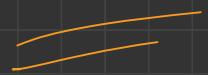
T. F. D



N.T.F.D



$$A \cap B = \text{NULL.}$$



① Reflexive

$$X = \{a, b, c, d, e\}$$

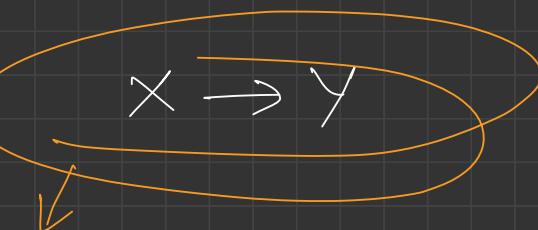
Subset $Y = \{a, b, c\}$

$\Rightarrow Y$ is a subset X ,

$$X \rightarrow Y.$$

② Augmentation

$$\overline{\rightarrow}$$



$$R(XY, Z)$$

$$XZ \rightarrow YZ$$

$\mathcal{Q} \quad R(A, B, C, D, E)$

$FD :-$

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

$CD \rightarrow \Sigma$

Can

$CD \rightarrow AC ?$

$C \leqslant$

$CD \rightarrow E \quad \& \quad E \rightarrow A \Rightarrow CD \rightarrow A \quad (\text{join, Hui})$

$C \subseteq CD \Rightarrow CD \rightarrow C$

So, $CD \rightarrow A \quad \& \quad CD \rightarrow C$

$\Rightarrow CD \rightarrow AC$ ✓

Why Normalisation

→ What if we have redundant data?

→ Introduces 3 anomalies → abnormalities

- ① Insertion
- ② deletion
- ③ Updation / modification

Student

<u>id</u>	<u>name</u>	<u>age</u>	<u>Branch_code</u>	<u>Branch_name</u>	<u>Branch.NOD</u>
1	A	18	1	CS	X
2	B	19	1	CS	X
3	C	18	1	CS	X
4	D	21	2	ECE	Y
5	E	20	2	ECE	Y
6	F	19	3	M.S	Z

① Inserthon

⇒ New student.

⇒ University → 'IT'

② Deletion

③ update / -

HOD change CS \rightarrow Q.

\Rightarrow id | name | age | Br code] Table |

Table 2 Branch info

Br code	Branch name	HOD name
1	CS	X
2	SCS	Y
3	MSE	Z
4	IT	Q

What we do in normalization

3) Table \rightarrow decompose \rightarrow into multiple tables.

[SRP] \rightarrow Single responsibility principle.

INF
= eg.

emp.

id	name	Phone.
1	A	88
2	B	12,99

INF



id	name	Phone
1	A	88
2	B	12
2	B	99

2NF \rightarrow

$R(\underline{A} \ B \ C \ D)$

$\Rightarrow \{AB\} \rightarrow P.K$

$A, B \rightarrow$ prime attributes,

$C, D \rightarrow$ Non-prime,

$\Rightarrow FD \Rightarrow B \rightarrow C$  Partial dependen.

$\approx AB \rightarrow C \checkmark$

$AB \rightarrow D$

A	B
Null	1
2	Null.
(Null Null)	
2	Y

{A B} P.K.

B → C

Null → C ?? X.

2NF Conversion

==

R₁ (A B D) A B → D

R₂ (B C) B → C

2NF Table (student project)

<u>studentID</u>	<u>Proj ID</u>	Student Name	Project name
S89	P09	Aria	Geo
S76	P07	Jacob	Choles
S56	P03	Ava	IOT
S92	P05	Alex	Cloud.

P.I.C : {StudentID, ProjID}

FID : $\begin{matrix} \text{Student ID} \rightarrow \text{Student Name} \\ \text{Project ID} \rightarrow \text{Project Name} \end{matrix}$

2NF = form

Student

Student ID	Project ID	Student Name.
S 89	P 09	Diva
S 76	P 07	Jacob
S 56	P 03	Ava
S 92	P 05	alex

Project

Project ID	Project Name.
P 09	Geo
P 07	Chitru
P 03	iot
P 05	Cloud.

3NF

R (A B C)

P.K { A }

F.D: -
A → B
B → C

2NF ✓

⇒

A	B	C
a	1	x
b	1	x
c	1	x
d	2	y
e	2	y
f	3	z
g	3	z

A → Prime.

B → C

↓ ↗
Non prime.

$B \rightarrow C - F, D$ (Functional dependency)

→ decompose 3NF

$R_1(\underline{A}B)$

$R_2(\underline{B}C)$

R_1	A	B
a	1	
b	1	
c	1	
d	2	
e	2	
f	2	
g	3	

R_2	B	C
	1	x
	2	y
	3	z

BCNF

e.g.

Stud-ID	Subject	Professor
101	Java	PJ
101	CPP	PC
102	Java	PJ2
103	C#	PC#
104	Java	PJ

- One student can enroll in multiple subjects
- for each subject, a professor is assigned to a student
- multiple professor can teach a single subject
- One professor can teach only one subject

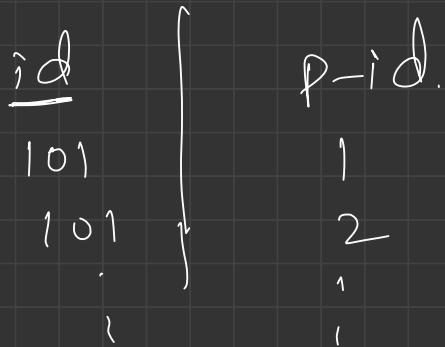
P.K = ? {Stud-ID, Subject}

① {Stud-ID, Subject} → Professor

② Professor → Subject

→) BCNF Conven
—

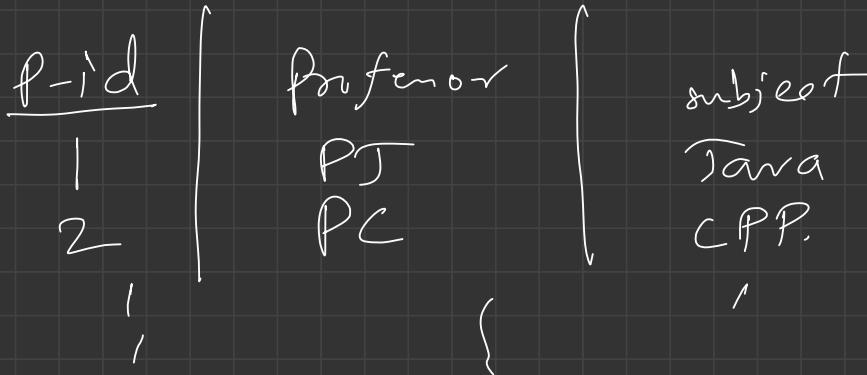
Student



p-id.

|
2
. |
|

Profess



Professor

PJ
PC

subject

Java
CPP.

|

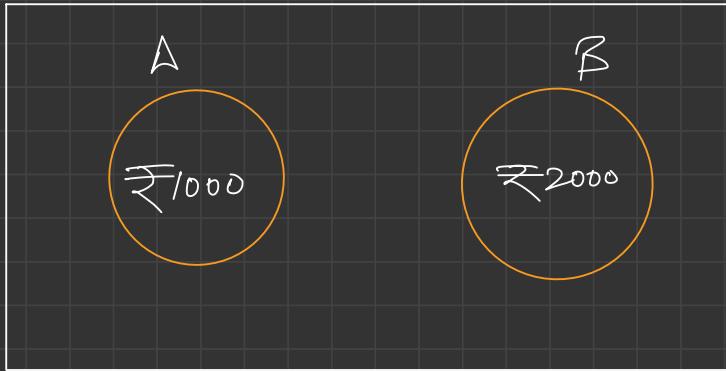




Lec-12

Transaction 1

ABC Bank



System: ₹3000

Task 1 → ₹50

from Bank A/C A to A/C B

T₁:

```
read(A)
A := A - 50;
write(A);
read(B);
B := B + 50;
write(B)
```

6 steps

Atomic →
They are considered single Task.
to be a

A C I D

Properties

$T_i : \text{read}(A)$

$$A := A - 50;$$

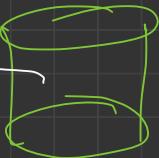
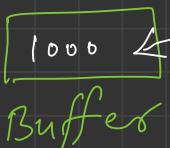
$\text{write}(A);$

$\text{read}(B);$

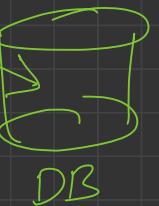
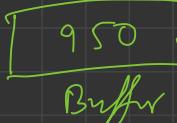
$$B := B + 50;$$

$\text{write}(B);$

$\text{read}(A)$



$\text{write}(A);$



(1) Atomicity

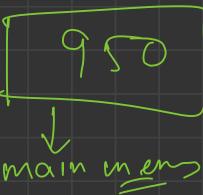
→ read 1000
 $950 = 1000 - 50$

950
 $\boxed{\text{write}(A)}$

B
read 2000

read 2000
2000 + 50
 $\text{write}(2050)$
2050

Commit



\rightarrow Transaction $\xrightarrow{\quad}$ Successful.
 $\xrightarrow{\quad}$ fail \rightarrow old state recovery.

DB \rightarrow maintain \rightarrow old state.

\downarrow intermediate state.

\swarrow Rollback $\xrightarrow[\text{Success} \times]{\quad}$ old state

② Consistency

③ Isolation |— Banking system

T_1 T_2 T_3 T_4

$t=0 \Rightarrow T_1$ (Google Pay) $t=10 = T_2$ (Netbanking)

$\text{read}(A) \rightarrow 1000 \rightarrow \text{read}_1(A) \rightarrow 1000.$

$A := A - 50 = 950.$

$\boxed{A = 950}$

$\text{write}(950)$

$1000 - 50 \quad \cancel{2-950}.$

$B \rightarrow$

$B \rightarrow 50 + \cancel{50} \quad \boxed{1000}$

④ Durability

$T_j \rightarrow \text{Success}$



all 6 step \rightarrow Success



DB update permanent (persistent)



even if there is a system failure after

T completed
executed.

→ Recovery - mgmt Component.

read (A)

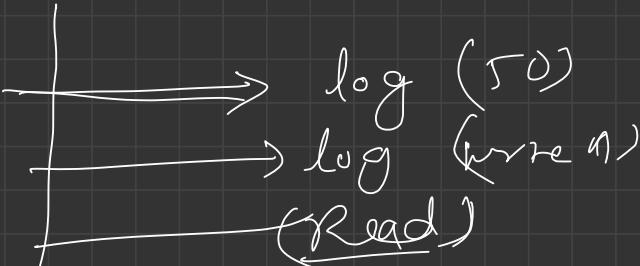
$A := A - 50$

write (A)

Read (B)

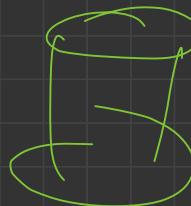
$B := B + 50$

write B,

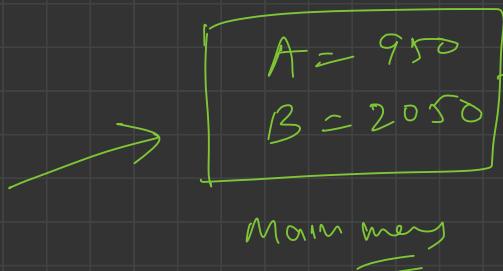


write

crossed out

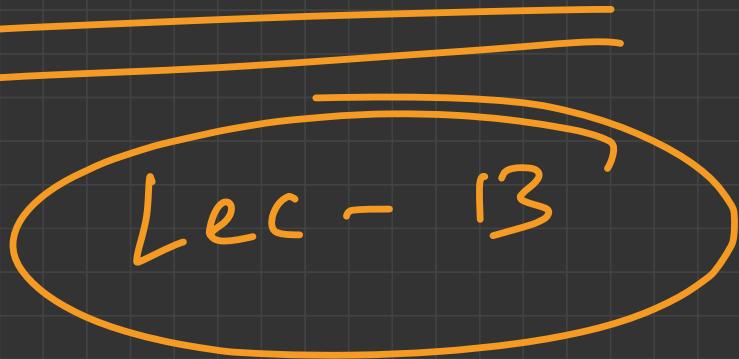


System crash



* States of Transaction (life cycles)

How to Implement Atomicity 2.

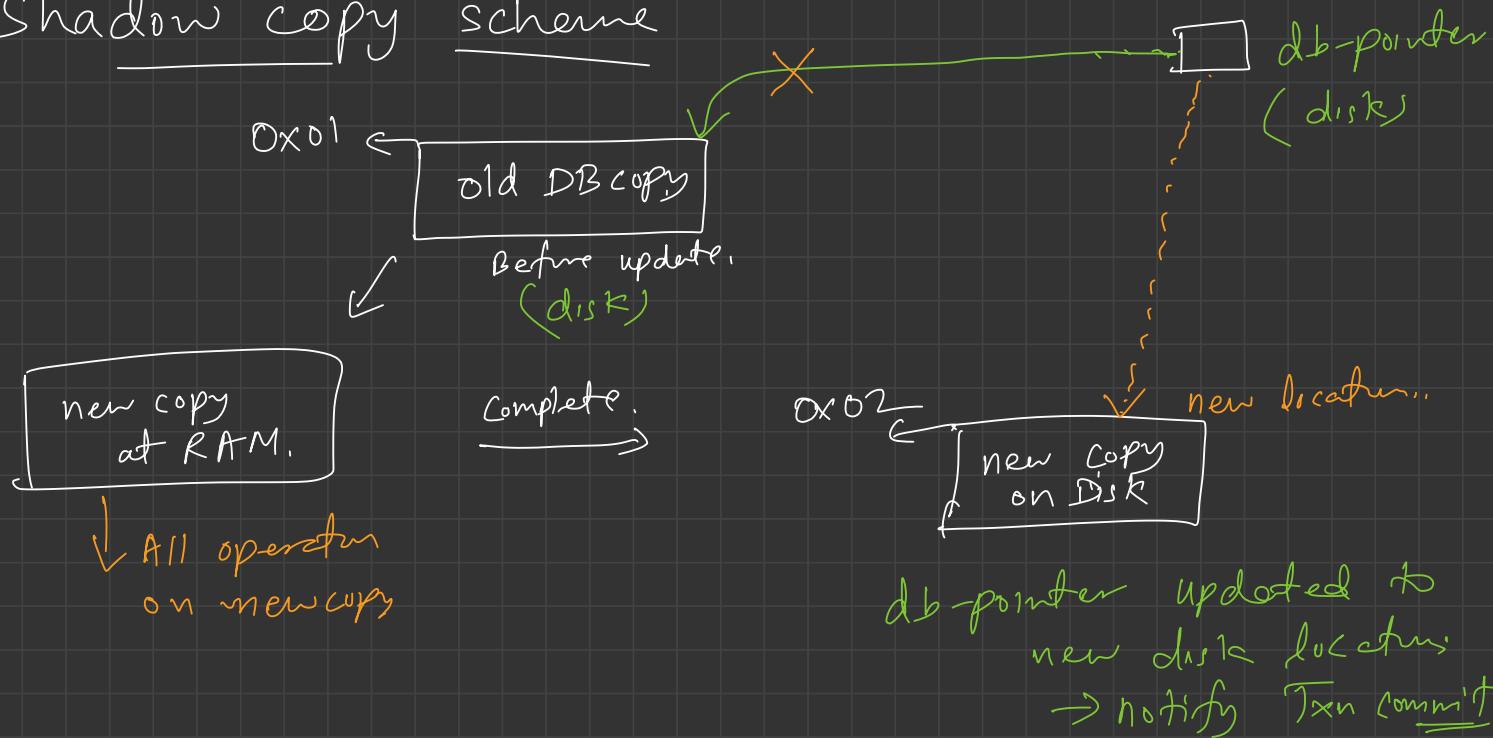




Lec-13

* How to implement Atomicity & Durability ?

① Shadow copy scheme



Log Based Recovery !-

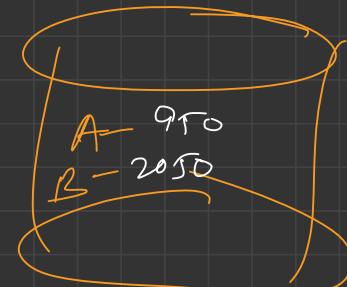
①



T_0 : $\text{read}(A)$
 $A := A - 50$
 $\text{write}(A)$
 $B := B + 50$
 $\text{write}(B)$

Logs (stable storage)

$\langle T_0, \text{start} \rangle$
 $\langle T_0, A, 950 \rangle$
 $\langle T_0, B, 2050 \rangle$
 $\langle T_0, \text{commit} \rangle$



① Deferred DB Modifications

② Immediate DB modifications

$\langle T_0, \text{start} \rangle$ \rightarrow old item value
 $\langle T_0, A, 1000, 950 \rangle$ \rightarrow new item value.
 $\langle T_0, B, 2000, 2050 \rangle$
 $\langle T_0, \text{Commit} \rangle$

→ Checkpoints

$C_1 \rightarrow \sum T_n \rightarrow \text{commit}$

$C_2 \rightarrow \sum T_n \text{ com}$

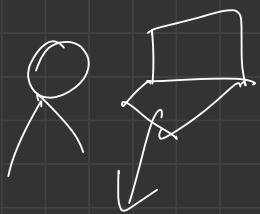
$H.W \rightarrow$ Checkpoints

Comment ✓

Google



Lec-14



SQL
Select
Where

|
DBMS.

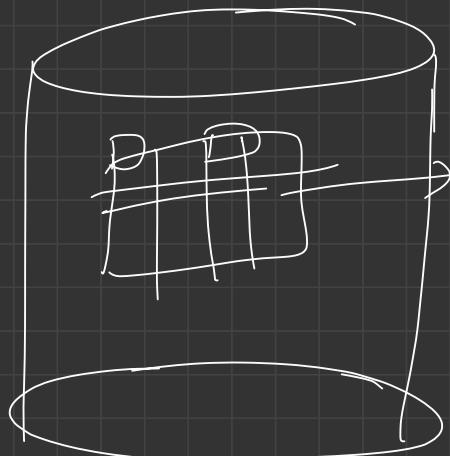


Table
student

Disk

Data struct

INDEX

INDEX file.

Rollno.	BP
1	B ₁
11	B ₂
21	B ₃
:	:
9991	B ₁₀₀₀

Data file in Disk. (10,000 records)

1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

- B₁

11				
12				
13				
14				
15				
16				
17				
18				
19				
20				

- B₂

21				
22				
23				
24				
25				
26				
27				
28				
29				
30				

- B₁₀₀₀

e.g. 843 Student

each Block has 10 records → Total Blocks
1000

Types of Indexing

(1) Primary index

(1) Based on key attribute (ordering datafile based on key attribute)

→ P.I / C.I
 |
 =

↳ Indexing → Sparse indexing

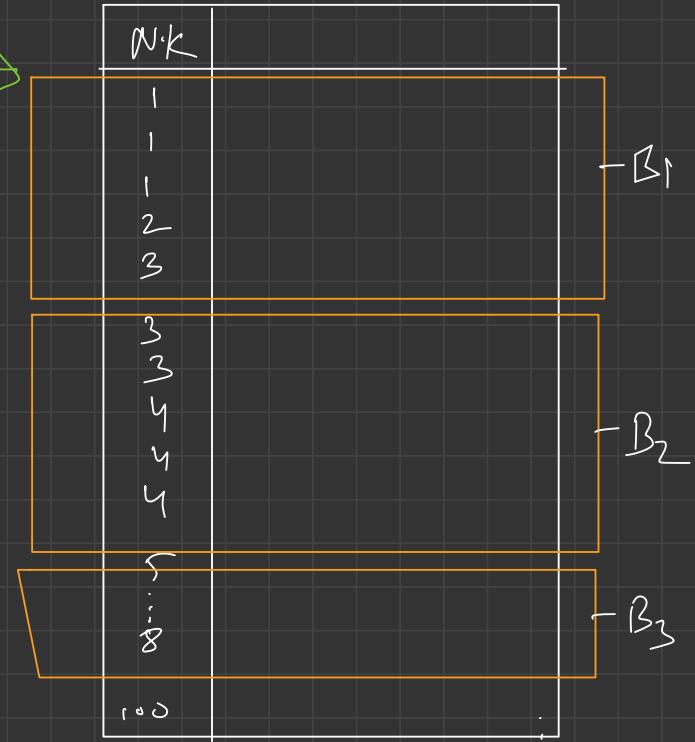
② Based on non-key attribute

↳ Clustering Indexing

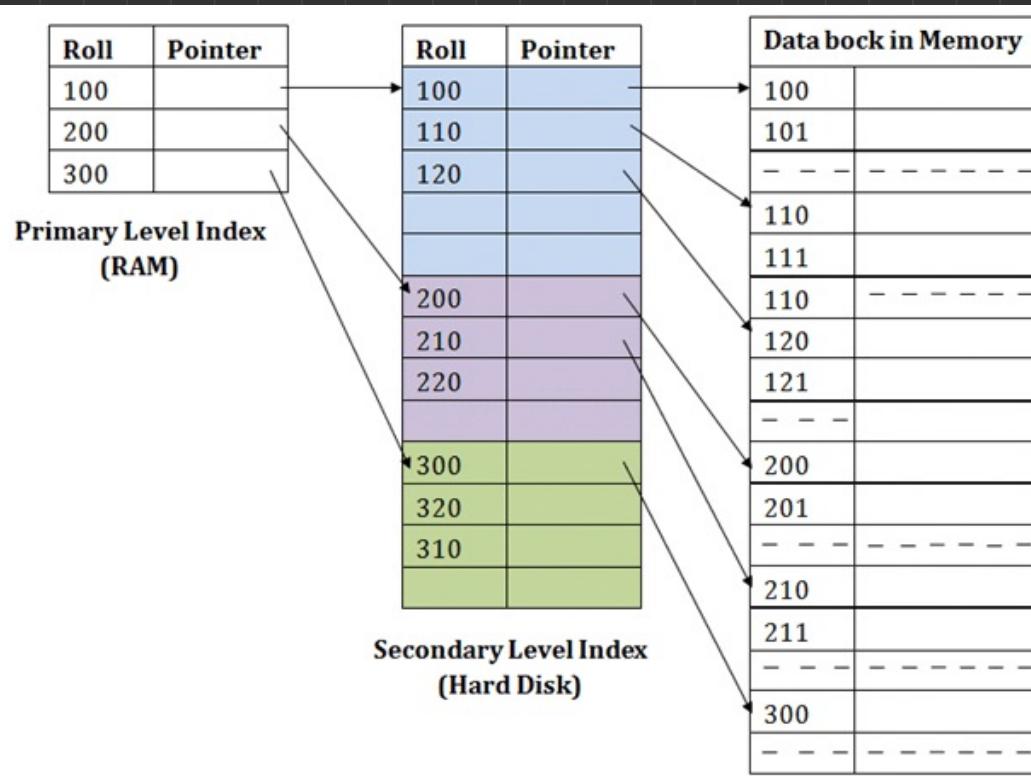
2) INDEX

N.K	BP
1	B ₁
2	B ₁
3	B ₁
4	B ₂
5	B ₃
.	.

→ Dense Indexing



Multi Level INDEXING



Secondary Indexing

- Datafile unsorted

- S.k → Key or Non key

S.K	B.P
1	
2	
3	

→ search key
=

S.K
1
19
23

→ B_1

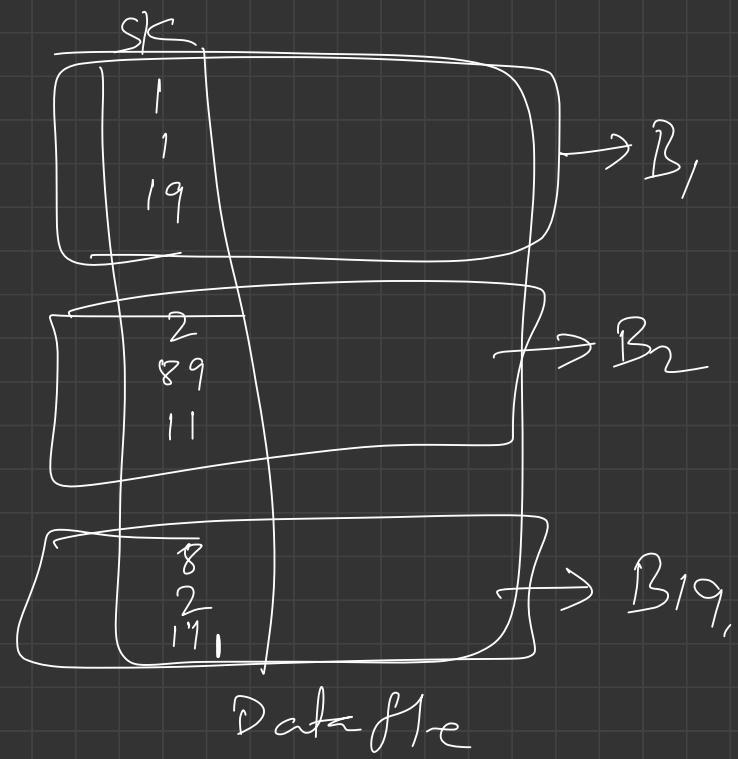
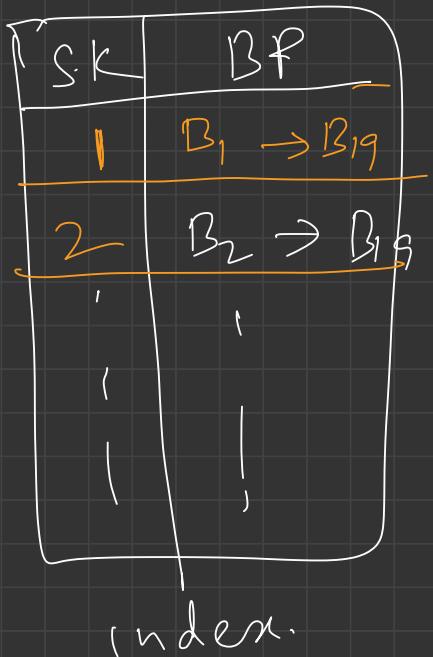
92
89
2

→ B_2

⋮
⋮
⋮

8
12
100

- Dense Indexing





Lec-15

NoSQL



"Not only SQL"



Non-Relational Model

→ SQL

→ Structured data.

→ constraints

→ fixed schema.

→ Vertical scaling

NoSQL

→ structured data
→ semi-structured

→ flexible
schema,

→ vertical
horizontal

⇒ Data Modeling in SQL vs NoSQL

① SQL

Mers	ID	first_name	last_name	cell	city
	1	Tata	Salt	811	Mumbai

Hobbies

ID	mer_id	hobby
10	1	scrapbooking
11	1	Games,
12	1	Biking

② NoSQL.

{

```
"id": 1,  
"first-name": "Tata",  
"last-name": "Salt",  
"cell": "811",  
"city": "Mumbai",  
"hobbies": ["Scrapbooking", "Games", "Biking"]
```

}

→ Advantages of NoSQL:

① Flexible schema:

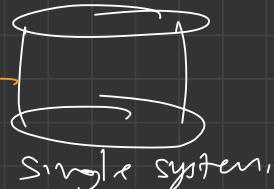
② Horizontal scaling:

① Scaling

Vertical

- hardware
- RAM
- CPU.

NoSQL



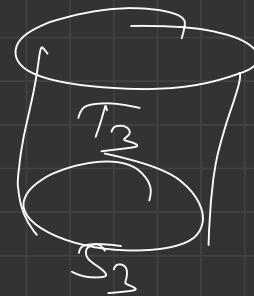
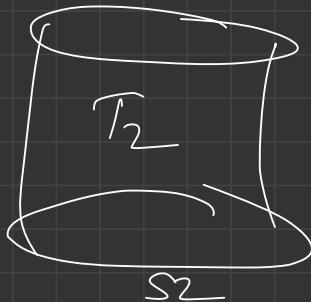
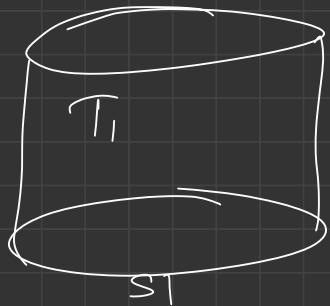
Horizontal scaling

- additional node
- Load share.



→ Intention

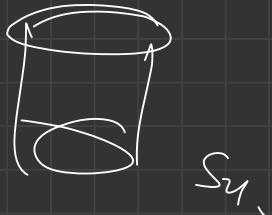
SQL → why No Normal scaling?



SQL → Table collection

→ SQL → Data retrieval → Joins

→ Slow → entirely slow



→ When to use NosQL

① Fast development

TikTok → X.

Instagram → Reels X

→ Types of NoSQL

① Key value stores

Key → value.

{

id : 'Yashas'

2

② C - Structures

Name	City	Age
Matt	Delhi	27
Dave	Taipur	30

Row wise →

Matt	Delhi	27	Dave	Taipur	30
0x01	0x02	0x03	~~~~~		

Column wise

Matt	Dave	Delhi	Taipur	27	30
------	------	-------	--------	----	----

③ Document based → MongoDB

lower document

{

id:

:
:
:
!

{ id: — .

}

access

{ id: — .
}

④ Graph Based stores

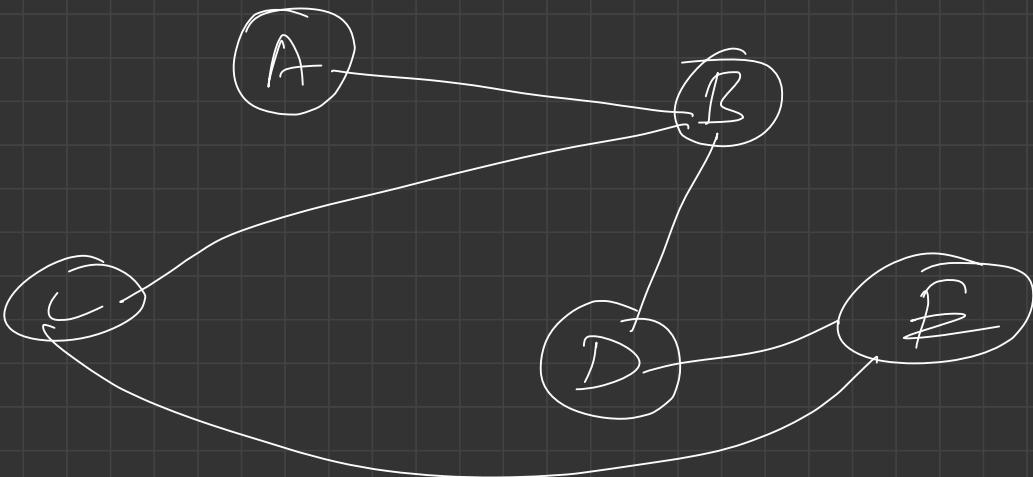
→ data →

Nodes (vertices)

Edges (Relationships)

Facebook →
=

Relationships



⇒ Disadvantages ↗



Lec - 16

Types of Databases

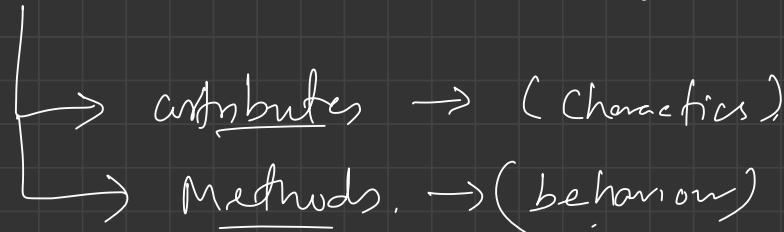
① Relational Databases

- Relational Model
- tables
- Related.

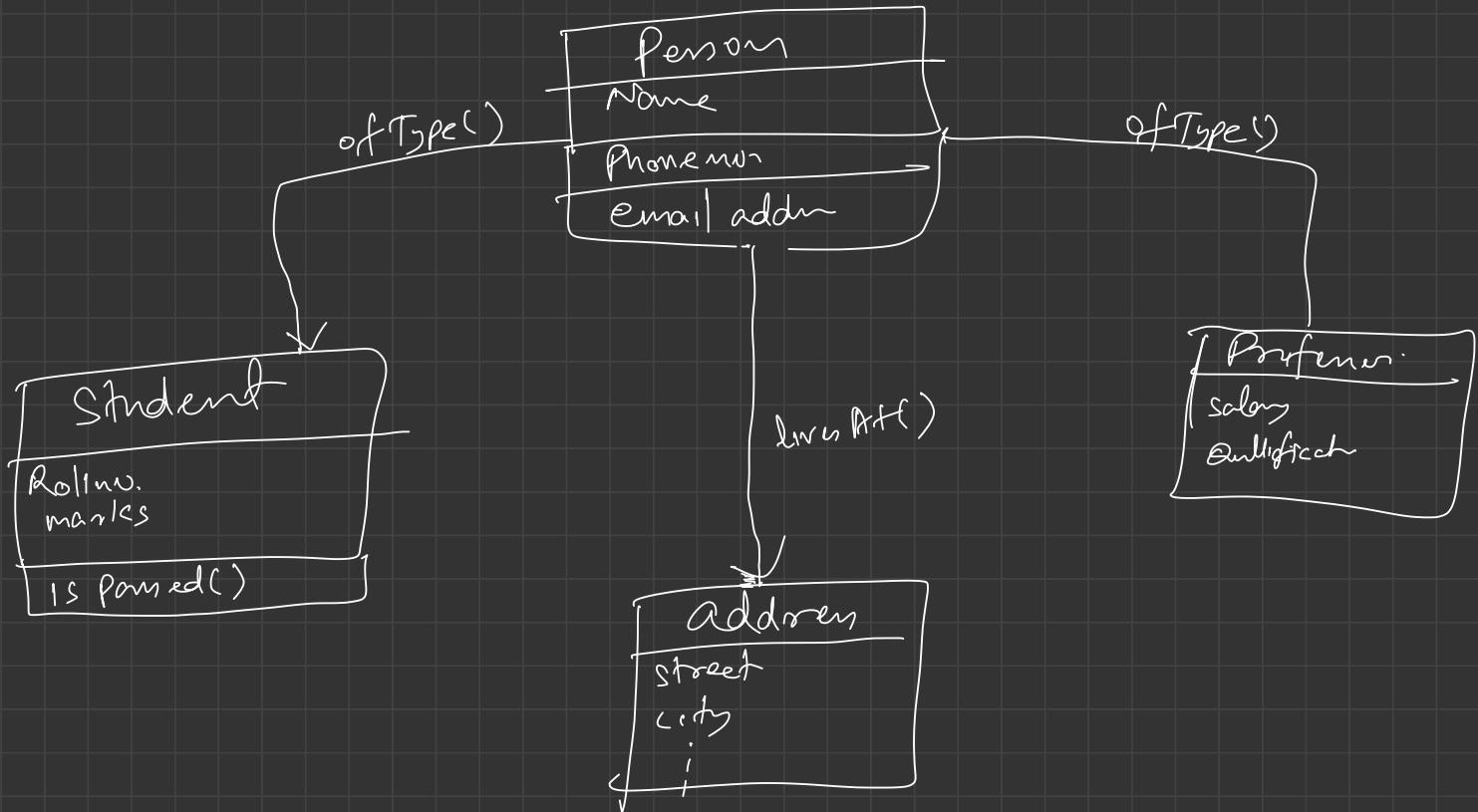
② object-oriented datamodeling

— OOPS similar concept

→ Class → instance → objects

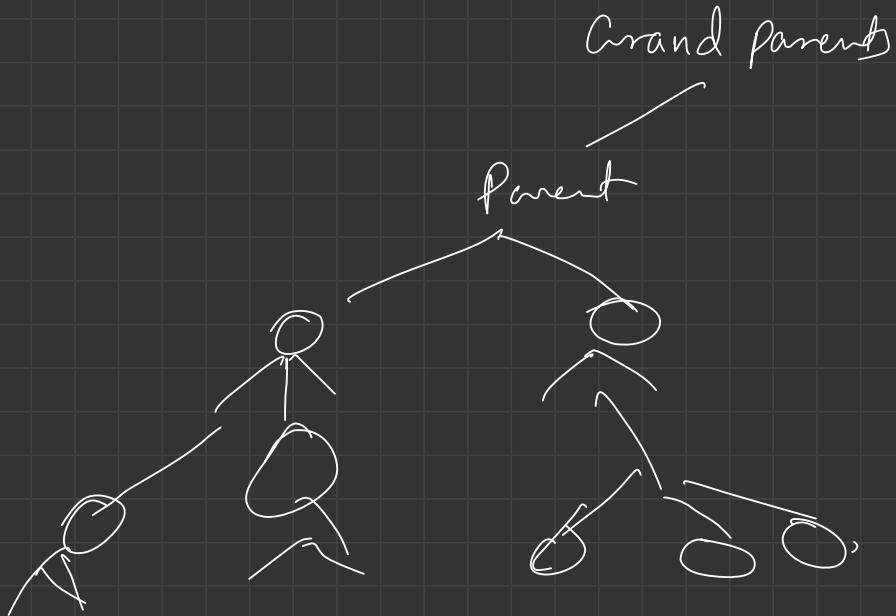


- Stores everything in form of objects.
- object can have , table , executable code .
- objects they communicate via methods .
- handles → object ID .
- information stored in DB is capable of being represented as an object .



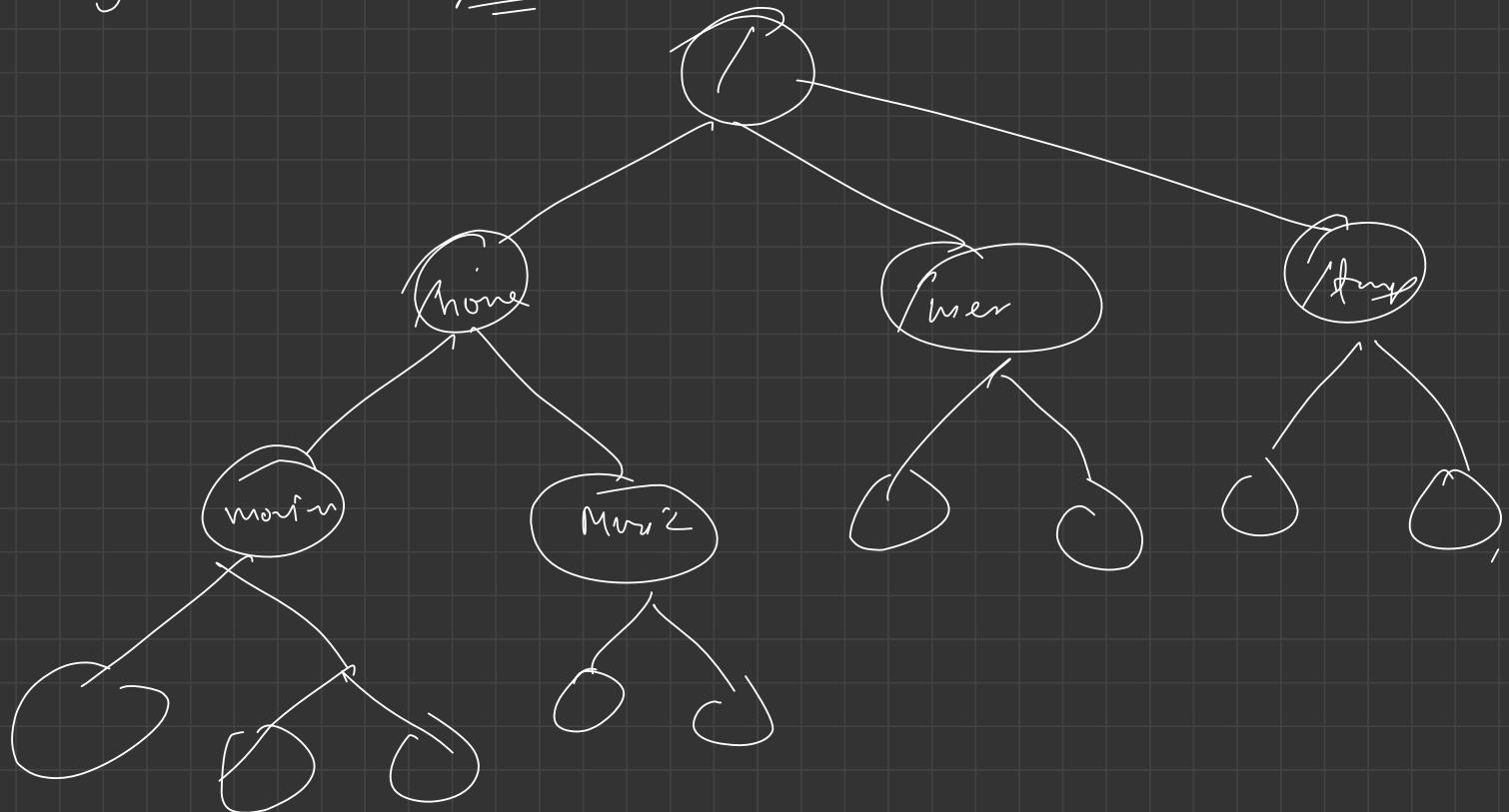
③ Hierarchical DB

e.g. Family Tree.



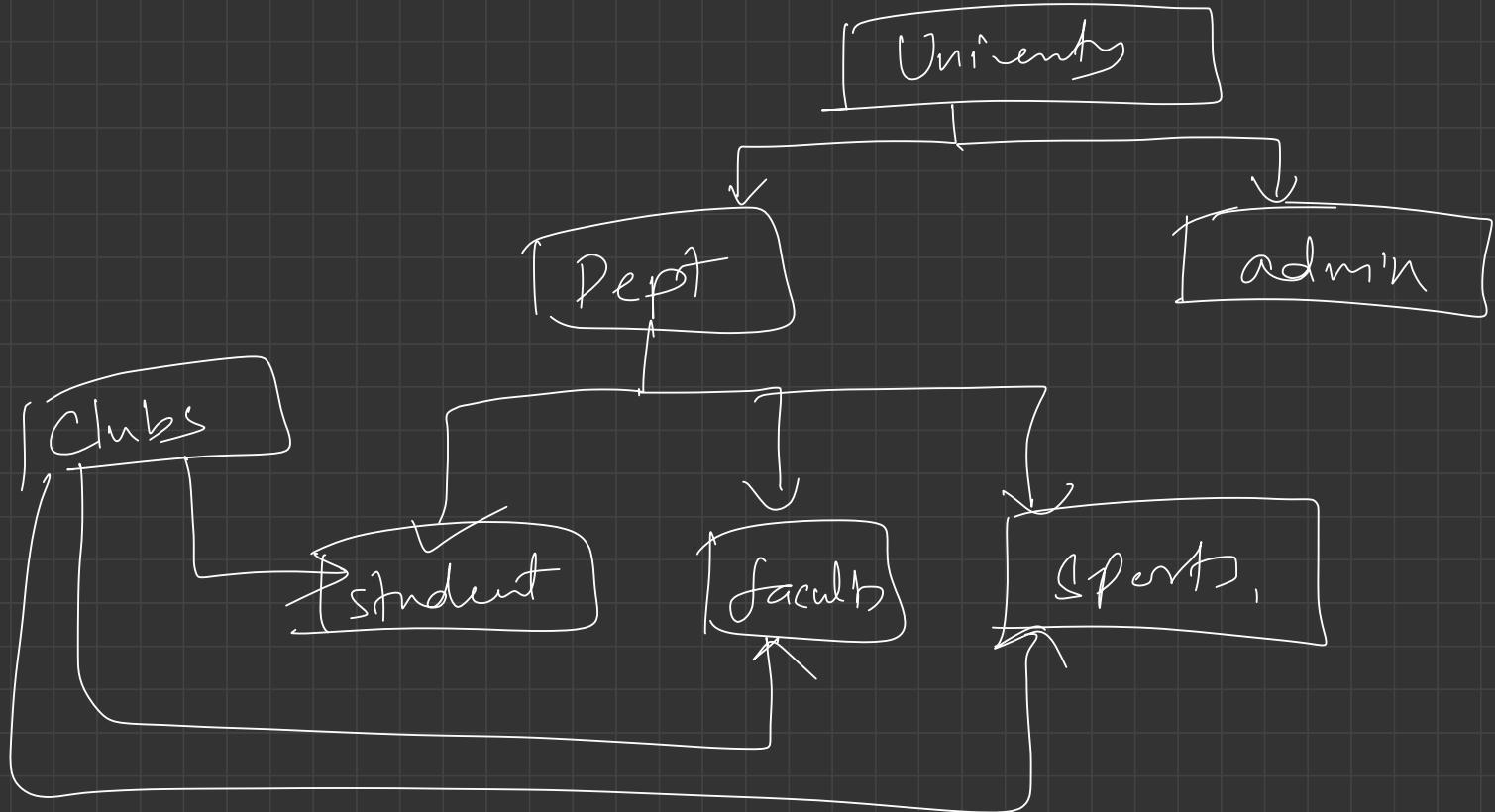
e.g.

Filesystem



(S)

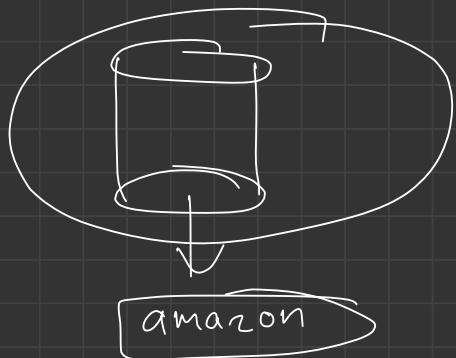
N/W - DB

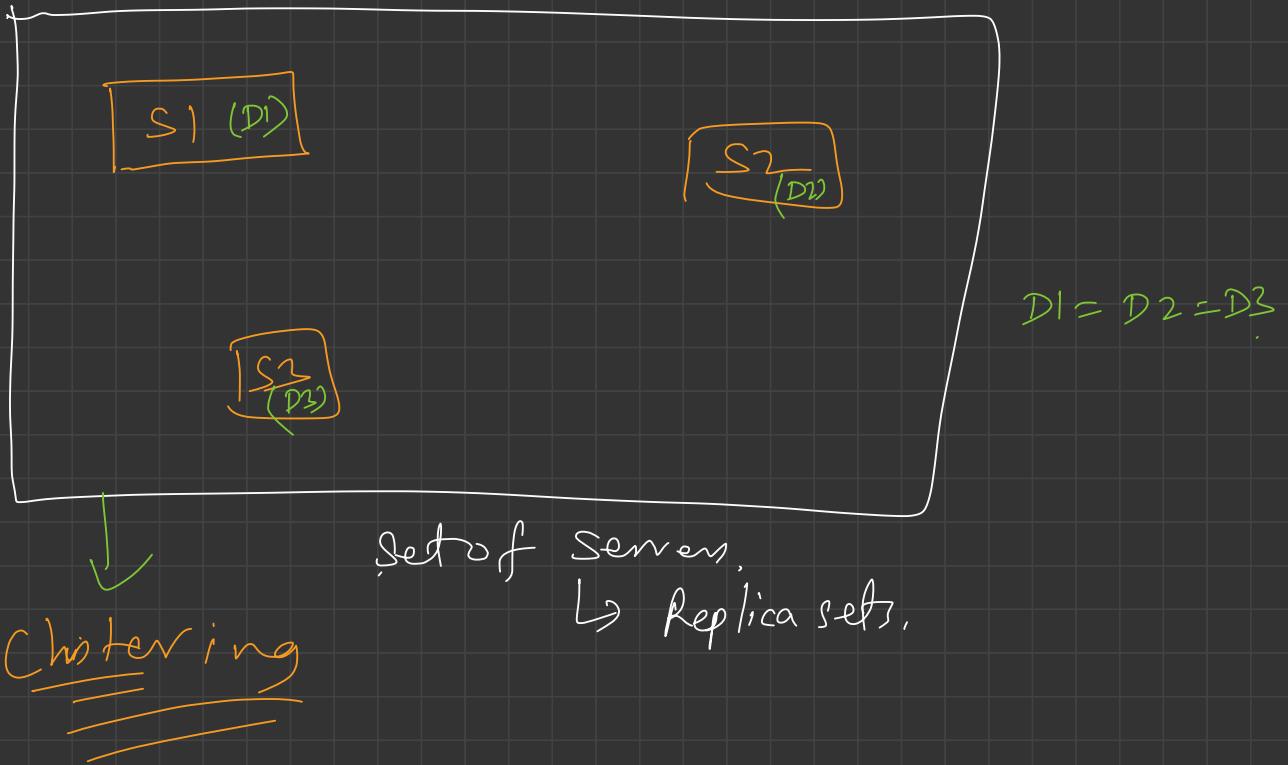




Lec-17

(Clustering | Replica Set)





① Redundancy ↗

(S1) → shutdown
→ maintenance
→ etc.

② Load Balancing ↗

③ Availability \rightarrow

High Avl.

$S_1 \rightarrow \text{down}$

$\rightarrow S_2 \neq S_3 \rightarrow \underline{\text{Backup}}$

⇒ homework \rightarrow (Content delivery Network)
 \rightarrow Internet



Lec-18

⇒ Data → store → DBMS



Bhot zadd
huge data → Mawes.



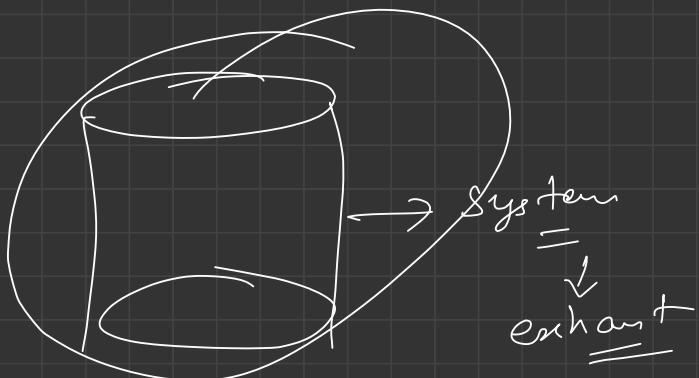
www hosted



Face book

mtb

Amazon

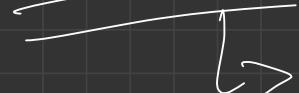


① Data huge (Manageability)

② no. Requests are large.
of.



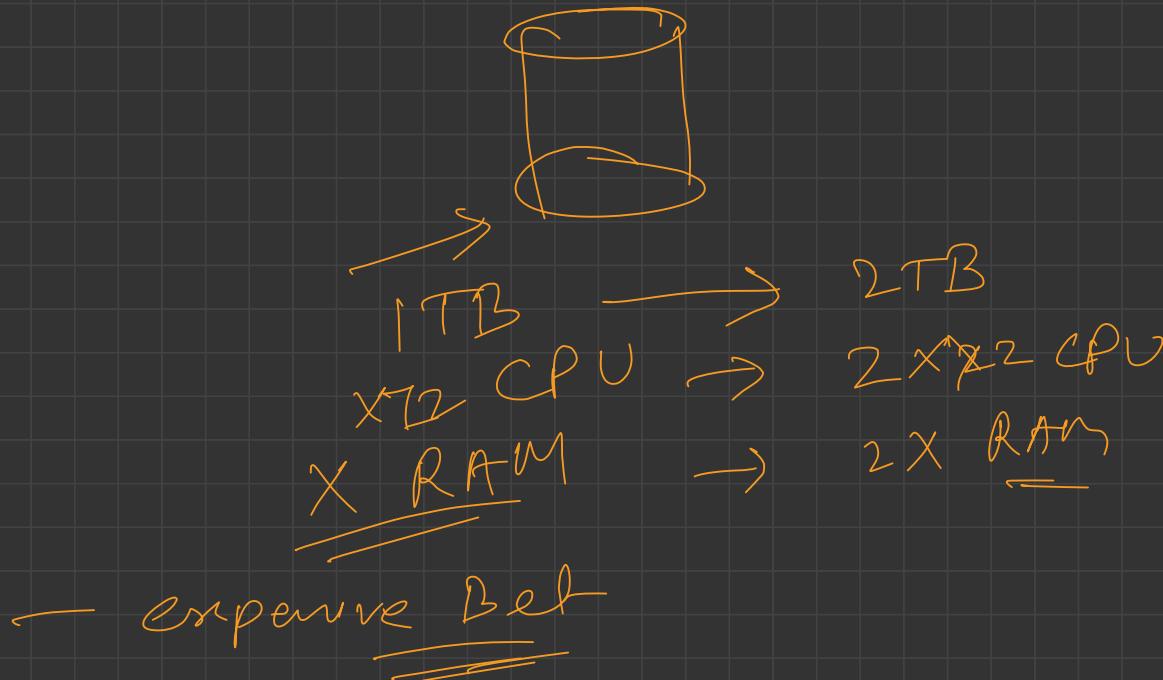
Data distribute



Distributed database

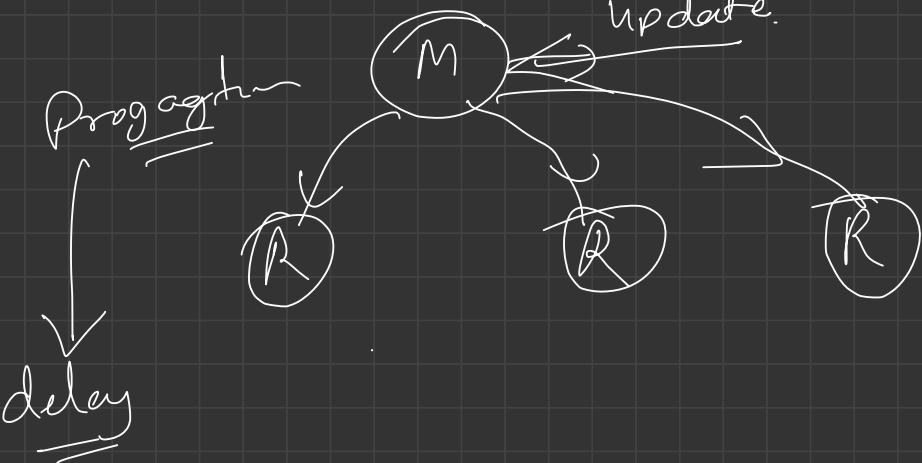
⇒ Database Optimization

① Scale-up (Hardware) ↑



②

Add Replica-set (Clustering)



③

Partitioning

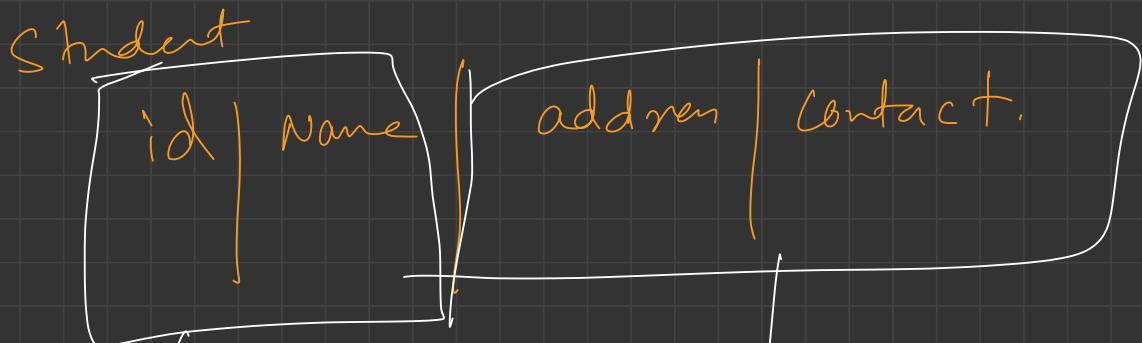
→ Partitioning

→ Scale-out (Horizontal scaling)

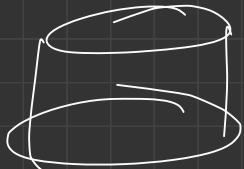
→ New nodes are added.

Student Table.

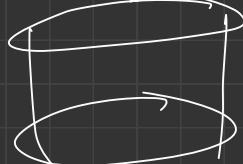
id	Name	Class	addrn	phone
x		,		
x				
x				



S1



S2



RShno.

id

|
→ 25K.

S1

S2

Server

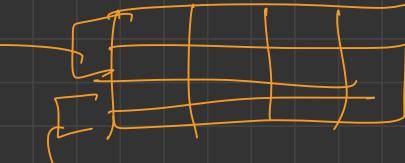
25001 → 100K

*Sharding

→ Horizontal partitioning → apply

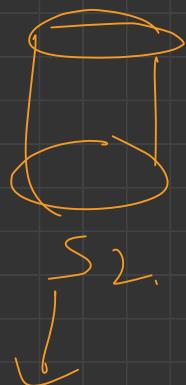
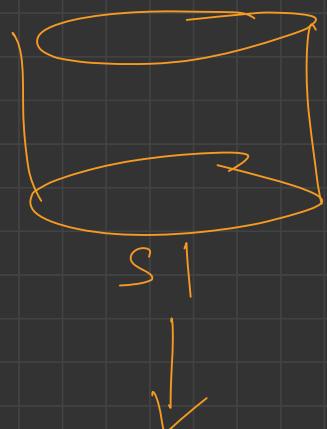
student

$1 \rightarrow 10K$



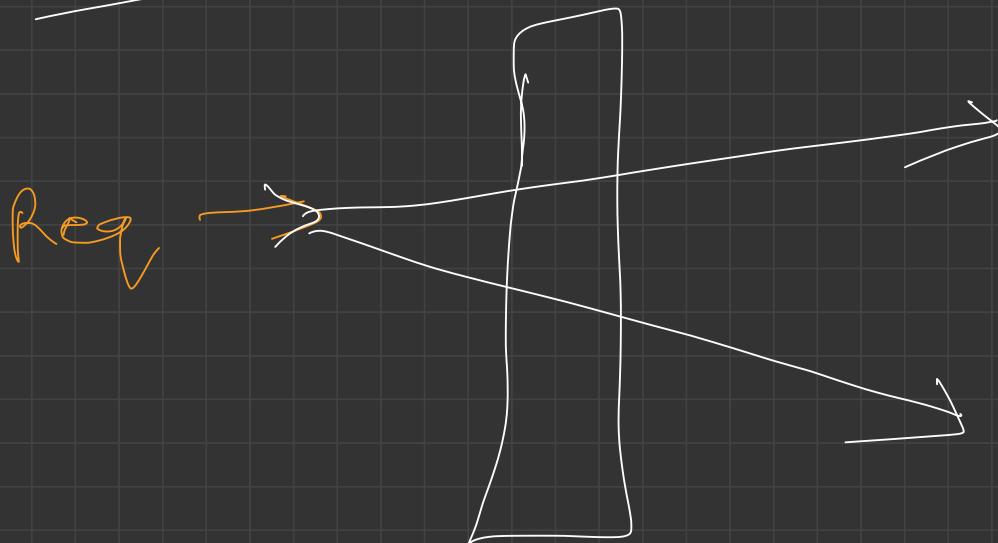
seaching

$10001 \rightarrow 100K$



Independent

Routing layer.



Routing layer.

DB instance

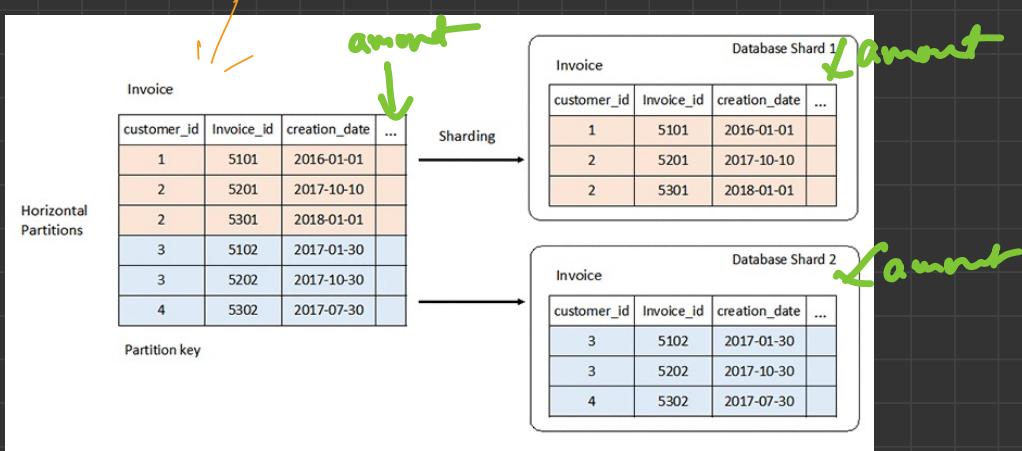
S1

$1 \rightarrow 10K$

$>10K$

$-100K$

DB instance



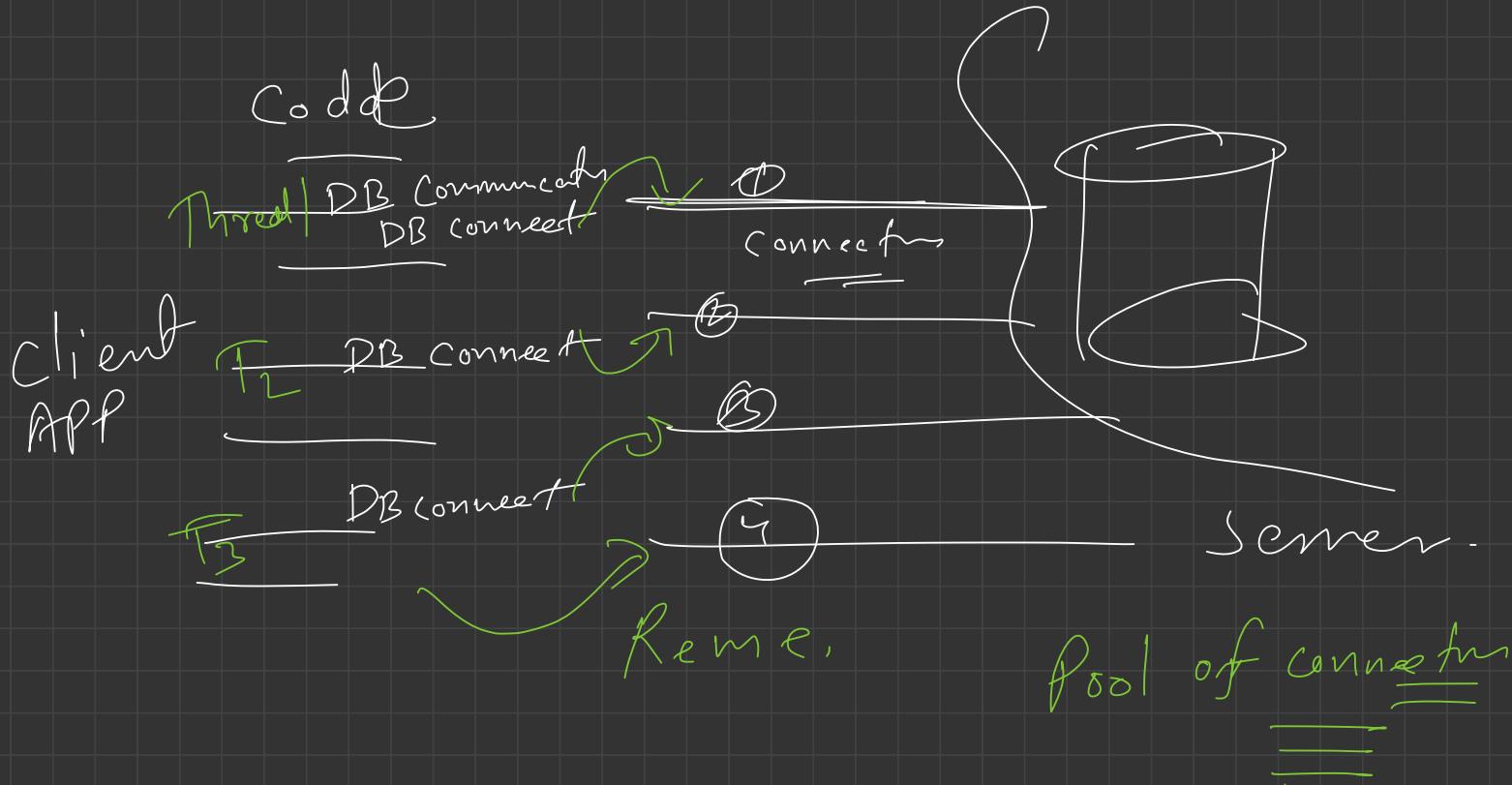


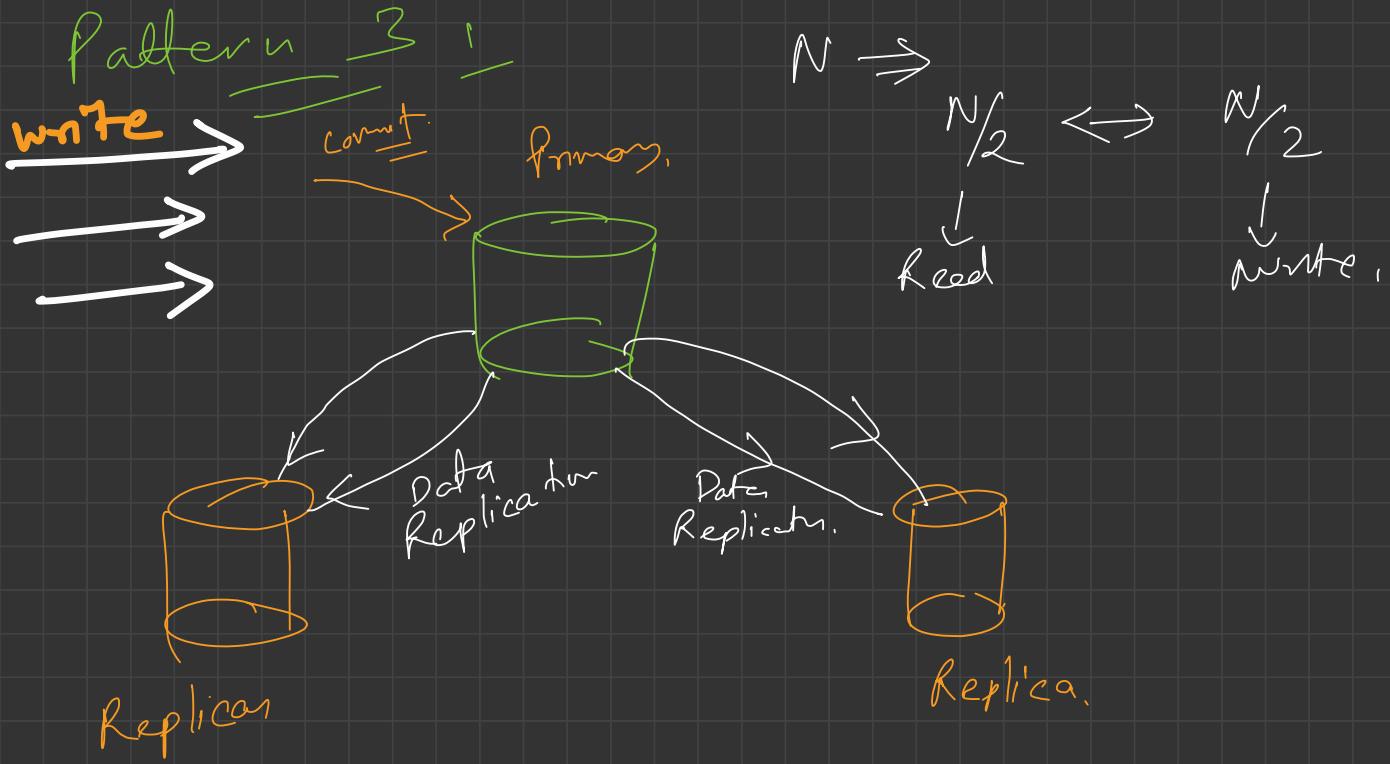
Lec- 19

→ Design

DB scaling Patterns

Cache DB Connections

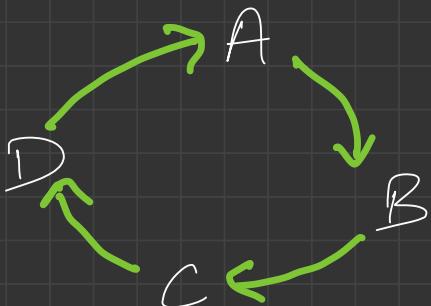




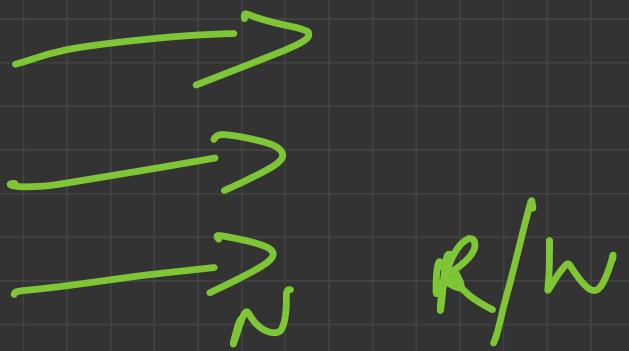
↑↑↑ Read.

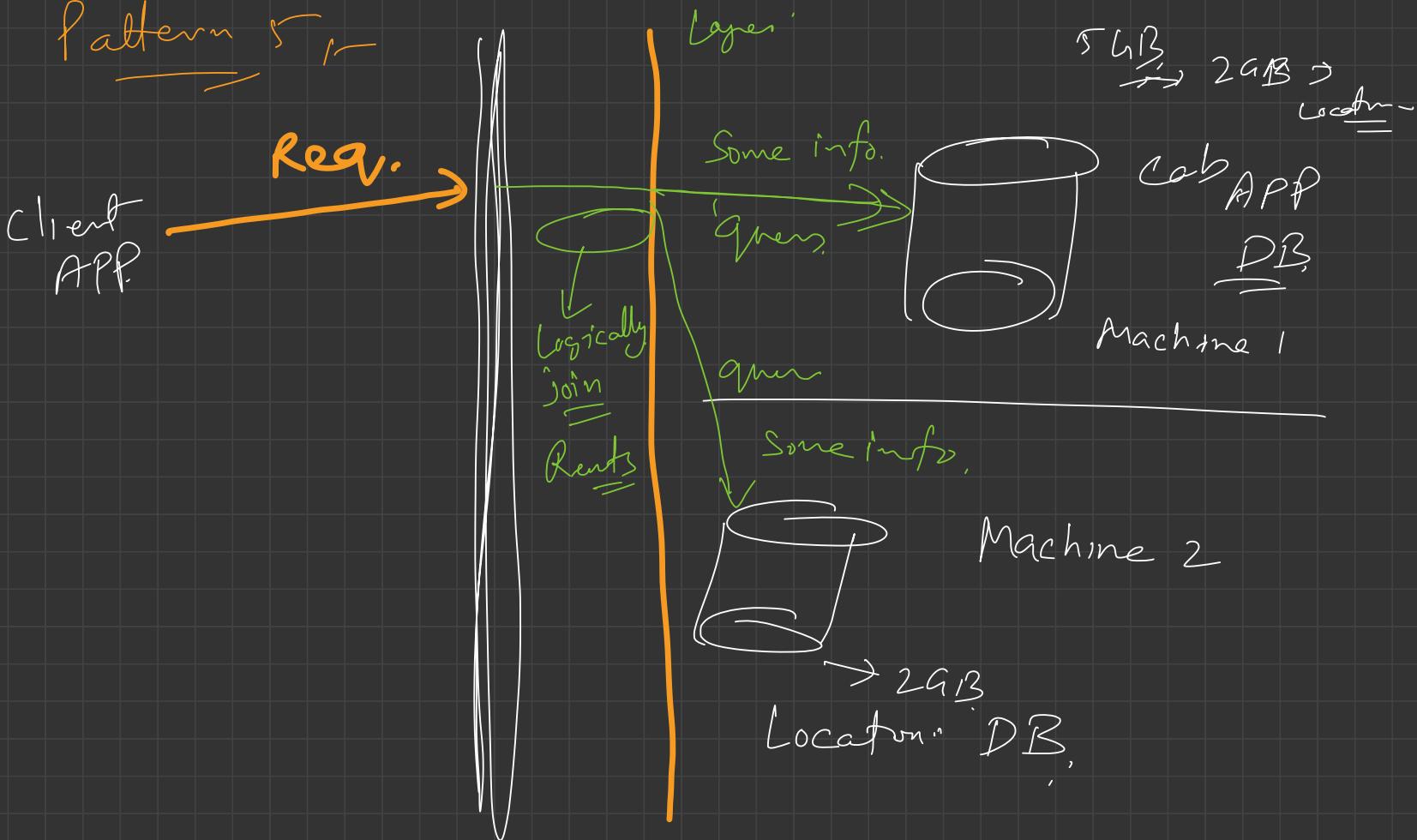
↑↑↑ Read

Pattern Y \mapsto



Write \rightarrow Any node
Y will handle it

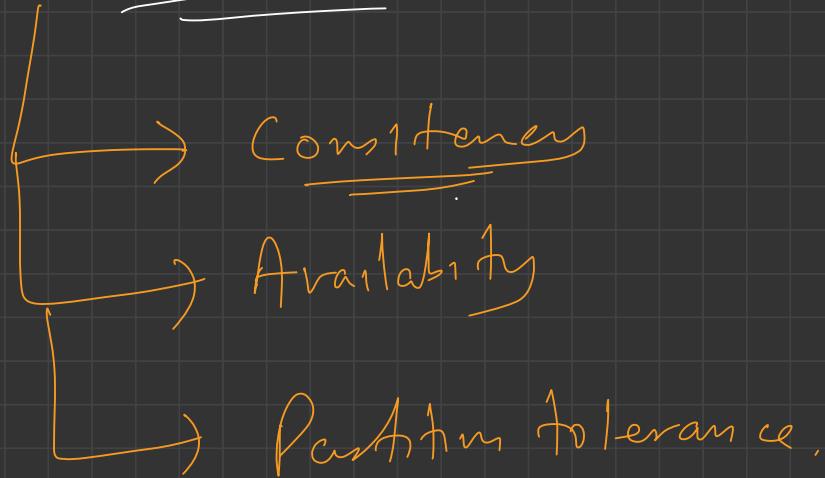


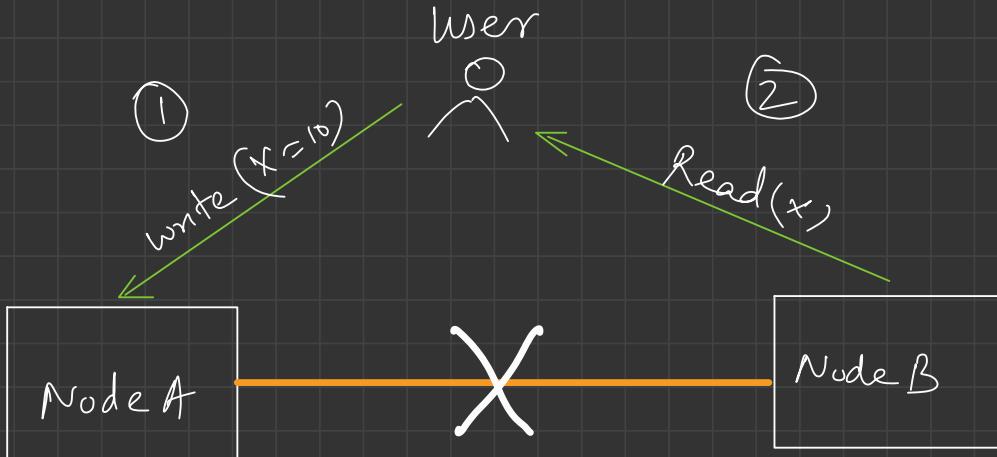




Lec-20

CAP Theorem





① Consistency lost \Rightarrow (Both req. should process)

② Availability lost \Rightarrow (if we fail one req.)

→ Home works

→ Banking system → ACID

→ Social Networking System

→ BASE Properties

①

Basically available

②

Soft state

③

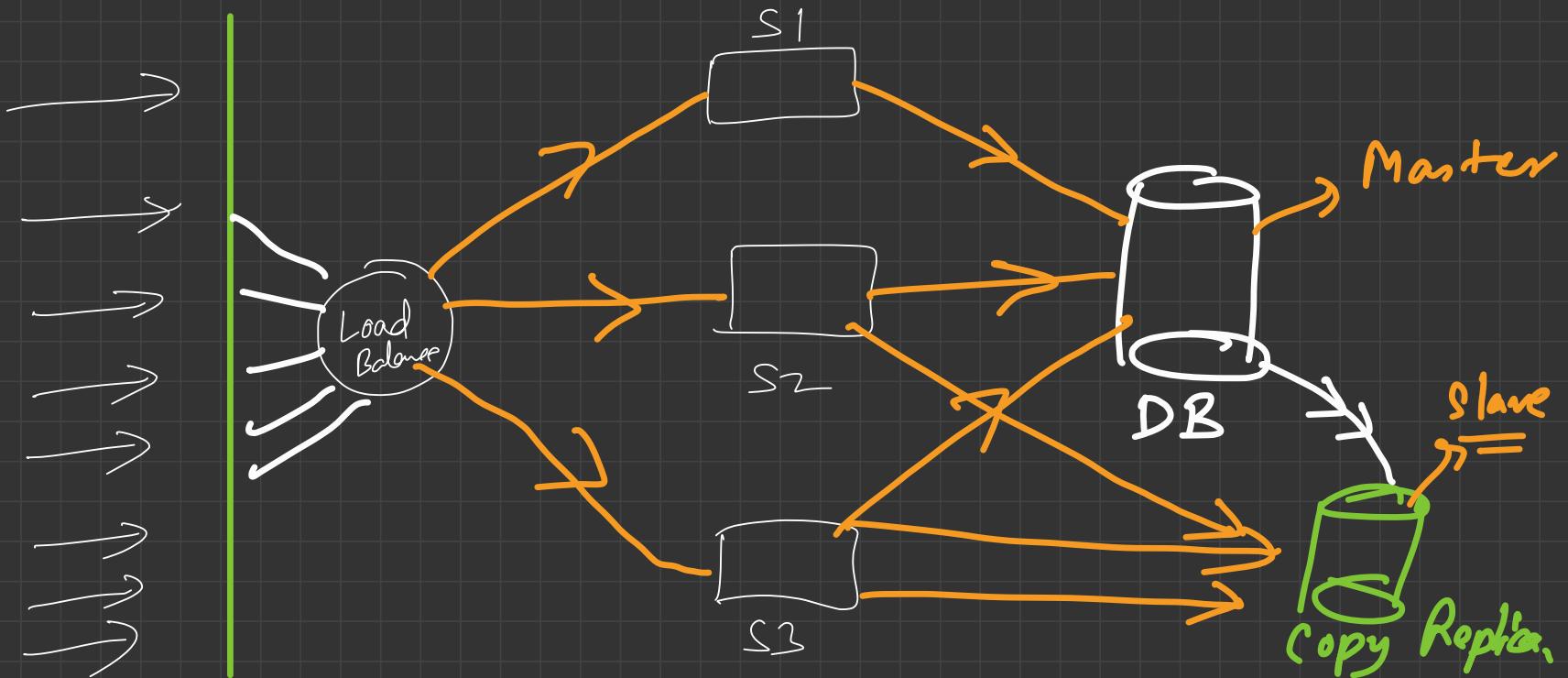
Eventually consistent

Comments
Brief



Lec-21

* Master - slave Architecture **



* Single Point of failure

→ Master node → write operation

→ Original DB

— Latest DB

→ Owner DB

→ Primary DB

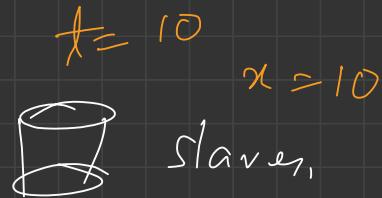
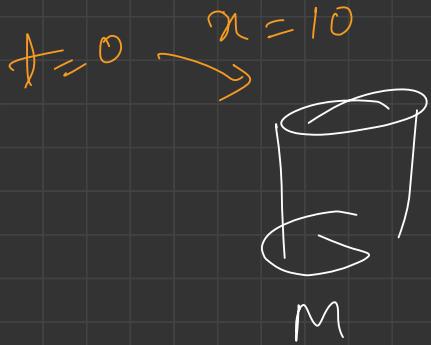
→ it has latest info.

— Primary Latest update

→ Slaves / Replicas → - Read operation

- DB Replication (has Replicat)

How Replication happens ?



→ ① Asynchronous / A sync.

→ FB Comment / Likes

② Sync.

→ Banking APP.

Q

Slaves gets update query ??

①

ignore / never allow slaves to accept
update

②

Allow it, & you have to make a
way to propagate it to master.

M - S

M - M

⇒ Lec-9 Pattern 4

* Advantages

- ① Backup
- ② Scale out read op.
- ③ Availability
- ④ Reliable
- ⑤ Latency Reduce

DBMS Placements Series 2022

Over

①

MySQL

~~Star R~~

Ber+

All the

placements

Rem's e

→ Dustans
S&L

~~☆☆~~ Googling ✓

instagran

✓ Lakeshore k12
— loveBaBBaR codehelp