Lec-1

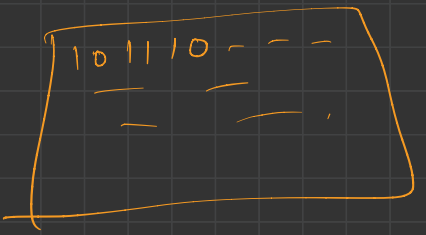
DBMS → ???

⇒ Data ?? ⇒ Bits / Bytes ⇒

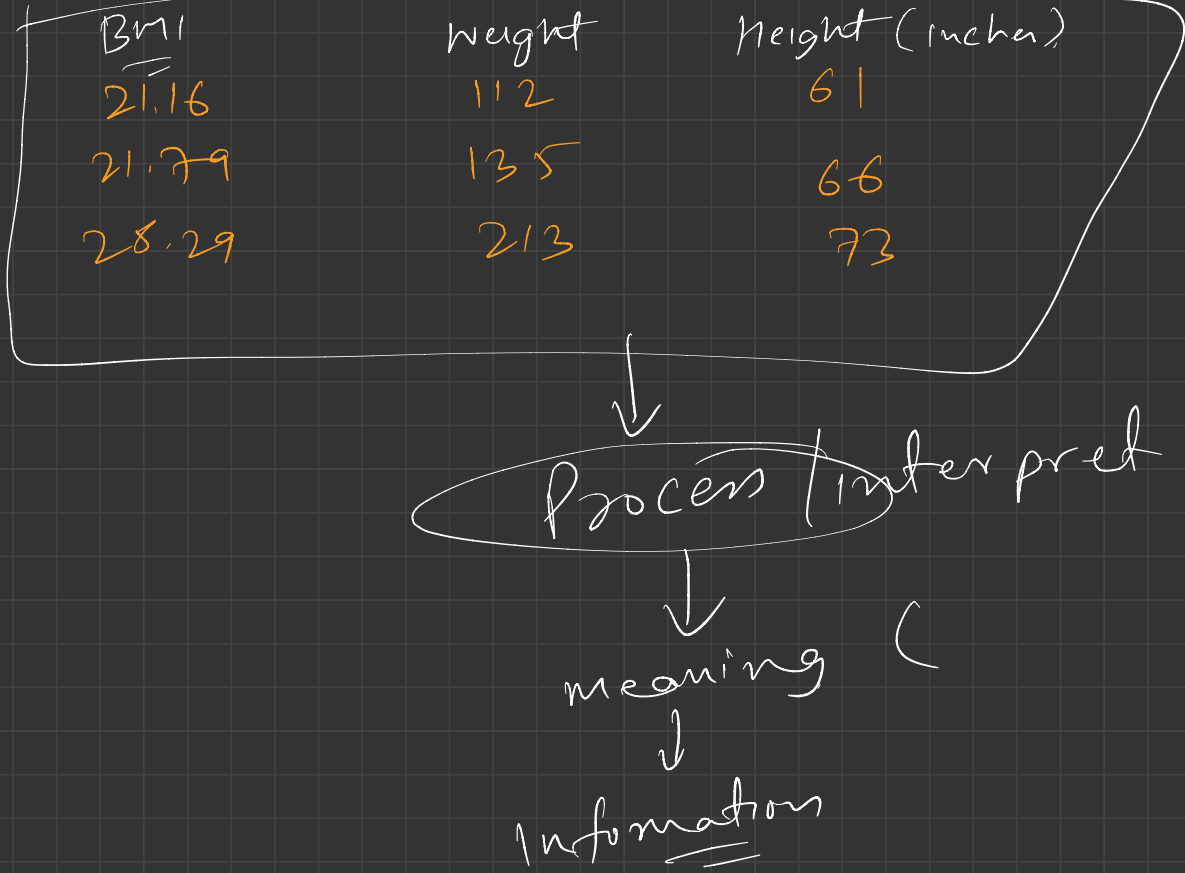
↳ collection of raw bytes

image → Collection of bytes.

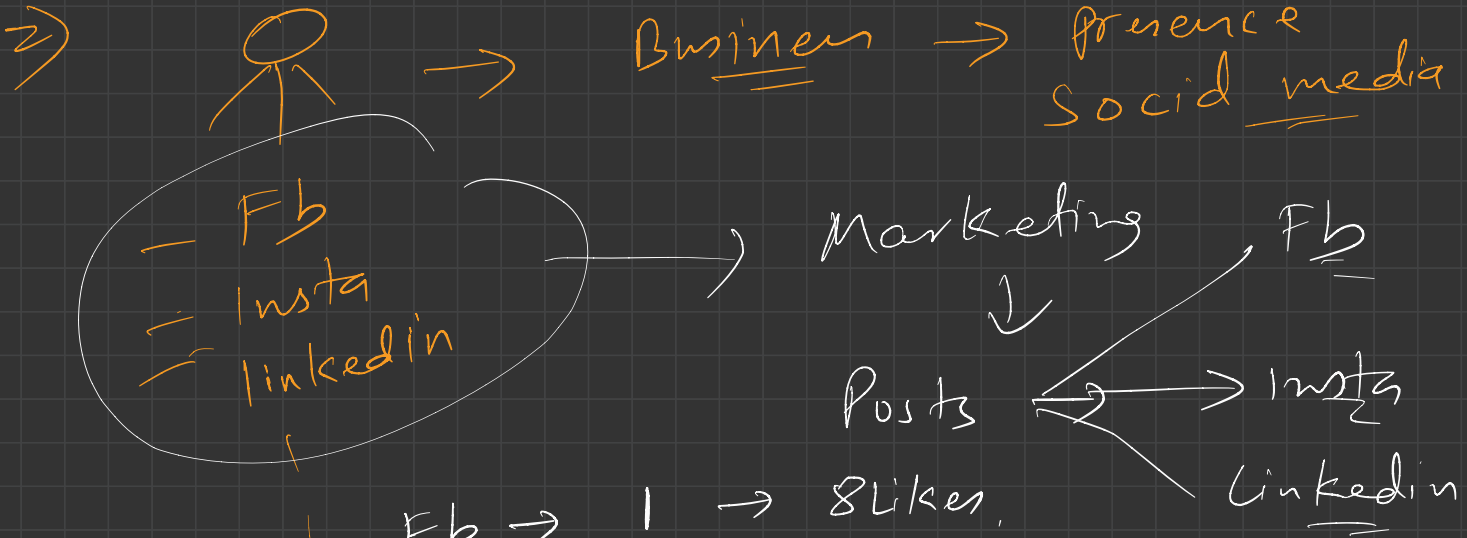
Integers → " " "



memory
HDD


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 → Likes count → 4 bytes integer

↓

Process

↓

Conclusion → Insta → max. likes

↓

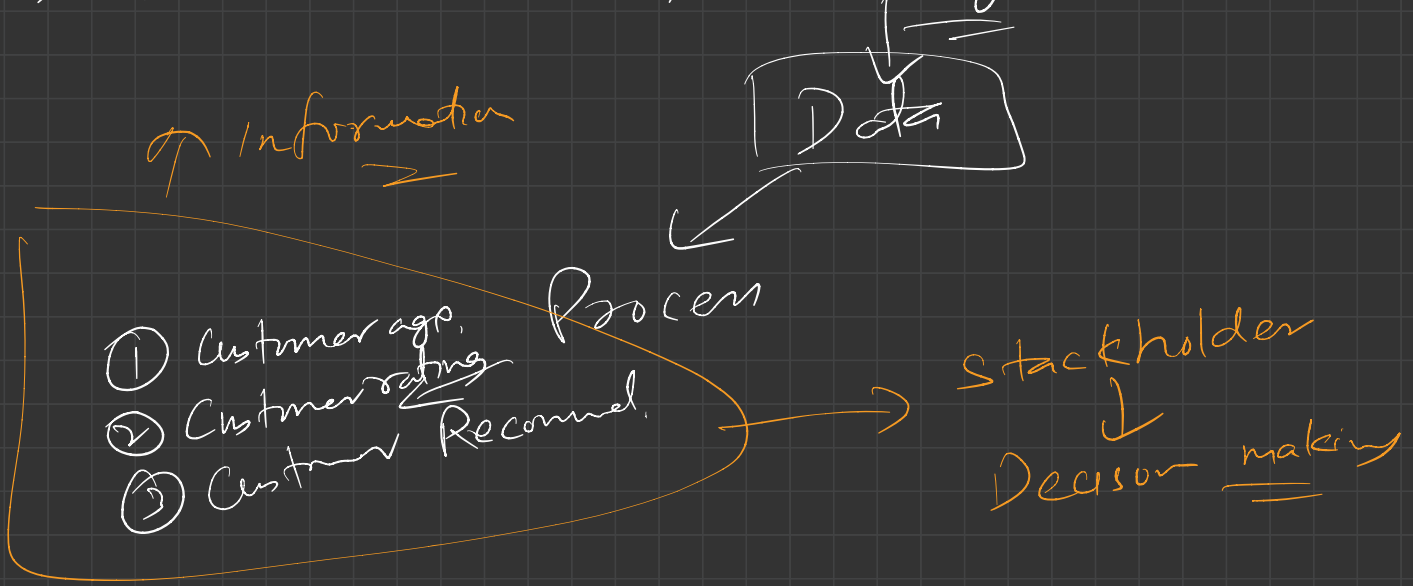
Decision making

↓

I should focus on Insta.

⇒ Amazon ⇒ Product ⇒ feedback

⇒ Feedback ⇒ text ⇒ (bytes)



→ Data is the new oil.

Person A

→ 10 Dishes

— Feedback X

→ Data X

Person B

10 Dishes

FB ✓

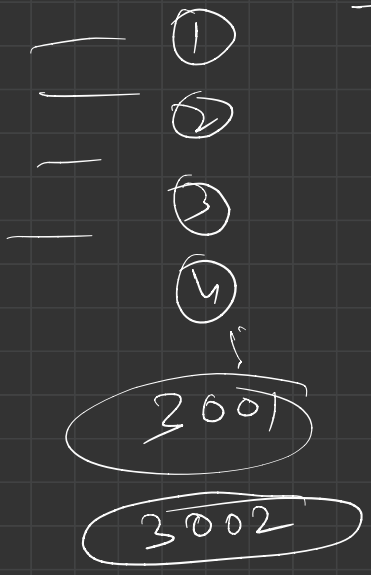
Data collect ✓

Database :-

→ DBMS →

1. DB → (store)
2. set of programs
   - → access
   - → add
   - → update
   - → Delete

school students



addition method

→ File system :-

① earlier :-

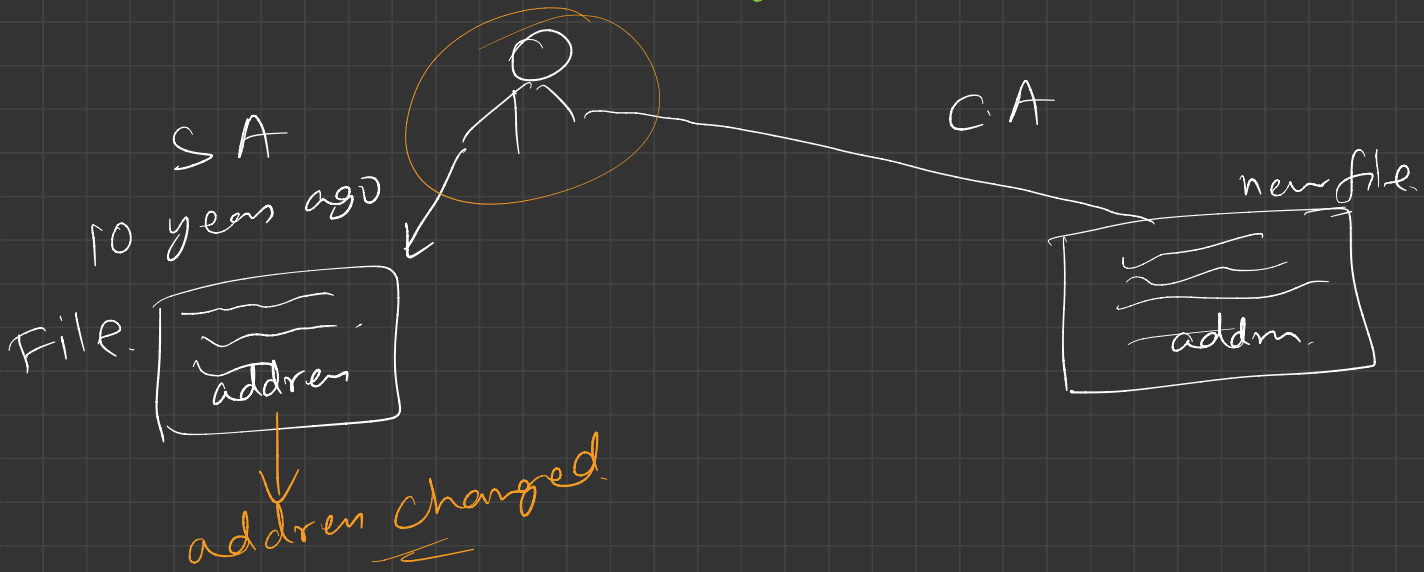
→ Bank → saving account

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

→ Bank → after 10 years → current a/c

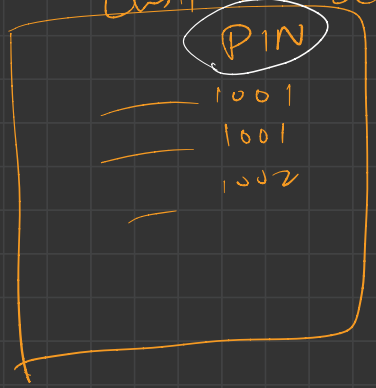
- Investment

① Data Redundancy & Inconsistency



② Difficulty in Data access

Customers data



→ Info

1001 PIN

Lec-2

*Abstraction → DBMS

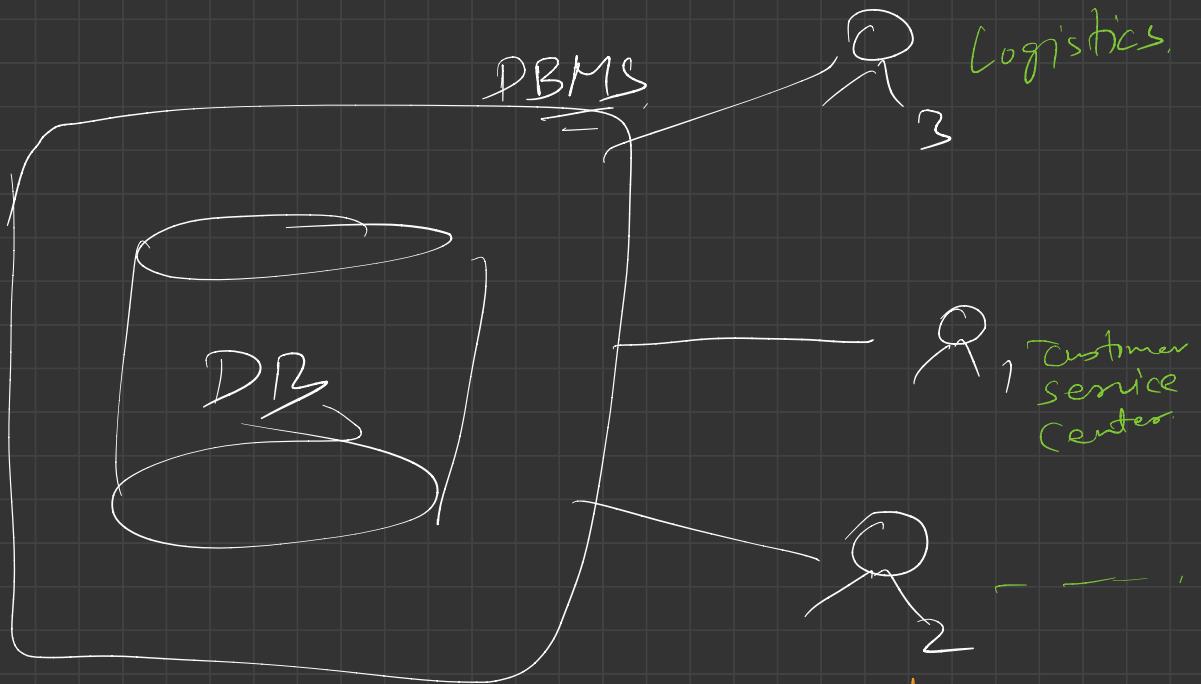
① => car driving

② Tally - Business soft

2) DBMS = Main ID

Amazon

Customer
- Name
- Add
- Phno.
- liking/disliking
- Age
- CC
- DC
- UPI
- Product bought
- ...



Logistics → view schema.

Name	add.	phno.

Customer service dept ⇒

Name	address	phno	Product bought

* Three schema architecture (view of data)

=) F.b → Profile picture image

↓

Collection of bytes → 100×100 → 1MB.

≈ Pixels.

↓

Compression

↓

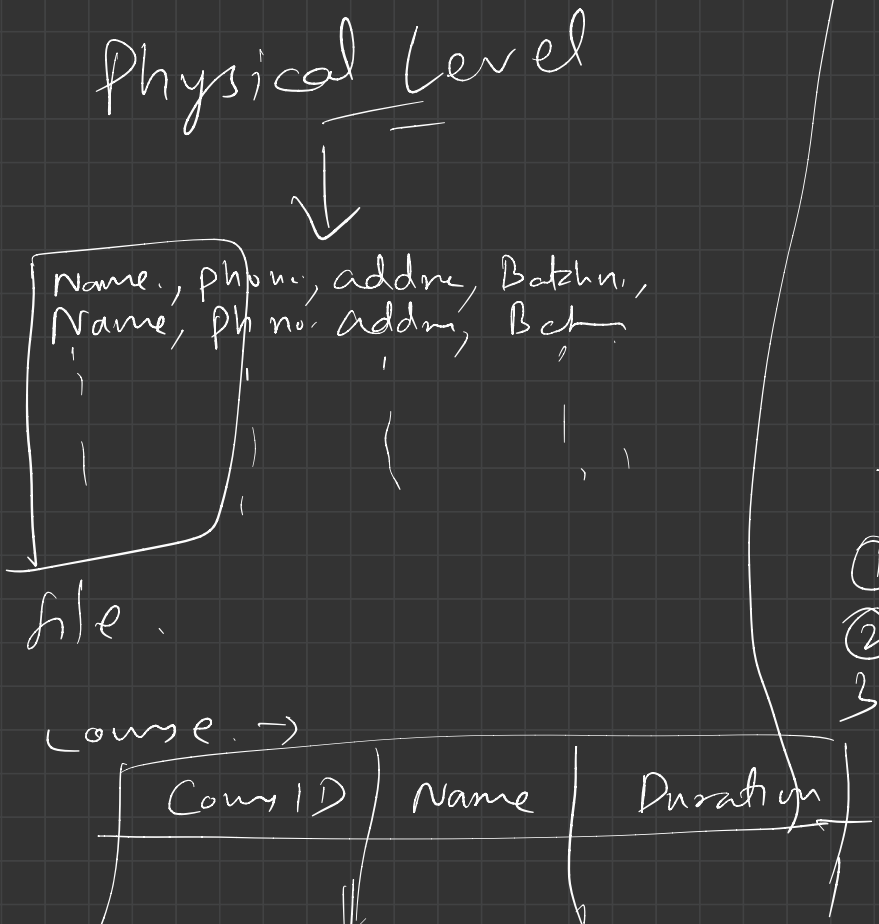
Run length encoding

↓

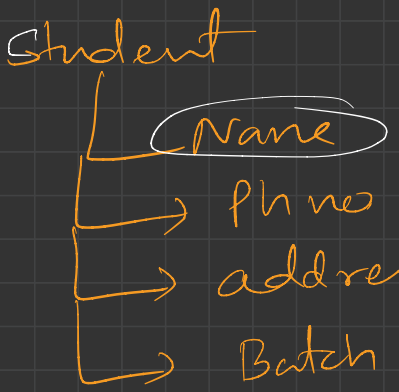
50KB.

* Conceptual Level.

Physical Level



Logical Level



Student

	Name	Ph no.	addrn	Batch.
①	xyz	8	—	2019
②	Lakshy	—	—	—
3				

Course →

CourseID	Name	Duration

CourseID

* Instance of DB

Student =>

	Name	ph	addres	Batch	Stu-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 → use → Language.

=> schema define → student → stu_ID, Name, add, Ph. ...

→ DDL

* Insert, delete, update, Retrieval.

stu_ID	Name	add	Ph
①			
②			
③			

* Data Mani. Laungge

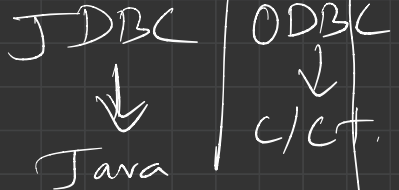
DDL → create table students (stu_ID int,
Name varchar(50),
add varchar(90),
ph. int
);

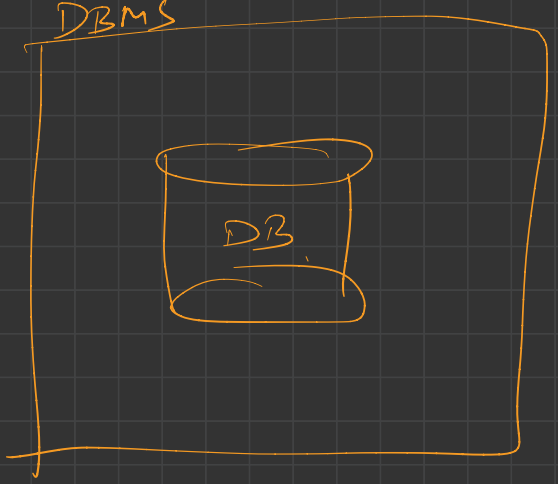
DML → select * from students;

* How app accem DB ?

App :⇒ Javascript, C/C++,

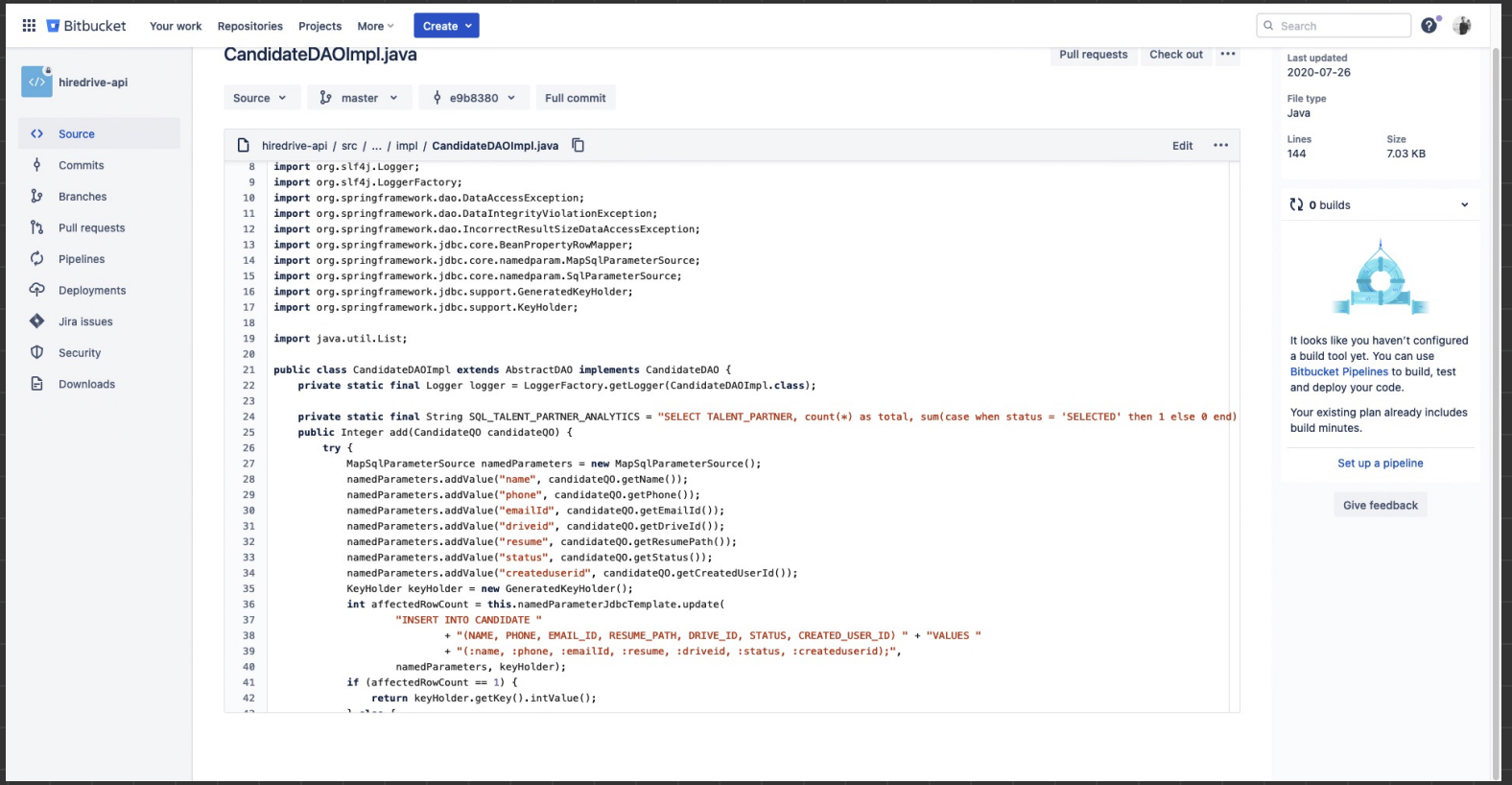
function lastOrderDetails {

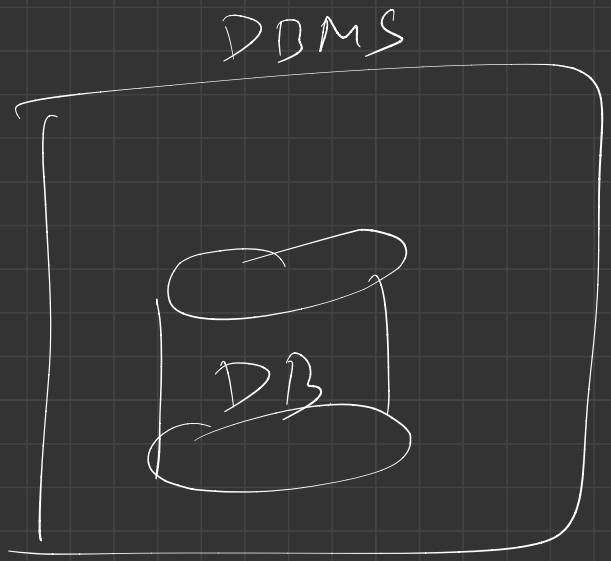




}

interface


Bitbucket
Your work
Repositories
Projects
More
Create
Search
hiredrive-api
Source
Commits
Branches
Pull requests
Pipelines
Deployments
Jira issues
Security
Downloads
CandidateDAOImpl.java
Pull requests
Check out
Source
master
e9b8380
Full commit
hiredrive-api / src / ... / impl / CandidateDAOImpl.java
Edit
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();
Last updated
2020-07-26
File 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



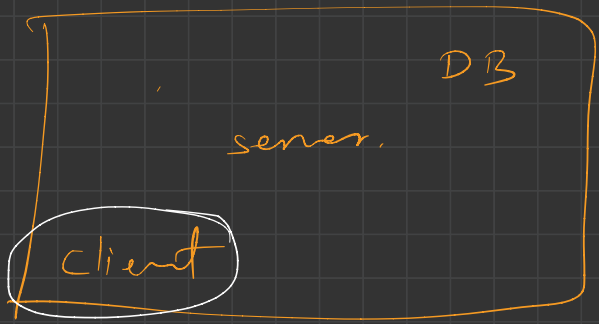
Control -> - data ??
— Data access APIs

* DBMS Application architecture

client machine

Server machine

① T1 archi =>



eg ① sql. learn.

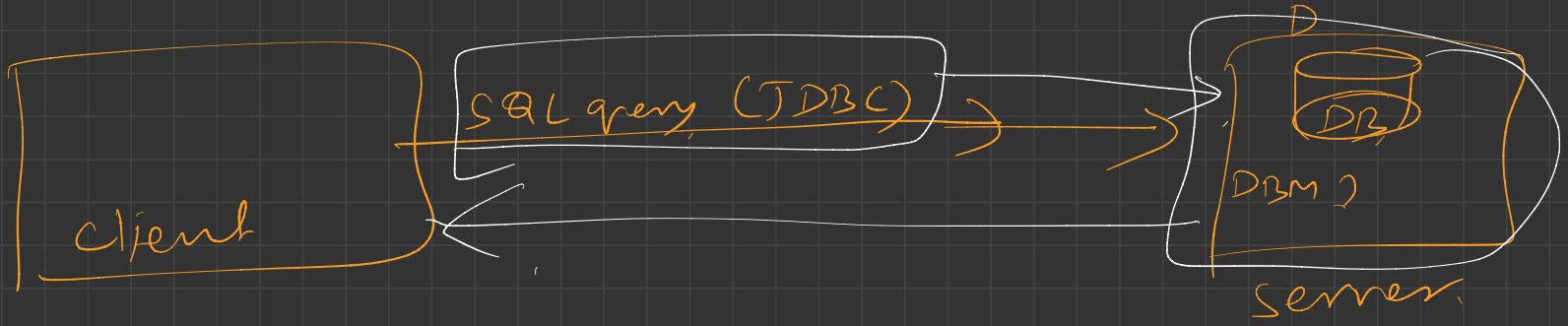
end-user → PC

DBMS server → PC

↓

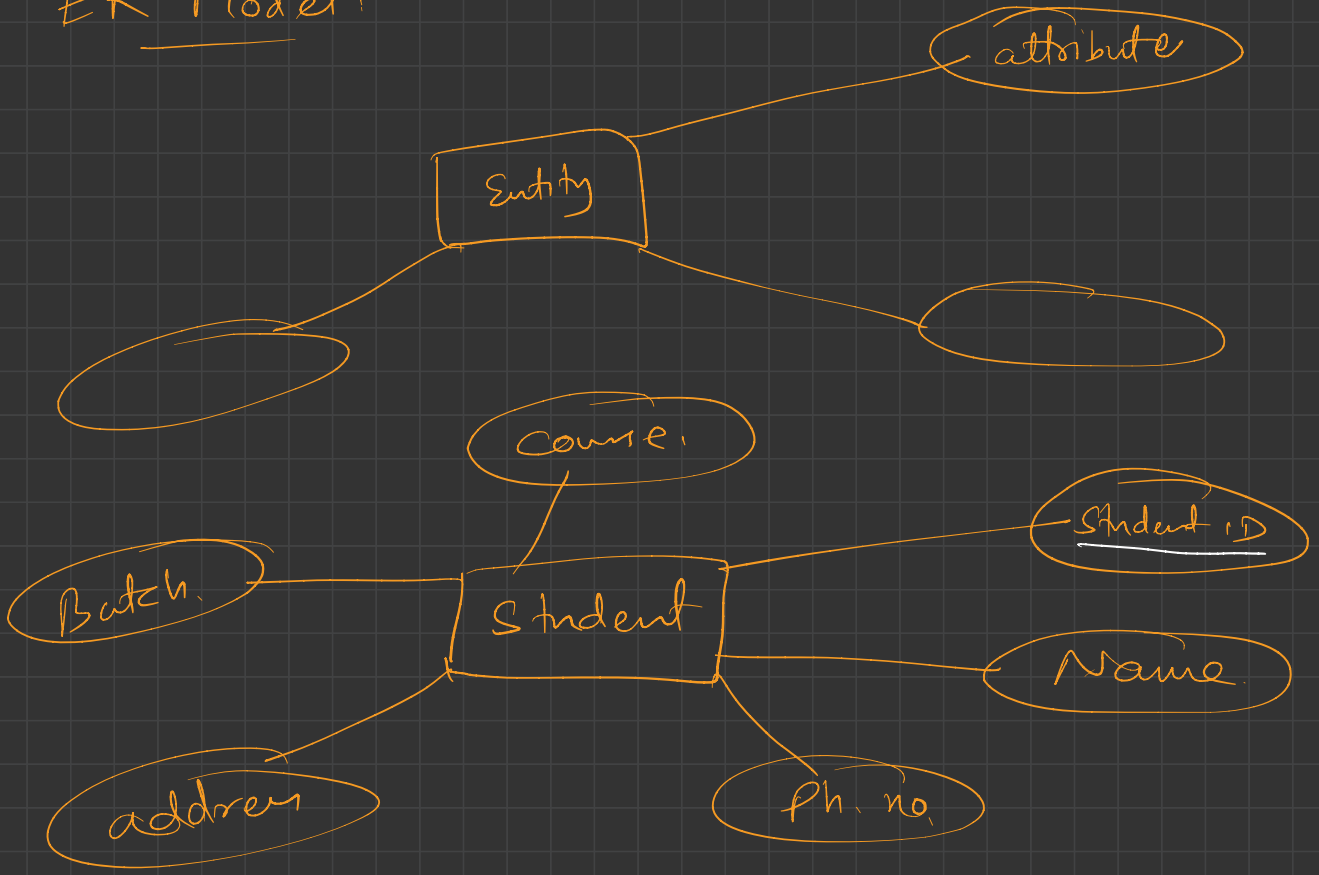
DB → PC (files)

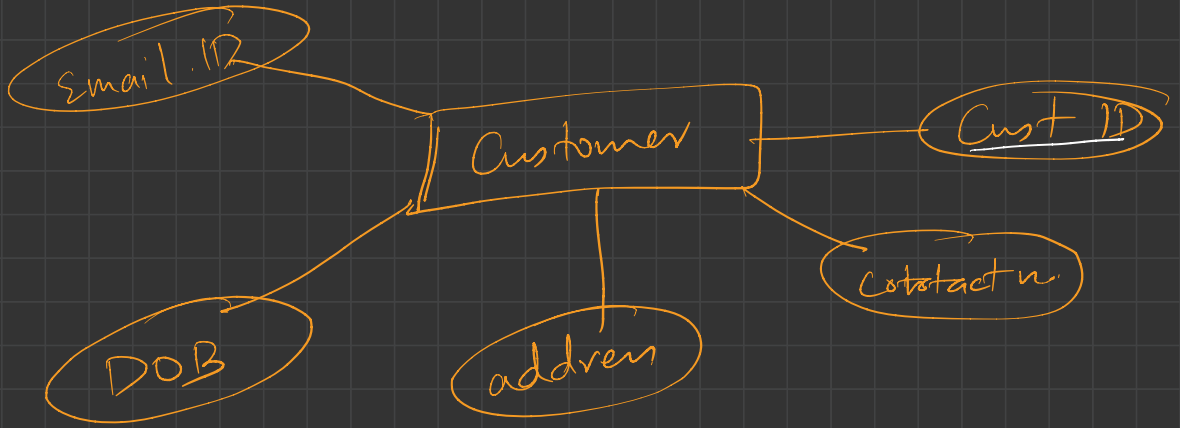
(2) T2 >



Lec -3

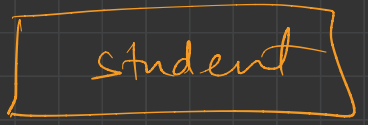
ER Model:





→ Unique attribute → Primary Key

Entity Set →



Attributes

Relationships :- Banking,

Customer , Loan , Account.

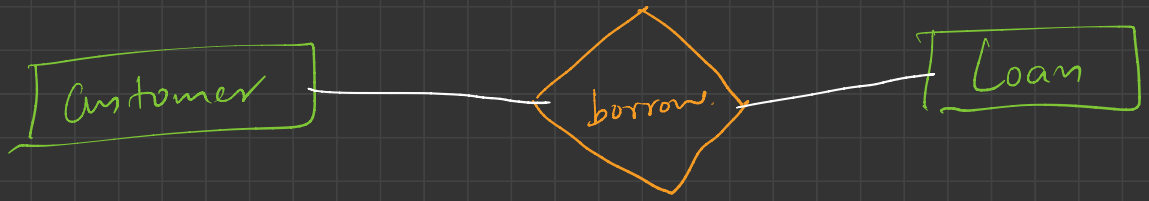
1. Customer borrow Loan.
2. Customer deposit Account.

=> Customer places order.

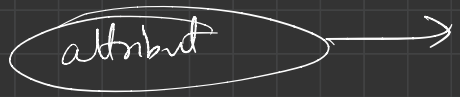
=> professor takes course.

=> student enroll Course.

=> Citizen has vehicle.

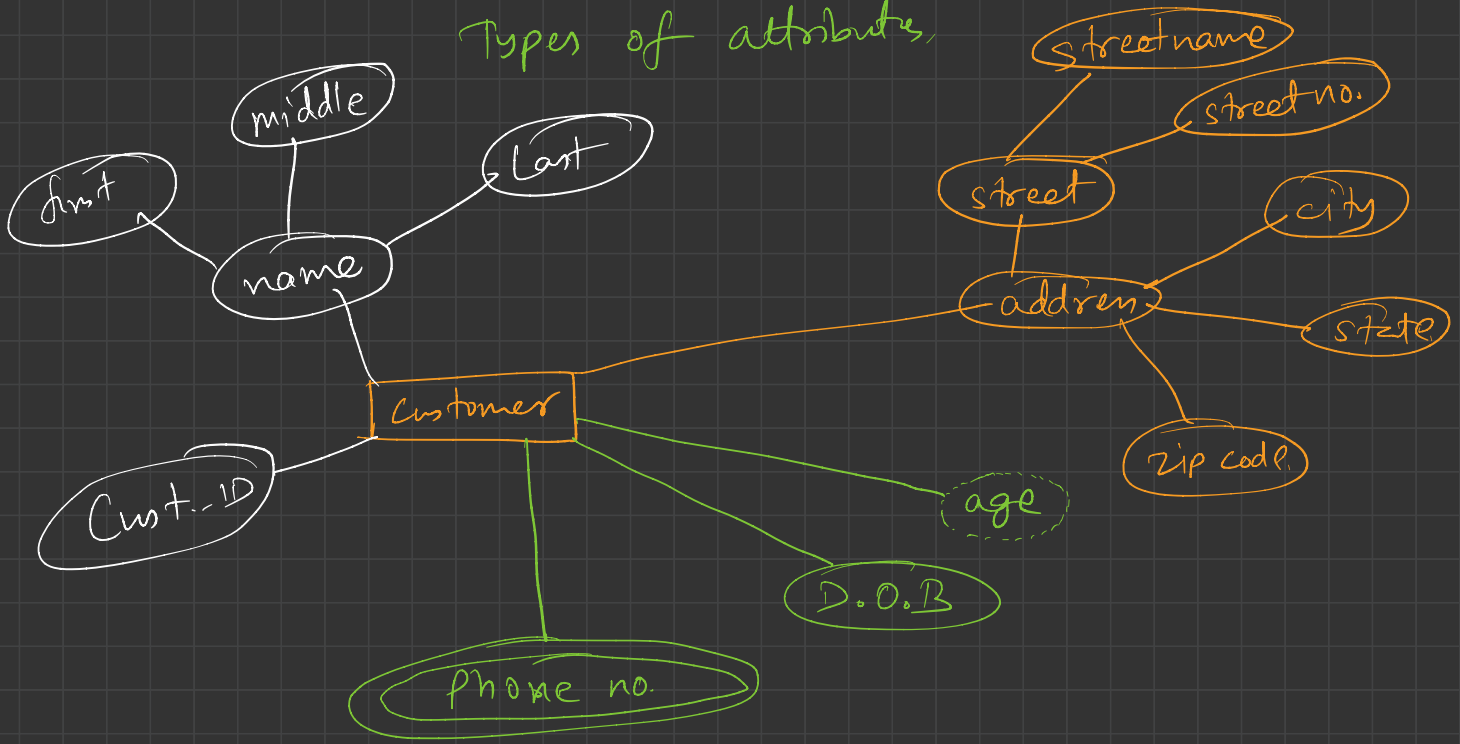


* Attributes



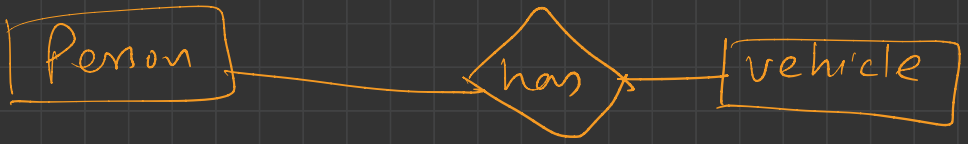
domain, values
permitted / non-permitted,

* Types of Attributes


Types of attributes
middle
first
Last
name
streetname
street no.
street
city
address
state
Customer
Cust.-ID
zip code
age
D.O.B
phone no.

* NULL value

Relationships :- association among 2 or more entities



Strong Relationship

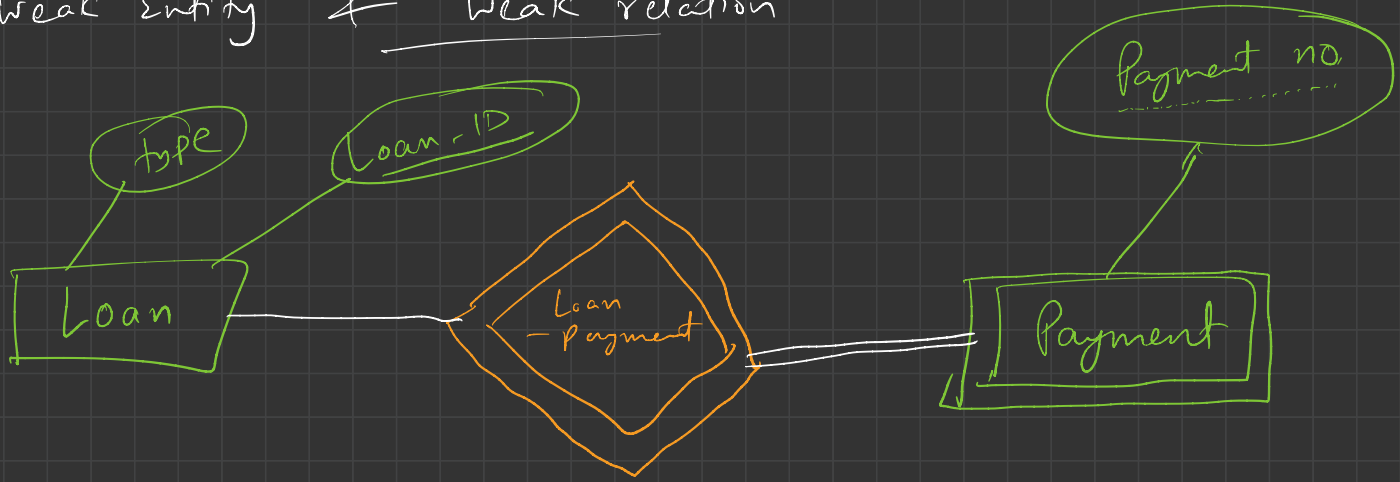
Weak Relationship

University system

Strong Entity

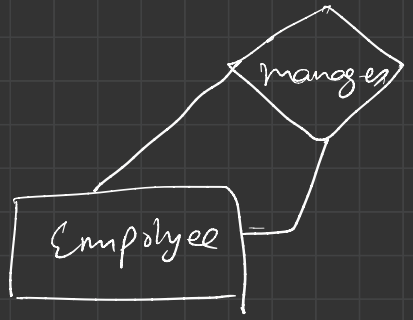
Weak Entity

weak Entity & weak relation



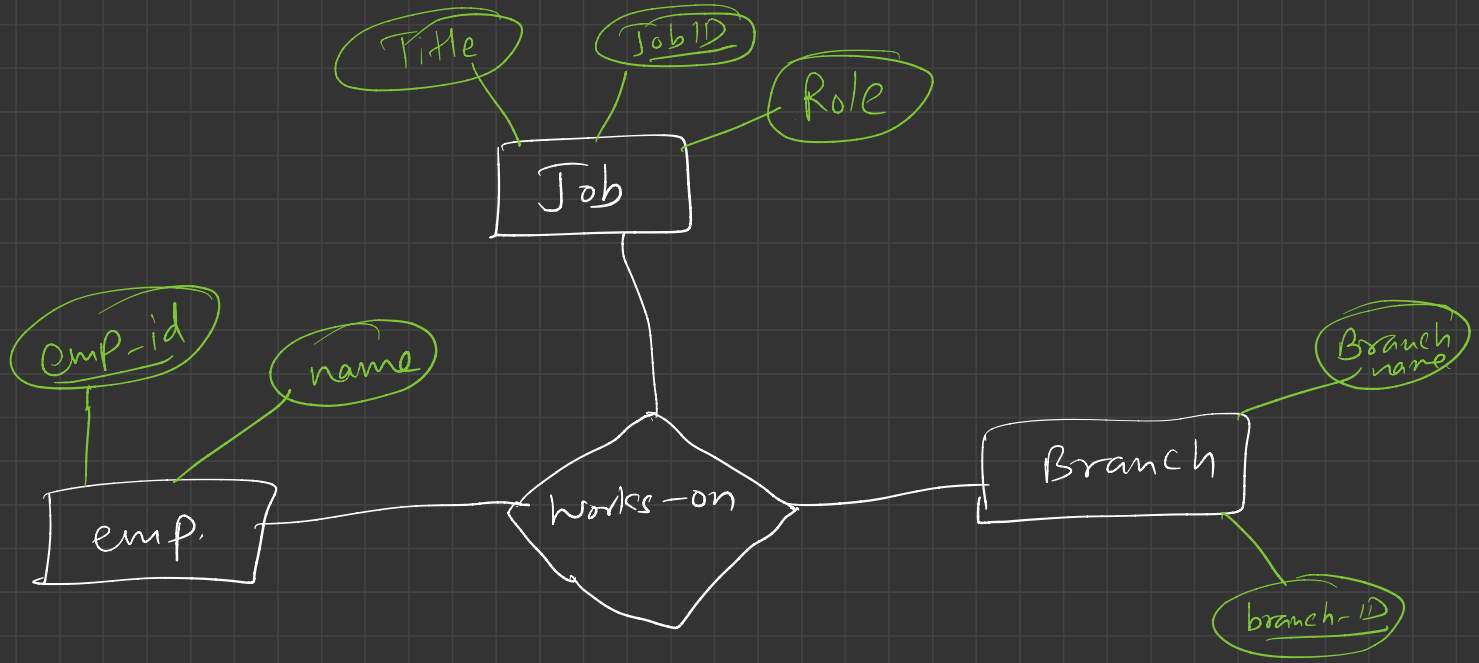
Degree of relⁿ :-

① Unary :->



② Binary relⁿ :-

③ Tenary relⁿ :-

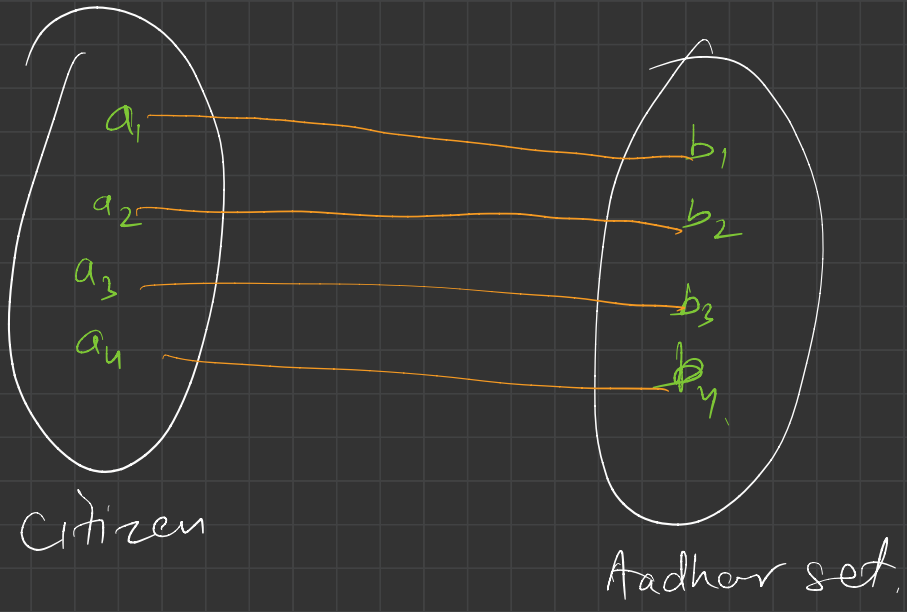


Ternary Rel^N

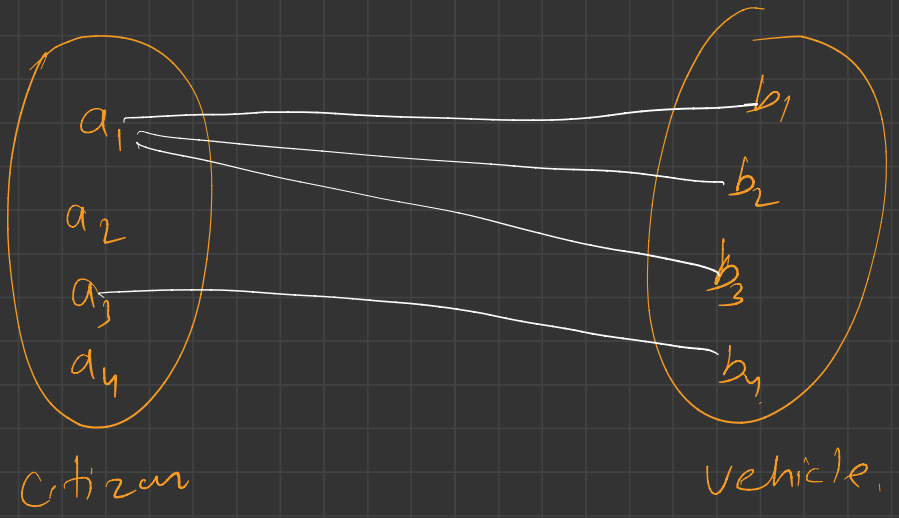
* Relationship constraints

① Mapping cardinality

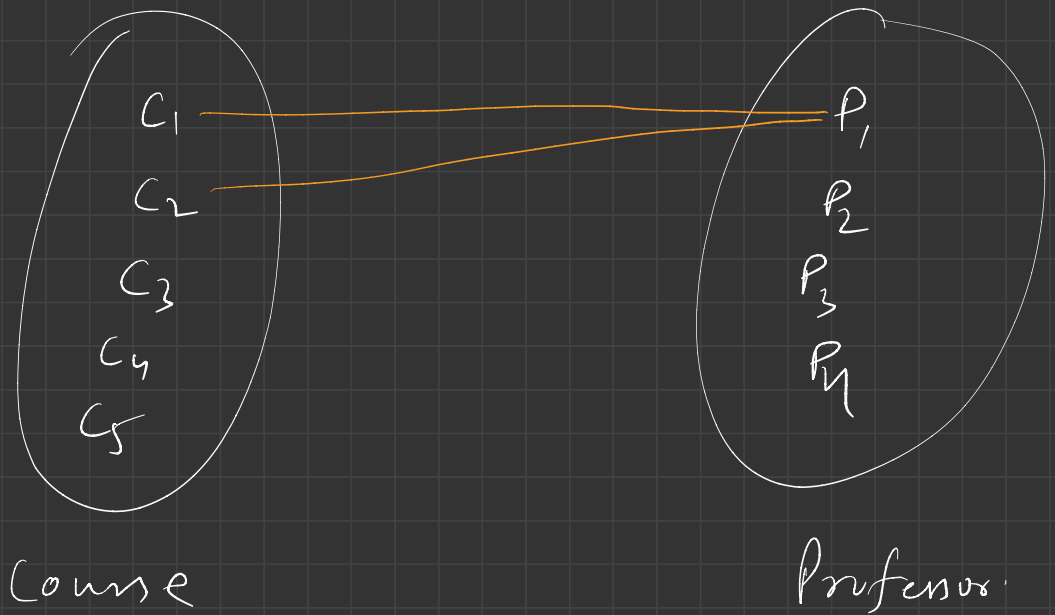
① 1 : 1



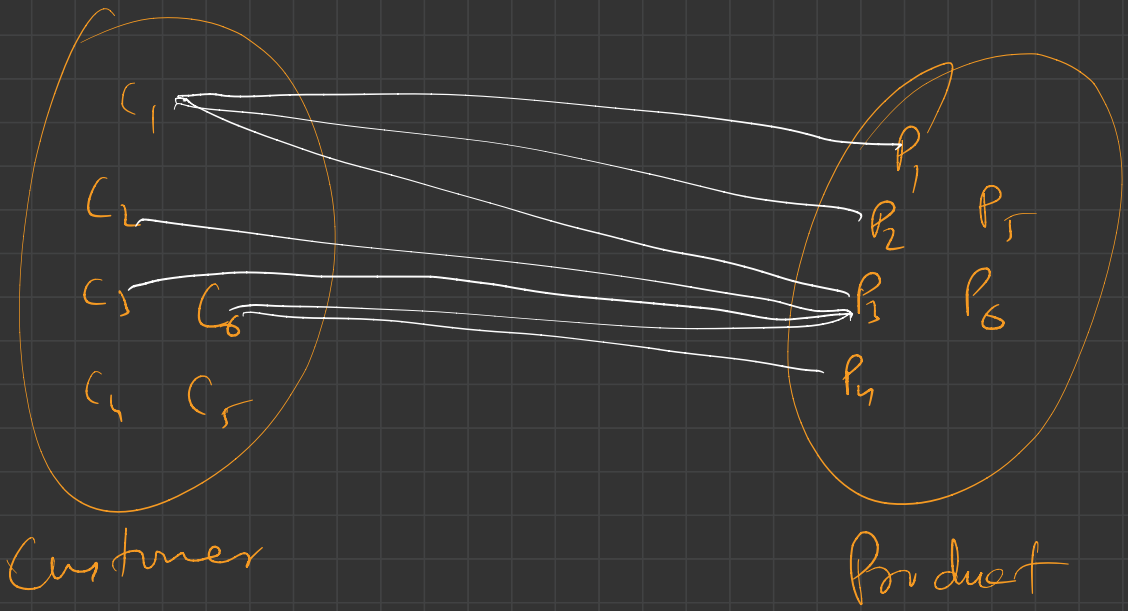
② 1 : N



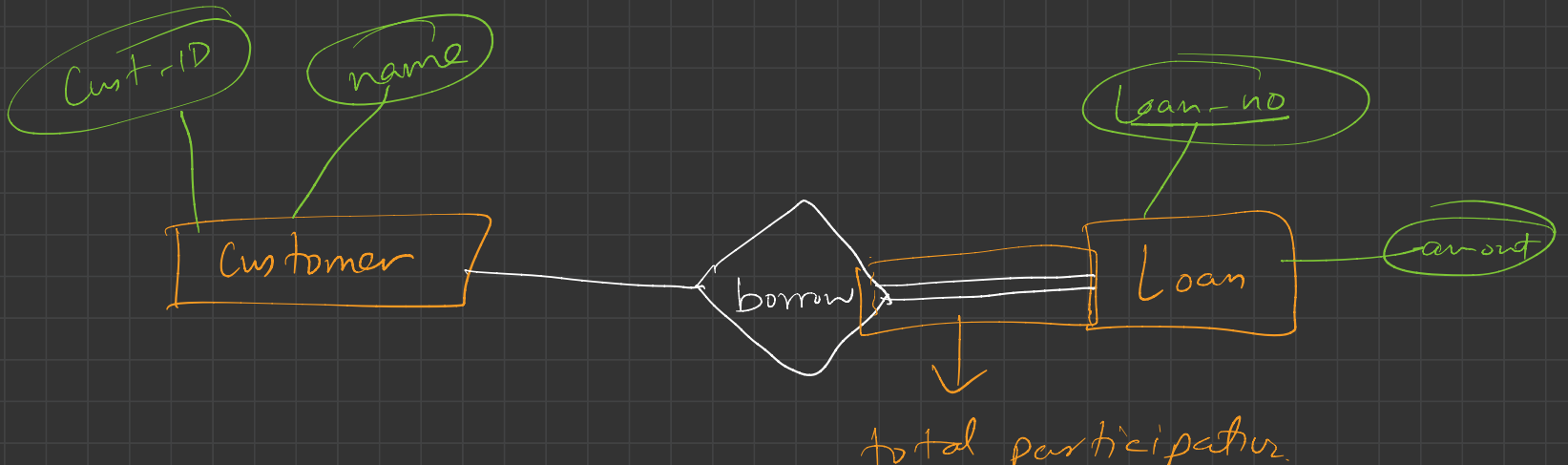
(3) N:1



(5) N : M



* Participation Constraint



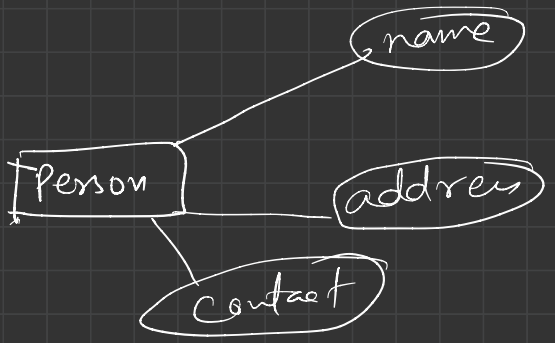
Cust ⟶ Loan → Partial Participation.

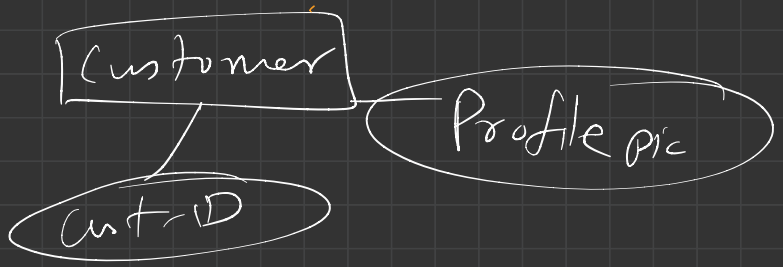
Loan ⟶ customer → Total "

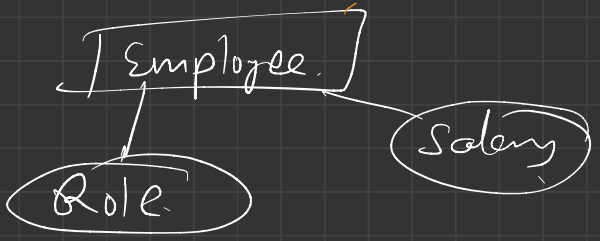
Lec-4

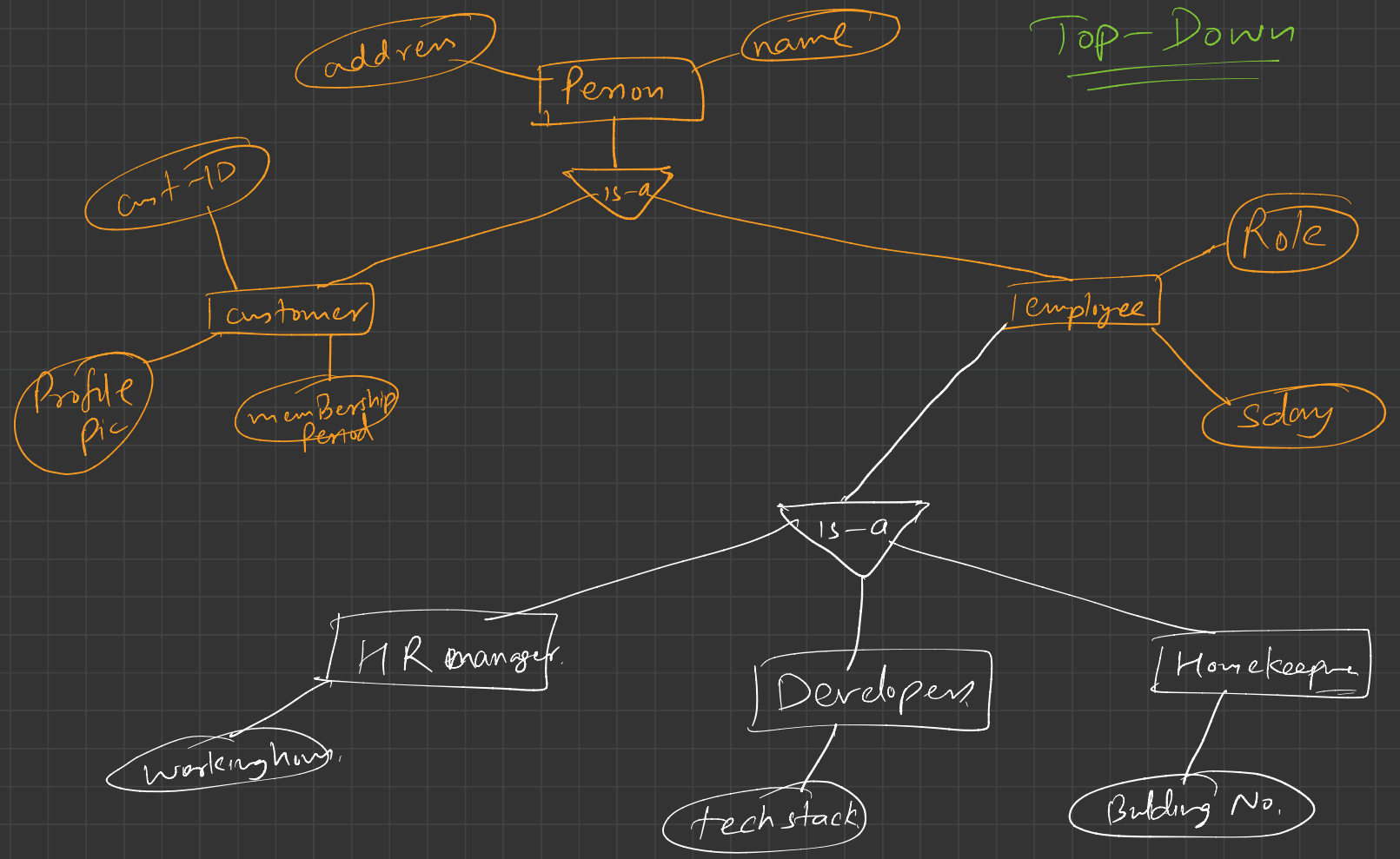
Extended ER Features

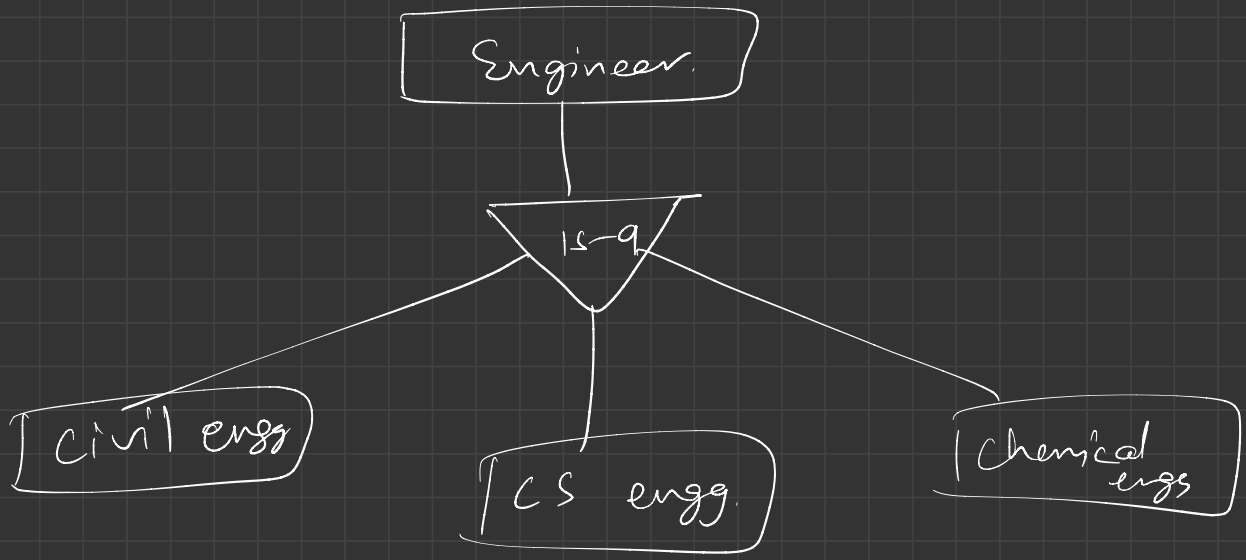
① Specialization

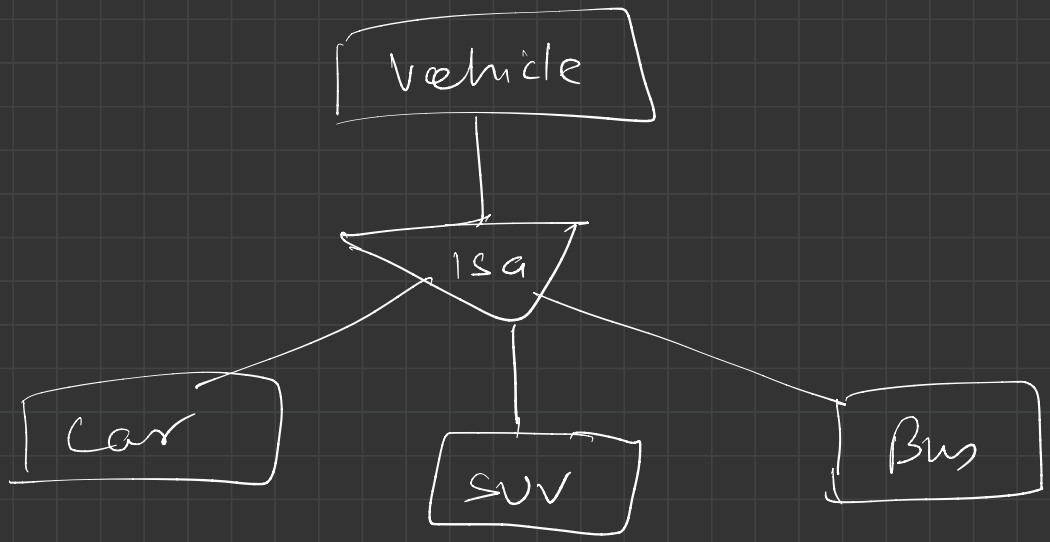






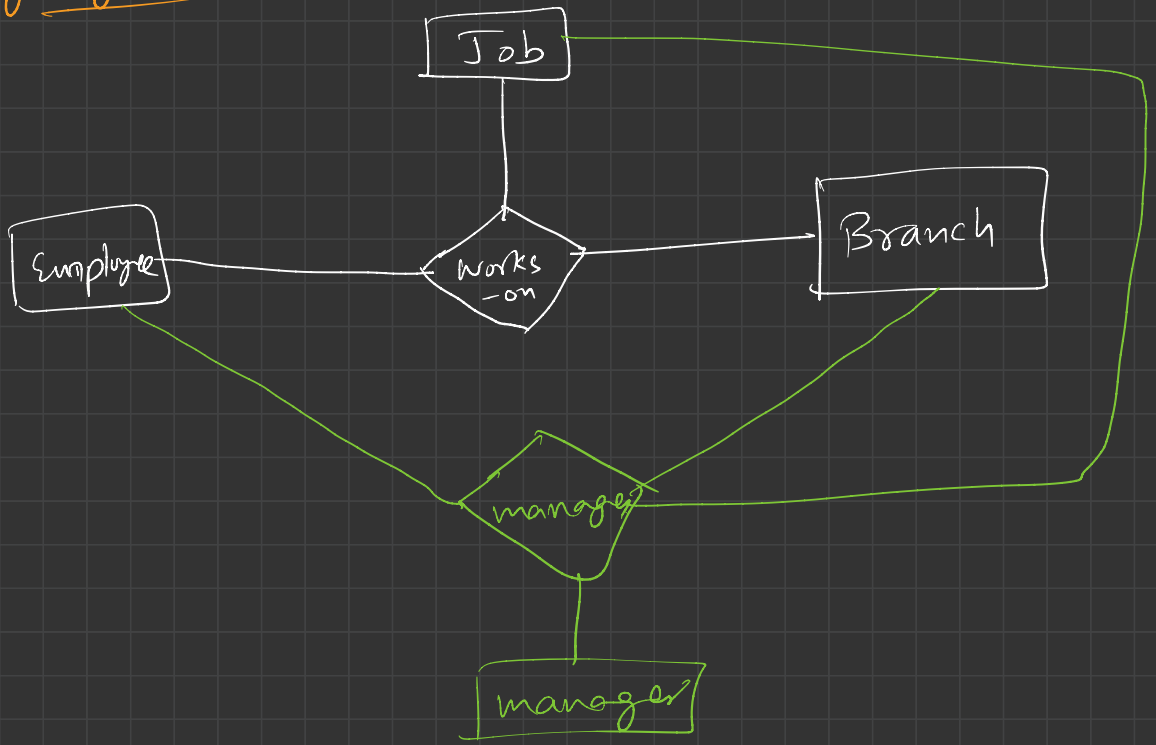

Top-Down
address
name
Person
is-a
Cust-ID
customer
Profile pic
membership period
Role
employee
Salary
is-a
HR manager
working hours
Developers
tech stack
Housekeeper
Building No.

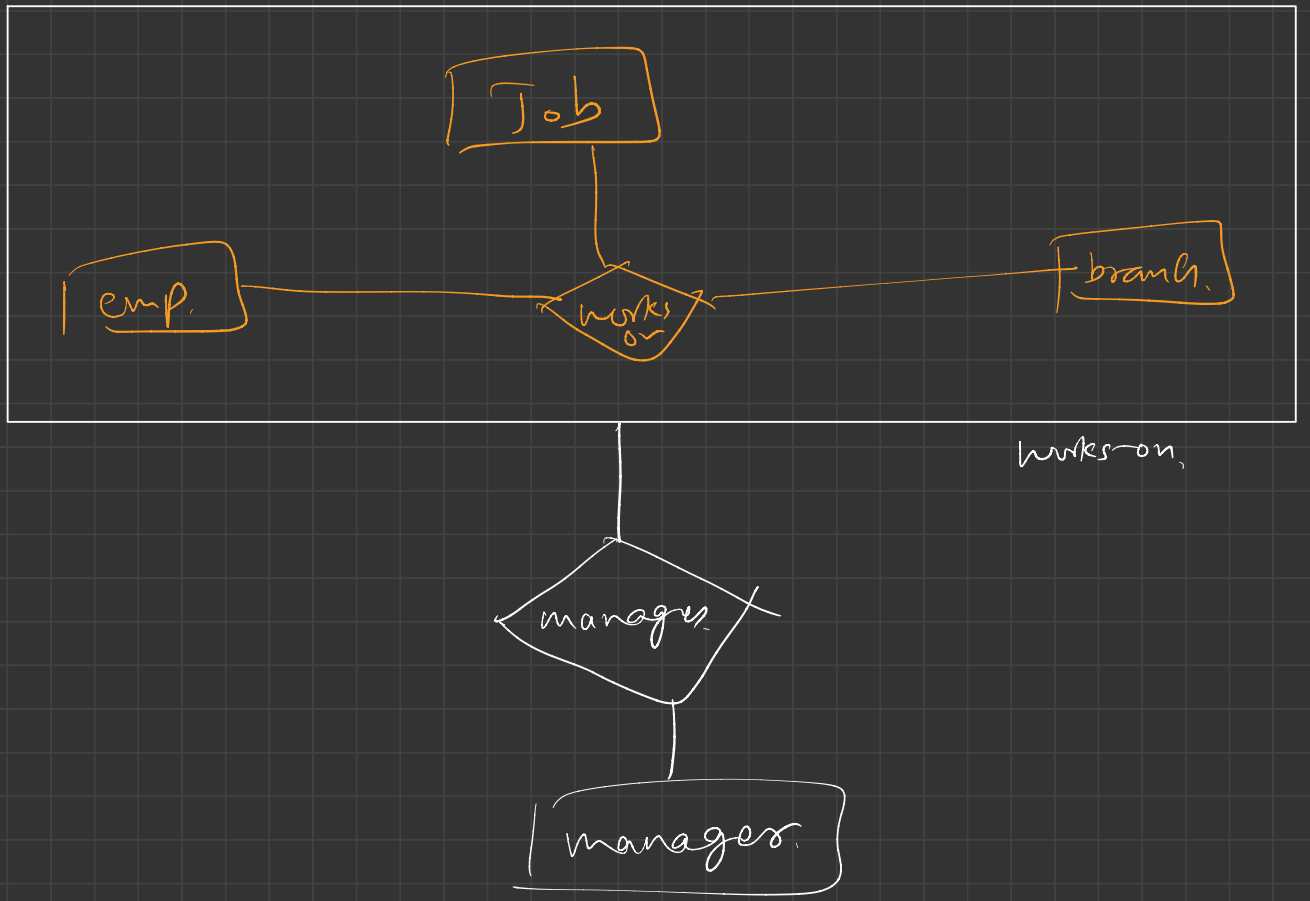

Engineer
is-a
civil engg
CS engg
Chemical engg


Vehicle
Isa
Car
SUV
Bus

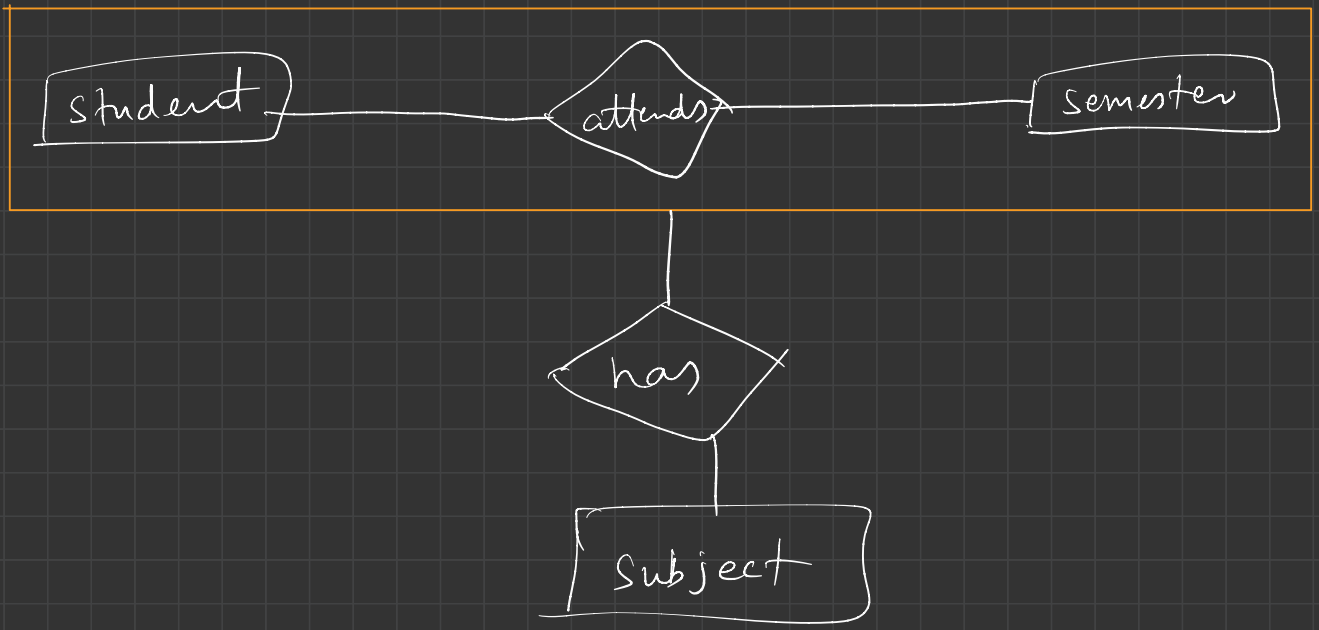
② Generalizations

Aggregation




Job
emp.
works on
branch
works_on
manager
manager

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. (many)
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,
age → derived.
address → Composite.
contact no. → multivalued.

③ Employee → emp-id, name, contact no., dependent name,
years of service, start-date
↓ derived attribute ↓ 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 Rel^n & constraints

① Customer borrow loan.

M : N

② Loan originated by branch.

N : 1

③ Loan loan-payment Payment.

1 : N

④ Customer deposit account

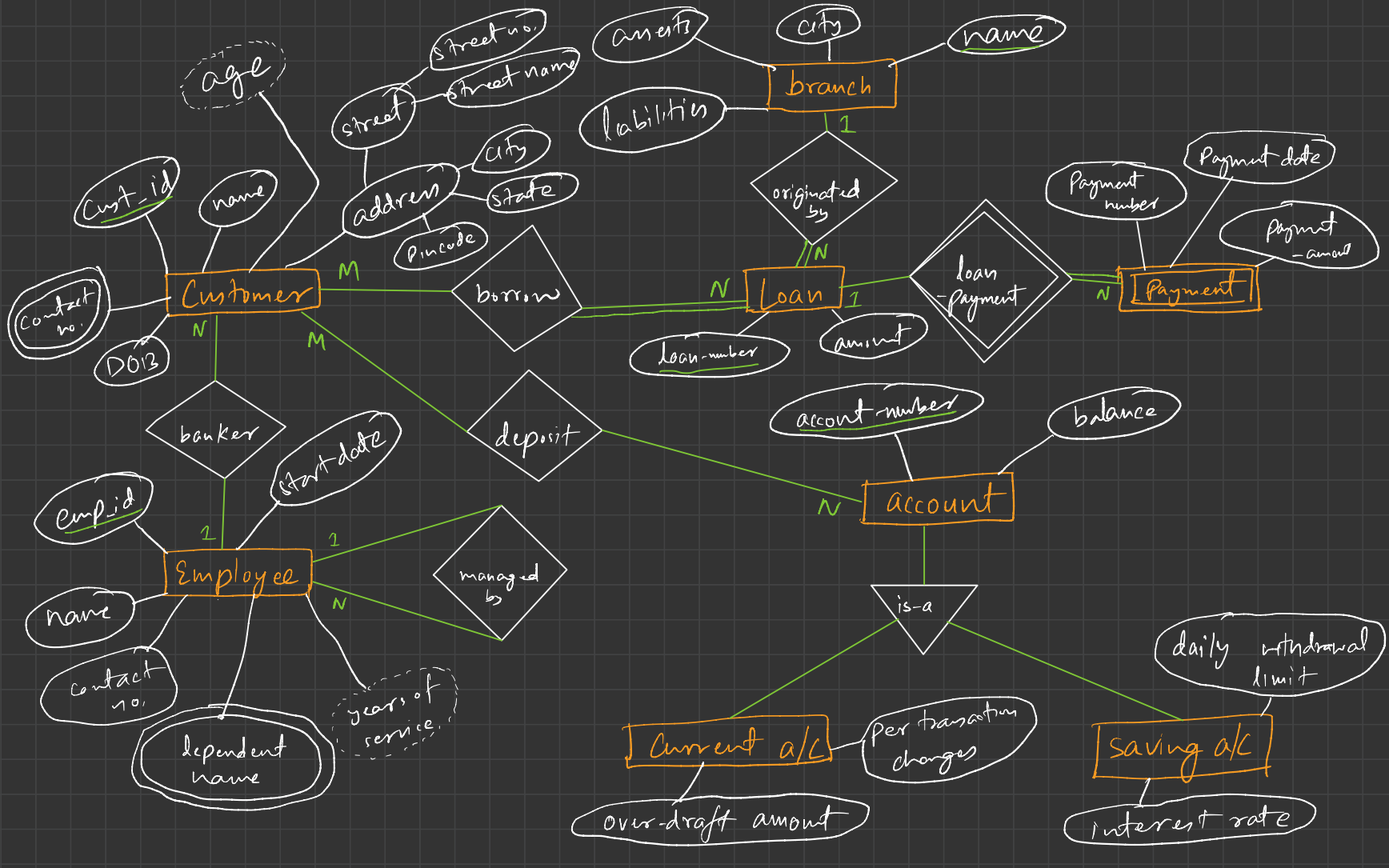
M : N

⑤ Customer banker Employee.

N : 1

⑥ Employee manage by Employee,

N : 1


age
street no.
street name
street
address
city
state
Pincode
Cust_id
name
Customer
contact no.
DOB
M
N
borrow
assets
city
name
branch
liabilities
1
originated by
N
Loan
N
1
Loan-number
amount
loan-payment
N
Payment
Payment number
Payment date
Payment amount
banker
start date
emp_id
1
Employee
1
N
managed by
name
contact no.
dependent name
years of service
deposit
N
account-number
balance
account
is-a
Current a/c
per transaction charges
over-draft amount
Saving a/c
daily withdrawal limit
interest rate

H.W

1. Online delivery system.
2. University

Lec-6

Facebook – DB formulate using ER model

* ER diagram

① Features & use case.

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

② Attributes + types

① user_profile → Name, username, email, password, Contact no.

Name → composite. email → multivalued. Contact no. → multivalued.

DOB, age. → derived.

② user_post → post_id, text content, image (multivalued), video (multivalued),

created_timestamp, modified_timestamp.

③ post_comment → post_comment_id, text content, timestamp

④ post_like → postlike_id, timestamp

③ Rel^n & Constants

① user_profile — friendship — user_profile.

M : N

② user_profile — posts — user_post.

1 : N

③ user_profile can post_likes

1 : N

④ user_profile. comments post_comments

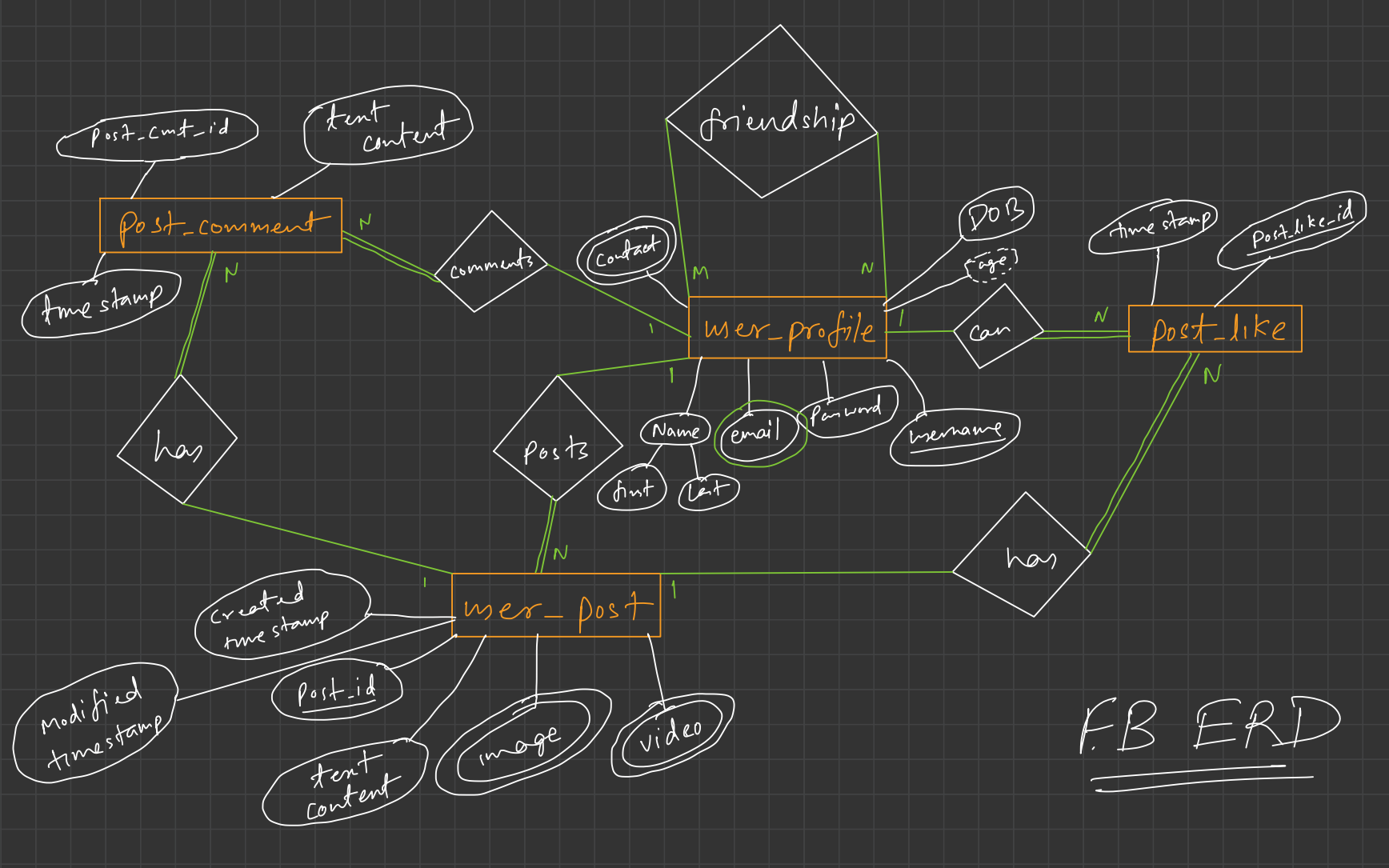
1 : N

⑤ user_post has post_comment

1 : N

⑥ user_post has post_like

1 : N


Post_cmt_id
text content
friendship
post_comment
N
comments
contact
M
N
DOB
age
time stamp
Post_like_id
time stamp
N
has
1
user_profile
1
can
N
post_like
Posts
Name
email
Password
username
first
Last
N
N
has
1
1
user_post
Created timestamp
Modified timestamp
Post_id
text content
image
video
FB ERD

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 ⇒

↓

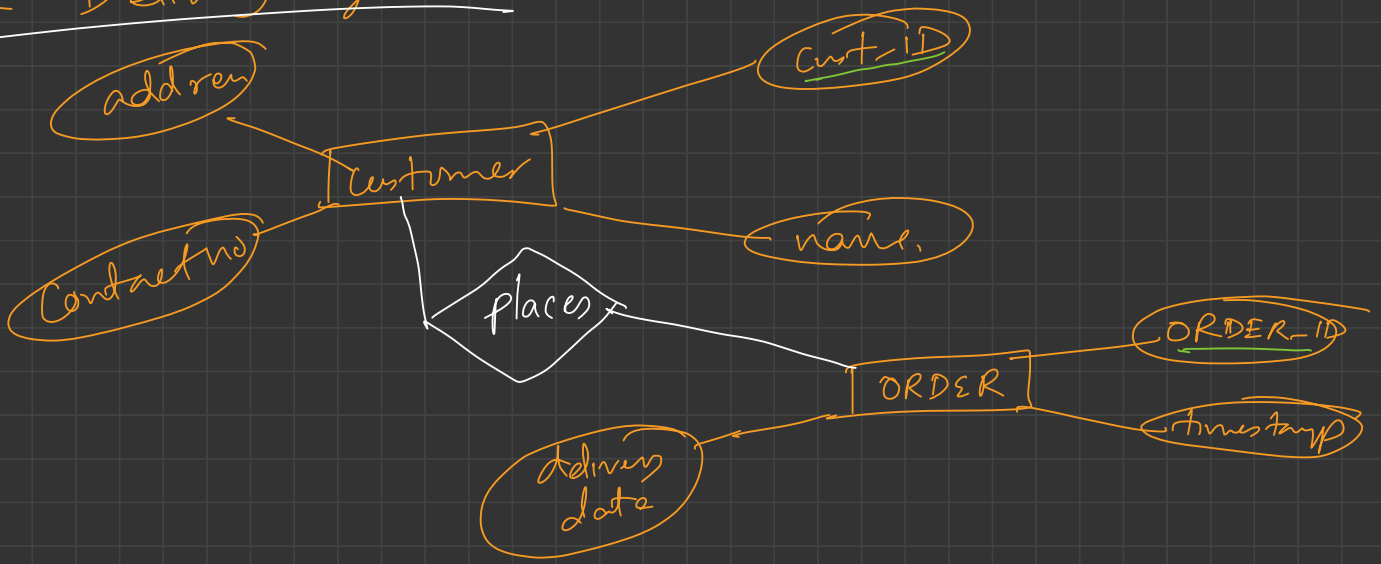
③ RDBMS ⇒ software DBMS

↓

software implementation of RM

MySQL, MS access, Oracle etc.

Online Delivery system



① Customer (Cust_ID, name, address, contact no.)

② ORDER (order_ID, timestamp, delivery_date)

⇒ Key ⇒ Forgein key

⇒ Keys ⇒ Primary key ⇒ each data point uniquely identify

Cust_ID	Name	contact	addrss	email

① Super key ⇒ {Name, contact}, {Name, email}

{Cust_ID, email}, {Cust_ID} {All}

{Cust_ID, Name, contact, email}

{Raj, 888}
{Raj, 811}

② C.K. → {Contact}, {Cust_ID}, {Cust_ID, contact}, {Cust_ID, email}

③ P.K → {Cust_ID} → P.K.

④ A.K → {C.K} − P.K. →

(4) Foreign Key :-

Customer (cust_ID, name, addrss, contact no)

Order (order_ID, timestamp, delivery date, cust_ID → {F.K})

Customer → Referenced Rel^n / Parent table.

Cust_ID	Name	addrss	contact
1	Kaz	- -	- -
2	Jo	- -	- -
3	dada	- -	- -
4	...	...	...

ORDER → Child table. / Referencing rel^n.

Order_ID	Timestamp	delivery date	Cust_ID
21	- - -	- -	1
22	- -	- -	2
23	- - -	- -	3

* Composite key -

A Surrogate key ⇒

Table of School A

reg.no	name
101	Ram
102	Monika
103	Tata.

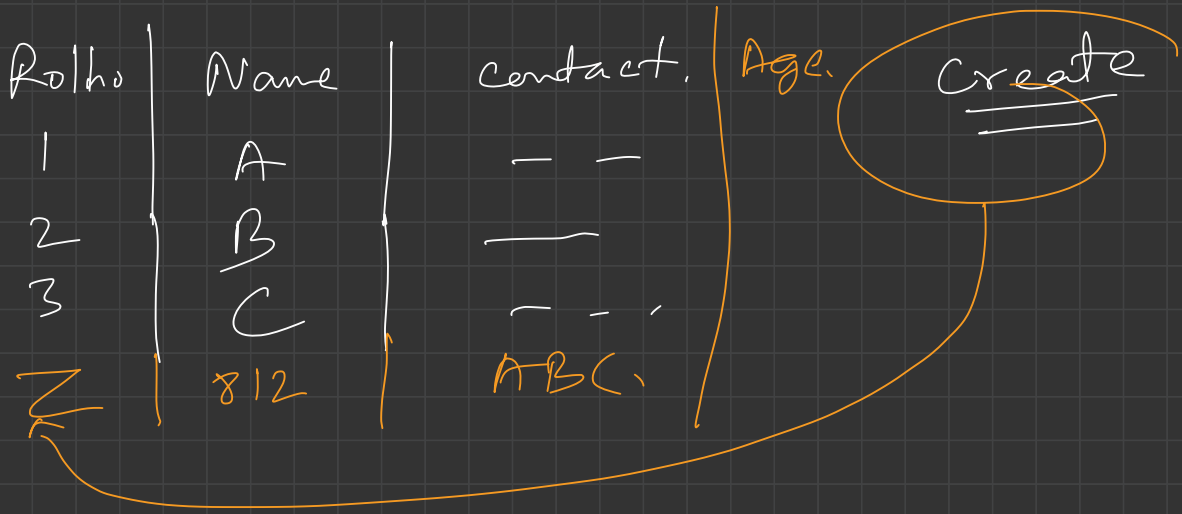
Merged. →

Surr-no	reg_no	name
1	101	Ram
2	102	Monika
3	103	tata
4	AB101	Tommy
5	AB102	Nota.

table of school B

reg_no	name.
AB101	Tammy
AB102	Nota.

CRUD



* Referential constraint

Customer :-
↓
Parent rel^n
Referenced table.

Cust_ID	Name	add	Contact no.
1	Kaz	..	— —
2	Jo	— —	— —
3	Dada		

ORDER :-
↳ Child Rel^n / Refereing table

order_ID	Timstamp	delivery date	Cust_ID → F.K.
21	— —	— —	1
22	— —	— —	2
23	— —	— —	~~3~~ 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 delete 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 NotNull,
    Name varchar(50) NOT Null,
    unique (ID)
    );
```

③ Default constraint :

↳ set default value of col,

```
create table Customer (
        ⋮
    prime_status int DEFAULT 0,
        ⋮
);
```

④ Check :-

— limit value Range (Domain)

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

(5) Primary Key constraint

→ Uniquely identif each tuple.

— P.K ≠ Null.

— Relation only 1 P.K.

```
(
  :
  PRIMARY KEY (ID)
  : )
```

⑥ Foreign Key Constraint

— Keeps reln 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 →

F.K.

P.K

| Loan-number | Payment no. | Payment-date | payment-amount |

* Compost attribut → → sep. attribute for each component.

Customer table

Customer-name	address-city	add-state	add-pincode	add-streetno.	add-streetname

* Multi-value attribute

F.K

dependent-name — new relation.

emp_id	dname.

{emp_id, dname}

↓

P.K..

☆ Generalization

Method 1

☆ Aggregation ⇒

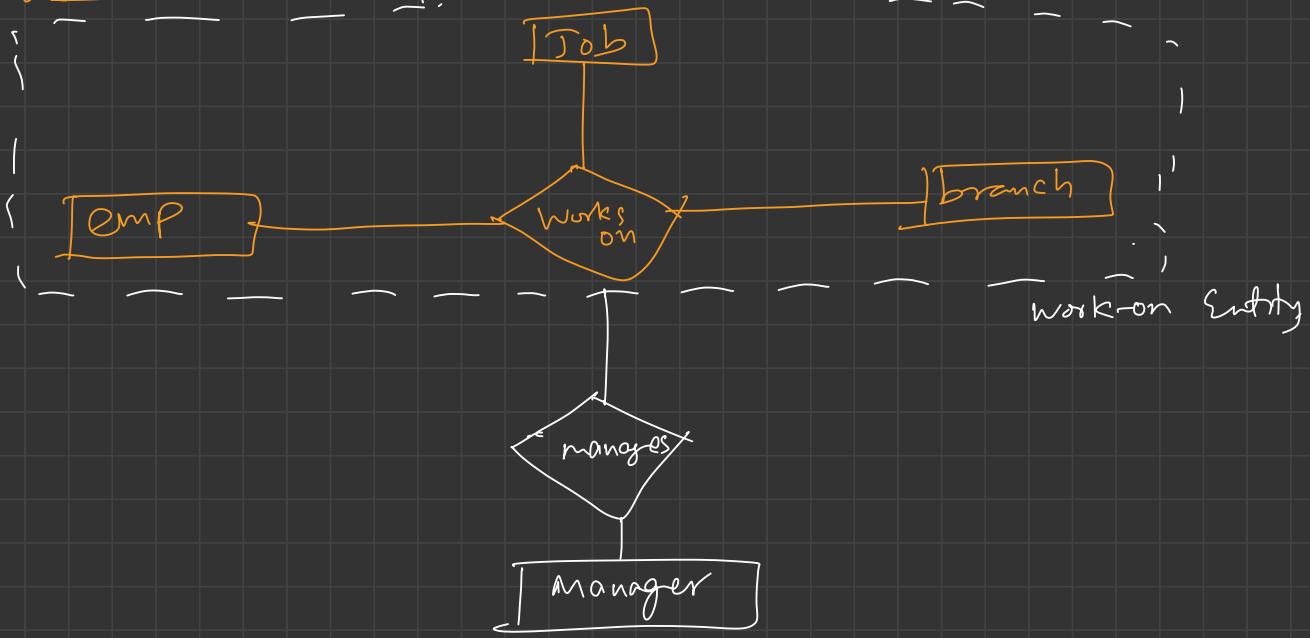
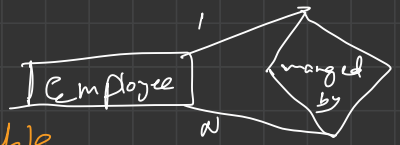


Table: manages (mgr_id, emp_id, job_id, branch_id)

↳ P.K.

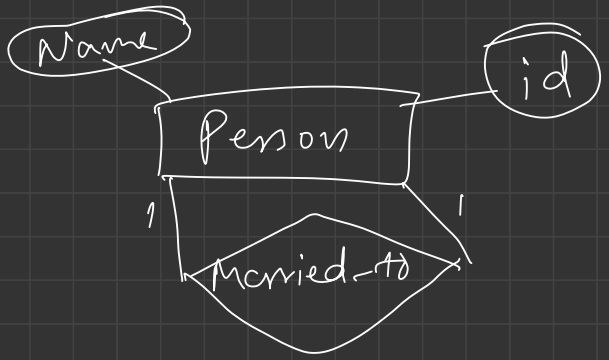
* Unary Relationship :—



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

Emp_id	name	joiningdate	Emp_mgr_id
201	...	...	205
202	...	...	205
205	...	...	Null.

* 1 : 1 ⟹



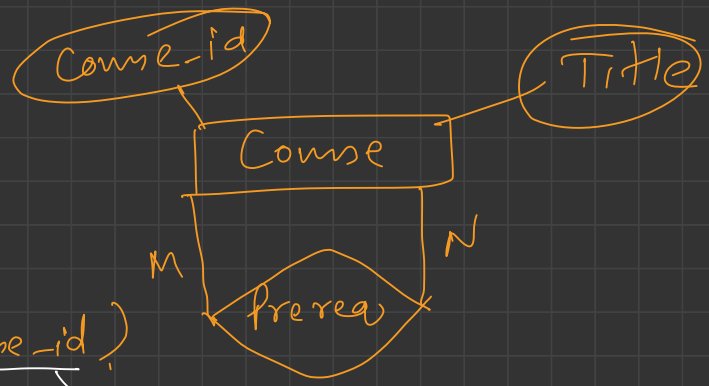
Person (id, Name, Spouse_id)
{ F.K }

* M : N ⇒

① Course (id , Title)

② Prereq (id , Prereq_Course_id)

{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 {f.k}, username {f.k})

Lec-9

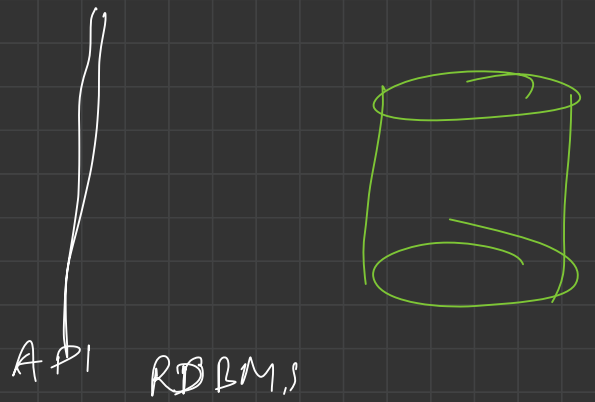
* SQL in One video

SQL => structured Query language

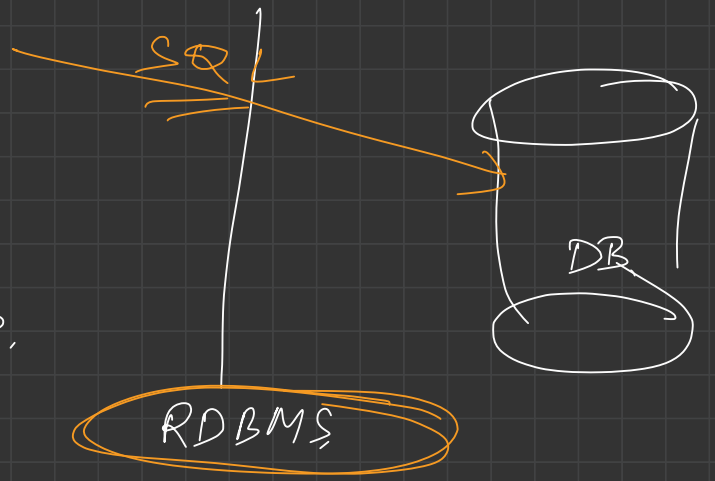
RDBMS => MySQL, Oracle, MS access etc.

CRUD

SQL queries.



CRUD → Create
Read
Update
Delete.



RDBMS → ① MySQL → Open source RDBMS

② MS SQL server → Meta, Adobe etc.

③ Oracle

④ IBM.

SQL	MYSQL
① Query Language.	① MySQL itself a RDBMS
② Way to access data	② CRUD done on it using SQL.

⇒ Install → ① MySQL Community Server
② MySQL Workbench

2) Workbench → DB create
Table

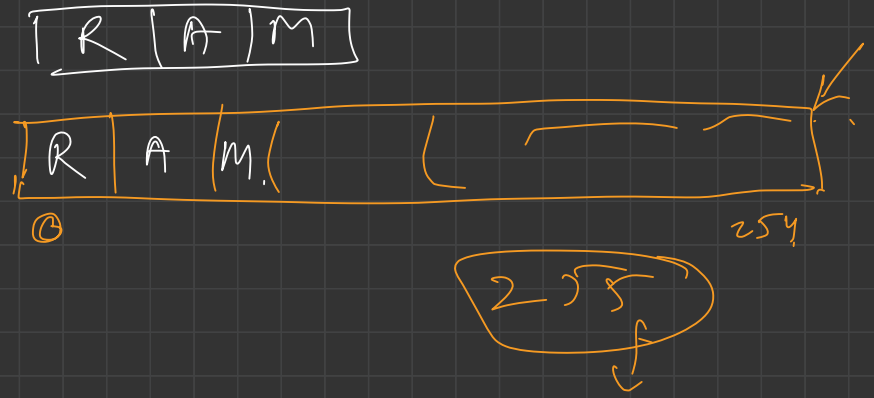
* SQL Data types

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

Variable datatype

VARCHAR (255) →

CHAR (255) →



Fixed datatype

SIGNED , Unsigned

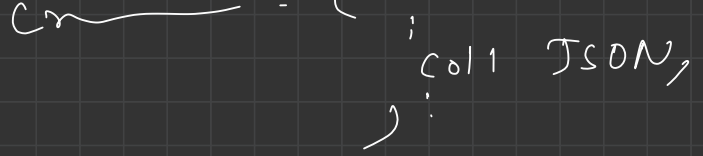
TINYINT ⇒ (−128 to 127)

UNSIGNED TINYINT ⇒ (0 → 255)

```
Create TABLE table_name (
        Col1 INT,
        Col2 INT UNSIGNED
);
```

⇒ Advanced DT |—

① JSON → JS object Notation



SQL Types of Commands

① DDL

⇒ DQL ⇒ ① DB

② table

User defined tables — Student

Dual table

↳ without using FROM

SELECT

Wildcardy

=>

'%pa%'

'abcpadb'

'pabc'

'adpa'

'_pa_'

apab

bpac

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 depart

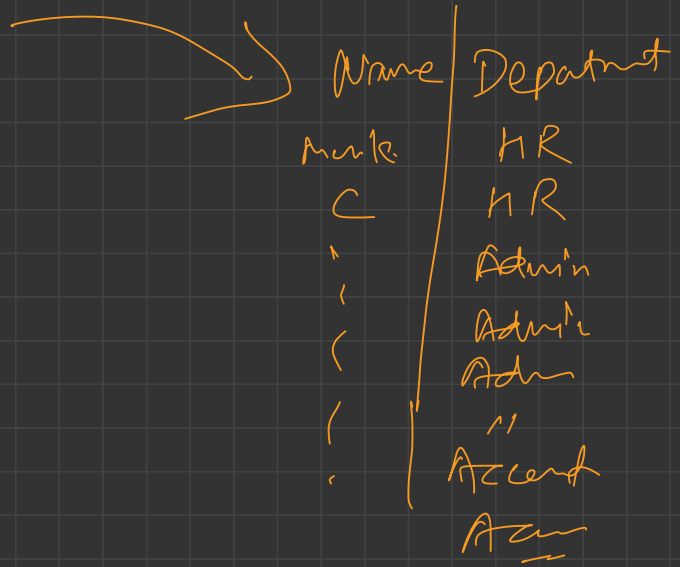
→ HR → ?
Account → ?
Admin → ?

→ Grouping → Aggregation (count)

→ GROUP BY → Aggregation fn.
COUNT, SUM
AVG, MIN, MAX.

Worker

Name	Department
Monika	HR
Nimi	Admin
C	HR
D	Admin
E	Admin
F	Account
G	Account
H	Admin



Q find avg. Salary per department?

* HAVING

→ Select where to filter

"GROUP BY" & "HAVING"

2) 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 Custme (
    ;
    Name VARCHAR(255) UNIQUE,
);
```

④ CHECK

```
create table account (
    :
    CONSTRAINT acc_balance_chk CHECK (balance >=100),
);
```

⑨ DEFAULT

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

* DML Data Modification Language:

* Referential Constraint:

① INSERT

② Delete Constraint

→ ON Delete Cascade

→ ON Delete Set Null:

Replace :- ① Data already present, Replace.

② Data not present, INSERT

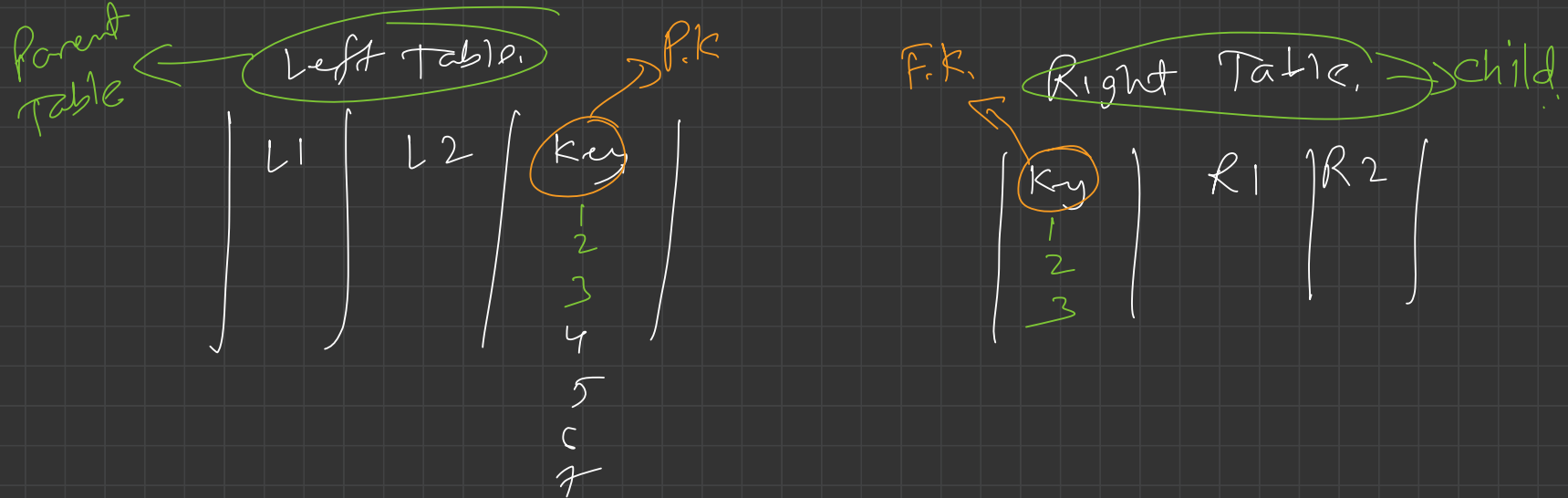
Replace Vs Update

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

JOINS

2) F.K → Relations

→ we → Data fetch → JOINS



① INNER JOIN

Resultant table

Key	L1	L2	R1	R2
1				
2				
3				

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

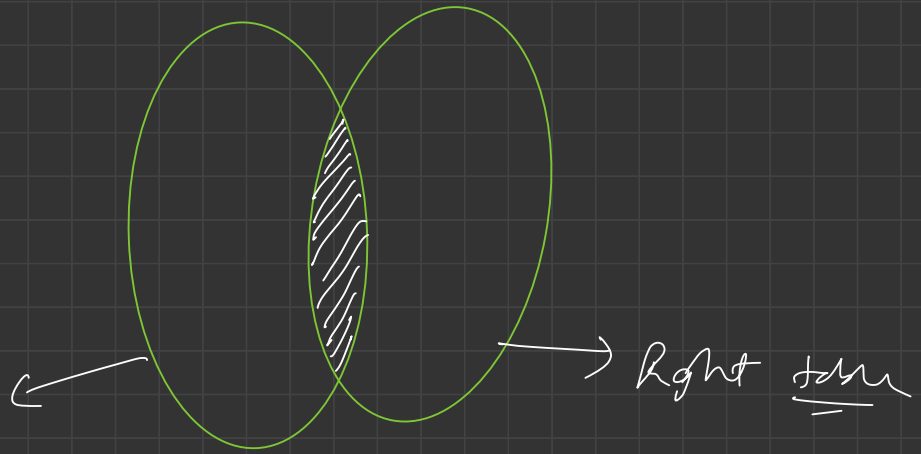
Syntax:

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

C.* → C.id, C.name.

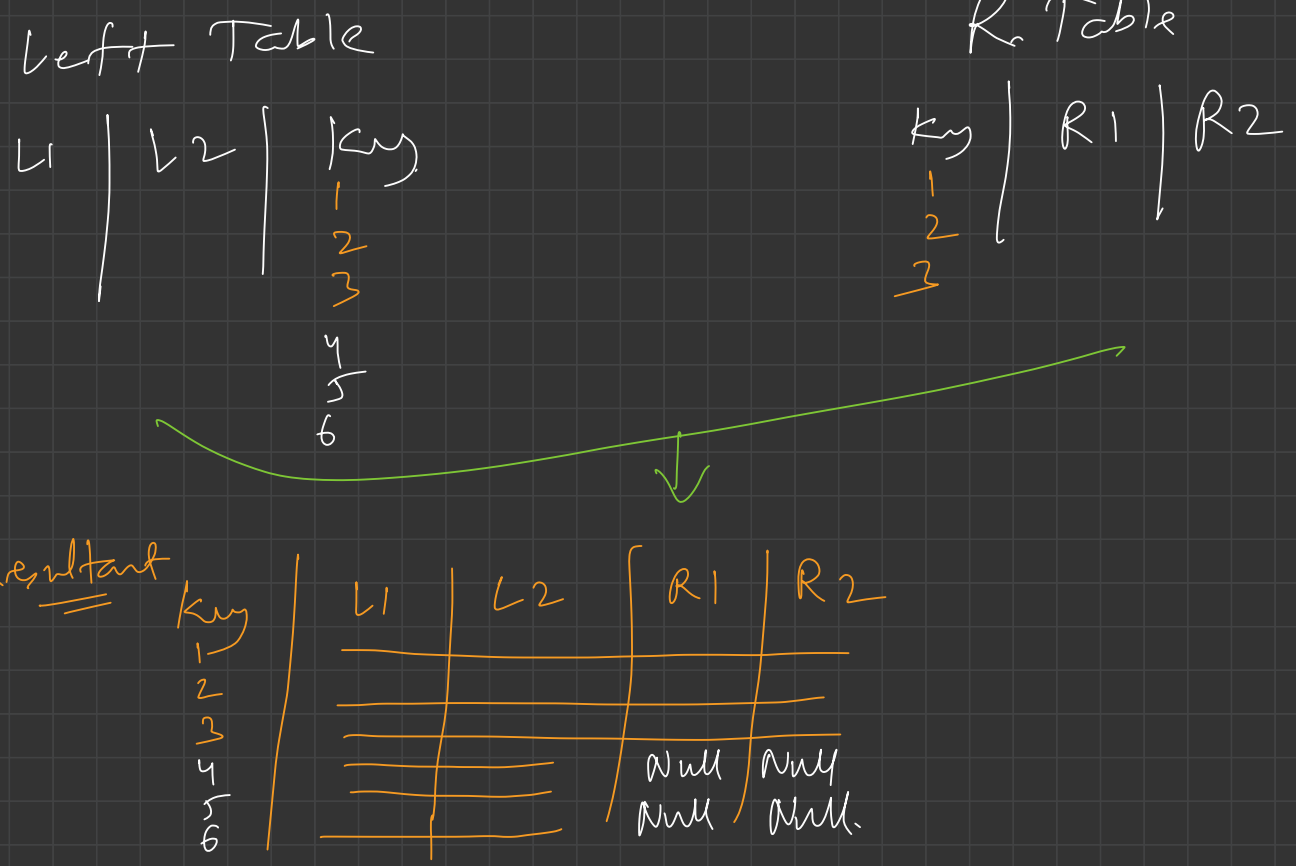
O.* → O.productm

aliasing



INNER JOIN

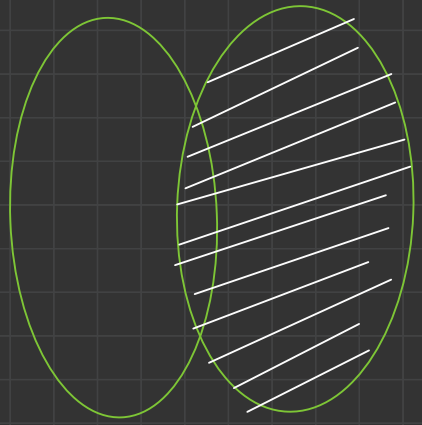
⑦ Left Join

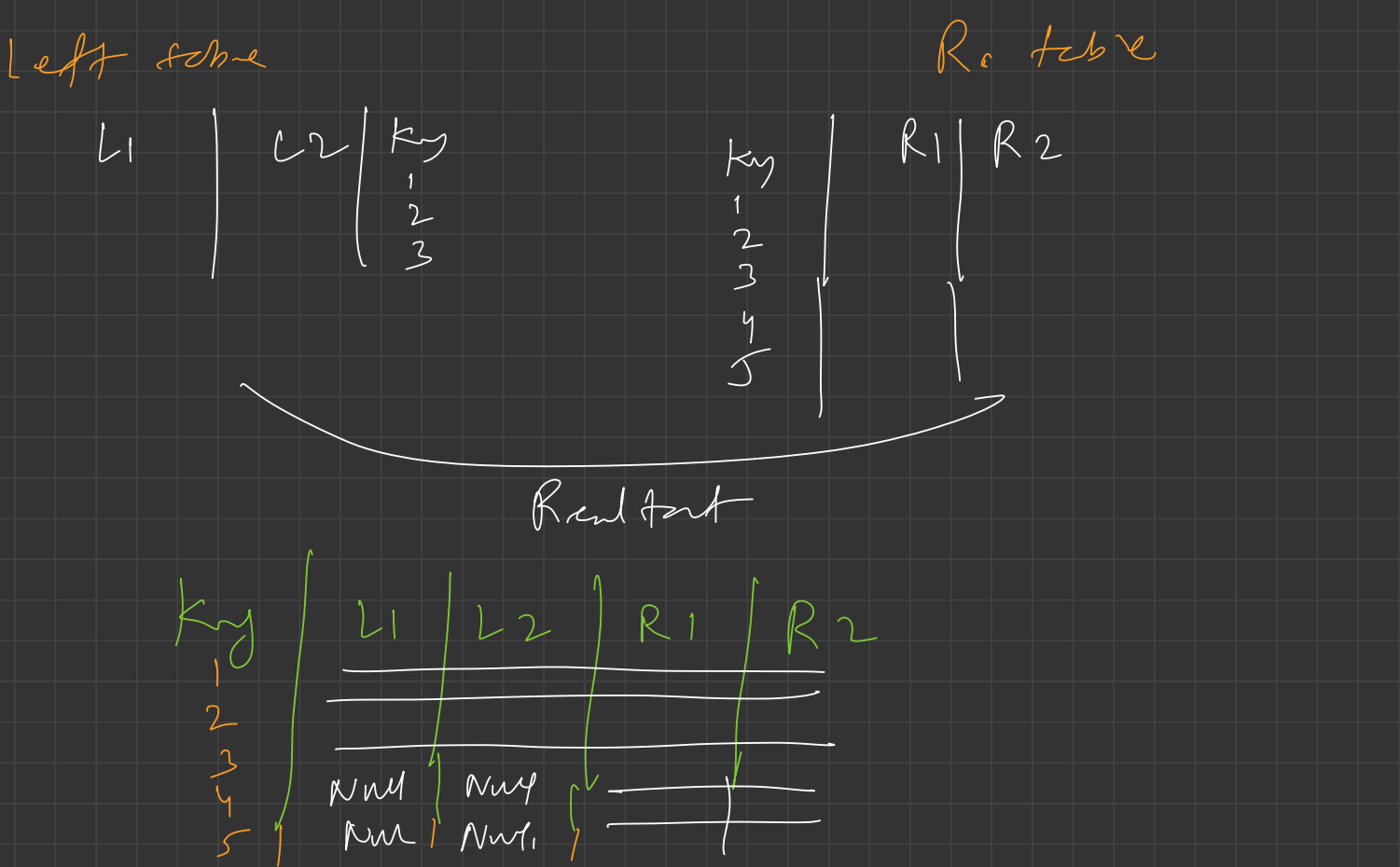

Left Table
L1
L2
Key
1
2
3
4
5
6
R. Table
Key
R1
R2
1
2
3
Resultant
Key
L1
L2
R1
R2
1
2
3
4
5
6
Null
Null
Null
Null

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

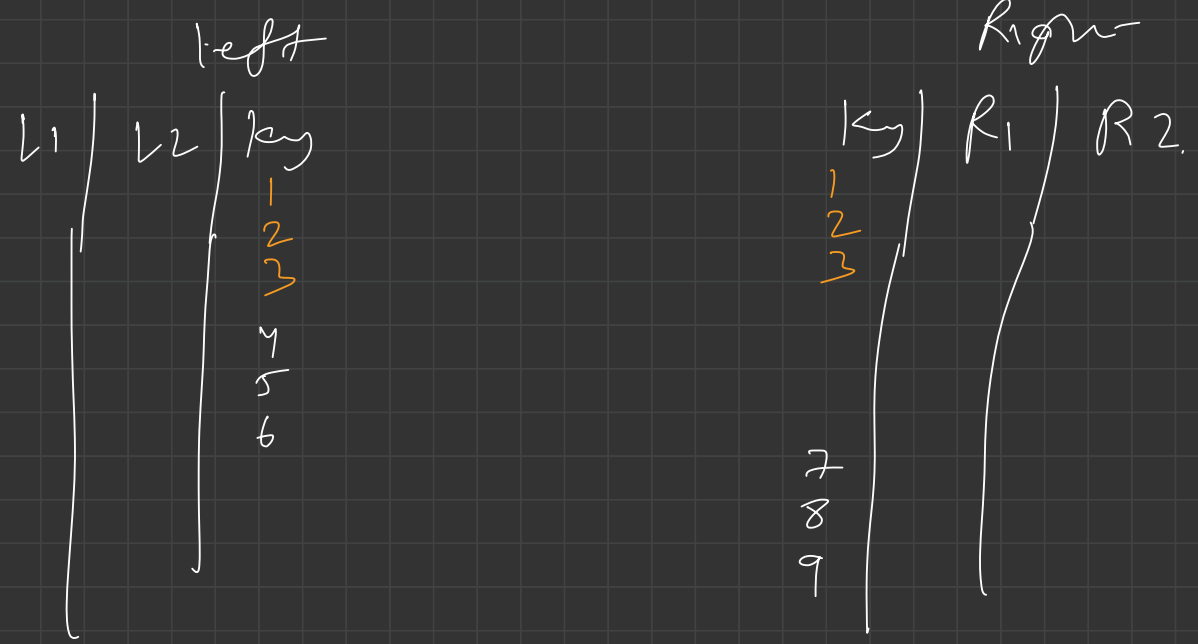
③ Right Join

— Intented in knowing only the right side info.




Left table
Right table
L1
L2
Key
1
2
3
Key
1
2
3
4
5
R1
R2
Result
Key
1
2
3
4
5
L1
L2
R1
R2
Null
Null
Null
Null

⑨ Full Join



MySQL Full Join X keyword X.

Emulated

Left Join ∪ Right Join

Syntax

```
select * from leftTable as l LEFT JOIN RightTable as r
ON l.key = R.key.

UNION

select * from leftTable as l RIGHT JOIN RightTable as r
ON l.key = R.key;
```

6 Cross Join

→ Cartesian product

⇒ T_L — 5 rows

T_R — 10 rows

Result :- $5 \times 10 = 50$ rows

eg.

L_1	L_2
1	A
2	B

R_1	R_2
3	C
4	D

2) Result

→ Industrial use X.

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

⑤ Self Join

- emlate.
- 'INNER JOIN'
- Alias 'AS'

eg. select e1.id, e2.id, e2.name
FROM employee as e1
INNER JOIN employee as e2
ON e1.id = e2.id;

→ Workbench server

* 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
NULL	NULL	NULL	NULL	NULL

* EMPLOYEE

id	fname	lname	Age	emailID	PhoneNo	City
1	Aman	Proto	32	aman@gmail.com	898	Delhi
2	Yagya	Narayan	44	yagya@gmail.com	222	Palam
3	Rahul	BD	22	rahul@gmail.com	444	Kolkata
4	Jatin	Hermit	31	jatin@gmail.com	666	Raipur
5	PK	Pandey	21	pk@gmail.com	555	Jaipur
NULL	NULL	NULL	NULL	NULL	NULL	NULL

* CLIENT

id	first_name	last_name	age	emailID	PhoneNo	City	empID
1	Mac	Rogers	47	mac@hotmail.com	333	Kolkata	3
2	Max	Poirier	27	max@gmail.com	222	Kolkata	3
3	Peter	Jain	24	peter@abc.com	111	Delhi	1
4	Sushant	Aggarwal	23	sushant@yahoo.com	45454	Hyderabad	5
5	Pratap	Singh	36	p@xyz.com	77767	Mumbai	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

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, 8 }

Table 1

col1	col2
A	1
B	1
C	2

Table 2

col1	col2
A	1
B	2
D	3

1. Union
2. Intersection
3. MINUS

→ UNION

$T_1 \cup T_2$

$T_1 \cup T_2 \rightarrow$

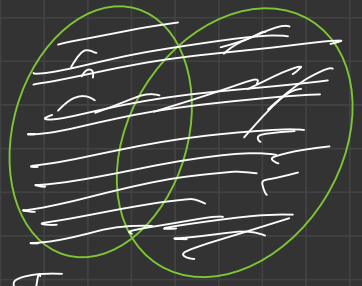
col1	col2
A	1
B	1
C	2
B	2
D	3

— ROWS.

Syntax :→

```
Select * FROM Table1
UNION
Select * FROM Table2;
```

Set operation
→ Combine two or more select statements

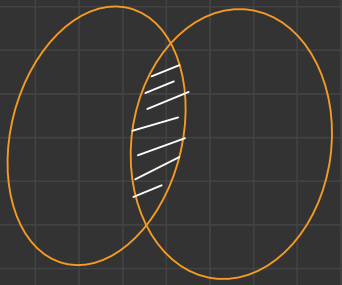


JOIN vs UNION

FULL JOIN :-

Col 1	Col 2	Col 1	Col 2
A	1	A	1
B	1	B	2
C	2	Null	Null
Null	Null	D	3

② INTERSECT



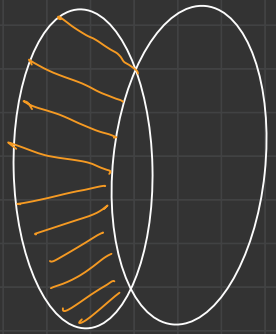
```
Select * from T1
INTERSECT
Select * from T2;
```

→ Emulate

```
select DISTINCT id from T1
INNER JOIN T2
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

ABC

Dept 1

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

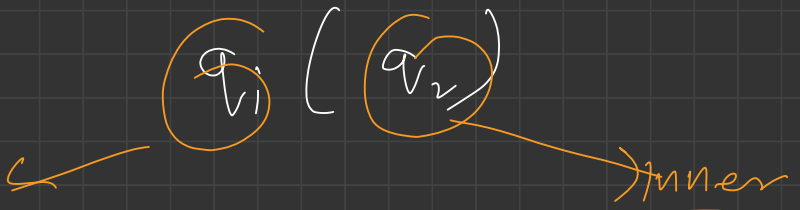
Dept 2.

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

Sub Queries

- alternative to joins

-



Generally, outer query depends on inner query

eg. `Select * from table where id IN ( select id from table where name = 'cat' );`

→ Workbench server

* 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
NULL	NULL	NULL	NULL	NULL

* EMPLOYEE

id	fname	lname	Age	emailID	PhoneNo	City
1	Aman	Proto	32	aman@gmail.com	898	Delhi
2	Yagya	Narayan	44	yagya@gmail.com	222	Palam
3	Rahul	BD	22	rahul@gmail.com	444	Kolkata
4	Jatin	Hermit	31	jatin@gmail.com	666	Raipur
5	PK	Pandey	21	pk@gmail.com	555	Jaipur
NULL	NULL	NULL	NULL	NULL	NULL	NULL

* CLIENT

id	first_name	last_name	age	emailID	PhoneNo	City	empID
1	Mac	Rogers	47	mac@hotmail.com	333	Kolkata	3
2	Max	Poirier	27	max@gmail.com	222	Kolkata	3
3	Peter	Jain	24	peter@abc.com	111	Delhi	1
4	Sushant	Aggarwal	23	sushant@yahoo.com	45454	Hyderabad	5
5	Pratap	Singh	36	p@xyz.com	77767	Mumbai	2
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

* Corelated Subquery

Outer (Inner)

- inner query that refers the outer query

```sql
-- Corelated 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 = 32

inner ⇒ 2

② e1.age = 44 ⇒ ∅ 1

③ 22 ⇒ 4

32.
44.
22
31.
21

④ e1.age. 31 ⇒ 3

SQL Views

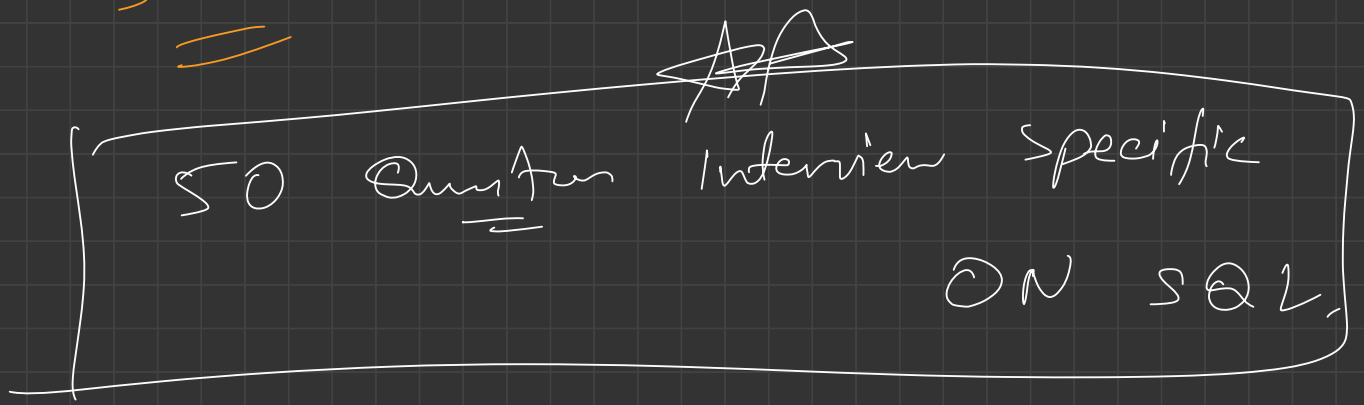
⇒ MySQL → views

Customer ⇒ id | name | age | address |

view ⇒ name | age → alias

— custom_view

2) SQL

2) 

Home work

1. Note making
2. Practice

Lec-10

Practice SQL Questions

- → DB
- → Tables / Relations
- → Data Relationships
- → Questions.
- → 5 Th Question

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
NULL	NULL	NULL	NULL	NULL	NULL

- 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 ⟺ 2,1

2,3 ⟺ 3,2

O/P

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

Sol' —

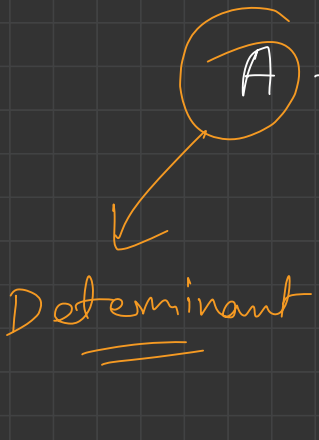
① JOINS

② CO-related.

Lec-11

Normalization ?

Functional dependency



$A \rightarrow C$

Table

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

eg. EMP

Emp_id	name	dept.	address

FD's ⇒

Emp_id ⟶ Emp_name.

Emp_id ⟶ dept

① Trivial F.D :- $A \rightarrow B$, B is a subset of A.

eg. {EMP_ID, Name} ⟶ Emp_ID

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

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

(2) Non-Trivial F.D ⇒

{EMP_ID, name} ⟶ {Emp_addrs}

A ⟶ B

Q. B ⊆ A? ans NO,

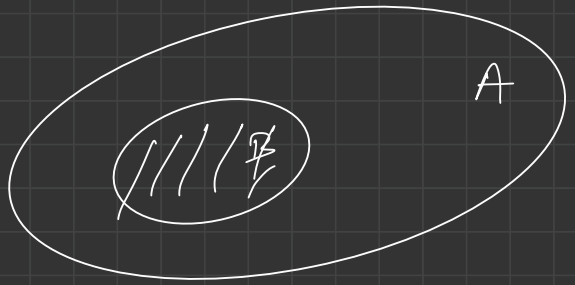
⇒ This is N.T.F.D

⇒ A ⟶ B, B ⊄ A, A ∩ B = NULL.

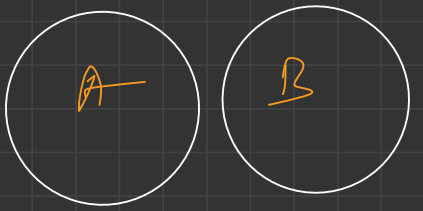
EMP_id ⟶ Emp_dept

Venn diagram

T.F.D



N.T.F.D



A ∩ B = NULL

① Reflexive

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

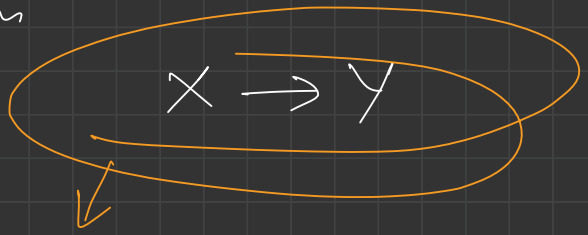
subset $Y = \{a, b, c\}$

⇒ Y is a subset X,

$X \rightarrow Y$.

② Augmentation

→



$XZ \rightarrow YZ$

R (X Y, Z)

Q R(A B C D E)

$\equiv$

$A \to B$

$A \to C$

$CD \to E$

$B \to D$

$E \to A$

$\equiv$ $BC \to CD \Rightarrow B \to D$, ✓

$EC \to AC$? $\to$ ✓

$BD \to CD \Rightarrow B \to C$?? X ✓

Q R(A, B, C, D, E)

FD :- $A \to B$

$A \to C$

$CD \to E$

$B \to D$

$E \to A$

Can $CD \to AC$??

$CD \to E$ & $E \to A$ $\Rightarrow$ $CD \to A$ (transitivity)

$C \subseteq CD \Rightarrow CD \to C$

So, $CD \to A$ & $CD \to C$

$\Rightarrow CD \to AC$ ✓✓

Why Normalisation

What if we have redundant data?

→ introduces 3 anomalies → abnormalities

1. Insertion
2. deletion
3. updation / modification

Student

id	name	age	Branch_code	Branch_name	Branch_HOD
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.E	Z

(1) Insertion

=> New student.

=> University -> 'IT'

② Deletion

③ Updation :-

HOD change CS ⇒ Q.

⇒ | id | name | age | Brcode | Table1

Table2 Branch info

Brcode	Branch name	HOD name
1	CS	X
2	ECE	Y
3	ME	Z
4	IT	Q.

What we do in normalization

=> Table → decompose → into multiple tables.

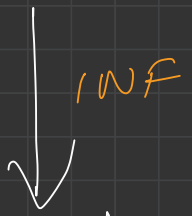
SRP → Single responsibility principle.

1NF

eg.

emp.

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



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

2NF →

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

⇒ {A B} → P.K

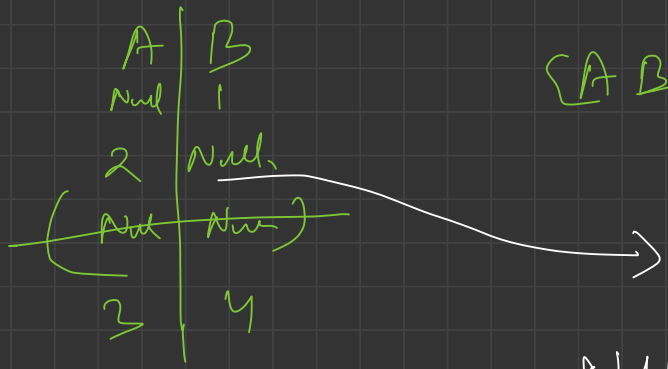
A, B → prime attributes.

C, D → Non-prime.

⇒ FD ⇒ B → C ← Partial dependency.

∴ AB → C ✓

AB → D



{AB} P.K.

B → C

Null → C ?? X.

2NF Conversion

R_1 (A B D) AB → D

R_2 (B C) B → C

2NF Table (Student Project)

StudentID	Proj ID	StudentName	Project name
S89	P09	Olivia	Geo
S76	P07	Jacob	Cities
S56	P03	Ava	IOT
S92	P05	Alex	Cloud

P.K : {StudentID, Proj ID}

FD :→ StudentID → StudentName

ProjectID → ProjectName

2NF form

Student

Student ID	Project ID	Student Name.
S89	P09	olvia
S76	P07	Jacob
S56	P03	Ava
S92	P05	alex

Project

Project ID	Project Name.
P09	Geo
P07	Cluster
P03	IOT
P05	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 → Prme.

B → C

↓ Nonprm.

→ Prme.

$B \Rightarrow C$ — F.D (Transitive dependency)

2) decompose 3NF

R_1 (A B) R_2 (B C)

R_1

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

B	C
1	x
2	y
3	z

BCNF

eg.

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 convers

Student

id	p-id.
101	1
101	2
:	:
:	:

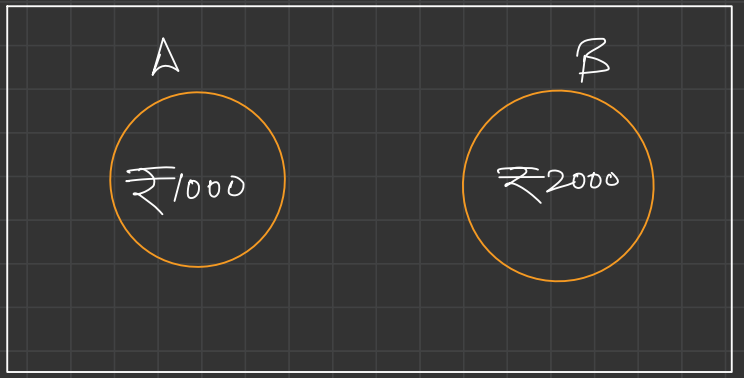
Profesor

p-id	Profesor	subject
1	PJ	Java
2	PC	CPP.
:		:
:		:

Lec-12

Transaction :-

ABC Bank



System ₹3000

Task :-> ₹50 from Bank A/C A to A/C B

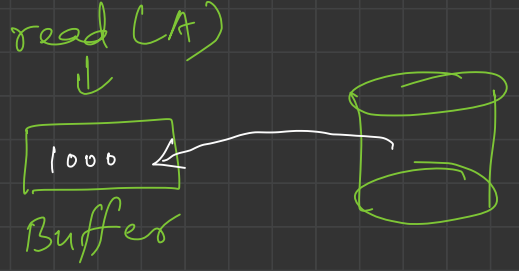
T_i :

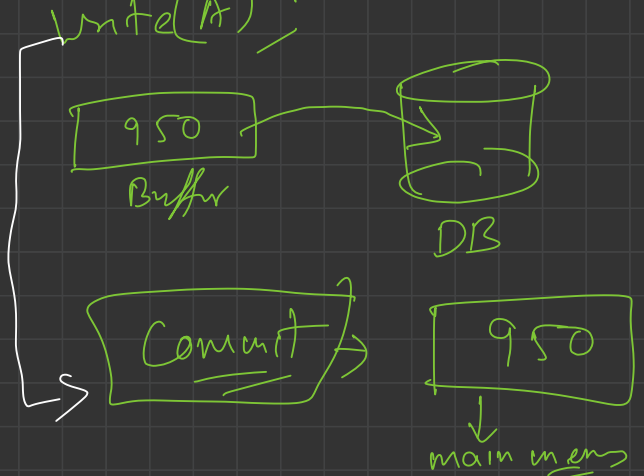
```
read(A)
A := A - 50;
Write(A);
read(B)
B := B + 50;
write(B)
```

6 steps

Atomic -> are considered they to be a single Task.

ACID Properties





T_i : read(A)
A := A − 50;
write(A);
read(B);
B := B + 50;
write(B);

(1) Atomicity

→)

A	B
read 1000	read 2000
950 = 1000 − 50	
write(A) 950	
	read 2000
	2000 + 50
	write(2050) 2050

950.

=> transaction → successful.

→ fail → old state recovery.

DB → maintain → old state.

↓ intermediate state.

↓ success ✗.

Rollback → 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)

read(A) → 1000 ⟶ $read_1(A)$ → 1000.

A := A − 50 = 950. 1000 − 50 = ~~950~~.

A = 950

write(950)

B ⟶

B ⟶ 50 + 50 = ~~100~~

(4) Durability

T_1 → success

↓

all 6 step → success

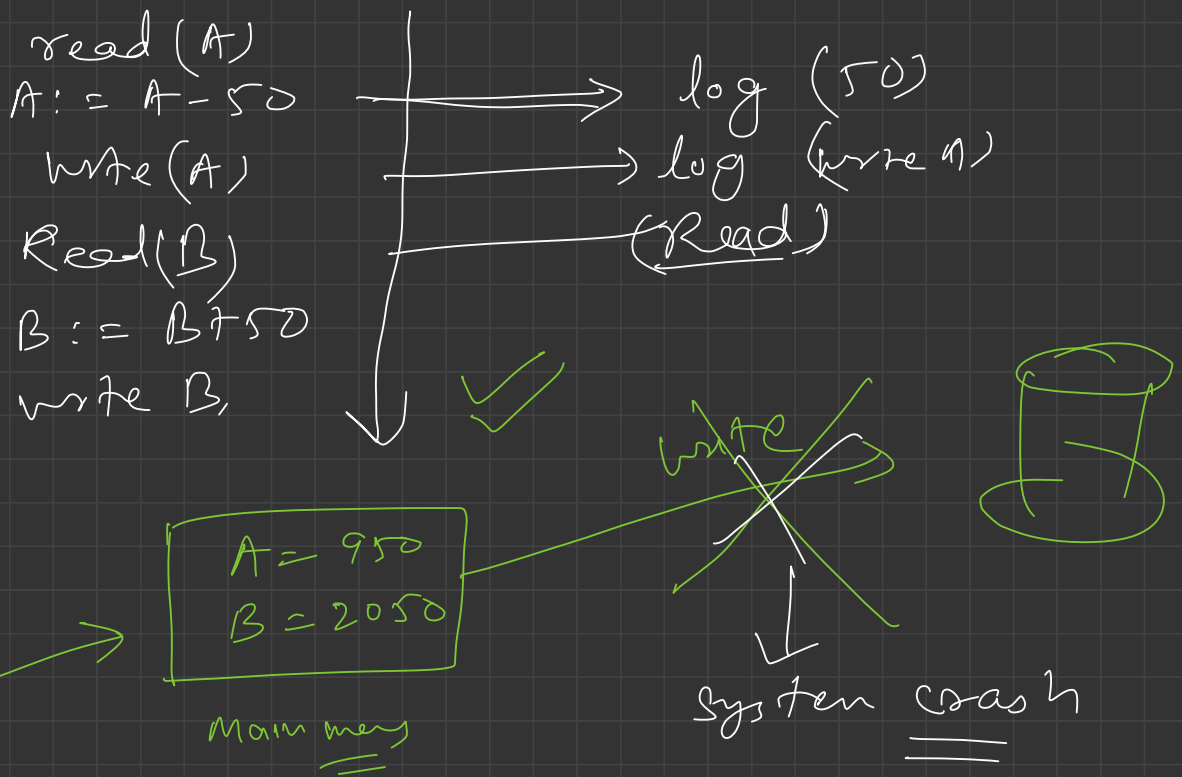
↓

DB updation permanent (persistant)

↓

even if there is a system failure after T completes execute.

→ recovery - mgmt Component.



States of Transaction (life cycle)

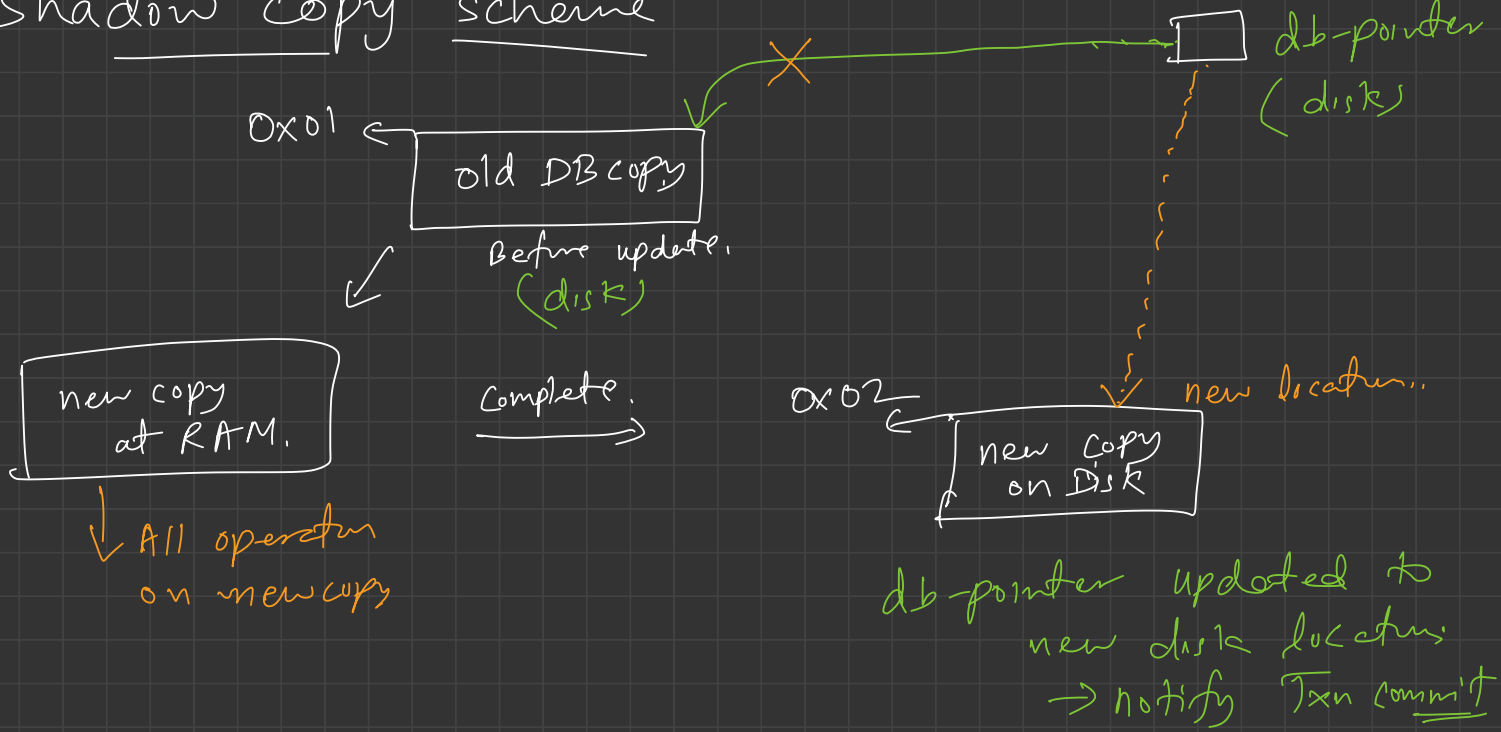
How to Implement Atomicity ?

Lec - 13

Lec-13

* How to implement Atomicity & Durability ?

① Shadow copy scheme



Log Based Recovery :-

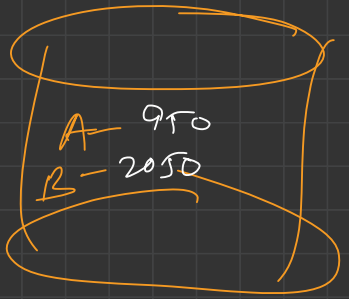
(1)

A	B
₹1000	₹2000

T_0:
read(A)
A := A - 50
write(A)
B := B + 50
write(B)

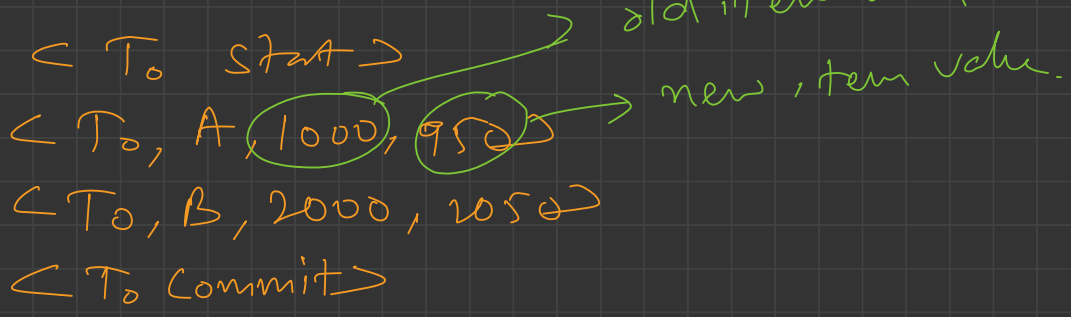
Logs (stable storage)

<T_0 start>
<T_0, A, 950>
<T_0, B, 2050>
<T_0 commit>



① Deferred DB Modifications

② Immediate DB modifications



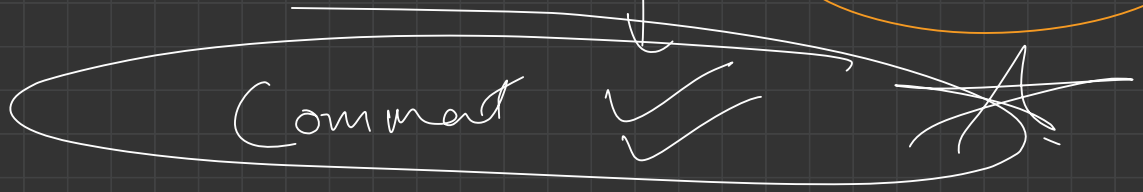
→ Checkpoints

$C_1 \rightarrow$ ΣT_n → commit.

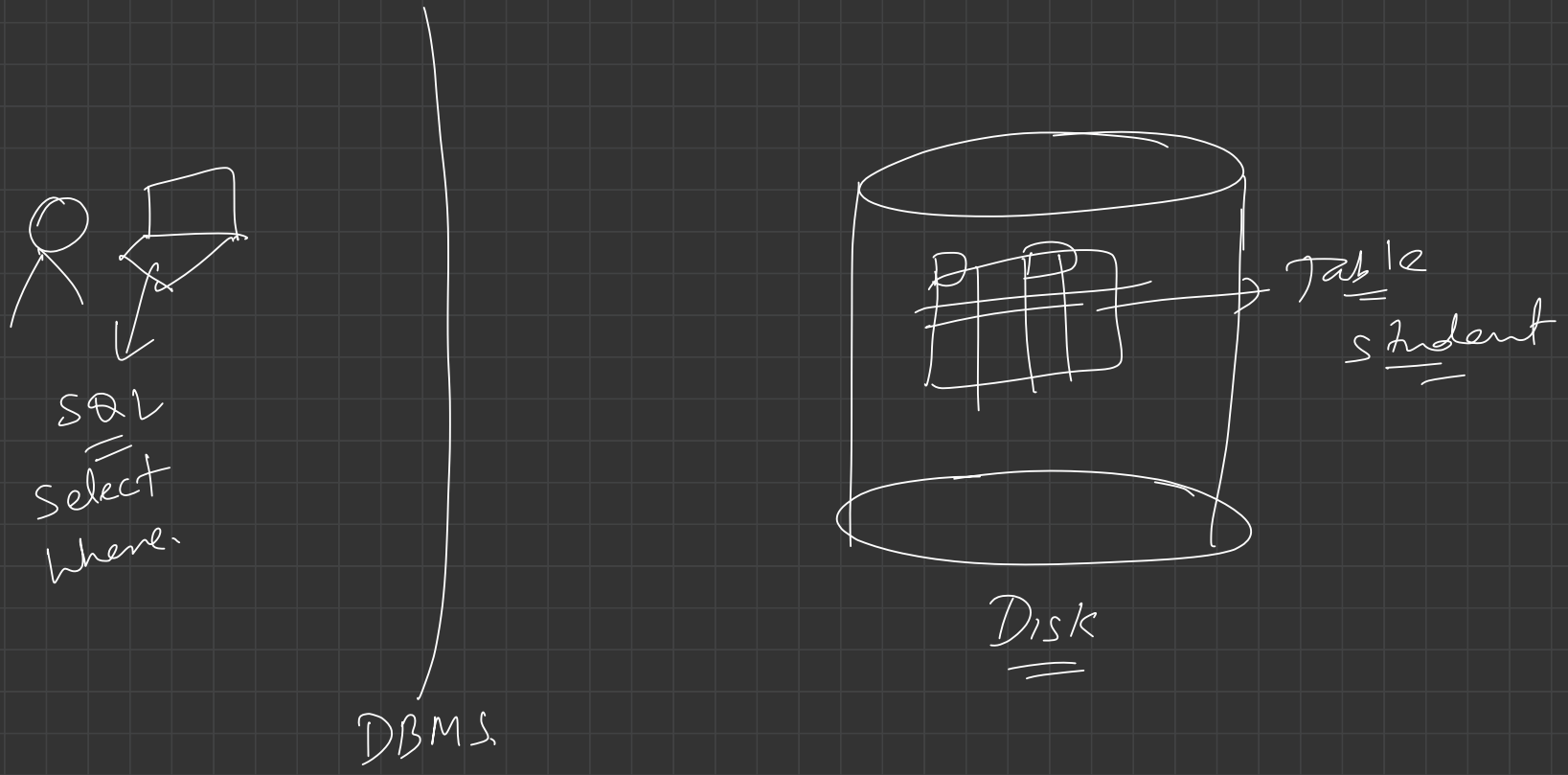
$C_2 \rightarrow$ ΣT_n commit.

H.W ⇒ checkpoints

Google

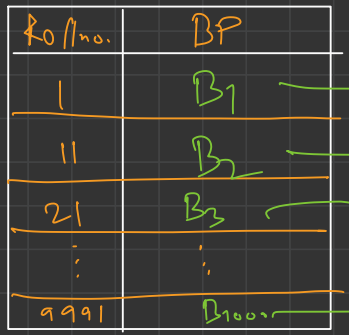


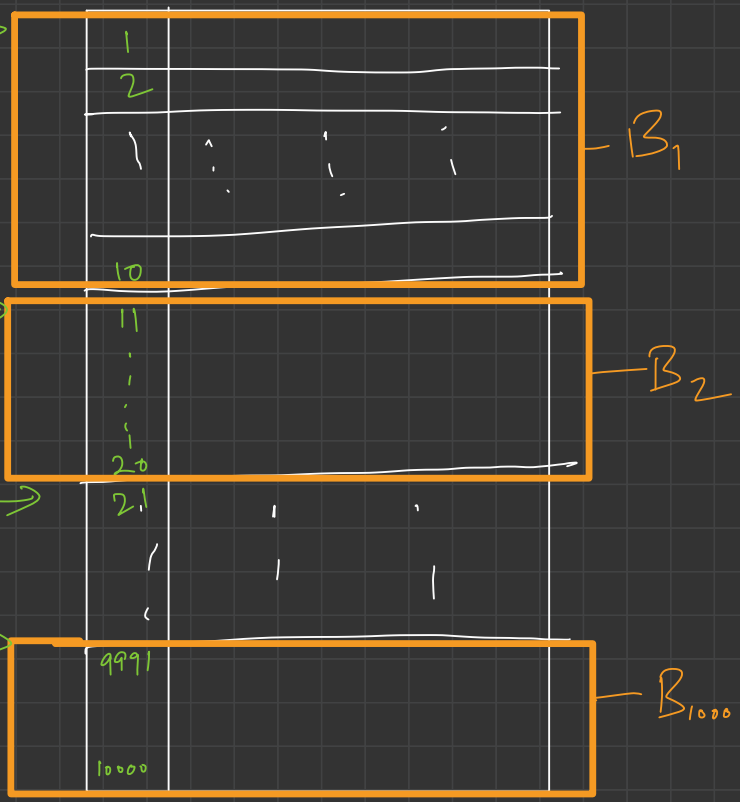
Lec-14



Data structure

INDEX





eg. 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.K / C.K

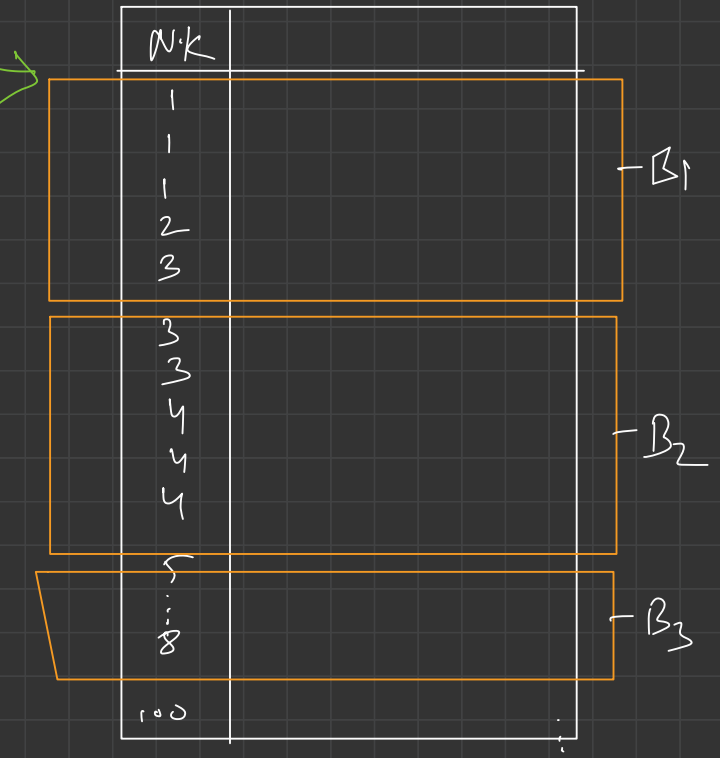
↳ indexing → Sparse indexing

② Based on non-key attribute

→ **Clustering Indexing**

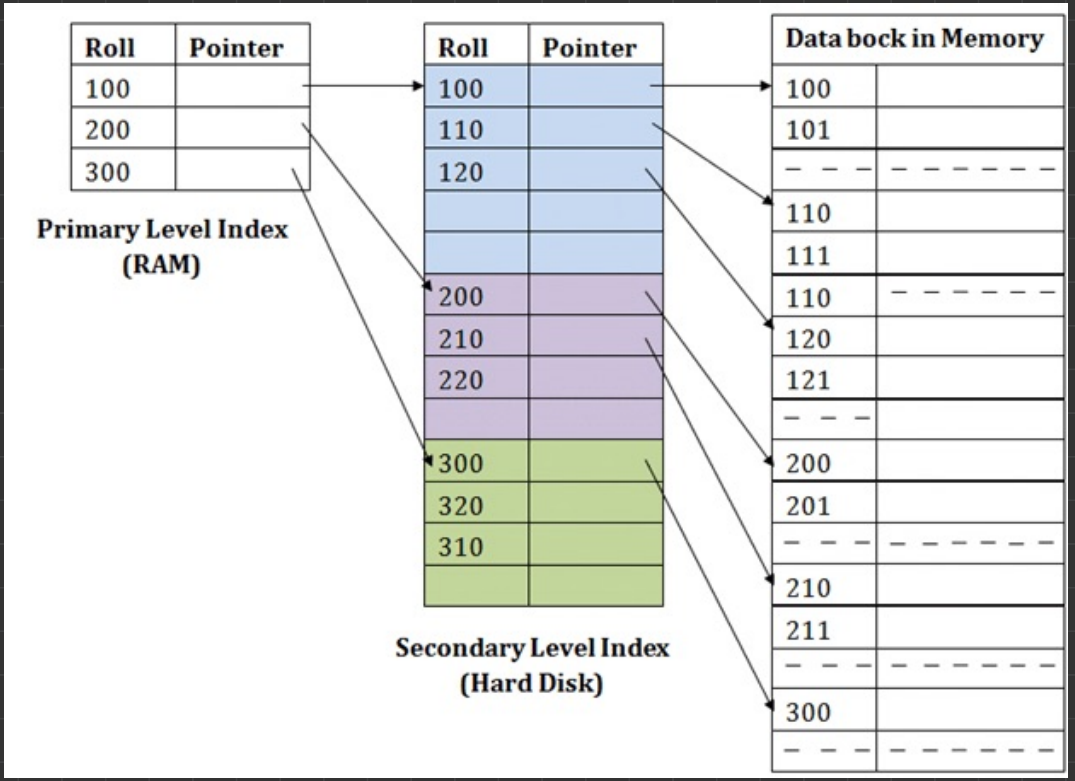
2) INDEX

N.K	BP
1	B_1
2	B_1
3	B_1
4	B_2
5	B_3
⋮	⋮



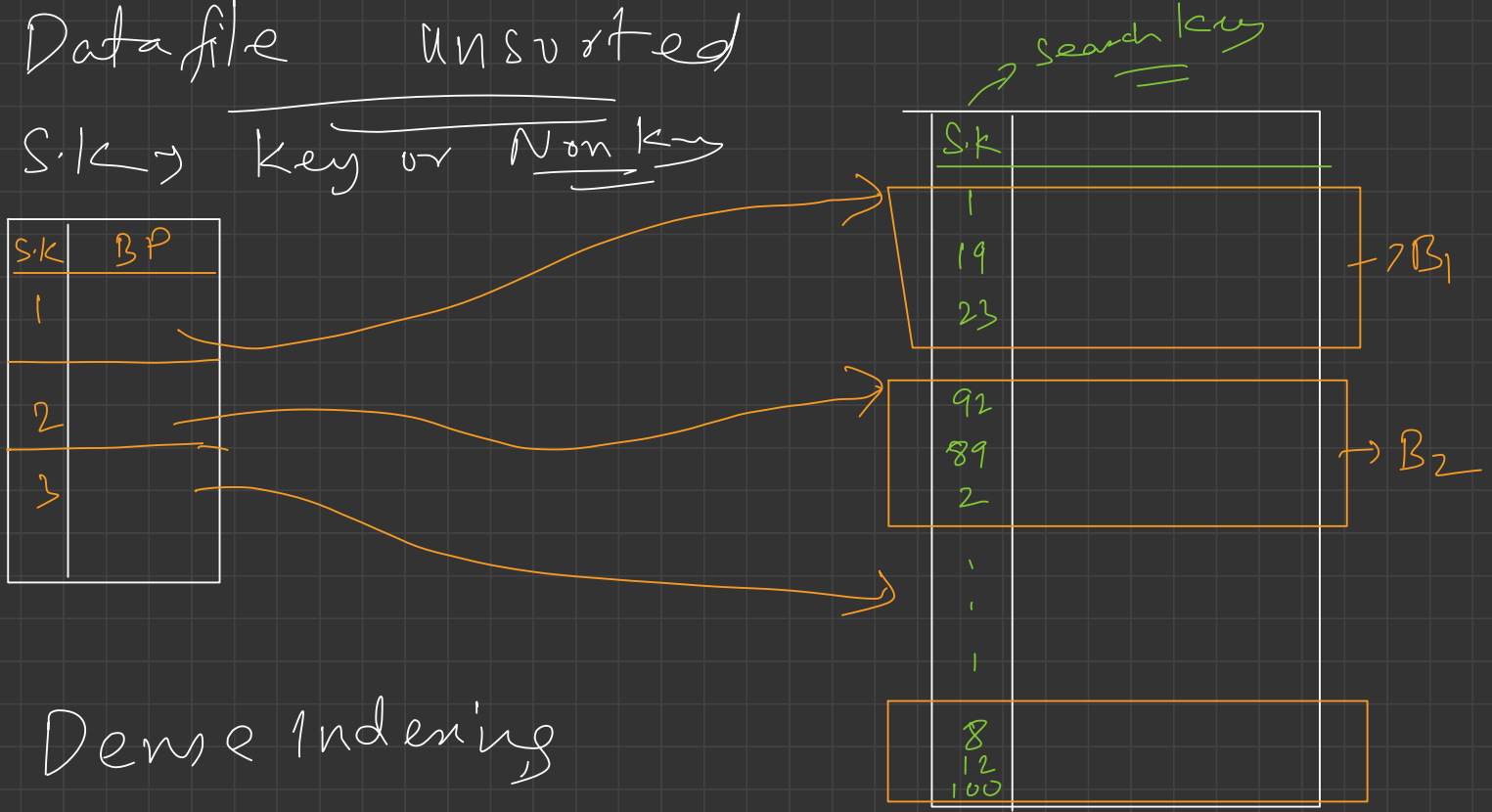
→ Dense Indexing

Multi Level INDEXING

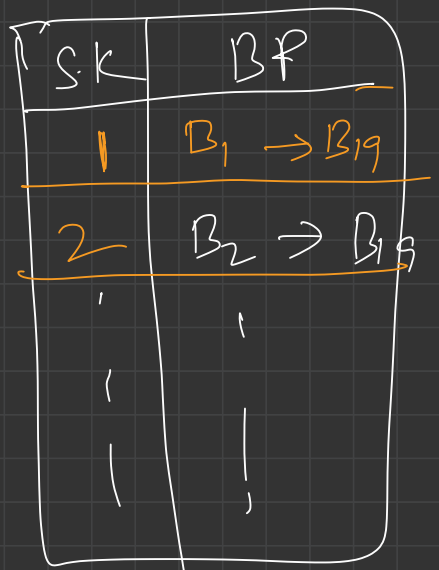


Secondary Indexing

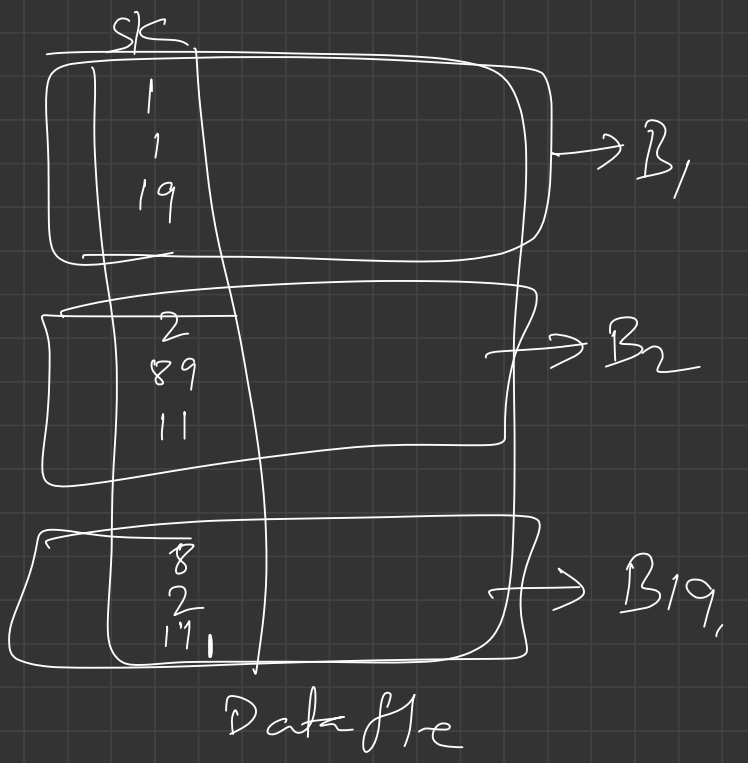
- Datafile unsorted
- S.Key key or Non key



- Dense Indexing



index.



Data file

Lec-15

NoSQL → Non-Relational Model

↓

"Not only SQL"

- → SQL
- → Structured data.
- → constraints
- → fixed schema.
- → Vertical scaling

NoSQL

- → structured data unstructured, semi structured
- → flexible schema.
- → vertical horizontal

⇒ Data Modeling in SQL vs NoSQL

① SQL

users

ID	first_name	last_name	cell	city
1	Tata	Salt	811	Mumbai

hobbies

ID	user_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"]
}
```

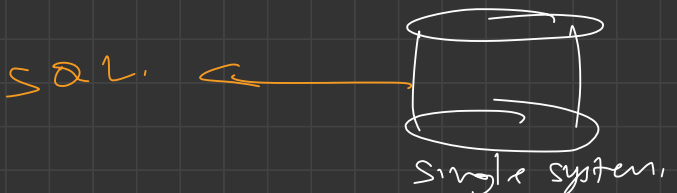
Advantages of NoSQL:

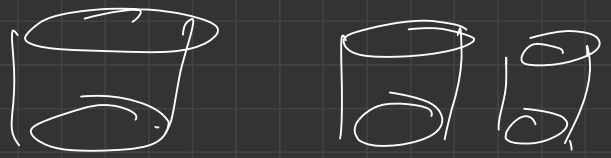
1. Flexible schema.
2. Horizontal scaling.

1. Scaling

Vertical
- Hardware
- RAM
- CPU.

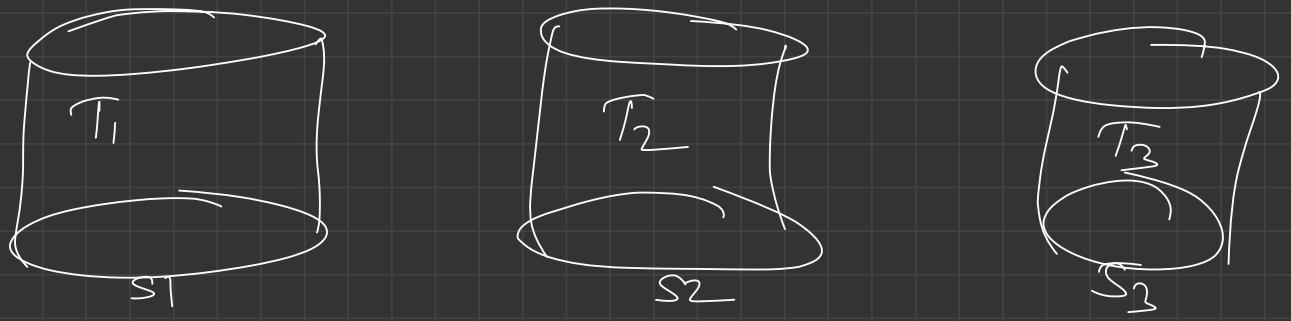
Horizontal scaling
- additional node.
- Load share.





⇒ Interview

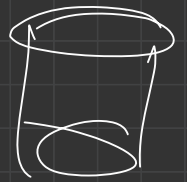
SQL → why No horizontal scaling ?



SQL → Table collection

→ SQL → data retrieve → 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 : 'Kathmandu'
}

(25) C - Structures

Name	City	age
Matt	Delhi	27
Dave	Jaipur	30

Row wise →

Matt	Delhi	27	Dave	Jaipur	30

0x01 0x02 0x03

Column wise

Matt	Dave	Delhi	Jaipur	27	30

③ Document based → MongoDB

Contact

{ id: —

}

inner document

{

id:

}

accen

{ id: — —

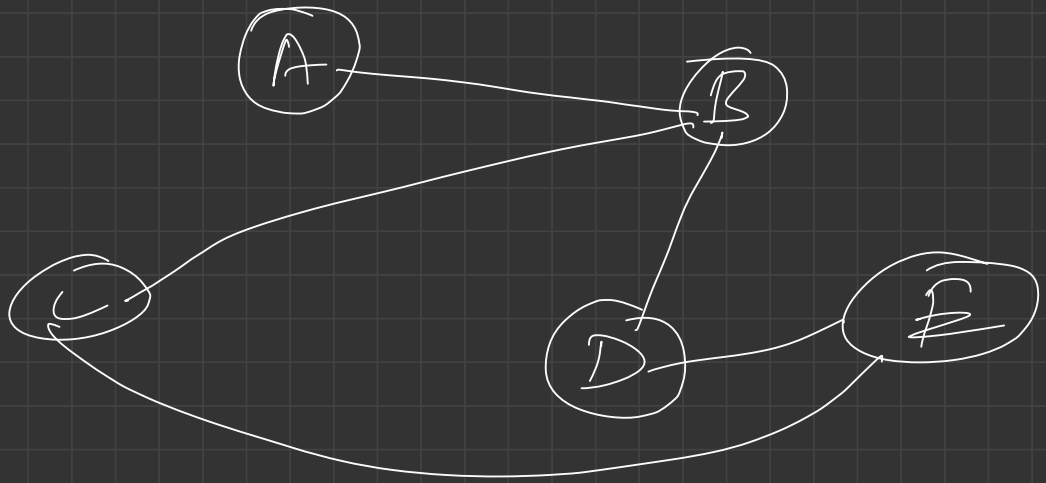
}

④ Graph Based store

→ data → Nodes (vertices) ← Edges (Relation)

Facebook →

Relation



⇒ Disadvantage :-

Lec - 16

Types of Databases

1. Relational Databases
   - Relational model
   - tables
   - → Related.

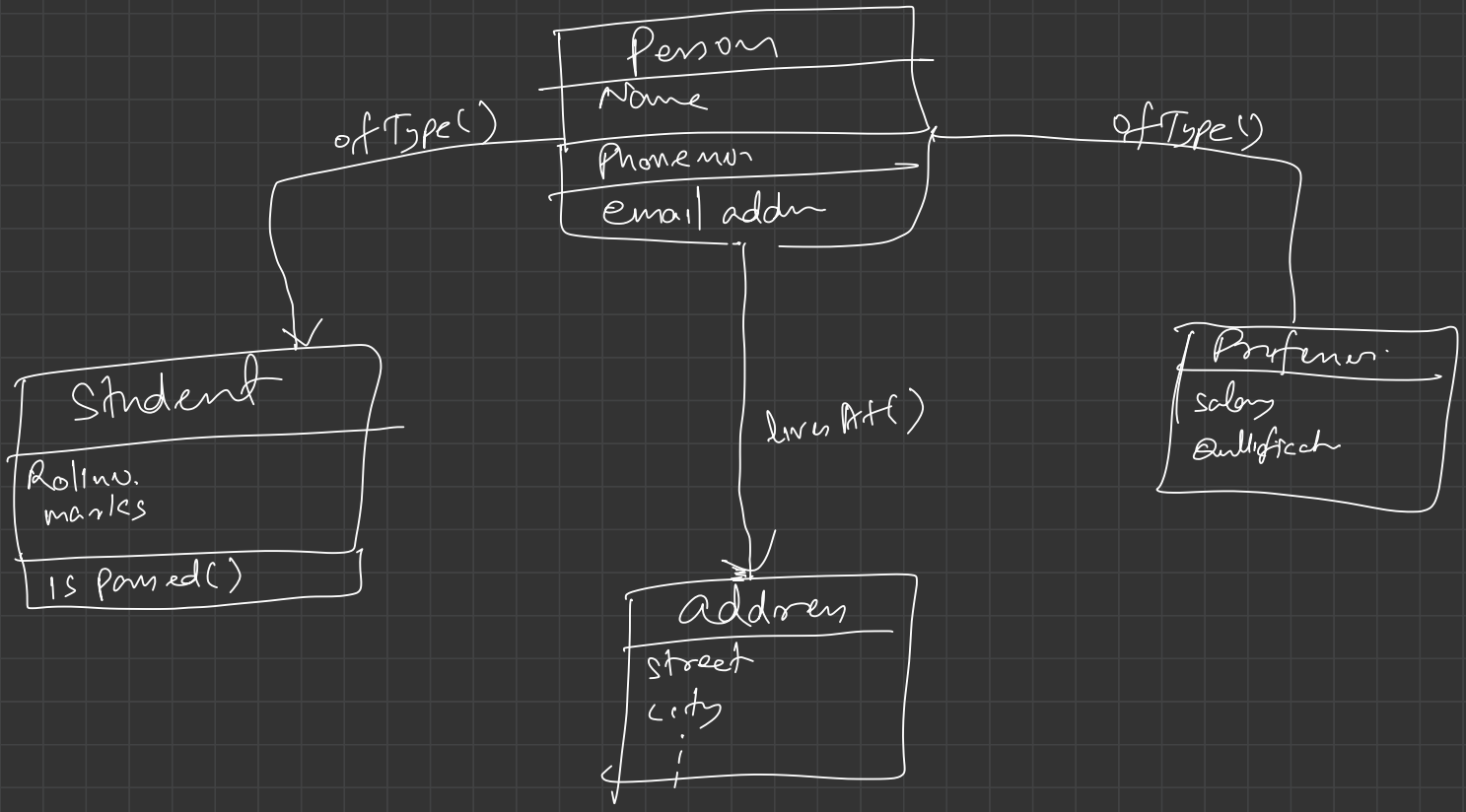
(2) object-oriented datamodeling

— OOPS similar concept

→ Class → instance → objects

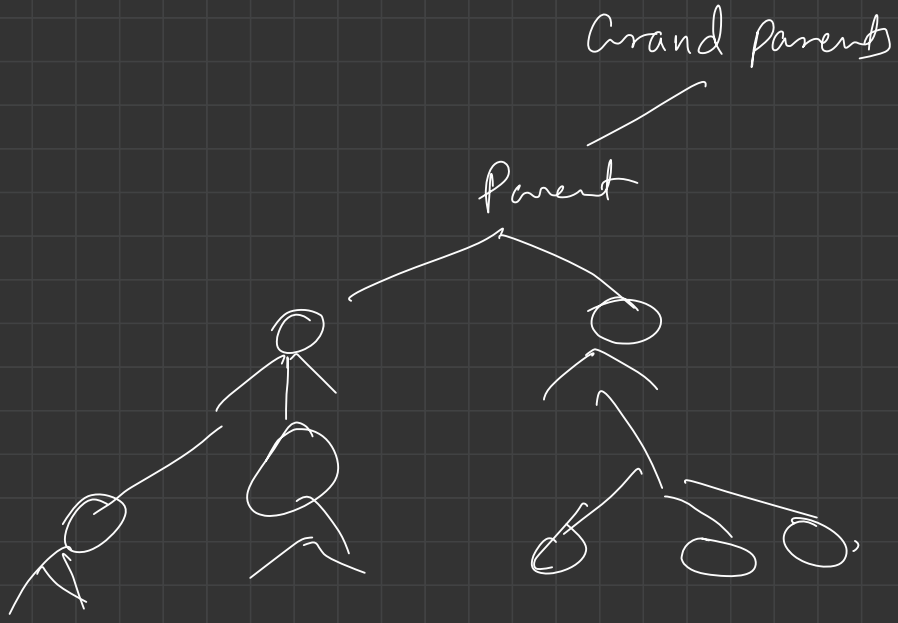
- → attributes → (Characters)
- → Methods. → (behaviour)

— 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.

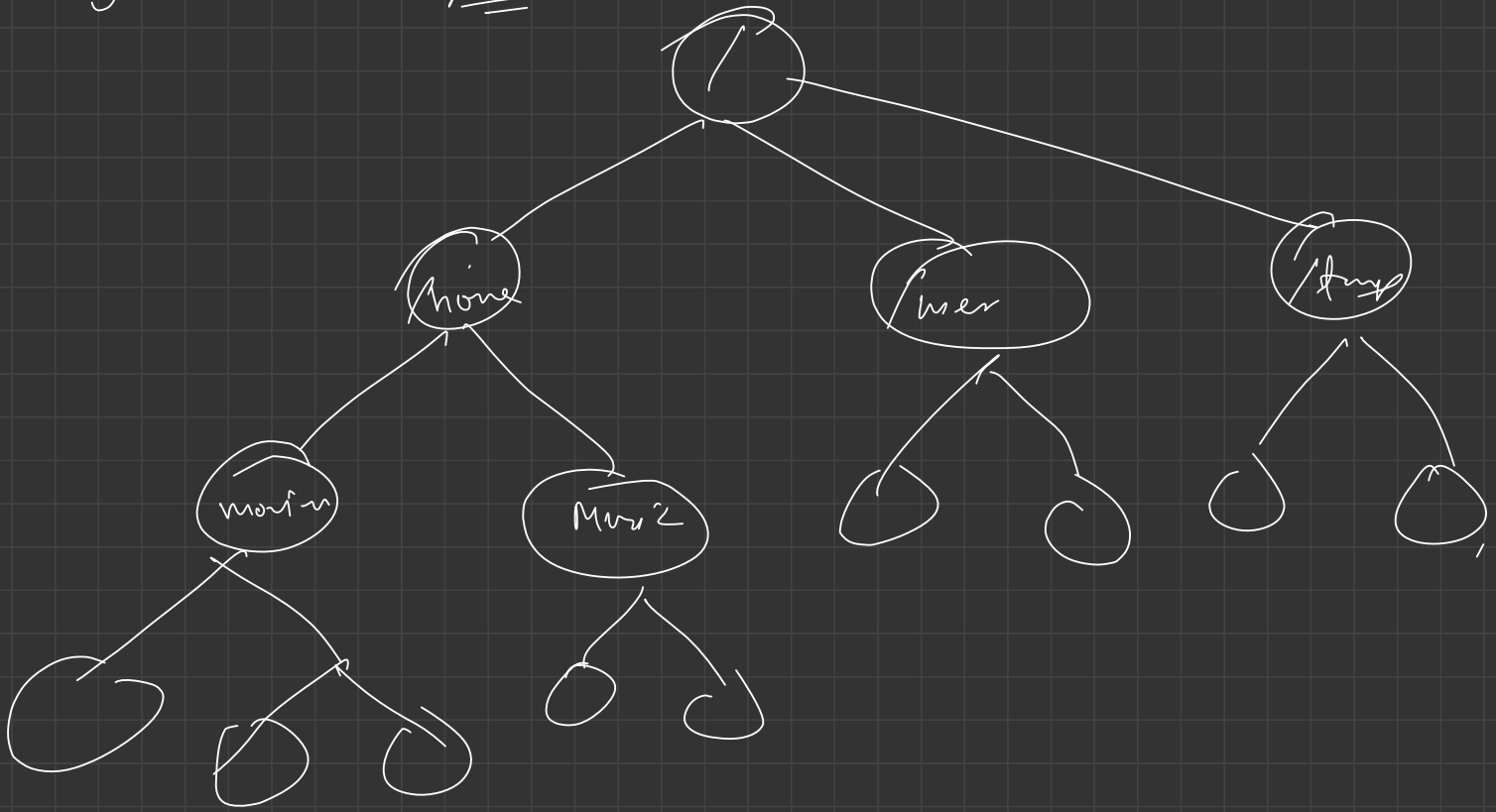

Person
Name
Phone no.
email addr
ofType()
ofType()
Student
Roll no.
marks
is Passed()
Has A()
Professor
salary
qualification
address
street
city

(3) Hierarchical DB

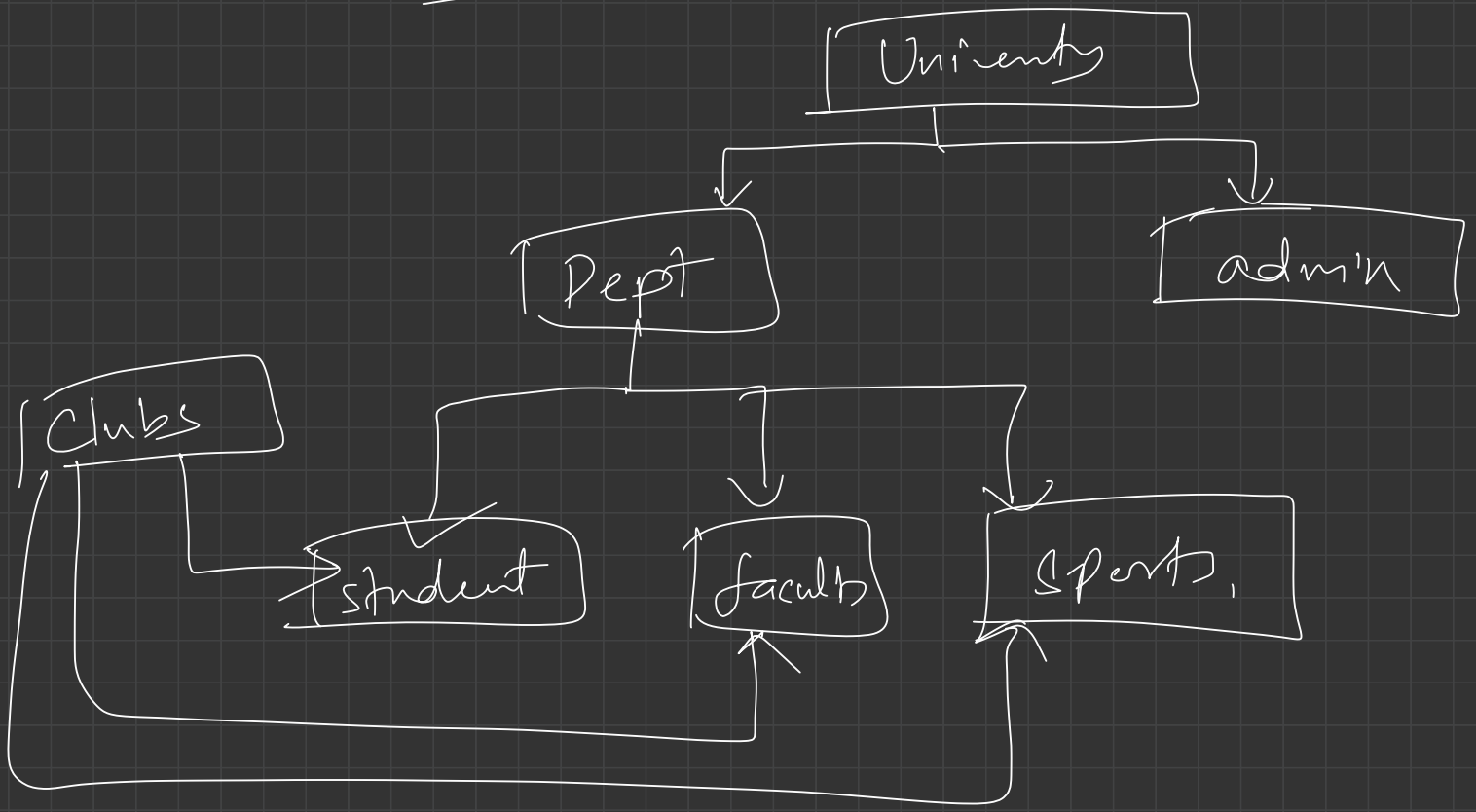
eg. Family Tree.



eg. Filesystem

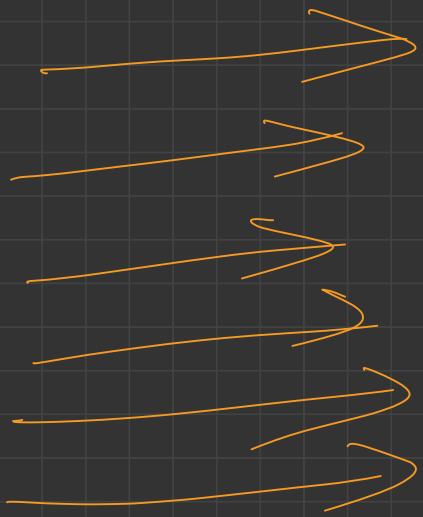


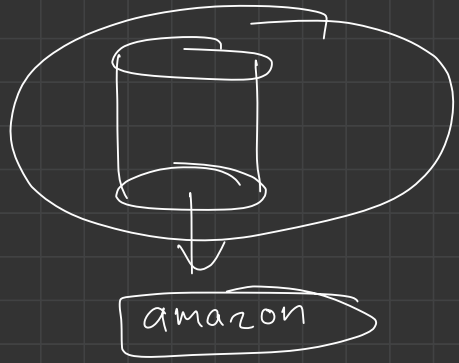
5 N/W DB

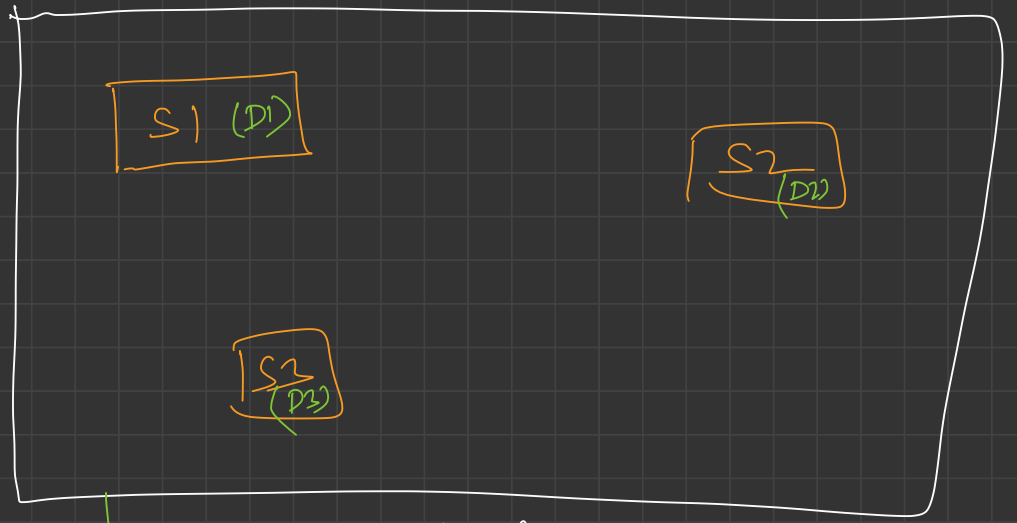


Lec-17

(Clustering / Replica set)







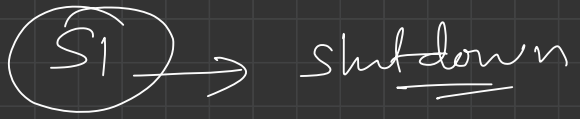
D1 = D2 = D3

Set of servers.
↳ Replica sets.

↓

Clustering

① Redundancy :-



② Load Balancing :=>

③ Availability ⇒

High ↑

S_1 ⟶ down

⟶ S_2 & S_3 ⟶ Backup

⇒ Homework

⟶ (Content delivery Network)

⟶ INTernet

Lec-18

⇒ Data → store → DBMS

↓

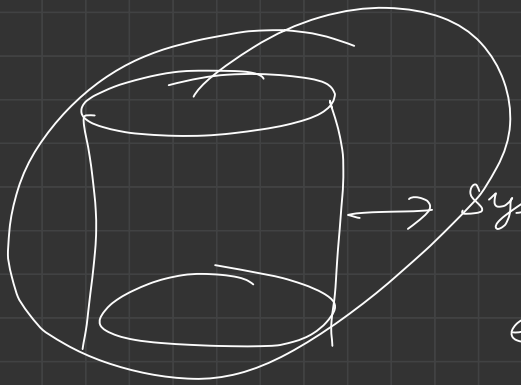
Bhot zada huge data → Masses.

↓

www hosted

↓

Facebook
insta
Amazon



→ System

↓

enhance

① Data huge (Manageability)

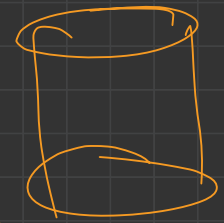
② no. of Request are large.

Data distribute

↳ Distributed database

Database Optimization

1. Scale-up (hardware) ↑



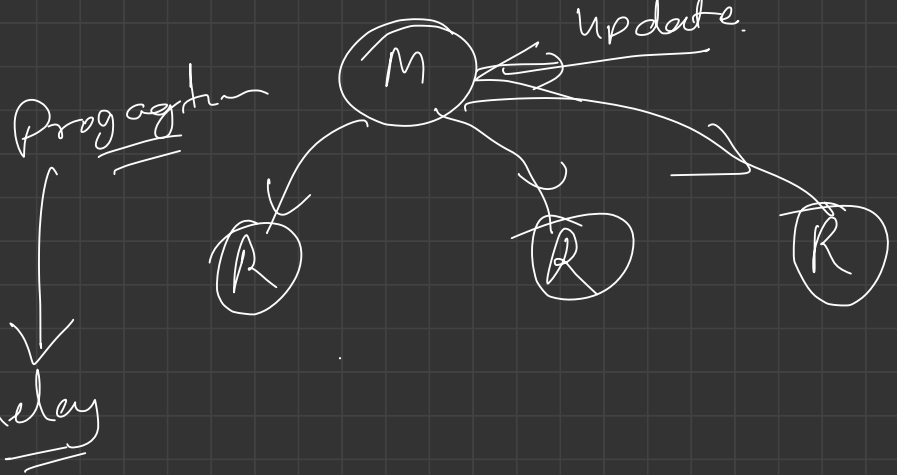
1TB → 2TB

X12 CPU → 2X12 CPU

X RAM → 2X RAM

— expensive Bet

② Add Replica-set (Clustering)



③ Partitioning

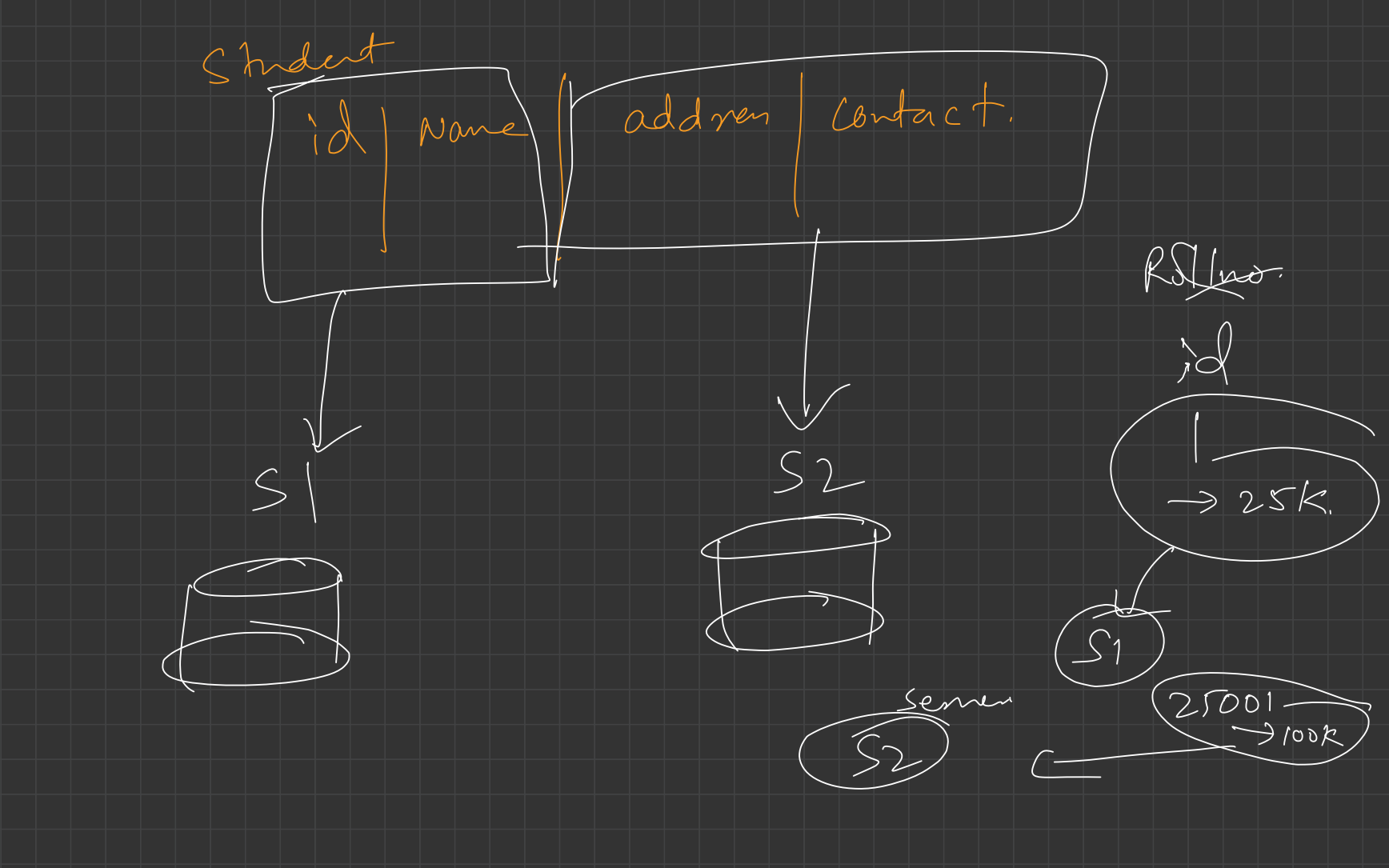
⇒ Partitioning

→ Scale-out (Horizontal scaling)

→ New nodes are added.

Student Table.

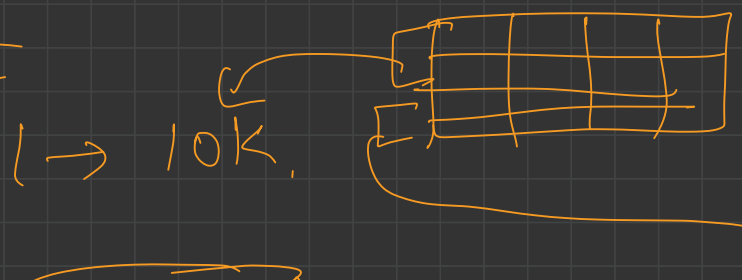
id	Name	Class	addm	phone
x				
x				
x				


Student
id
Name
address
Contact.
S1
S2
RollNo.
id
1
→ 25K.
S1
25001
→ 100K
Server
S2

*Sharding

→ Horizontal partioning → apply

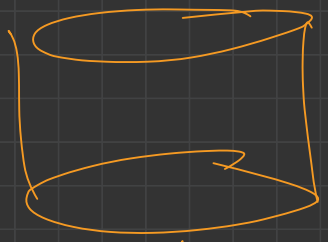
Student



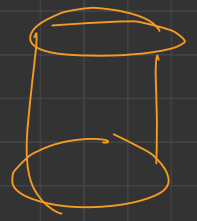
1 → 10K.

10001 → 100 K

fechinga



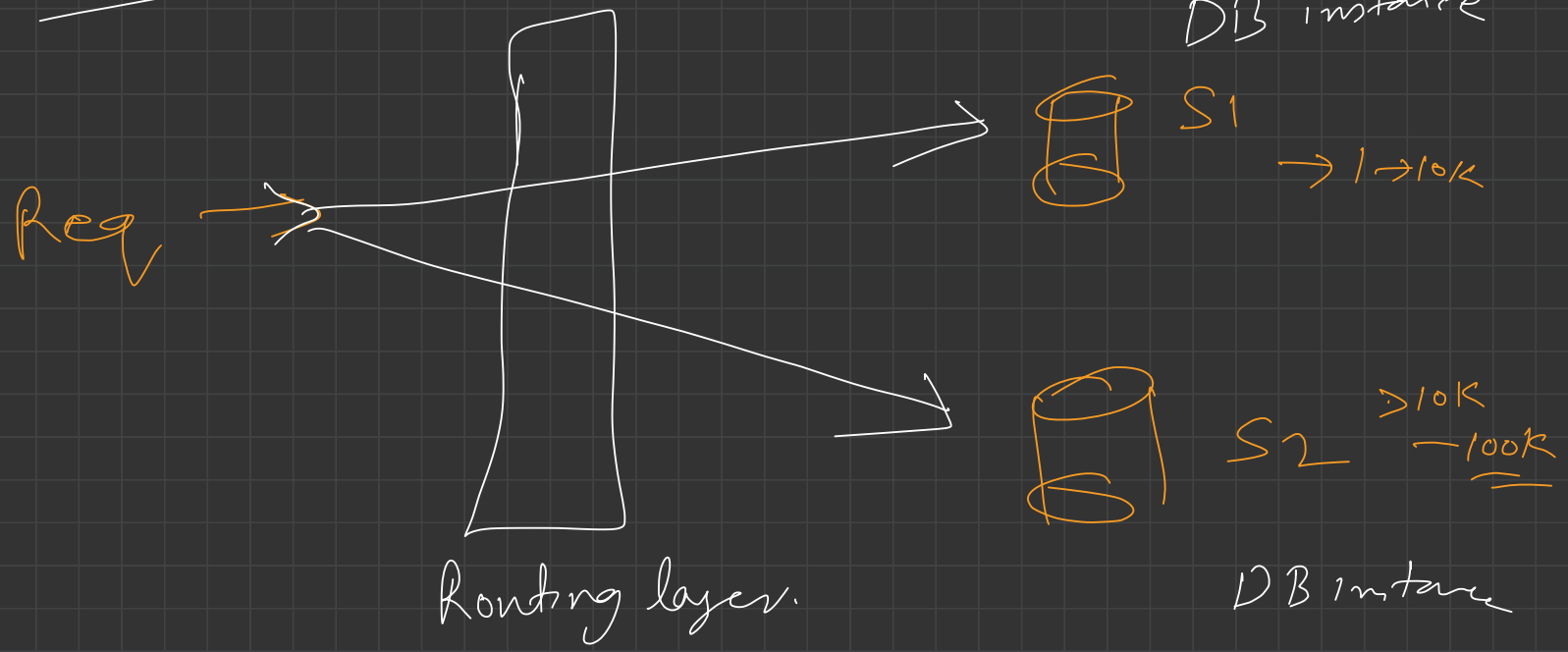
S1

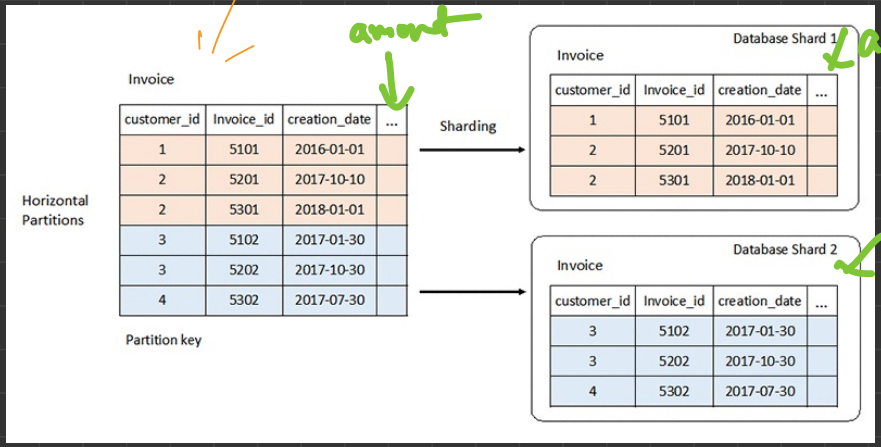


S2.

Independent

Routing layer.



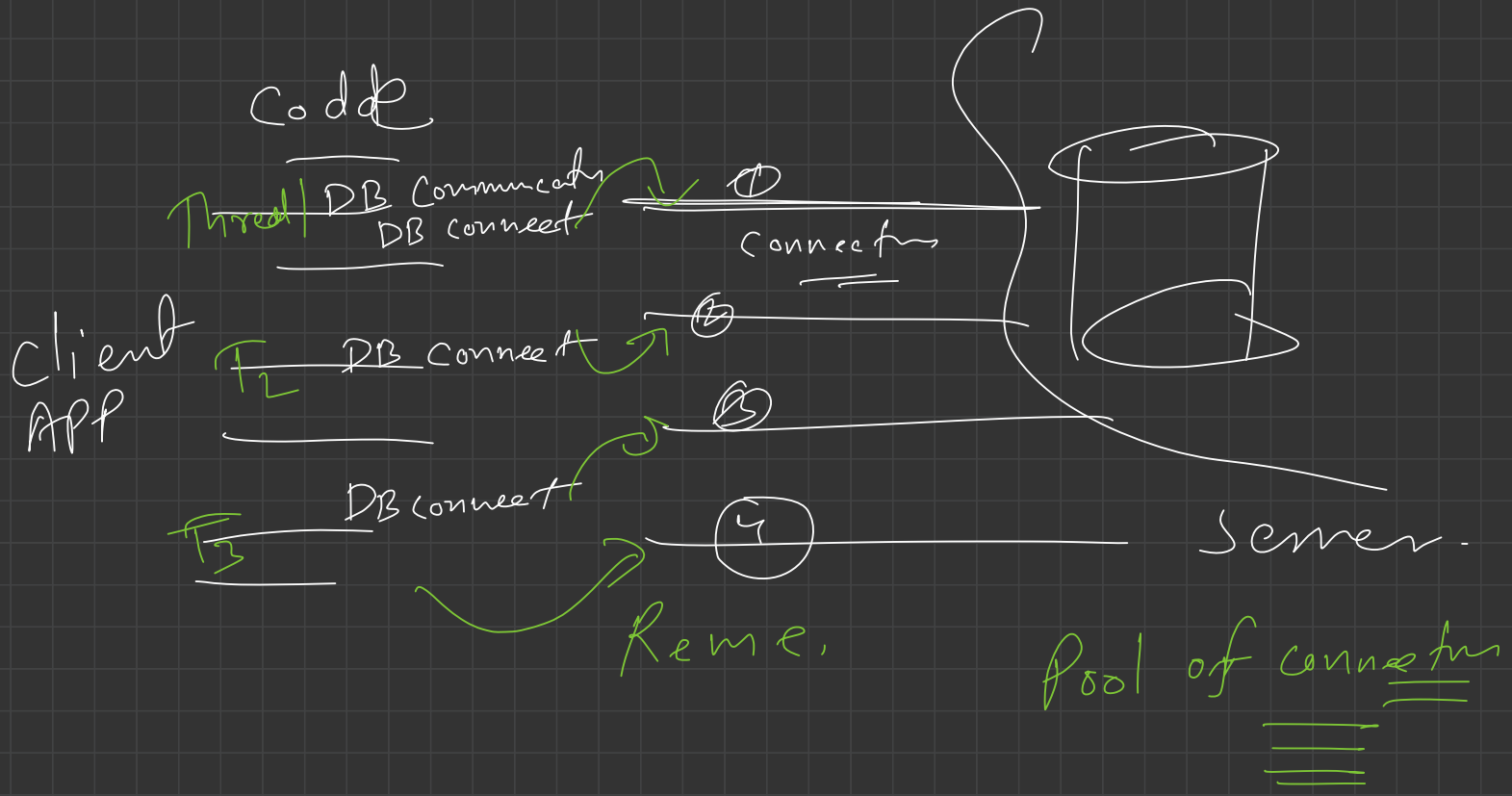

Invoice
customer_id | Invoice_id | creation_date | ...
1 | 5101 | 2016-01-01
2 | 5201 | 2017-10-10
2 | 5301 | 2018-01-01
3 | 5102 | 2017-01-30
3 | 5202 | 2017-10-30
4 | 5302 | 2017-07-30
Horizontal Partitions
Partition key
Sharding
Database Shard 1
Invoice
customer_id | Invoice_id | creation_date | ...
1 | 5101 | 2016-01-01
2 | 5201 | 2017-10-10
2 | 5301 | 2018-01-01
Database Shard 2
Invoice
customer_id | Invoice_id | creation_date | ...
3 | 5102 | 2017-01-30
3 | 5202 | 2017-10-30
4 | 5302 | 2017-07-30

Lec- 19

→ Design

DB scaling Patterns

Cache DB Connections

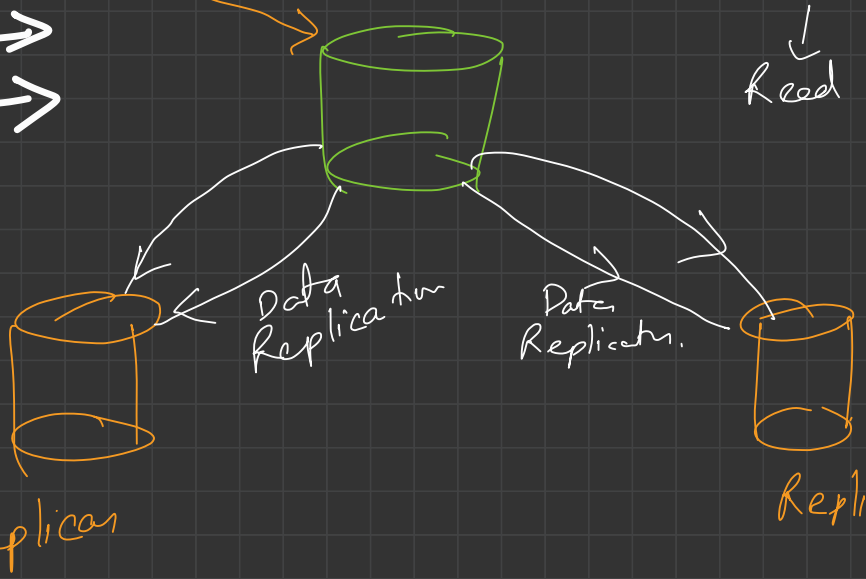


Pattern 3 : 1

write

commit

Primary



Replica

Replica

N ⇒ N/2 ⟷ N/2

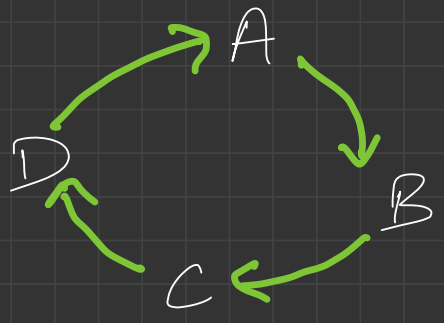
N/2 → Read

N/2 → write

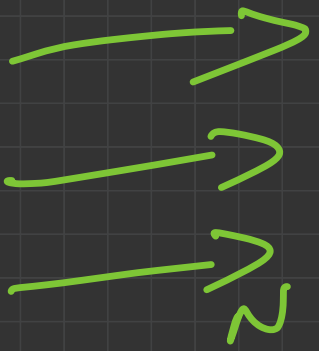
Read

Read

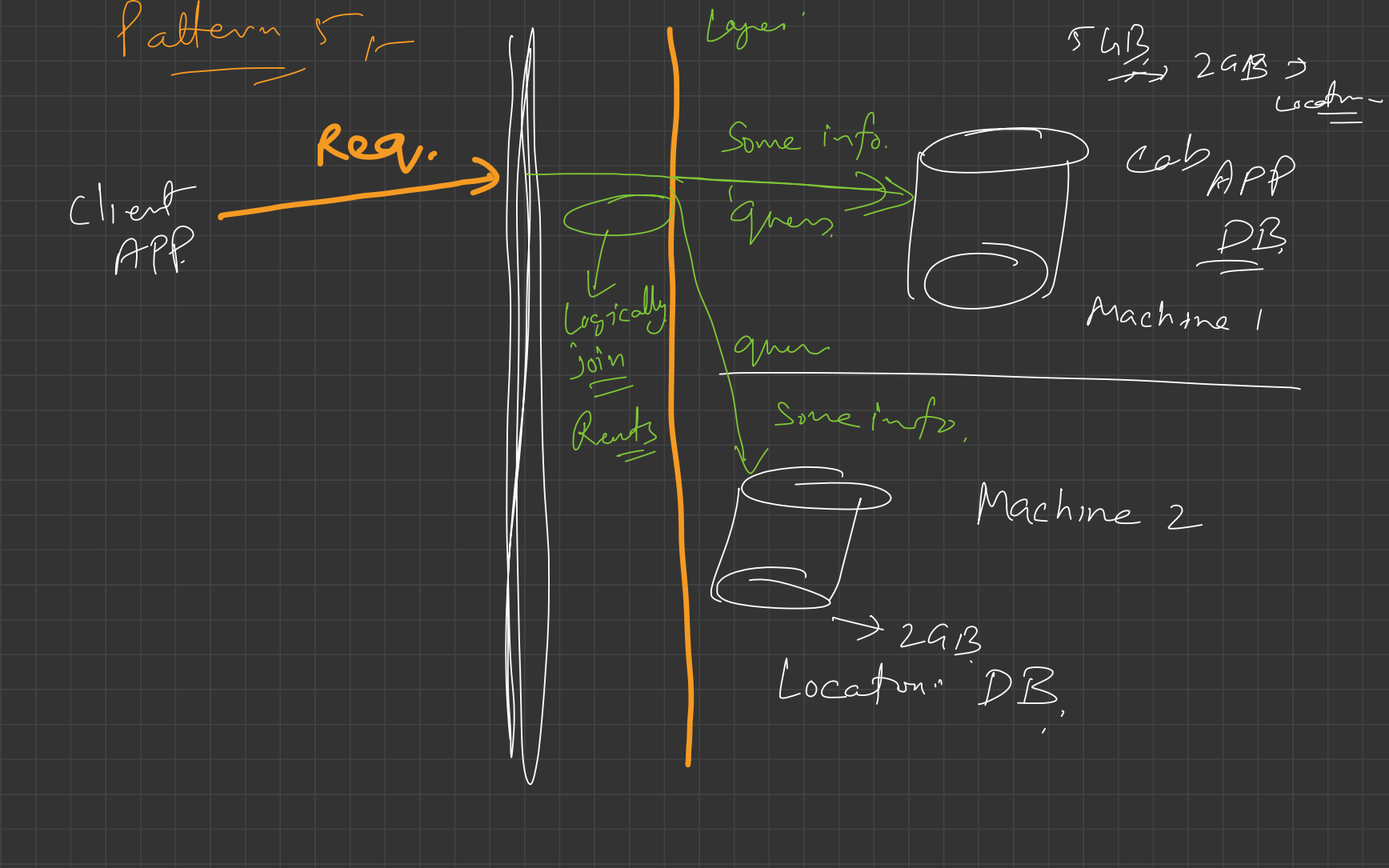
Pattern 4 =>



Write => Any node will handle it

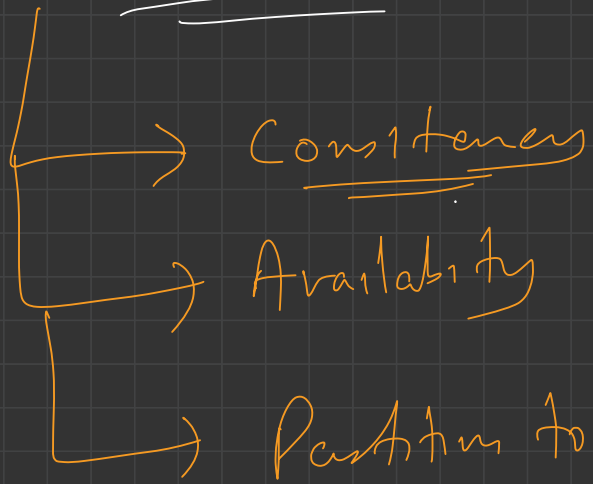


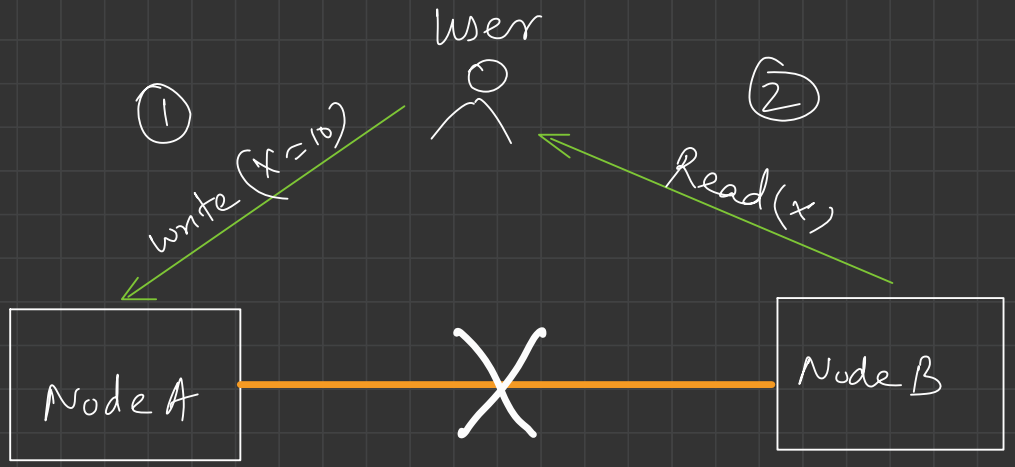
R/W


Pattern
Layer
5 GB
2 GB
Location
Req.
Client APP
Some info.
query
cab APP DB
Machine 1
Logically join Results
query
Some info.
Machine 2
2 GB
Location DB

Lec-20

CAP Theorem





① Consistency lost ⇒ (Both req. should procen)

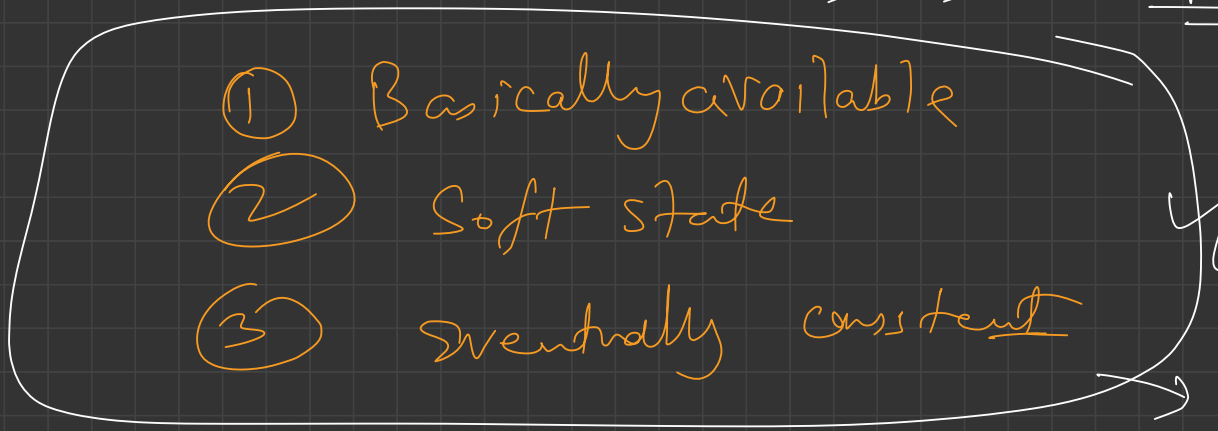
② Availability lost ⇒ (if we fail one req)

→ Homework

→ Banking system → ACID

→ Social Networking system

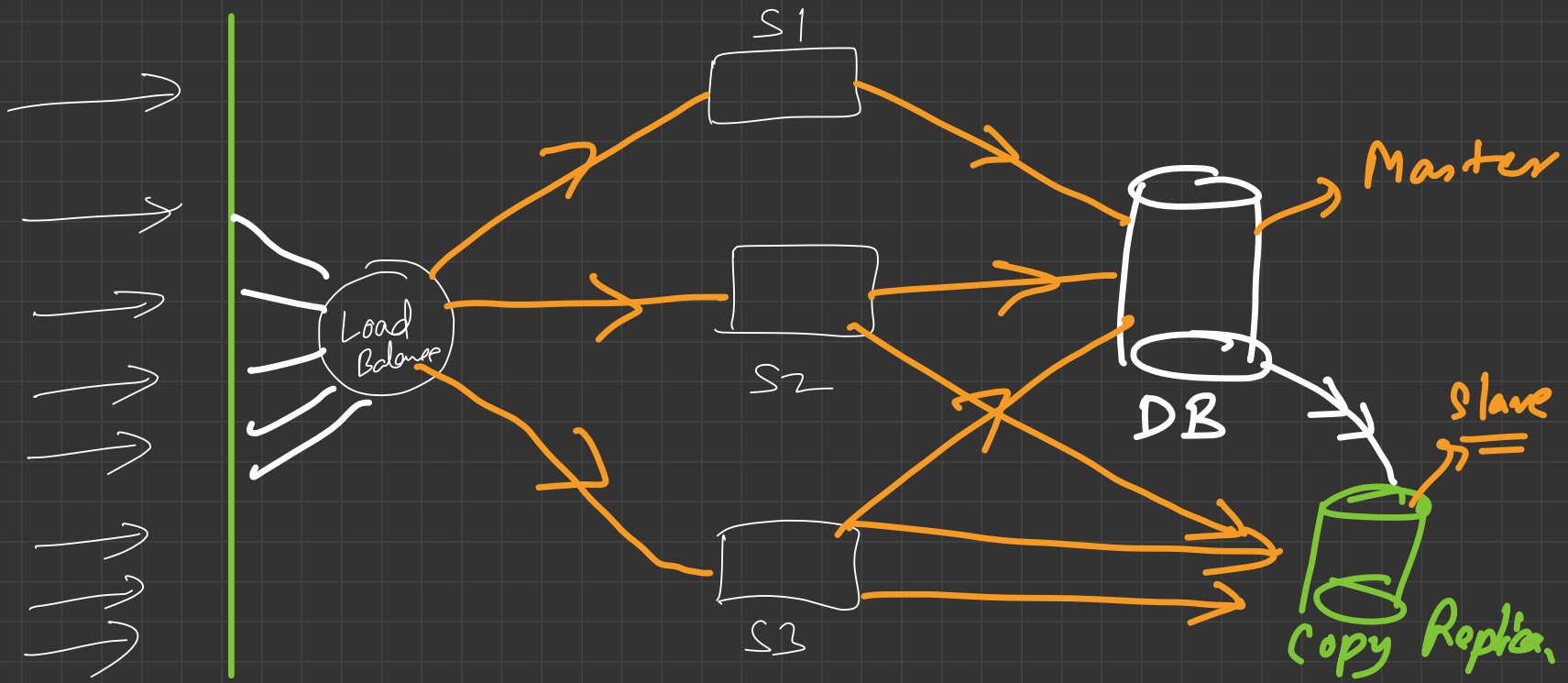
→ BASE properties



Comments Brief

Lec-21

* Master - slave Architecture ☆☆

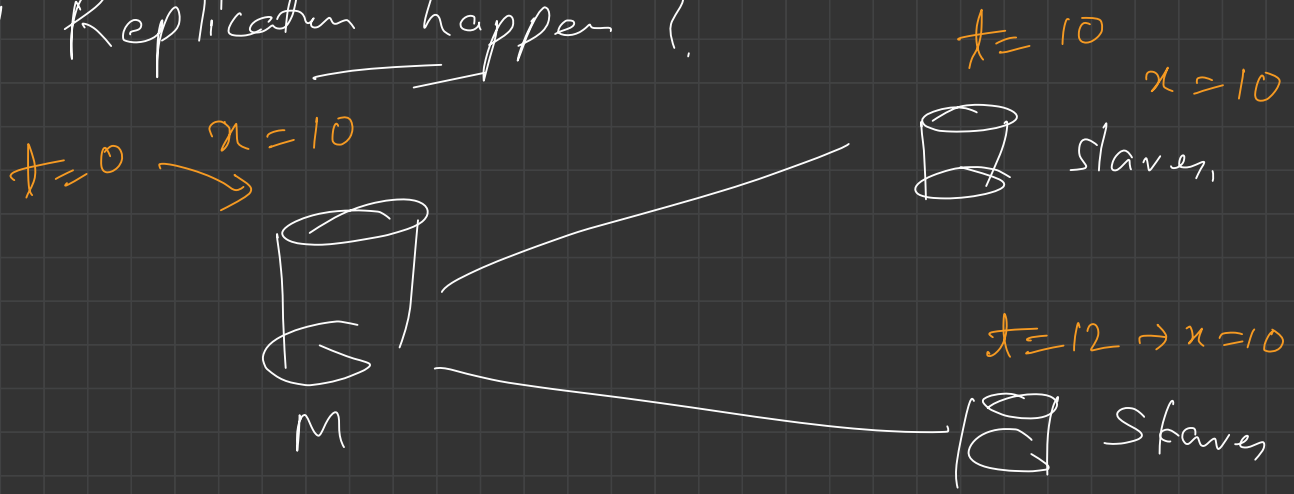


* Single Point of failure

→ 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 (more Replica)

How Replication happen ?



→ ① Asynchronous / Async.

→ FB Comment / Likes

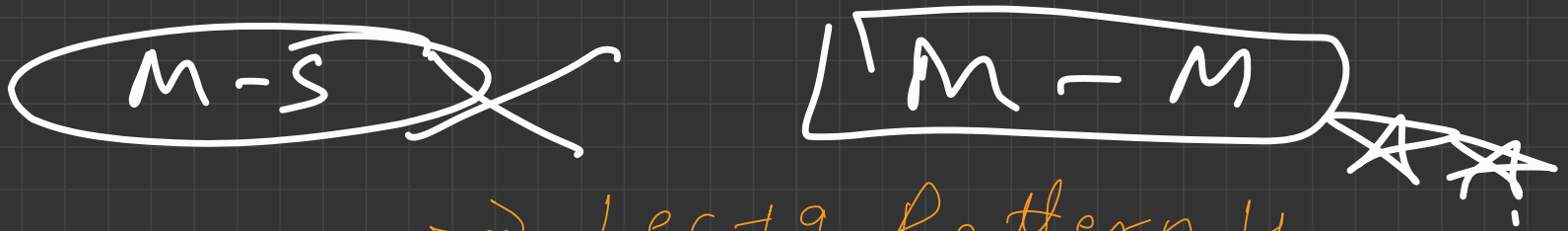
② Sync.

→ Banking App.

Q Slaves gets update query ??

1. ignore / never allow slaves to accept write

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



=> Lec+9 Pattern 4

* Advantages

1. Back up.
2. Scale out Read op.
3. Availability
4. Reliable
5. Latency Reduce

DBMS Placements Series 2022

Over

① MySQL

All the Best ★★ Placements

Revise

⇒ Questions

SQL

** Googling ✓

✓ lakshay k12

✓ love Ba BBar

(instagram)

codehelp