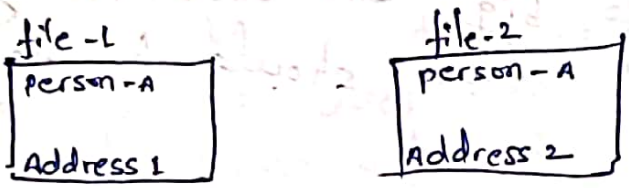
Advantages of DBMS / Disadvantages of File management system.

1. Data Redundancy and inconsitency

Redundancy - Meaning. Something in unnecessary because it is more than needed



- different address for same person is inconsistency.
- Two different files for same person is redundancy
- However everything is possible in file management system also but we have to write individual separate programms for them.

2. Difficulty in accessing data - Reformatting is required

3. Data isolatation
   - Different formats are available for same data.

4. Data integrity problem - Data should be trustworthy.

5. Atomicity problem
   - ↳ Single entity (In a single shot)
   - ↳ In a transaction money is debited from an account and credited to an account.

6. Concurrent access anomalies
   - ↳ get request at the same instance should be processed in a queue.
   - ↳ A/c Balance - 1000
     Two debit requests of same instance of ₹ 1000 rupees. only one transaction should be Completed.

7. Security problem:
   Data access problems, like confidentiality of data.

However everything is implemented in file management system is possible but we have to write every program individually and manually.

But now a days the volume of data on internet is too big and accessibility and processing of data should be fast.

Lecture-2

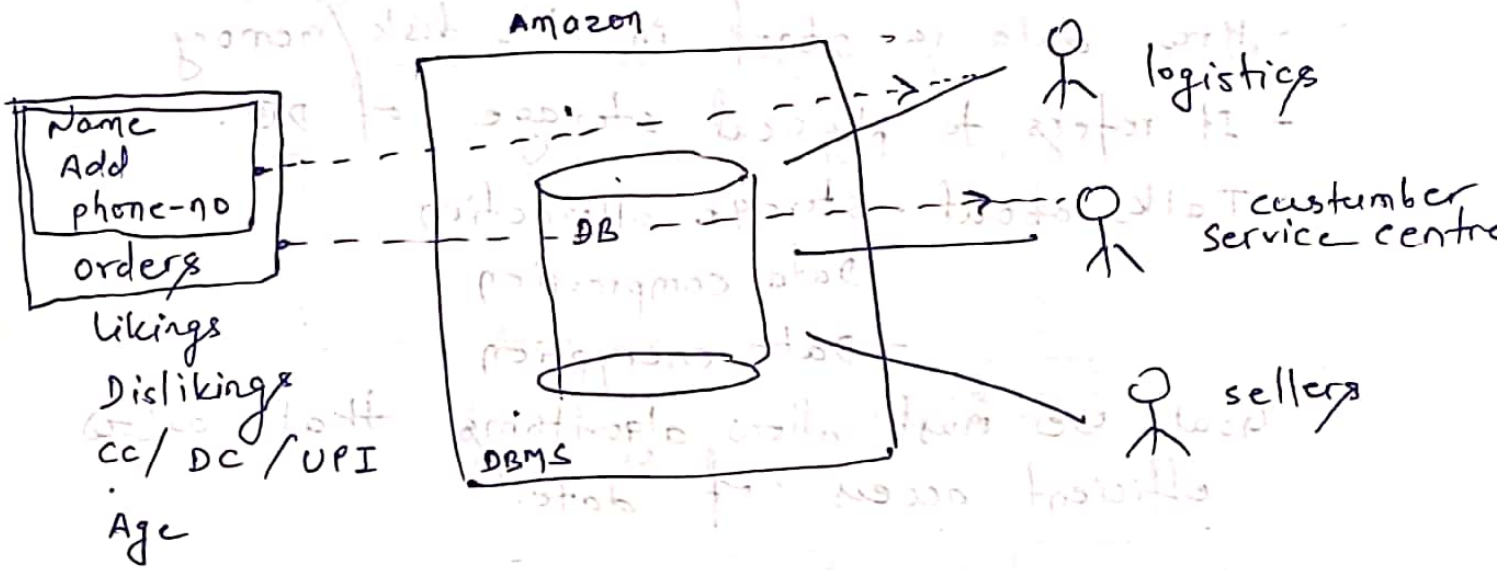
Abstraction:

Ex- Car-driving → Steer, A/B/C, Gears

Driver should take care of very less thing. At the backend whatever is happening not needed to known by the user.

- Abstraction simply hides the complex system behind and shows only the part needed for user and make interaction user friendly.
- System hides underlying complexity.

- DBMS provides abstract view to the users.



1. View of

1. view of data. (Three schema architecture)

- schema - Design - How it looks?
- Main objective of three level architecture is to enable multiple users to access the same data with a personalised view while storing the underlying data only once.
  - → physical level / internal level
  - → logical level / Conceptual level
  - → View level / external level

1. Physical level / internal level
   - How data is stored in the disk / memory.
   - It refers to physical storage of DB.
   - Talks about - storage allocation
     - Data compression
     - Data encryption

   Goal: We must allow algorithms that allow efficient access of data.

2. Logical level / conceptual level

physical level

name, phone, add, batch
name, phone, add, batch
, , , , ,
, , , , ,
, , , , ,
, , , , ,

logical level

Sl. No.	Name	phone-No.	add	batch
01	S1	xxx	add1	b1
02	S2	xxx	add2	b2
03	S3	xxx	add3	b3

- It describes what? data are stored in DB, and what relationship exist among those data.

DBA: Database administrator
: logical level abstraction.
Goal of logical level: Ease of use.

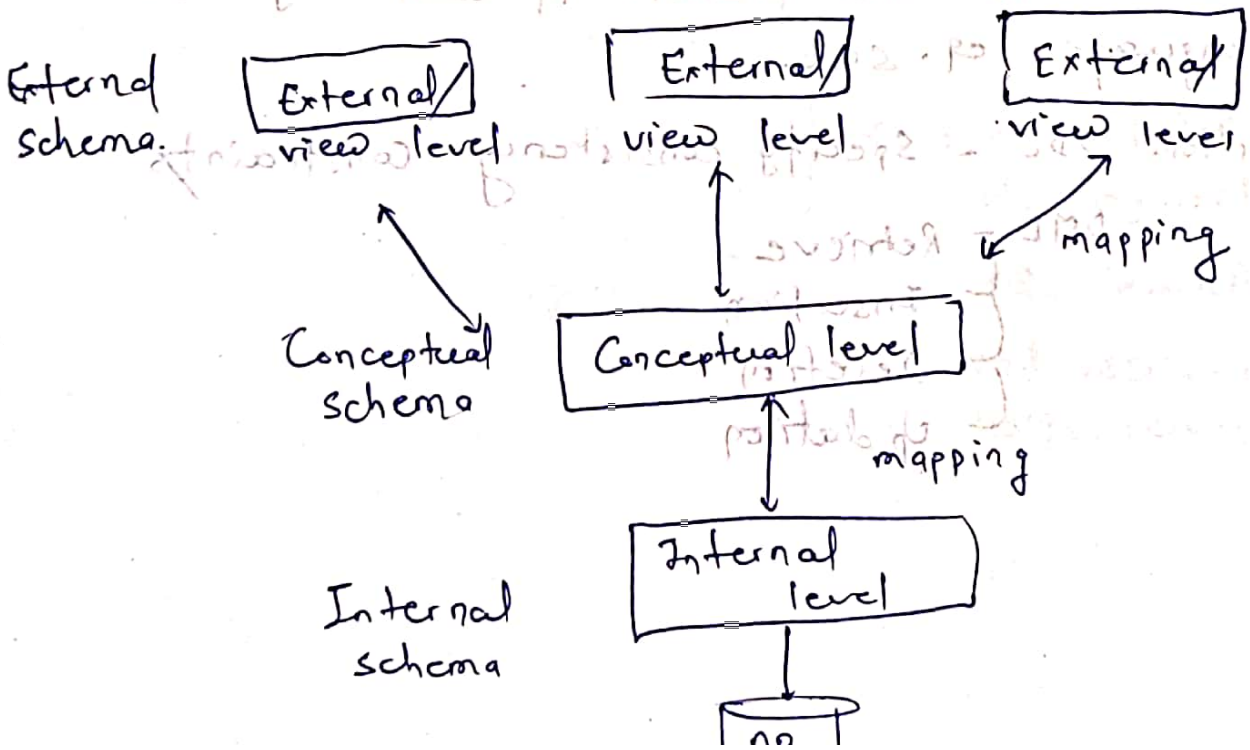
3. View level / External level
- provides different view to the end user.
- Each view is called subschema.
- provides security mechanism to prevent users to accessing certain parts of DB.
- Each level of abstraction is independent of other levels. i.e. user at any level does not have to care about other levels.

logistics - subschema (view to DB)

Name	add	phone-no

service - subschema (view to DB)

Name	add	phone	products



Instance of DB
↳ No. of objects / students / rows at any time.

- Overall Design of DB is called DB schema.

DB schema → attributes of table (Variables)
↳ Consistency constraints (primary key).
↳ Relationships.

Data Models
- provides a way to describe the design of a DB at logical level.

eg: ER model
Relationship model
Object-oriented model
Object-relational data model

Database languages

DDL: (Data definition languages)
DML: (Data manipulation languages).

DDL and DML are present in single DB language: eg. SQL.

Function: DDL - Specify consistency constraints

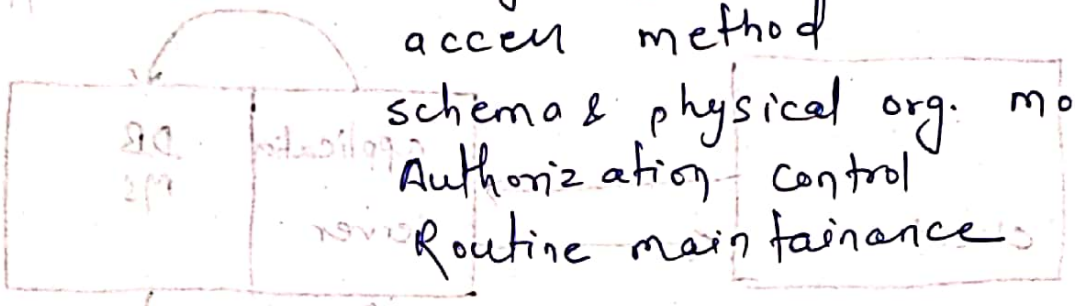
DML → Retrieve
Insertion
Deletion
Updation

How DB accessed from application programs?

Host language: c/c++/java.

1. ODBC : open DB connectivity , microsoft
2. JDBC : java DB connectivity , java

* DBA : Administrator : Works at logical user
  - Controls the both Data and programs that access those data.
  - functions: schema Definition
    storage structure
    access method
    schema & physical org. modification
    Authorization control
    Routine maintainance

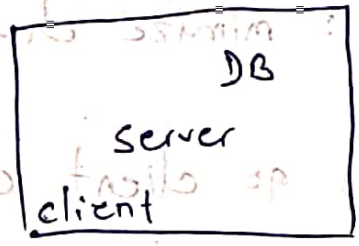


DBMS application arcitecture

remote user
end user

actual DB system is running

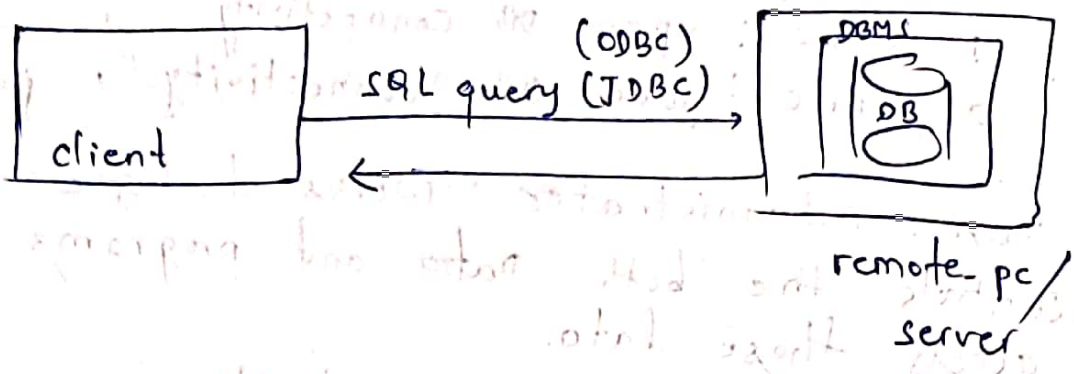
1. T1



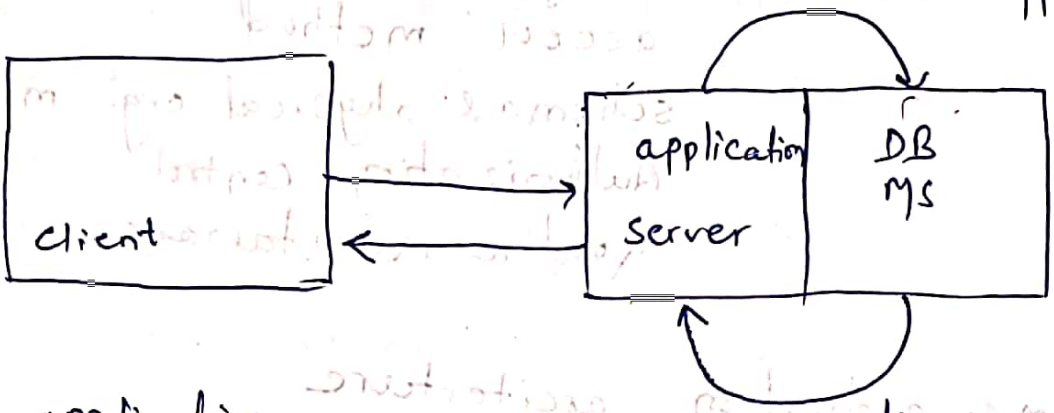
system

eg: 1. Web-application development phase
2. SQL learning phase
end-user → my PC
DBMS-server → my PC

2. T2



3. T3 : use on large scale/word wide application



- application server works as middle layer

advantage

- Scalability due to distributed app. servers. (more than one).
- Data integrity : minize chances of data corruption.
- Securioty : as no client can access DB directly.
- No direct connection Between client and DB different from T1 and T2 arcitecture

Lecture-3 : ER data model

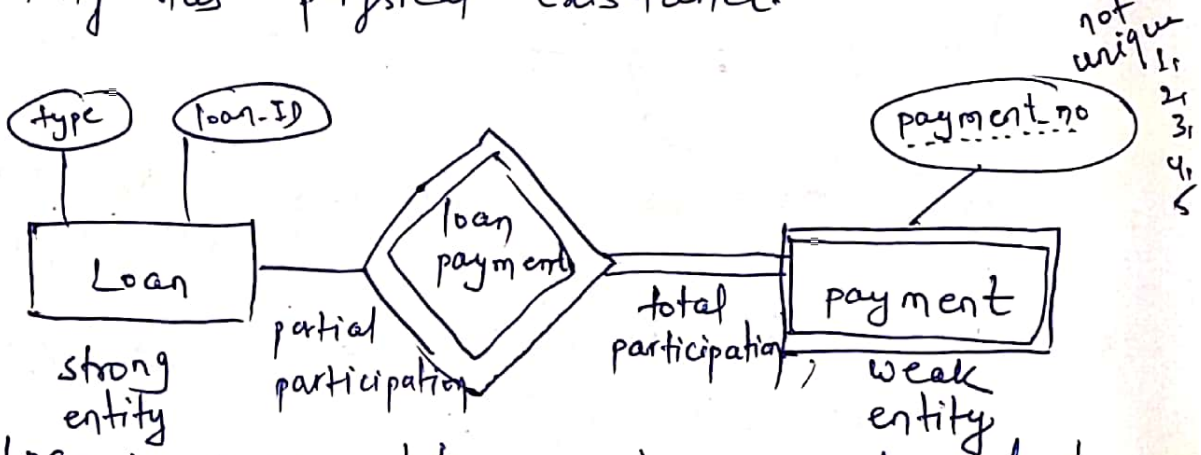
* Data model works on logical / conceptual model.
- ER: Entity relationship model

Data Model : Collection of conceptual tools for describing data, data semantics and consistency constraints.

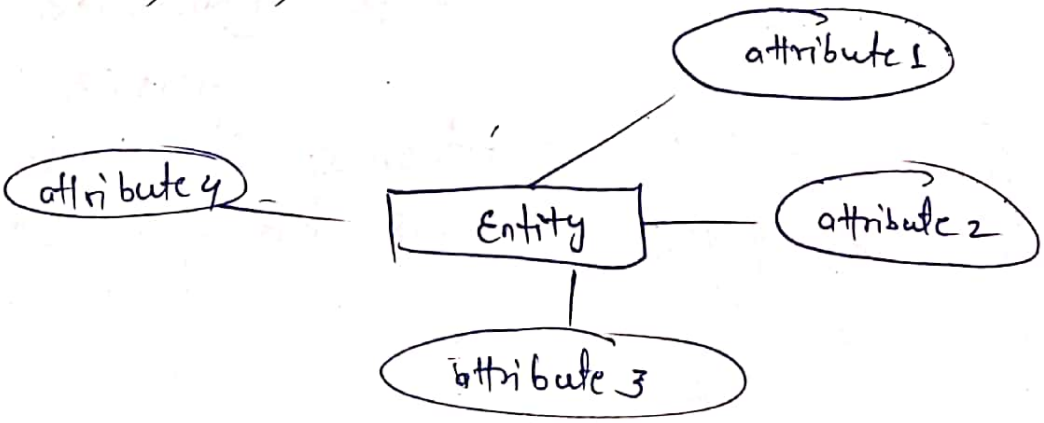
Entity: An entity is a thing or object in the real world.

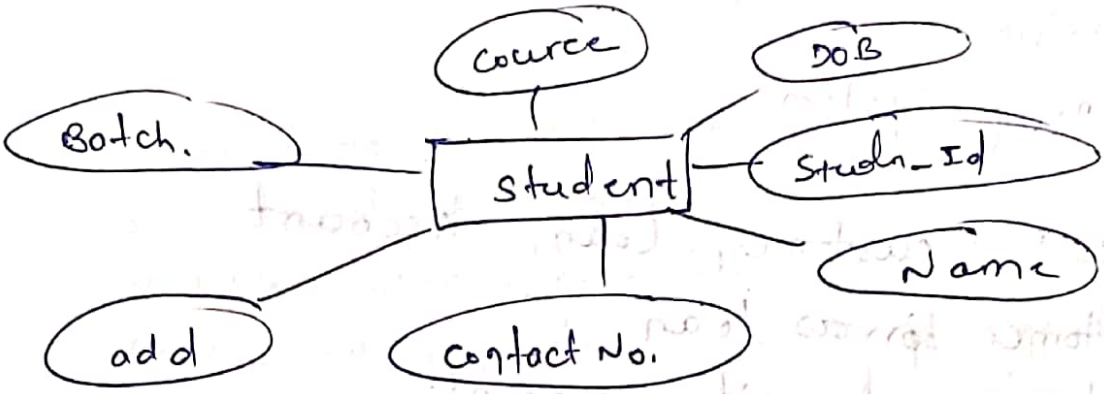
- strong entity : Uniquely identified
- weak entity : depends on strong entity for existance and can't be uniquely identified.

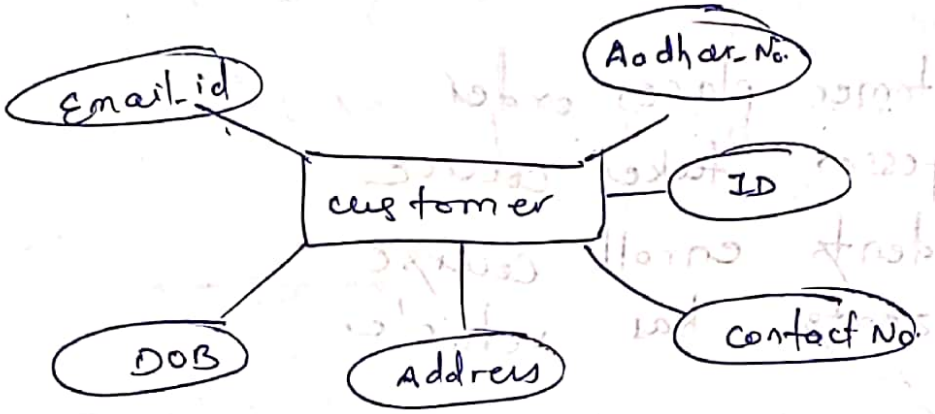
- Entity has physical existance.



- Loan is strong entity as it is independent
- payment is weak entity as it depends on loan for his existance.







. unique attribute is also called primary key.

* Entity-set

  student$_0$ { student1, student2, ---- student n }.

  - Set of entities that share same properties/ attributes.

  eg- student is an entity set.

  - Customer of Bank.

* Attributes.

  An entity is described by attributes.

  ~~Types~~

Relationship

Banking system

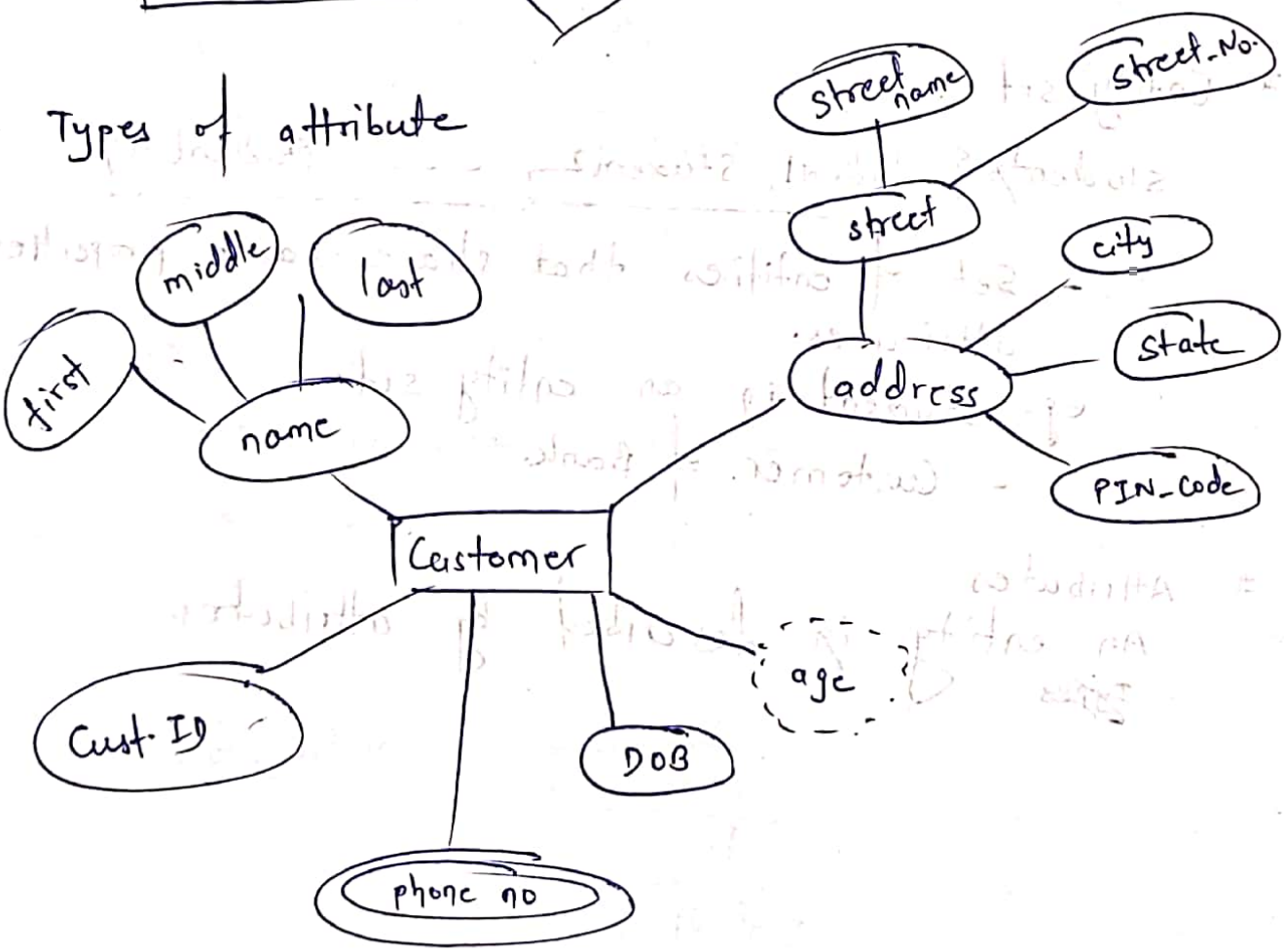
entities : customer, loan, Account

a) Customer borrow loan.

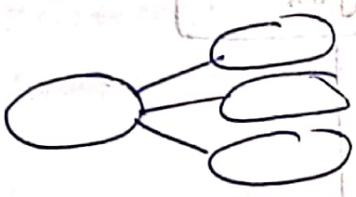
b) Customer deposit account.

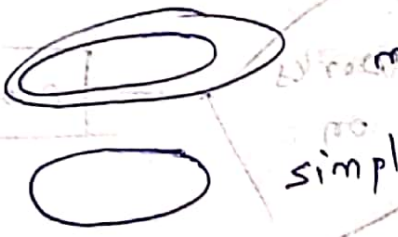
- Customer places order
- professor takes cource
- Students enroll course
- Citizents has vehicle.

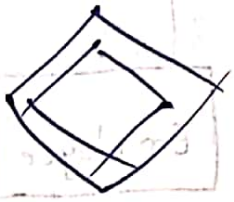
Types of attribute

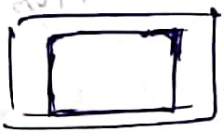


1. Simple - can't be divided further (AC_No, roll_no).
2. Composite - Name, address (street, city, state, PIN).
3. Single valued - Only one value attribute (ID, Loan_No).
4. Multi-valued - more than one value (Mobile No.)
5. Derived - depends on other (age depends on DOB)
6. NULL ⓐ entity does not have a value
   - middle name
   ⓑ Unknown — missing entry
   not known yet (solary for new employee



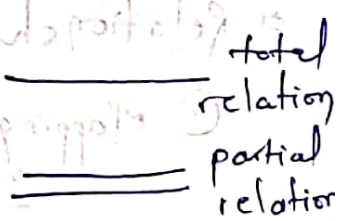


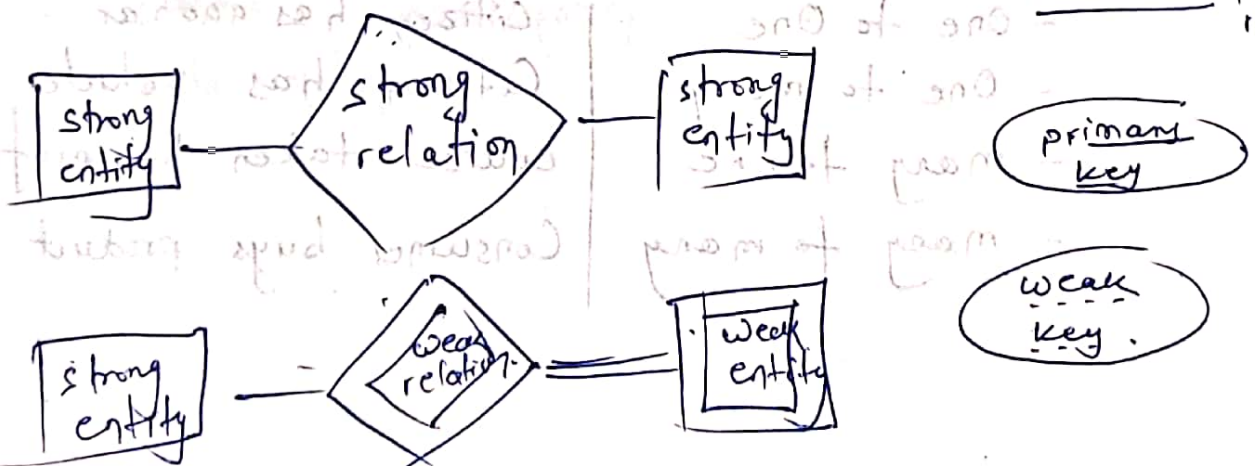




Relationship
association among 2 or more entities.

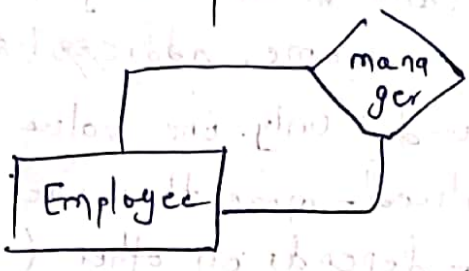
strongentity - unique primary key
weak entity - not a unique primary key.



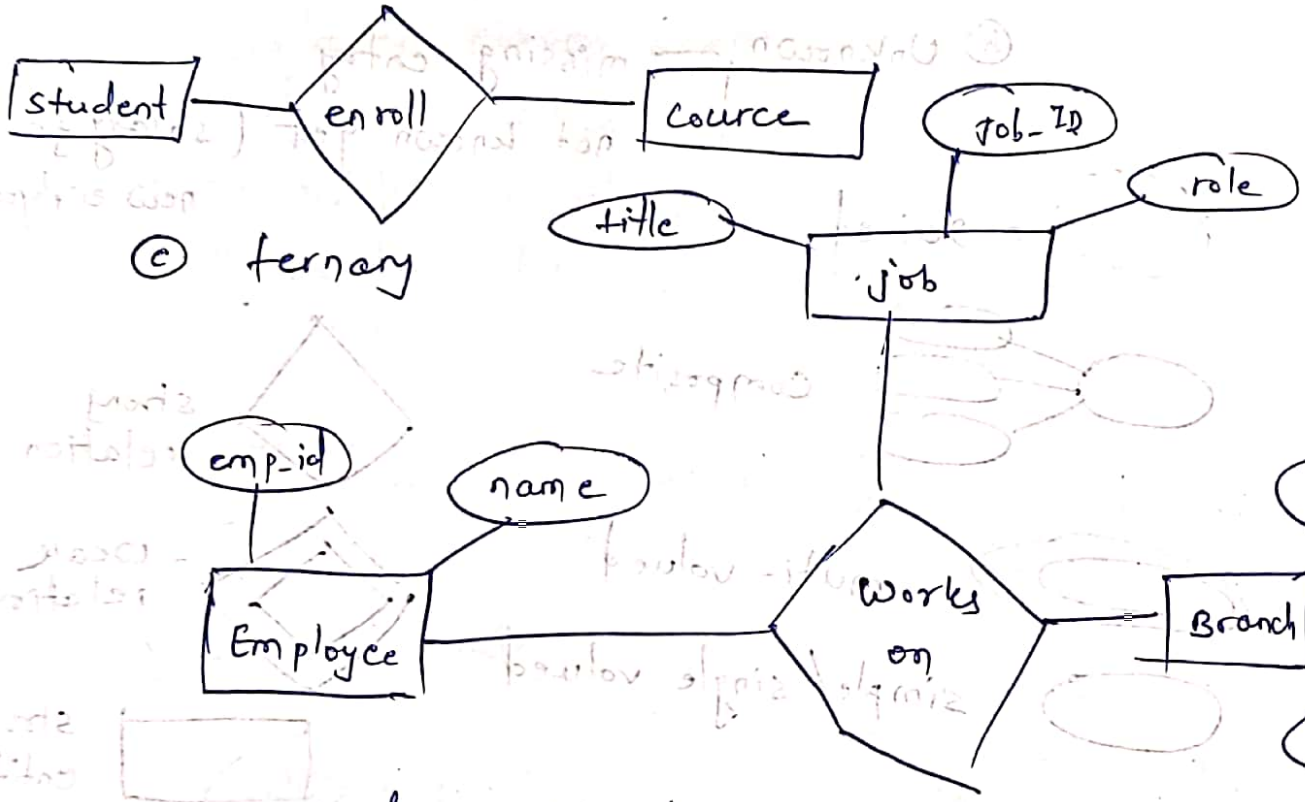


* Degree of relation : No of entities in a relation

① Unary



② Binary



③ ternary

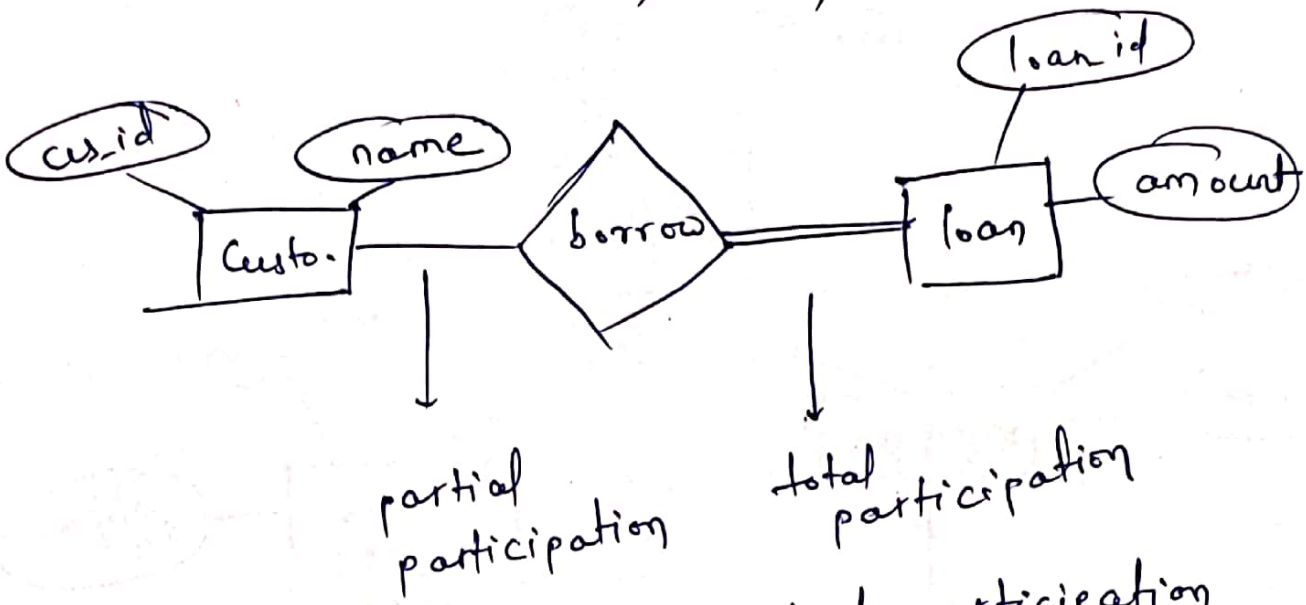
Most of relationships are binary.

* Relationship constraints

① Mapping cardinality

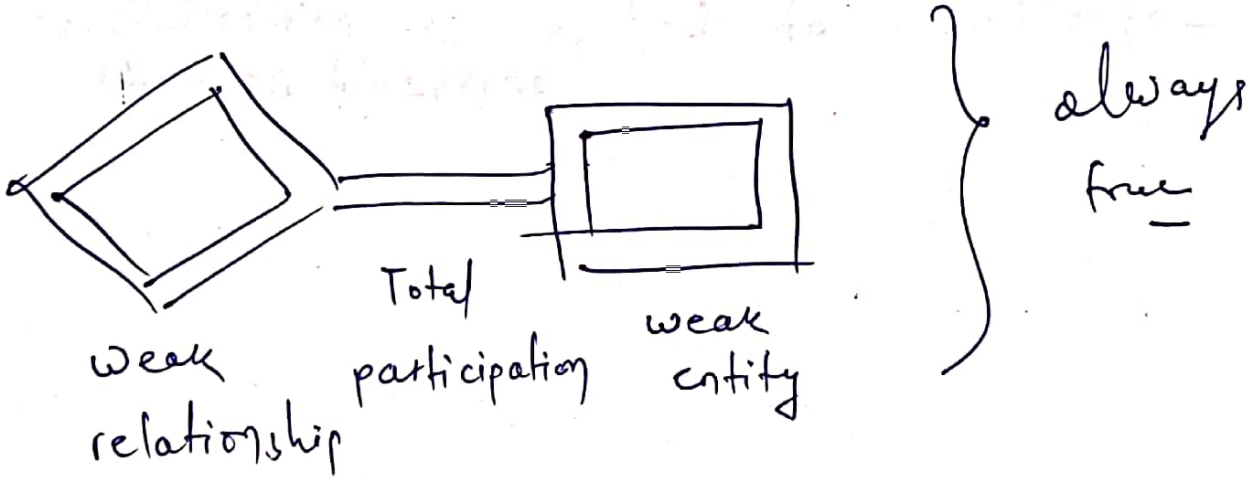
- One to One	Citizen has aadhar
- One to many	Citizen has vehicle
- many to one	cource taken by professor
- many to many	Consumer buys product

② Participation constraints.



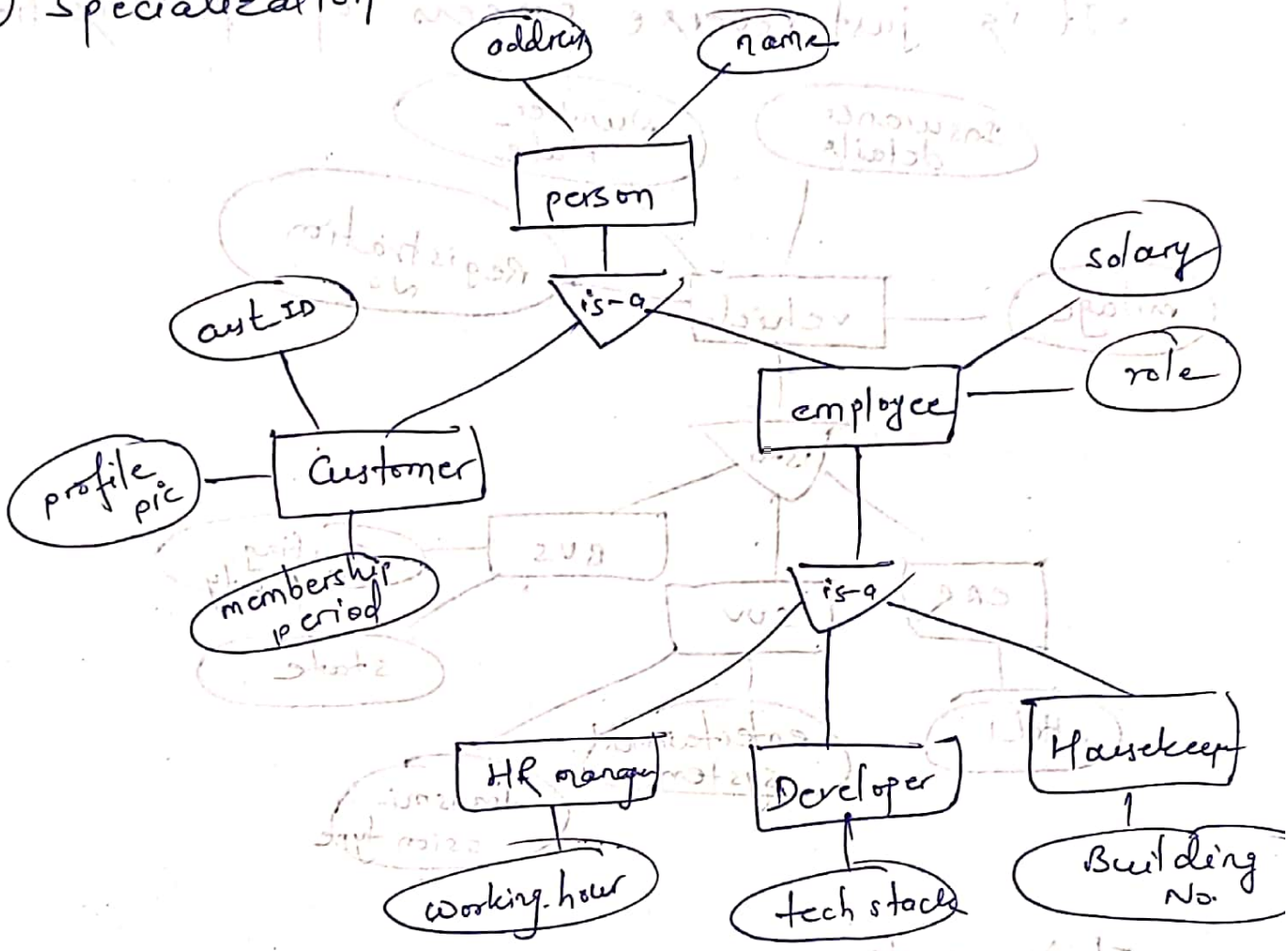
customer → loan : partial participation
loan → customer : Total participation

- Weak entity has total participation constraint but strong entity may not have total participation constraint.



Lecture: 4 Extended ER features

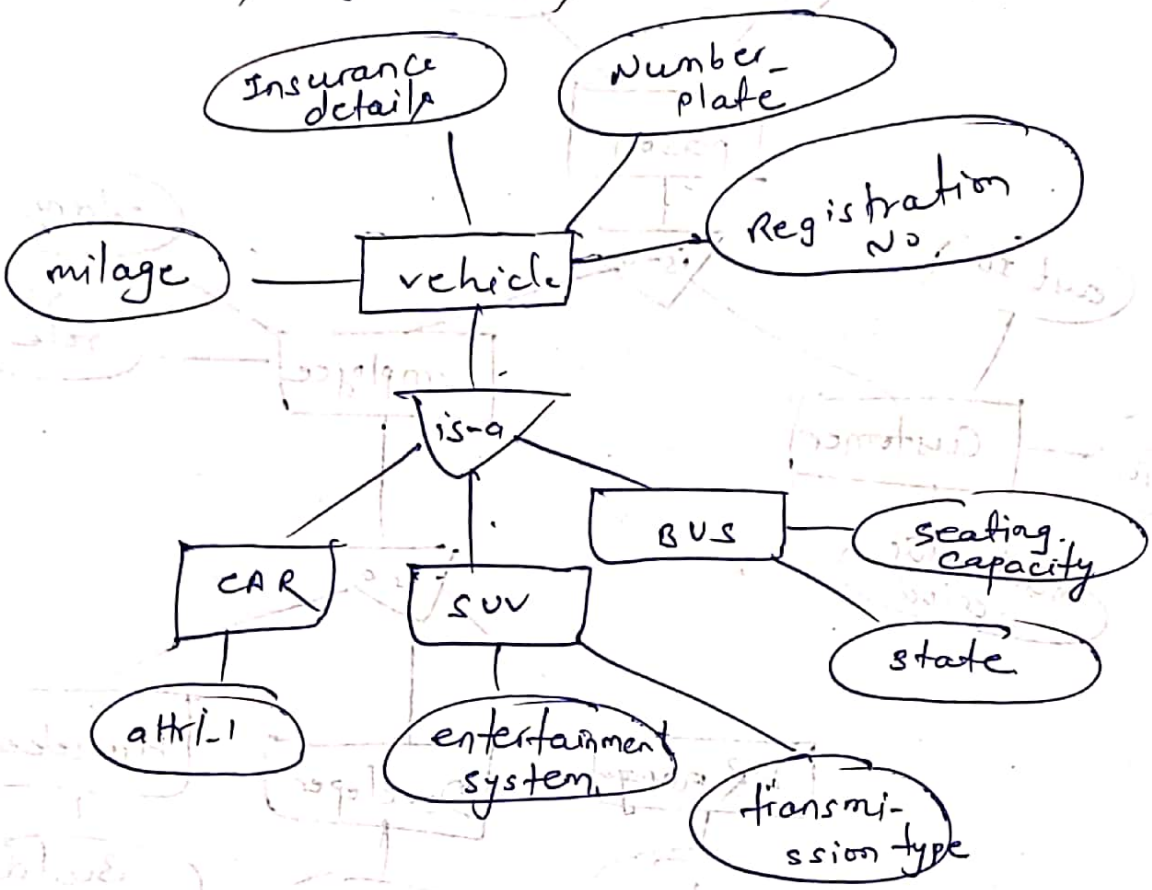
① Specialization



- It is a top-down approach
- We have created subclasses of the superclass.
- Specialisation is applied to overall refine the DB-blueprint.

2. Generalisation

- It is just reverse process of specialisation.



It is simple

- If we first though of vehicle_ then it is top down approach/specialisation.
- If we first though of car/bus/suv it is bottom-up approach/generalisation.

Lecture: 5 How to think and formulate ER diagram:-

Q.1) Draw ER diagram of
① Online delivery system
② University

Steps
① Identify entity sets
② Identify attributes and their types
③ Identify relation & constraints
- relation → weak, strong
- constraints → mapping cardinality, Participation Constraint

Q.1) Draw EE diagram of Banking system

① Identify entity sets
- customers of Bank - accounts and takes loans
- employee
- ~~manager~~
- Branches of Banking system
- Loan
- Savaing account
- payment (weak entity) (Loan dependent)
- current account
- banker

② Attributes

ⓐ branch → name, city, assets, liabilities

ⓑ customer → Id, name, address, contact no, DOB, age
address → composite
contact no → multi-valued
age → derived

ⓒ Employee → employee_Id, name, contact no, (single valued)
dependent name, years of service, date of joining.
dependent name → multi-valued
years of service → derived

ⓓ saving A/c → account_no, balance, interest rate, daily withdrawl limit.

ⓔ current A/c → account_no, balance, per transaction charges, overdraft amount.

ⓕ superclass: Account of ⓓ & ⓔ
account-no
balance

ⓖ Loan → loan number, amount.

ⓗ weak entity payment → payment no, date, amount.

③ Relation & Constraints

ⓐ Customer borrows Loan.

1 : N → a customer have multiple loans
M : 1 → a loan is of joint customers (education loan)

M : N

=

Total participation: If a loan exist, it is of some customer

(b) Loan originated by branch:

1 : 1 → a loan is originated by one branch
N : 1 → a branch can originate many loans

N : 1

Total participation: Loan must be originated by same branch.

(c)

Loan — Loan-payment — payment

- It is a weak ~~entity~~ relation.
- weak relation shows total participation

1 : N one loan have multiple pay.
1 : 1 one pay. regards one loan

1 : N

(d) Customer deposits in account

1 : N a customes have multiple accounts
M : 1 an account have mutiple holders (joint a/c)

M : N

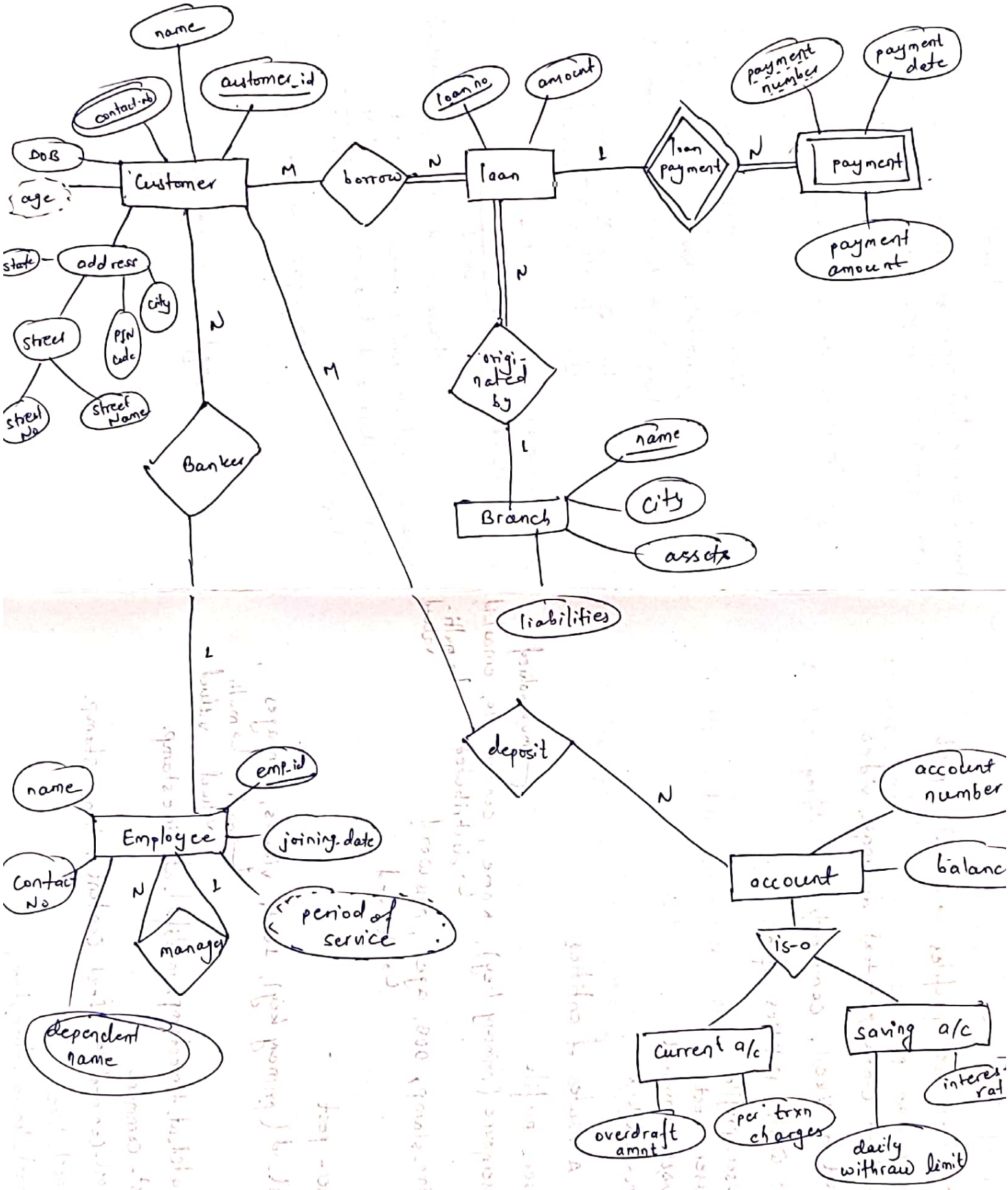
(e) customer banker employee

N : 1

(f) Employee manage by employee

N : 1

ER model of Banking system.



lecture:6 : DB of facebook

Features of facebook

① profile → user_profiles → friends, post

② user can post

③ post → contains → text, image, video

④ post → likes, comments

step-1 : Write Entities of DB

1. User_profile
2. user_post
3. post_comment
4. post_likes

step-2 : Attributes of entities

1. User_profile

usergame (primary key), Name (→ distributed), contact No (→ multivalued), email_id (→ multi-valued), created_time_stamp, DOB, age (→ derived), password

2. User_post

post_id (primary key), text, videos (→ multi valued), images (→ multi valued), created_at_timestamp, modified_at_timestamp.

3. post_comment

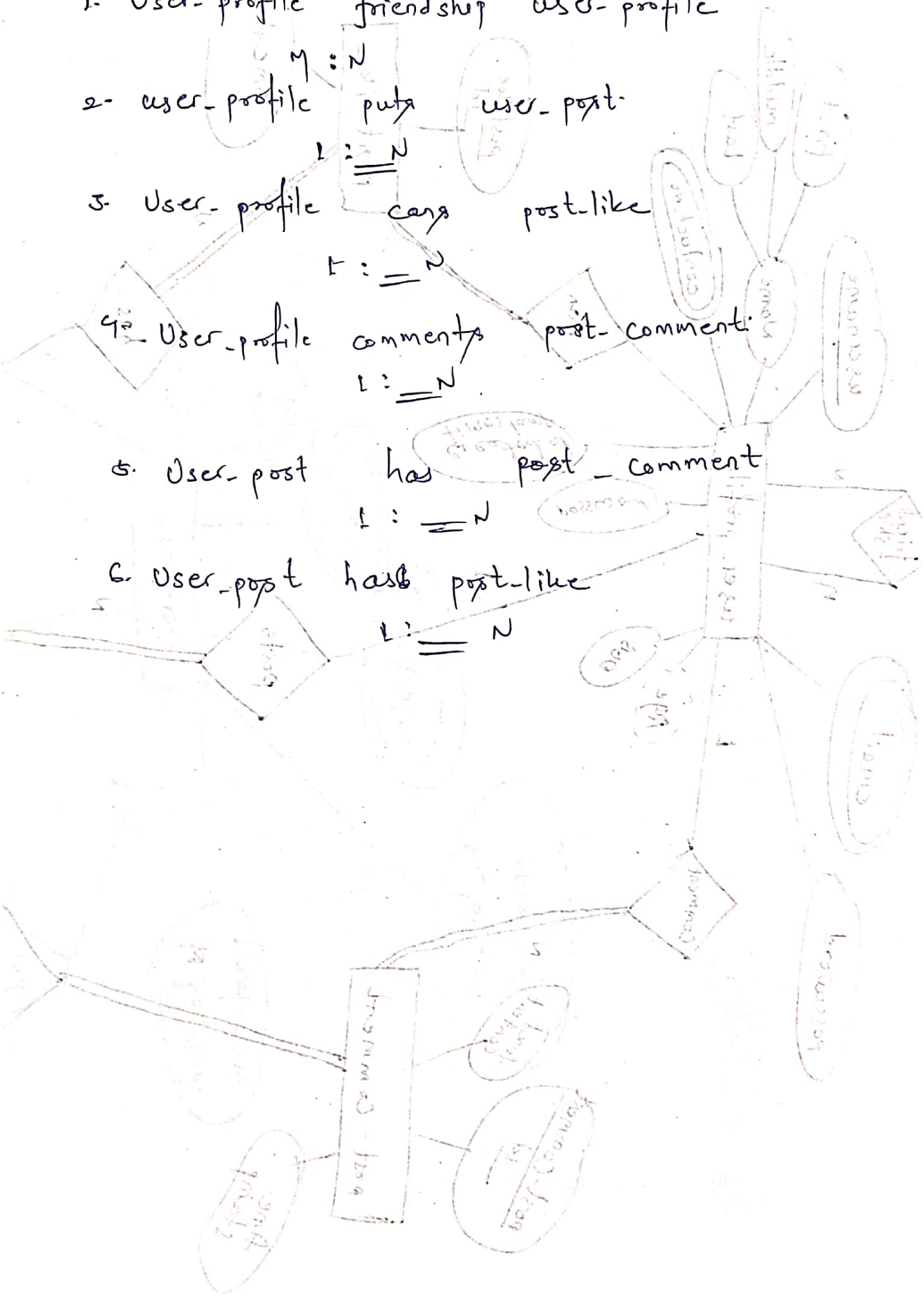
post-comment_id, text content, time stamp.

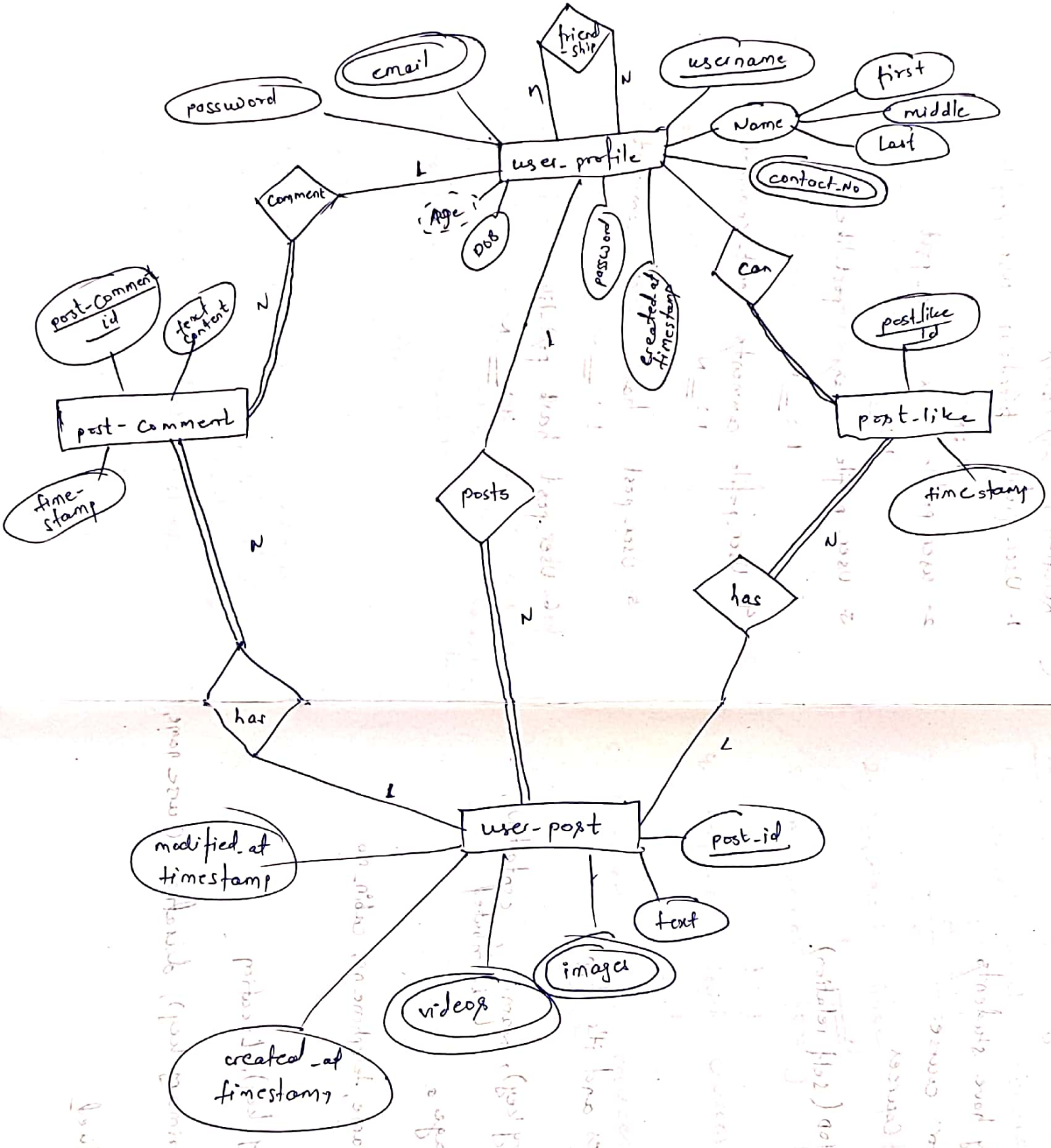
4. post_like

postLike_id, timestamp

③ Relation & Constraints

1. User_profile friendship user_profile
   M : N
2. user_profile puts user_post
   1 : N
3. User_profile cans post_like
   1 : N
4. User_profile comments post_comment
   1 : N
5. User_post has post_comment
   1 : N
6. User_post hasb post_like
   1 : N





H.W

1. Create ERD for University database

features

1. students → University have students
2. students enroll for course.
3. professors → take Cources
4. Departments
5. Department have HOD (self relation).

Step-1 Identify Entity set

1. students
2. professors
3. Department
4. Course

step-2 : Identify attributes and it's types

1. Student

student_id (primary key), Name, contact_No,
↳ distributed

email_id, DOB, age.

2. Professors

proffessor_id, Name, telephone_no, cabin_no

3. Department

name (primary key), Location

4. Course

course No (primary key), duration, Course name

prerequisites,

↳ muti-valued

Step-3 : Relation and Constraints

1. Student enrolls course

1 : N
M : 1

M : N

2. Department have professors.

1 : N

3. Department head by professors.

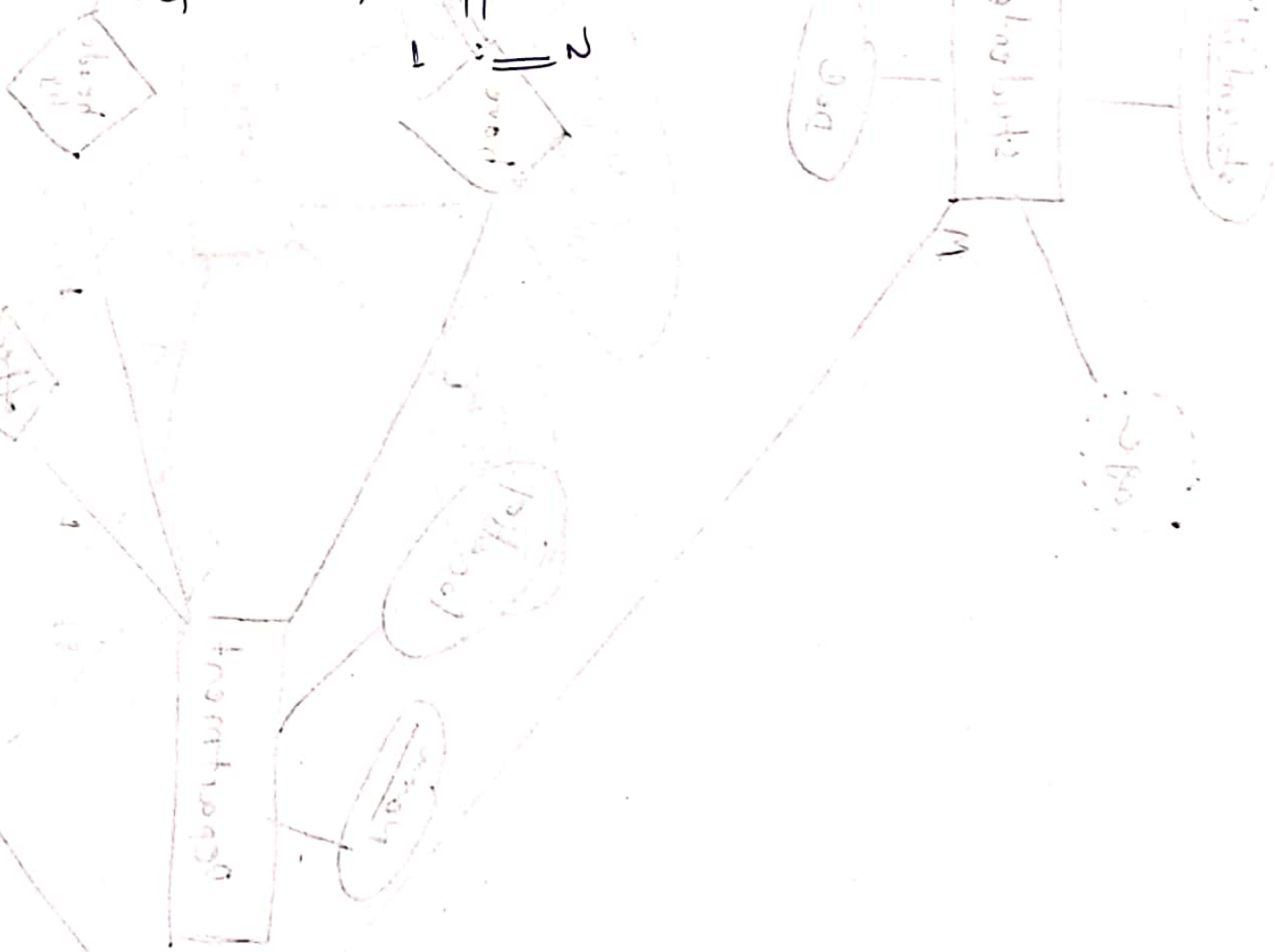
1 : 1

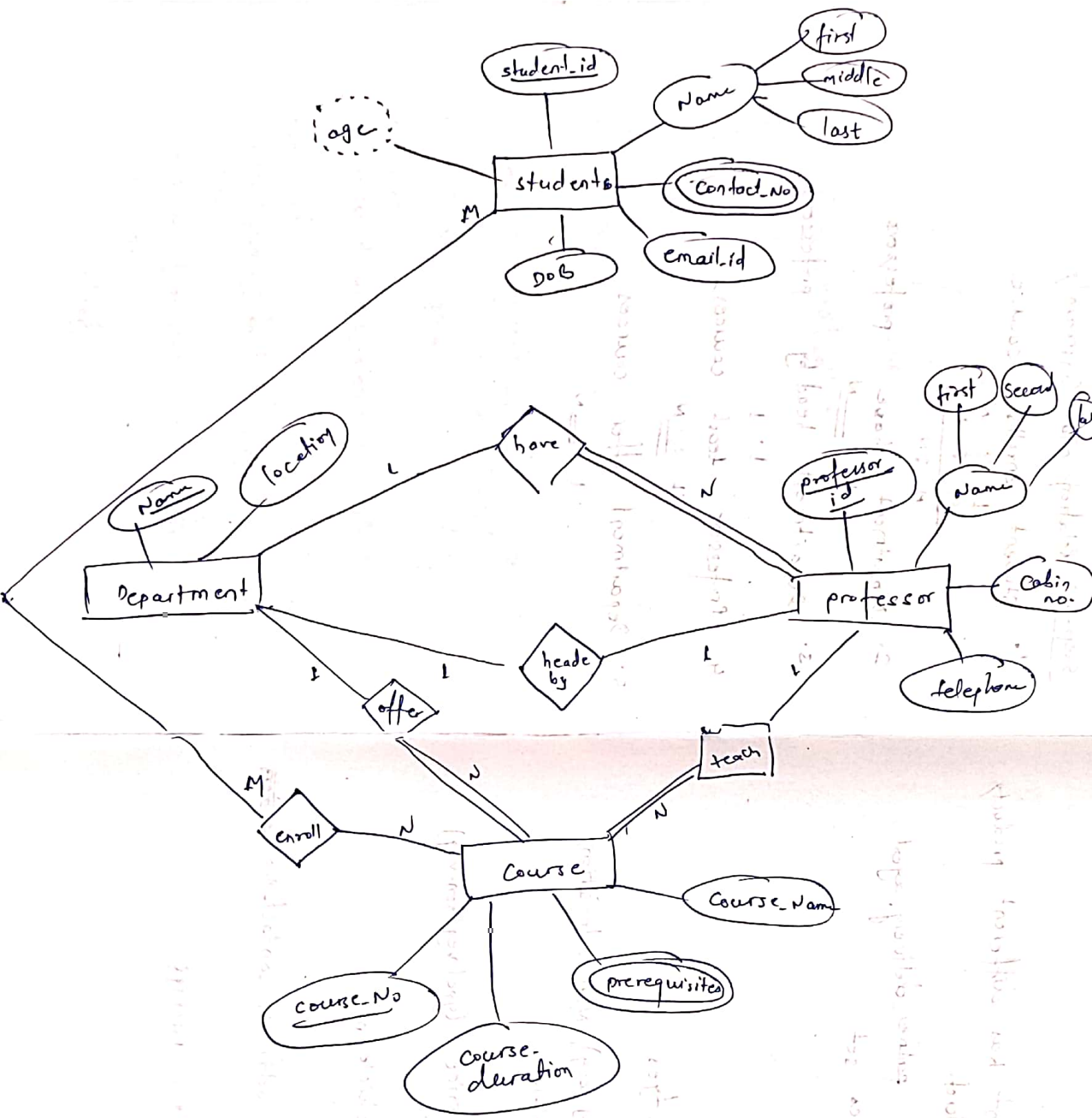
4. Professors teach cources.

1 : N

5. Department offer cources

1 : N





ERD of Online Deliveg System ~~(Ekart-logistics)~~
food.

features:

1. Reteurants → Resturant has different products
2. Customer order ~~product~~.
3. Product delivered by ~~partner~~ delivery. agent

Step-1: Identify the entities set

1. Resturant
2. product
3. delivery_agent
4. Customer
5. feedback

Step-2: Attributes and it's tyes.

1. Resturant

   Resturant_id (primary key), Name, location, Contact_No, email_id.

2. Product

   product_id, name, price, type(veg/non-veg).

3. delivery agent

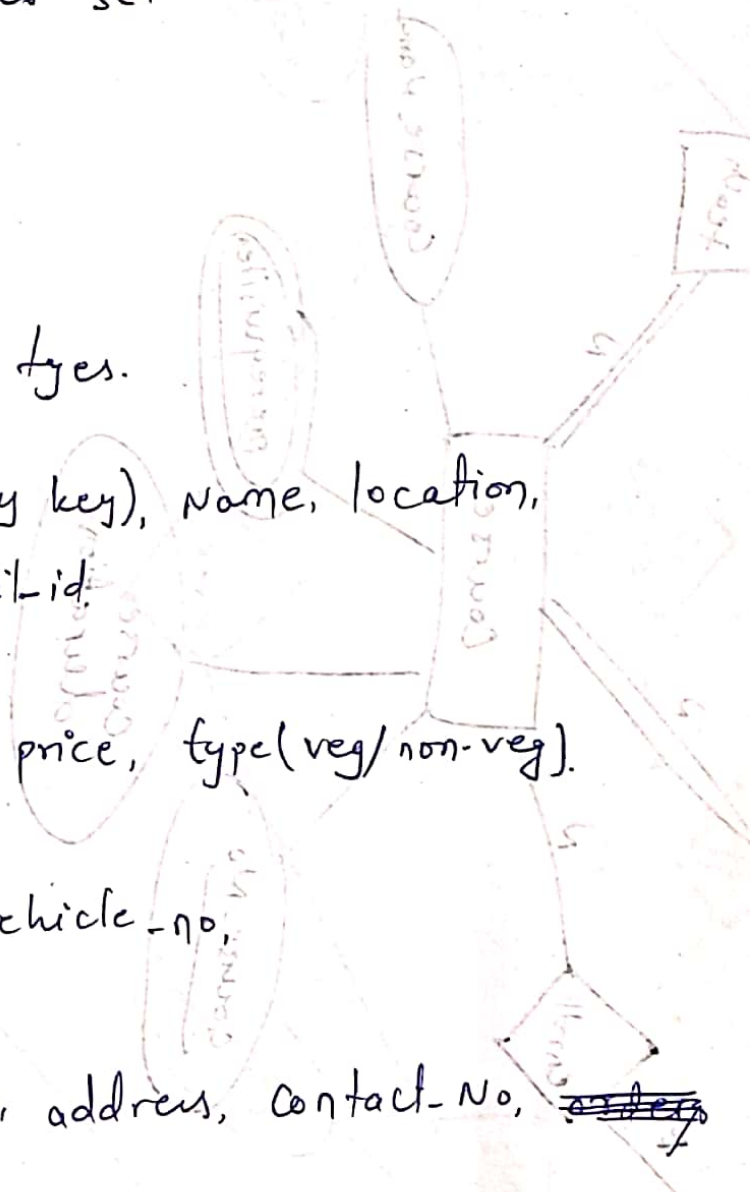
   agent_id, name, vehicle_no,

4. Customer

   customer_id, name, address, Contact_No, ~~orders~~

5. feedback

   feedback_id, text, ratings.



Step-3 : Relation and Constraints

1. Restaurants make products

1 : N
M : 1

M : N

2. ~~Restaurants~~ gets feedback

1 : 1 N

3. Customer gives feedback

1 : 1 N

4. delivery agent delivers product

1 : N
M : 1

M : N

5. Customer order product

1 : N
M : 1

M : N

DBMS: Database Management System

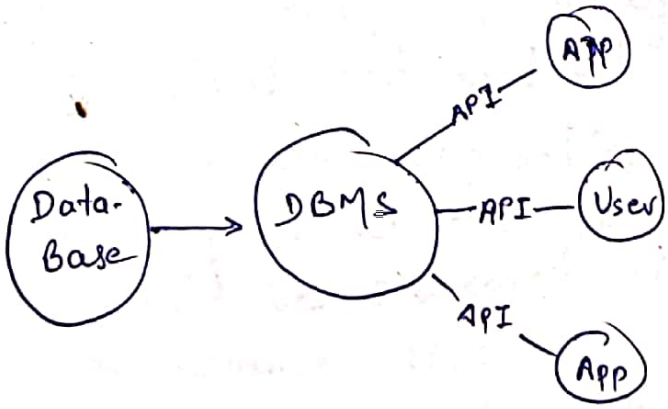
Lecture-1

- Meaningful data is information.
- Everything (Image/ text/ integer) is stored in terms of (bits and bytes) in computer memory.

① Data $\xrightarrow[\text{processing}]{}$ ② Information

- Data is the new oil.
- What is Database?
  Database is an electronic place/system where data is stored in a way that it can easily accessed managed and updated.

DBMS → ① DB (storing of data)
② set of programs, help us to
- access
- add
- update
- delete



Lecture:7 : Relational model

- This model is very similar to ER model, where we tabulate the things only.
- In relational model table is relation itself.
- Attributes are columns of table.
- Each row is touple and represents one customer
- Degree of tables = No. of tables's attributes.

DB design ① ER_model

↕

② relational_model

↓

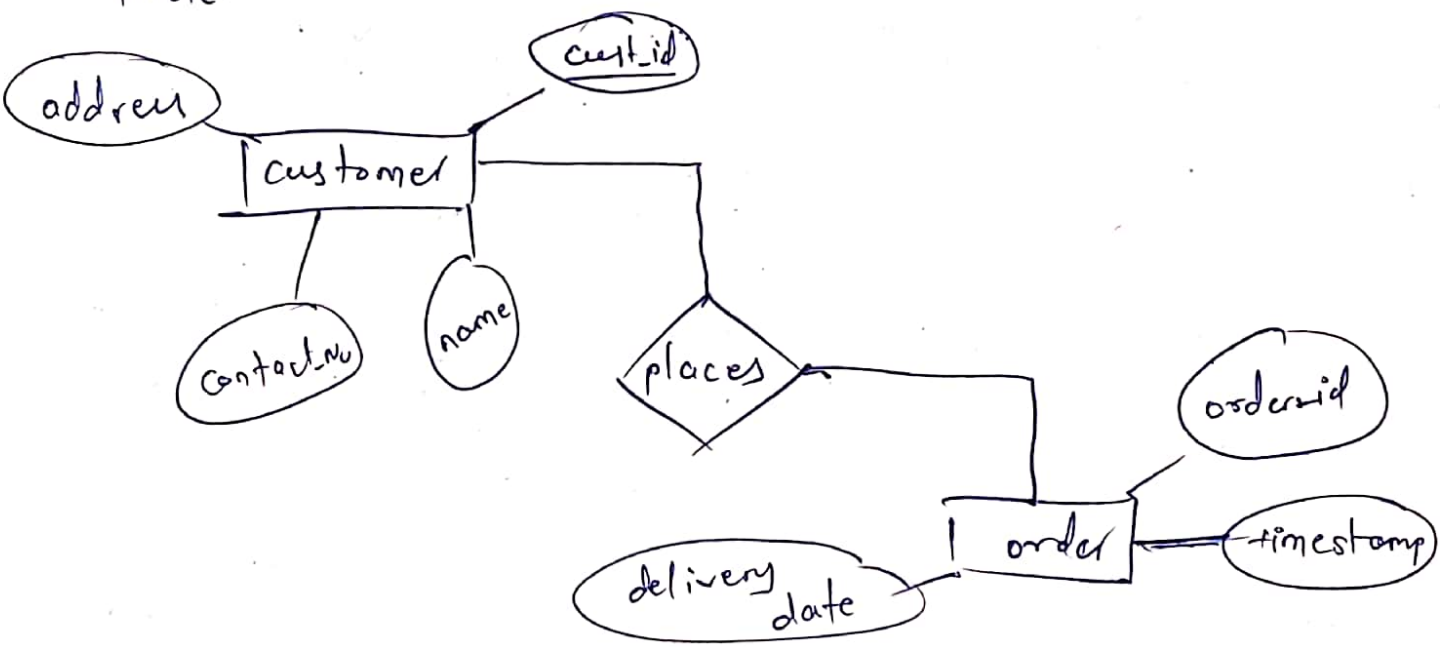
RDBMS → software DBMS

↓

Software implementation of R

eg: MySQL, Oracle, microsoft acce

- To keep track of relation like (places) from customer places order the RM provides functionality of "foreign key".
- A single row represents a data point in the table.



① Custome (Cust_id, name, address, contact_no)
— relation1 / table1

② ORDER (order_ID, time_stamp, delivery_date)
— relation2 / table2

relation-1 / table-1

	Cust_id	name	address	Contact_No
①	id_1	abc	add1	xxxx
②	id_2	xyz	add2	xxxx

Degree of table1 : 4
Cardinality : 2
↳ Total no. of tuples in given relation.

Relational key : set of attribute which can uniquely identify a tuple.

Properties of a table/relation
→ Name of relation should be distinct among all.
⇒ The values has to be atomic.
atomic - that can not be broken down.
⇒ The name of each attribute/ Column must be unique.
⇒ Each tuple must be unique in table.
(No redundant data should be available)
⇒ The sequence of row & col has no significance.
⇒ Tables must follow integrity constraints.

Relational Model keys:

1. Super key: Any plc of attribute that can identify each tuple uniquely.

Attributes: Name, add, phone_no, email, cust_id

{ {cust_id}, {Name, cust_id}, {name, email},
{name, phone_no}, {Name, add, phone_no, cust_id}}

2. Candidate key: ck can't be NULL
: ck has no redundant attribute

eg: name is a redundant attribut
address is a redundant attribu

→ we will discard all elements of super key containing them.

{ {cust_id}, {e_mail}, {cust_id, email}, ... }

3. primary key: ck having minimum no. of attributes is pk.

{cust_id}

4. Alternate key = Candidate key − primary key

all ~~subsets~~ elements of ck except pk.

5. foreign key:

(Customer (cust_id, name, address, Contact_No)
places
order (order_id, timestamp, delivery date, cust_id)
↑
foreign key.

- Foreign key is primary key of some other relation / table

→ It create relationship between two tables.
→ Customer - parent - Refranced relation
order - child - Refrencing relation

6. Composite key

Primary key having at least 2 attributes.

7. Compound key

PK which is formed using 2 FK.

8. Surrogate key

- Synthetic PK
- Generated automatically by DB
- May be used as PK.

School-1

101	ABC
102	BCD
103	DEF

School-2

101	JKL
102	KGF

⟹

Surr-k	Reg_No	Name
1	101	ABC
2	102	BCD
3	103	DEF
4	101	JKL
5	102	KGF

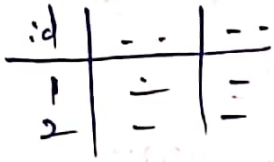
13 Integrity Constraints ⇒ It purpose is that we accidently does not corrupt the DB.

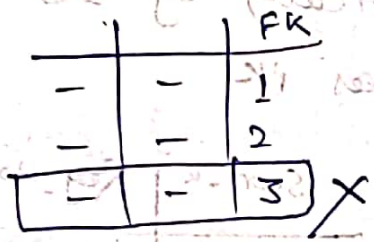
CRUD → Delete
→ Update
→ Read

Create

- Domain Constraints
  - Specifies the domain : Restricts value of attn
  - Restrict data types
- Entity Constraints
  - pk must present (PK != NULL)
- Refrential Constraints (Important)

① Insert Constraint
value can't be inserted in child table, if the value is not present in parent table.



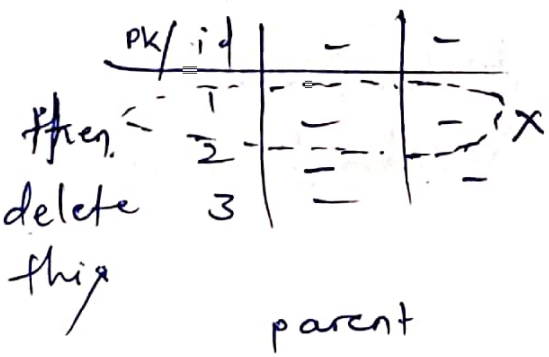


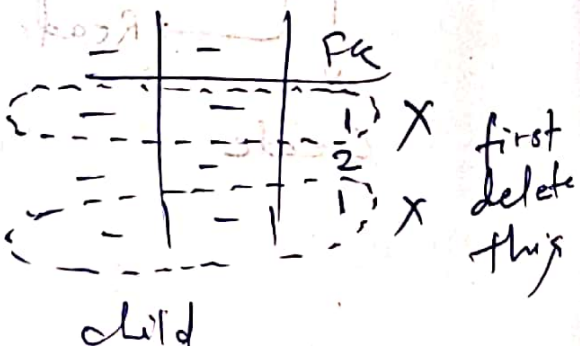
3 is not present in parent table

② Delete Constraint : Value can't be deleted from parent table, if the value is lying in child table

↳ on Delete Cascade
↳ on delete NULL

on delete cascade : delete corresponding entry in child table.





on cascade null : Same as on cascade delete only we need to mark them NULL instead of deletion.

PK	–	–
1	–	–
2	–	–
3	–	–

parent

–	–	FK
–	–	NULL
–	–	2
–	–	NULL
–	–	

child

8.) CAN FK have NULL value? yes: on cascad NULL

14 Key constraints : While defining any attribute in a table, we use key constraints.

① Not NULL : By default attribute/Column can be NULL.

```
create Table Customer (
    ID int not null,
    Name varchar (50) not NULL
    age int,
);
```

② Unique : Ensures all the values in the column are different.

- Both unique & PK constraint provide uniqueness.
- You may have many unique constraints per table, but only one P.K. constraint per table

③ Default constraint
- set default value of column.

④ check constraint
- limit value range (Domain)

```
(
    check (age>=10)
)
```

⑤ PK constraint : PK ≠ NULL
per relation/table only one PK constraint

⑥ FK constraint
- keep 2 tables related

```
create table order (
    PRIMARY KEY(cust_ID) Refrancing
        Customer (cust_id)
);
```

Lecture: 8

Transformation from ER to Relational model

1. ER diagram notations to relations

(a) Strong Entity

- Becomes an individual table with entity name, attributes becomes column of relation.
- Entity's PK is used as relation's PK.
- FK are added to establish relationship with other relations.

(b) Weak entity

- a table is formed with all the attributes of entity.
- PK of it's corresponding strong entity is will be added as FK
- PK of relation will be composite (FK + partial discriminator key).

(c) Single valued attributes

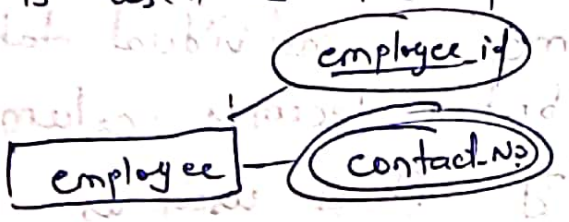
represented as columns directly in relations/tables.

(d) Composite attributes

name → firstname
lastname
} two different columns will be allocated as two different attributes.

address → street → street_name
street_no
→ village/city
→ pincode
} Four different Columns will be allocated.

(e) Multivalued attributes
↳ A whole new separate table is created as
  ↳ name of the attribute
  ↳ PK of entity is used as FK of the table



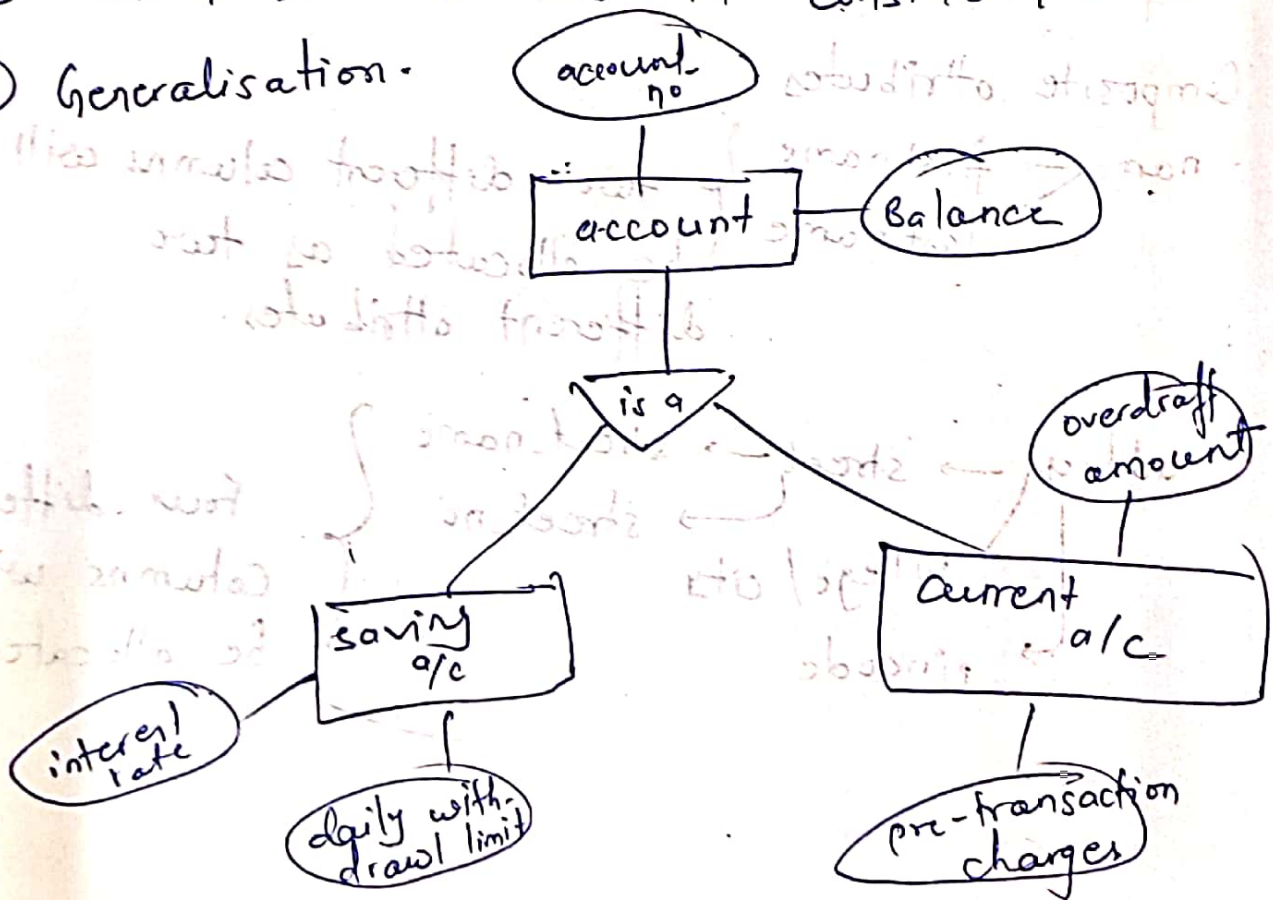
Contact No.

contact_No	Fk (employee_id)
xxx	1
xxx	2
xxx	1
xxx	2
xxx	3
xxx	3
xxx	4
xxx	

PK = {contact No, employee_id}

(f) Derived attributes are not considered in table

(g) Generalisation.



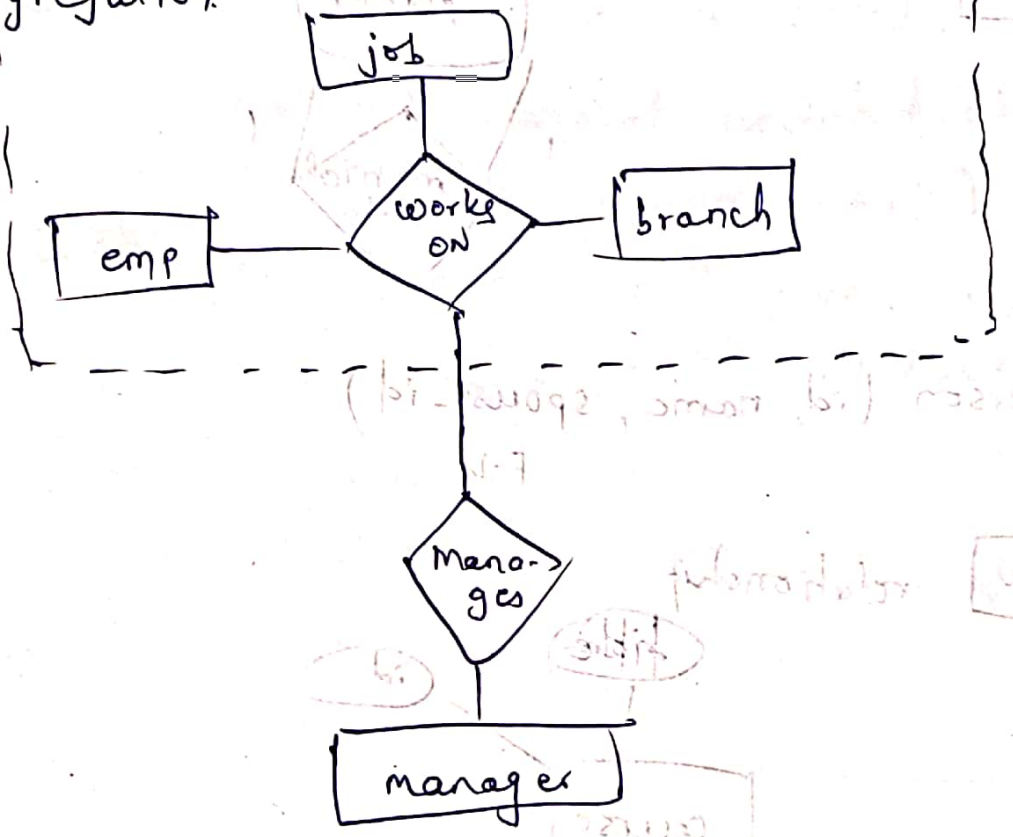
Method-1 : Create three separate tables.

1. Table1 : account : (a/c no, balance)
2. Table2 : Saving a/c : (a/c no, interest-rate, daily-w_lim
3. Table3 : current a/c : (a/c-no, overdraft, per txn limit)

Method-2 : Create two table in separate

1. Saving a/c (ac_no, balance, interest_rate, daily_w_lim
2. current a/c (ac_no, balance, overdraft, per_txn_limit)

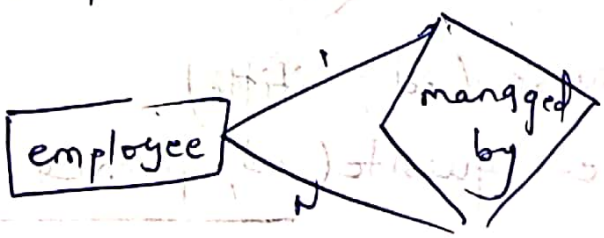
(b) Aggregation.



Table_name : manages (mgr_id, emp_id, job_id, brand_id)

composite Primary key.

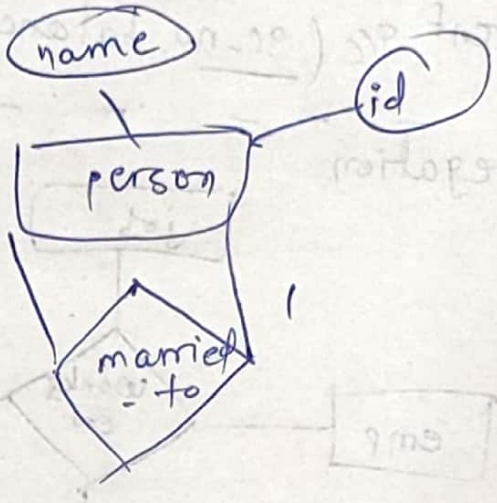
(i) Unary - relationship



we will add another attribute in employee table which will be FK.

emp_id	name	joining date	Emp_mgr_id
201	- -	- -	205
202	- -	- -	205
205	- -	- -	NULL

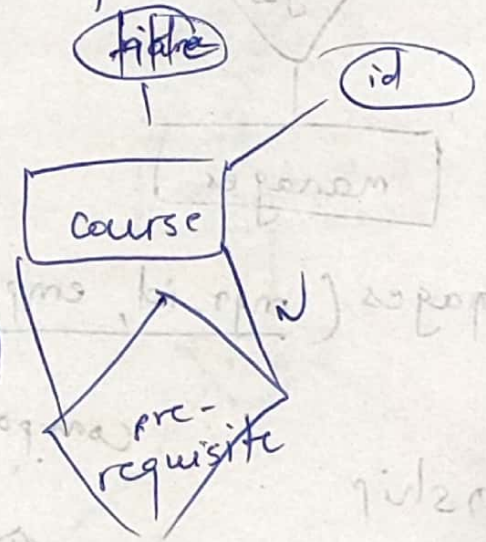
1:1 relationship



person (id, name, spouse_id)

F.K

M:N relationship



① course (id, title)

② pre-requisite (id, prereq_course_id)

Composite primary key.

FB relational Model

(1) user_profile (username, first_name, middle_name, last_name, DOB, password)

(2) user_profile_email (username [FK], email)

(3) user_profile_contact_No (username [FK], contact_No).

(4) friendship (profile_req [FK], profile_accepted [FK]) compound key

(5) post_like (post_like_id, timestamp, post_id [FK], username [FK])

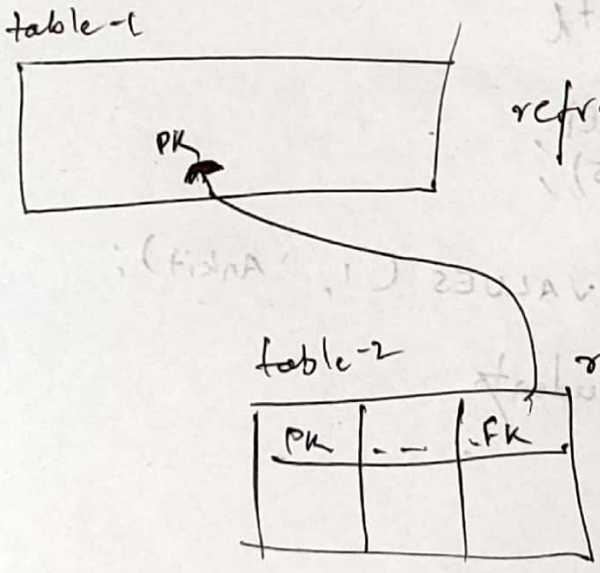
(6) user_post (post_id, created_at, updated_at, text_content, username [FK])

(7) user_post_image (post_id [FK], image_url)

(8) user_post_video (post_id [FK], video_url) compound key

(9) post_comment (post_comment, text_content, timestamp, postid, username) compound key
[F.K] [F.K]

Imp



Lecture-9 : SQL in 1 video

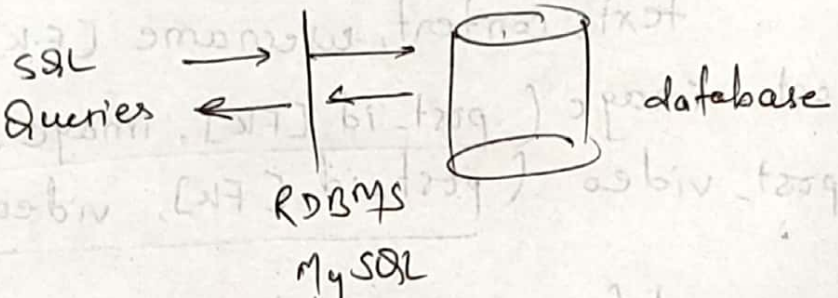
SQL: structured query language.

RDBMS ⇒ MySQL, oracle, MS access, IBM

We are now using MySQL as RDBMS.

↳ It is open source

SQL	MySQL
It is a language	It is a software or RDBMS
It is a way to access data	CRUD operation done on it using SQL language



SQL data-types : Refer notes

```
CREATE DATABASE temp;
USE temp;
CREATE TABLE student(
    id INT PRIMARY KEY,
    name VARCHAR(255),
    );
INSERT INTO student VALUES (1, 'Ankit');
SELECT * FROM stududento.
```

VARCHAR is recommended over CHAR.

↳ variable size saves memory.

DATE : YYYY-MM-DD
DATETIME : YYYY-MM-DD- HH:MM:SS
TIMESTAMP: YYYYMMDDHHMMSS
TIME : HH:MM:SS

size TINY < SMALL < MEDIUM < INT < BIGINT

(w3school)_ refrence for SQL.

SQL also supports signed and unsigned.
(default)

```
create table tname(
    col1, INT,
    col2 INT UNSIGHNED,
);
```

Advanced - Data-types - JSON better than text
↳ key: value pair
JS object Notation

Types of command in SQL: DDL

① Database Definition language (DDL)
we defining relation schema.

1. create: create table, DB, view
2. Alter Table: Modification in table structure
   eg: change datatype, add/remove column.
3. DROP: delete table, DB, view
4. TRUNCATE: remove all tuples from table
5. RENAME: rename DB, table, column name

② Data retreival language (DRL),
Data query language (DQL)
: retreive data from the tables

1. SELECT

③ DML: Data modification language
: used to modify already present data in the DB

1. INSERT: insert data into a relation.
2. UPDATE: update relation data.
3. DELETE: delete row(s) from the relation

④ DCL: Data control language:
grant/revoke authorities from user

1. GRANT
2. REVOKE

⑤ TCL: Transcation control language

1. START transcation: begin
2. COMMIT: apply changes
3. ROLLBAK: discard changes and end transcation
4. SAVEPOINT: ....

MANAGING DB (DDL)

1. CREATE DATABASE IF NOT EXISTS db_name
2. USE db_name
3. DROP DATABASE IF EXISTS db_name
4. SHOW DATABASES
5. SHOW TABLES

MANAGING DB (DQL)

1. SELECT

SELECT col1, col2, col3 FROM table_name;

execution : right to left

use select without using from? Yes

using DUAL Tables

SELECT 55+11	66
" now();	DATETIME (current)
" ucase();	converts to ucase

2. WHERE

reduce rows based on given conditions

SELECT * FROM customer WHERE age > 18;

" " " " " PINCODE = 555;

3. BETWEEN / AND

SELET * FROM custome WHERE age BETWEEN 18 AND 60;

Note: 18 & 60 inclusive

4. IN / OR

reduces OR condition

M-1 → SELECT * FROM worker WHERE DEPARTMENT = 'HR' OR DEPARTMENT = 'Admin';

M-2 → SELECT * FROM worker WHERE DEPARTMENT IN ('HR', 'Admin', 'Accounts');

5. NOT

SELECT * FROM Worker WHERE DEPARTMET NOT IN ('HR', 'Admin');

- they will show workers in accounts department

6. IS NULL

SELECT * FROM customer WHERE Pincode is NULL.

7. Pattern seaching

'%' - any no. of characters from 0 to n.

'_' - only one character

SELECT * FROM Worker WHERE First_name LIKE '%i%';

- aimim
- iqoo
- kamini
- sonia

SELECT * FROM Worker where First_name LIKE '_i%';

- sikha
- sita
- simita
- xi
- vipul

8. SORTING

Select * from Worker ORDER BY salary;

salary DESC ; } optional
salary ASC ;

ASC (Default)
DESC (Custom)

9. DISTINCT

select department FROM worker;

sclect DISTINCT departmen FROM worker;

10. DATA Grouping

find no. of employees working in different departments.

HR → ?
ACCOUNT → ?
Admin → ?

GROUPING → Aggregation functions
(COUNT, SUM, AVG, MIN, MAX)

→ select department from worker group by departme

o/p - HR
- Admin
- Accounts

→ selcct department, COUNT(*) from worker group by department

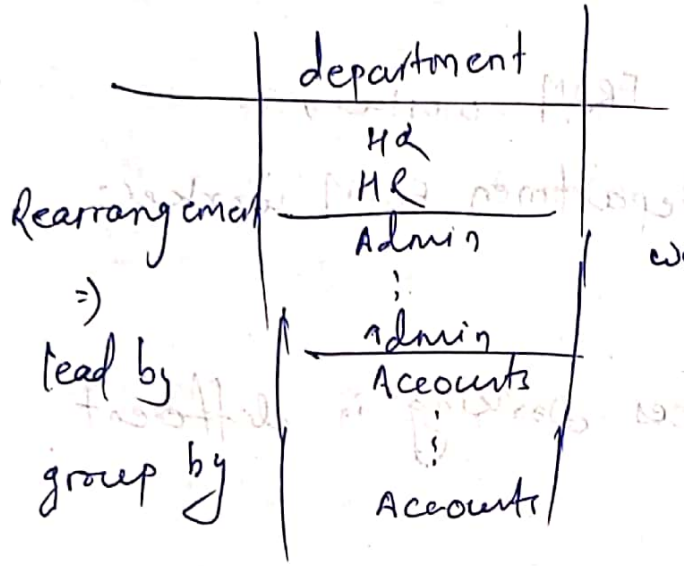
→ select department, COUNT(department) from worker group by department

HR-4
Admin-6

Q.) Find avg. salary of each department

SELECT department, AVG(salary) from worker group by deaprtment;

SELECT department, MIN(salary) from worker group by department.



II.) HAVING

To use HAVING we need group by.

Q.) Department count having more than 2 coorkers?

SELECT department, COUNT(department) from worker group by department having count(department)>2;

o/p. Admin-4

WHERE vs HAVING

we cannot use count with WHERE as to such grouping was there

- Both have same function of filtering the row based on certain conditions.
- WHERE is used to filter rows from the table based on specific condn.
- HAVING is used to filter the rows from groups based on specific conditions.
- HAVING used after GROUP BY
  WHERE used before GROUP BY.
- FOR using HAVING, GROUP BY is necessary.
- WHERE can be used with- SELECT
  - UPDATE
  - DELETE

  GROUP BY used with SELECT only.

Constraints of DDL

1. Primary key constraint
   - Not NULL
   - unique
   - Only one per table
   - INT (good practice)

2. Foreign key.
   FK refers PK of other table

```
CREATE TABLE order(
    id INT PRIMAR KEY,
    delivery_date DATE,
    order_placed_date DATE,
    FOREIGN_KEY (cust_id) REFERENCES
      customer(id)
);
```

3. Unique

```
CREATE TABLE #1(
        :
        First_Name VARCHAR(255) UNIQUE;
        :
    );
```

4. CHECK

```
CREATE TABLE Customer(
        :
        CONSTRAINT age_chk CHECK(age>12);
        :
    );
```

5. DEFAULT

```
create table account(
    :
    balance INT NOT NULL DEFAULT 0;
    :
)
```

6. ALTER OPERATIONS

Changes schema.

1. ADD
   → Add new column

   ALTER TABLE customer ADD age INT NOT NULL;

2. Modify
   → change datatype of an attribute

   ALTER TABLE customer MODIFY name CHAR(1024);

3. CHAGE COLUMN
   → Rename column name

   ALTER TABLE customer CHAGE COLUMN name ~~to~~ customer_name VARCHA(1024);

4. DROP COLUMN
   → DROP a column completely

   ALTER TABLE customer DROP COLUMN middle_name;

5. RENAME
   → Rename the table

ALTER TABLE customer RENAME to customer_details.

Note: DESC table_name

→ Description
Gives all information about table

DML (Data ~~manipulation~~ language)
modification

1. INSERT
2. UPDATE
3. DELETE
4. REPLACE

1. INSERT

→ single entry
INSERT INTO Customer (id, cname, Add, Gender, City, PIN)
VALUES (1251, 'Ramkumar', 'Add1', 'M', 'Jal', '144002');

→ multiple entry
same as above
INSERT INTO Customer (attributes)
VALUES ( ),
( ),
( );

direct syntax
INSERT INTO Customer values (1, 'codehelp', 'delhi',
'M', 'Delhi', '110006');

partial data
INSERT INTO Customer (id, cname)
VALUES (121, 'Bob');

2. UPDATE

single row → UPDATE customer SET address = 'Mumbai', Gender = 'M'
WHERE id = 121;

multiple rows → UPDATE Customer SET pincode = 110000;

Note

set .

SET SQL_SAFE_UPDATES = 0

disable the safe updates

to enable set the value to 1;

UPDATE customer SET pincode = pincode + 1;

3. DELETE

single row → DELETE FROM customer WHERE id = 121;

entire table is deleted → DELETE FROM Customer;

this command will delete the entire customer table.

→ ON DELETE CASCADE

→ ON DELETE SET NULL

Refrential Constraints

↳ INSERT (child table में तो FK होनी चाहिए)

↳ DELETE (parent table में तो सारा child table का data delete करना)

Customer table
order_details table (
:
FOREIGN KEY(cus_id) refrences Customer(id) ON
DELETE CASCADE.
);

Customer table
order details table(;
FOREIGN KEY (cus_id) refrances Customer (id) ON
DELETE SET NULL;
);

4. REPLACE
↳ If present, ~~update~~ REPLACE
↳ If not present, INSERT

REPLACE INO Customer(id, city)] New-entry
VALUES (1251, 'Colony');

REPLAC INTO Customer (id, city)] Replace & update in table
VALUES (1251, 'Mumbai');

REPLACE INTO Customer SET id=1300, Name='Mao',
City='Utah';

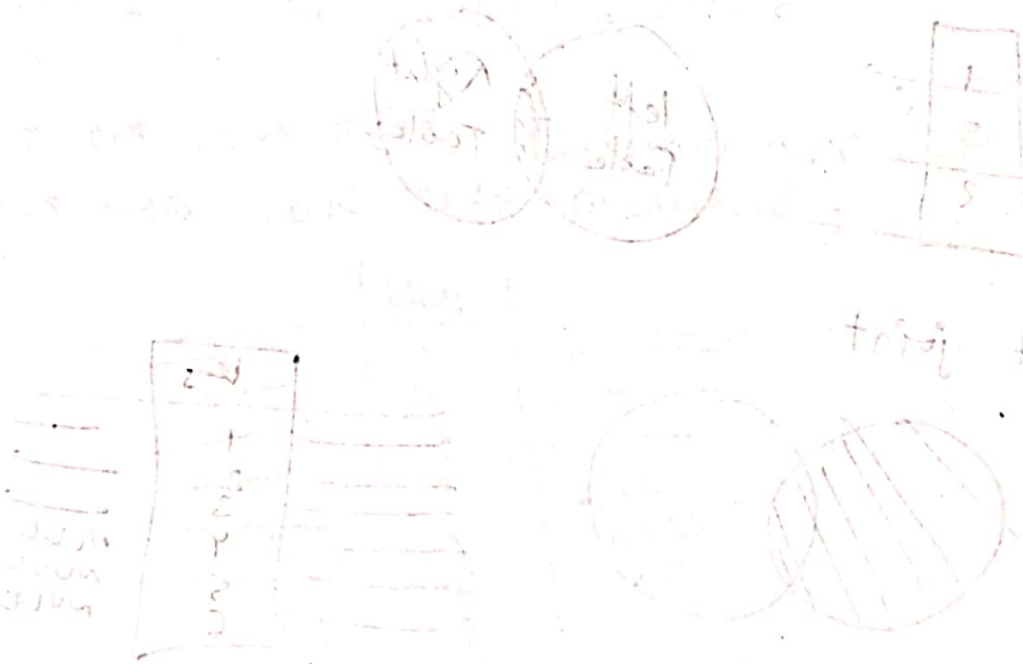
most used syntax

```
REPLACE INTO Customer (id, cname, city)
SELECT id, cname, city
FROM customer WHERE id=500;
```

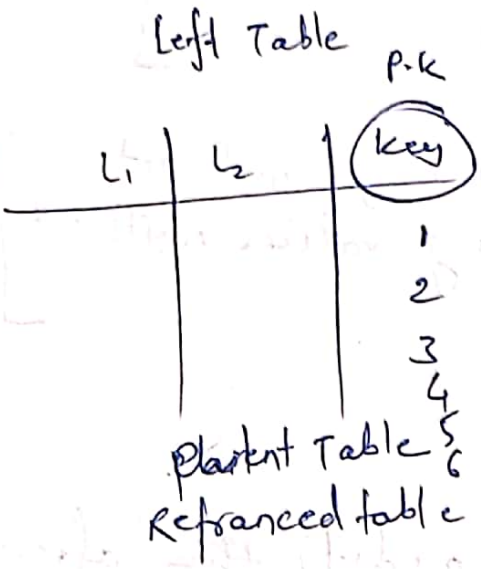
[at id=500 id, cname, city will not change. and remaining values will set to NULL.]

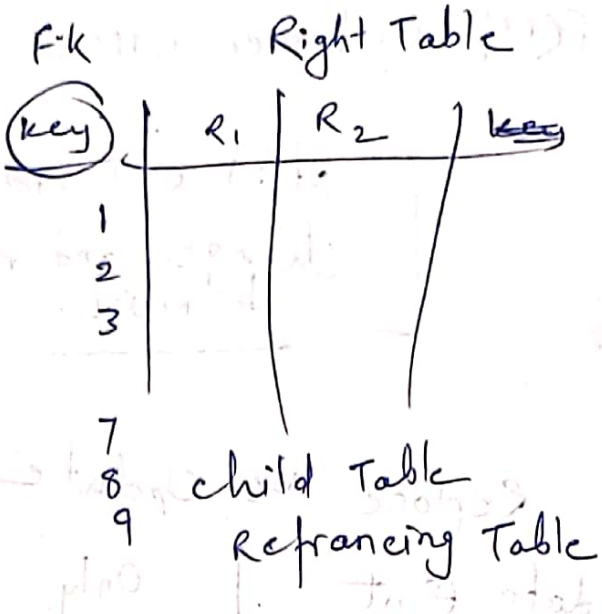
Replace Vs Update

Replace	Update
if data ⊕nt replace, otherwise make a new entry.	Only it can modify the already present data, it will not make a new entry in table.



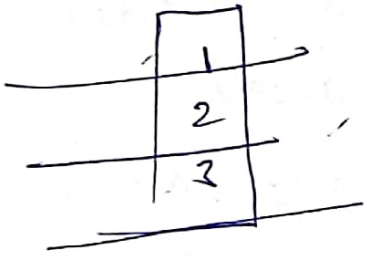
JOINS

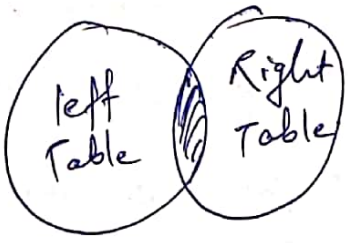




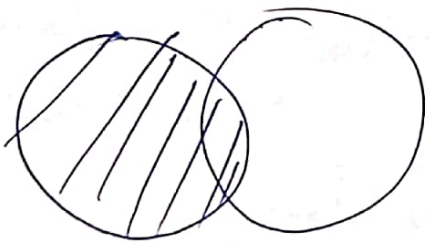
- INNER JOIN
  - Returns a resultant table that have matching values from both the tables or all the tables.
  - To apply join we, they should have a common attribute

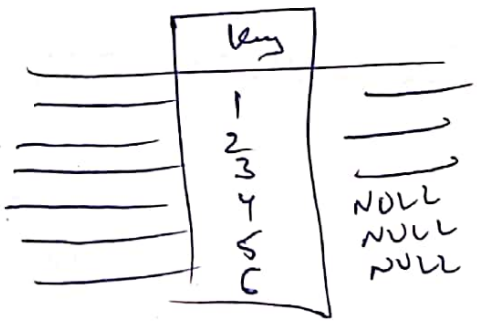
```
Select c.*, o.* FROM customer AS c INNER JOIN
    orders as o ON c.id = o.cust_id ;
```





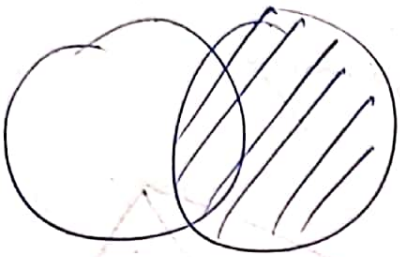
- Left joint

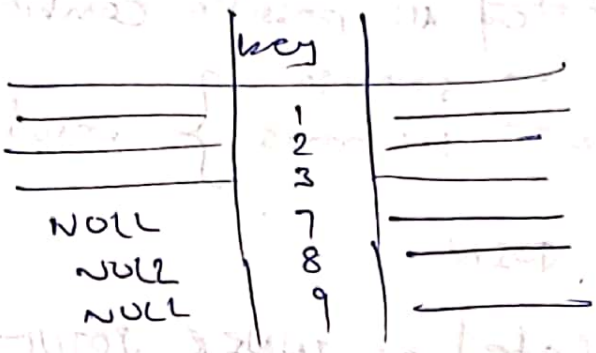




SELECT c.*, o.* FROM customer as c LEFT JOIN orders as o ON c.id = o.cust_id;

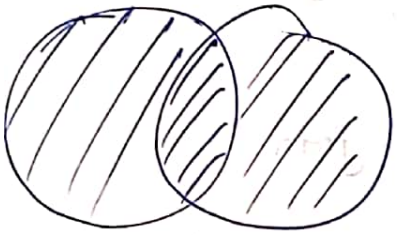
- Right join





SELECT c.*, o.* FROM Customer as c RIGHT JOIN orders as o ON c.id = o.cust_id;

- FULL JOIN

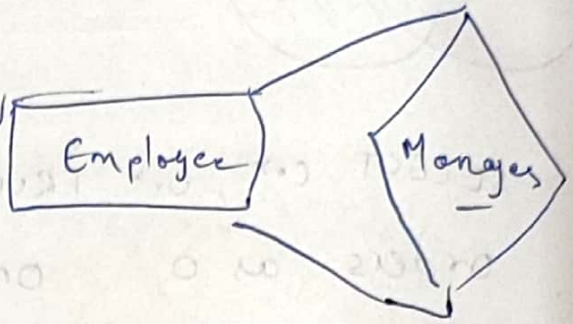


No such key word is here
- emulate
- LEFT JOIN UNION RIGHT JOIN

SELECT c.*, o.* FROM customer as c LEFT JOIN orders as o ON c.id = o.cust_id
UNION
SELECT c.* , o.* FROM Customer as c RIGHT JOIN orders as o ON c.id = o.cust_id;

- CROSS JOIN
  - This return all certain products / no industrial use
  - ~~emulated~~ All possible combinations
  - $t_1 \rightarrow$ 5 rows, $t_2 \rightarrow$ 10 rows } resultant = 50 rows

- SELF JOIN
  - emulated as INNER JOIN ON AS word




```
SELECT e1.id, e2.id, e2.name FROM
  employee as e1 INNER JOIN employee as e2
    ON e1.id = e2.id;
```

Q.) Can we use JOIN without using join keyword?

⇒ Yes

```
SELECT * FROM leftTable INNER JOIN righttable
ON left.table.id = Right.table.id;
```

```
SELECT * FROM Left table, right table WHERE
    lefttable.id = Right table.id;
```

INNER JOIN → ,
ON → WHERE

replacements

* SET operations (UNION / INTERSECTION / MINUS)

Set have unique elements

{1, 2, 3, 4, 5, 6} insert 2 again {1, 2, 3, 4, 5, 6}

1. UNION

Table1

Col1	Col2
A	1
B	1
C	2

Table 2

Col1	Col2
A	1
B	2
D	3

Join - Horizontal modification affects columns

SET/UNION - Vertically affect the (affects the rows) o/p

T1 ∪ T2 =

Col1	Col2
A	1
B	1
C	2
B	2
D	3

Syntax:

```
Select * From T1
UNION
Select * From T2;
```

Note: Set operation combines two or more "select" statements.

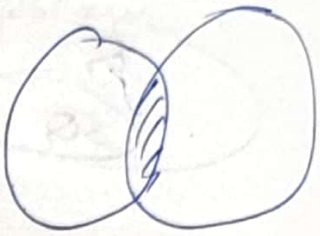
Full JOIN o/p

o/p to col1;

Col1	Col2	Col1	Col2
A	1	A	1
B	1	B	2
C	2	NULL	NULL
NULL	NULL		

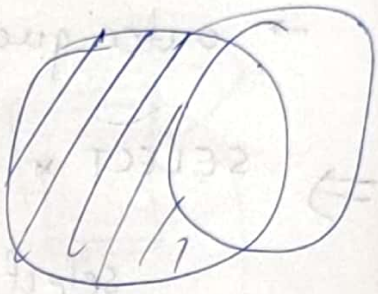
2. INTERSECT (No such keyword available)

Emulate: using INNER JOIN



```
select DISTINCT id FROM T1
    INNER JOIN T2 using(id);
```

3. MINUS



emulate:

```
Select dept1.* from dept1 LEFT JOIN
dept2 using (empid)
where dept2.empid is NULL.
```

JOIN	SET operations
1. Join multiple tables based on matching conditions	1. Combination of resulting set from two or more select operations
2. Column wise	2. Row wise.
3. Data types of two tables can be different	3. Datatypes of corresponding Column from each table must be same.
4. Can generate both distinct and duplicate rows.	4. Generate distinct rows only.
5. Horizontal	5. vertical
6. No. of columns selected may or may not be same from each table	6. No. of columns selected must be same for each table

SUB-QUERIES

parent query
child query
first child query will execute

→ Alternative to joins.
→ outer query depends on inner query.

⇒ SELECT * FROM TABLE WHERE id IN (
select id from table where name = "Ravi"
);

if will return table of id's having name RAVI
{1, 2, 3, 7}

⇒ Corelated Subquery:
Inner query that refers to outer query.

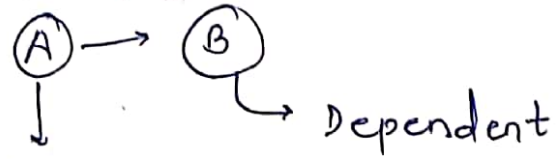
⇒ SQL - views (SQL)

CREATE VIEW custom_view AS SELECT fname, age FROM Employee.

ALTER VIEW Custom_view AS SELECT fname, lname, age From Employee

DROP VIEW IF EXISTS Custom_views;

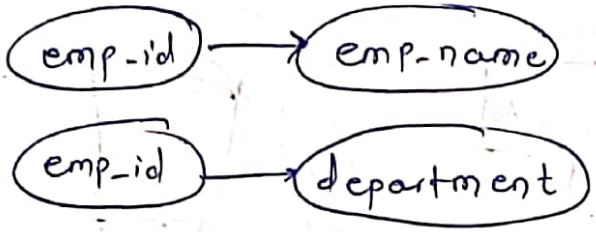
Lec-11 : Normalisation



Table

A	B

Functional dependency



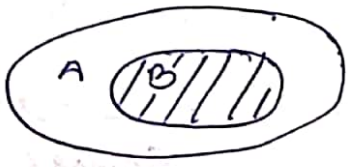
emp_id can be used to determine emp_name & department.

① Trivial FD

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

eg. {Emp_id, name} ⟶ {Emp-id}

A ⟶ A
B ⟶ B
emp_id ⟶ emp_id



② Non-trivial FD

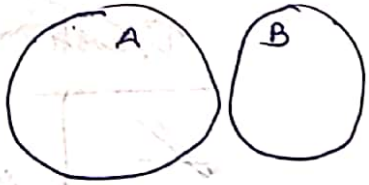
{Emp_id, name} ⟶ {Emp_address}

$A \rightarrow B$ where B is not the subset of A.

or, $B \not\subseteq A$

or, $A \cap B = NULL$

{emp_id} ⟶ {Emp_department}



$A \cap B = NULL$

Rules of FD (Armstrong's axioms)

① Reflexes

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

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

if Y is subset of X

then X can determine Y.

$X \rightarrow Y$

② Augmentation

if. $A \rightarrow B$

then adding an attribute to this FD will not change anything.

then, $AC \rightarrow BC$

③ Transitivity if $A \rightarrow B$

and $B \rightarrow C$

then $A \rightarrow C$

Normalization

Why ~~Redundancy~~ ?

To remove redundancy.

Relation: R (A B C D E)

FD: $A \rightarrow B$

$A \rightarrow C$

$CD \rightarrow E$

$B \rightarrow D$

$E \rightarrow A$

Answer: Q.1) $BC \rightarrow CD$ ✓ Augmentation on $B \rightarrow D$

Q.2) $CD \rightarrow AC$ ✓

$CD \rightarrow E$ & $E \rightarrow A$

Now, $CD \rightarrow A$

Reflexive & $CD \rightarrow C$

$CD \rightarrow AC$

Q) what happens if we have redundant data?

→ It introduces anomalies of three types

① Insertion anomaly
{ when certain data (attribute) can not be inserted into the DB without the presence of other data.

② deletion anomaly
{ It refers where deletion of data results in unintended loss of some of the imp. data

③ updation/modification anomaly
- The update of a single row requires multiple rows of data to be updates
- can potentially introduce data inconsistency.

SRP: Single Responsibility principle
one table should handle only one responsibility like Branch-info, student-data, course-info.

Q) what is Normalization:
We will decompose the table untill we receive SRP

Table ⟶ decompose into multiple tables.

Types of Normal forms:

① 1NF
1. Every cell must have atomic value.
2. Relation must not have multi-valued attributes.

② 2NF
1. Relation must be 1NF
2. Should not be any partial dependency.
   - All not-prime attributes must be fully dependent on PK
   - Non-prime attributes can not depend on the part of PK.

R(A B C D)

AB = primary ~~key~~ attributes
CD = non-primary ~~key~~ attributes

A → C (partial dependency)

here AB → C
AB should determine C.

AB → D
AB should determine D.

③ 3NF
1. Relation must be 2NF
2. No transitivity dependency exists.

R(A B C)

PK :→ A

FD : A → B
PD : B → C

R(ABC) ← 2NF

R_1(AB) R_2(BC) ← 3NF

non-prime attribute should not found a non-prime attribute.

④ BCNF (Boyce-codd normal NF)

1. Relation must be in 3NF
2. FD $A \rightarrow B$, A must be super key.
   We must not derive prime attributes from any prime or non-prime attribute

Advantages of Normalisation

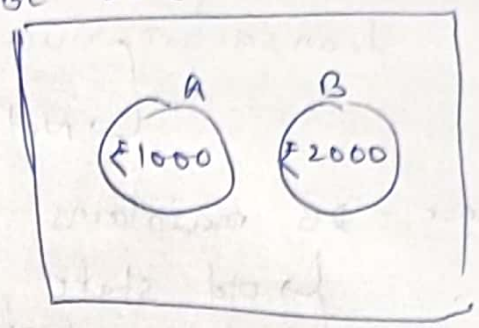
1. Normalisation helps to minimise data redundancy.
2. Greater overall database organisation.
3. Data consistency is maintained in DB.

Lecture-12 - Transactions

T_i :

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

ABC Bank



6 logical steps create one transaction.

→ Atomic: Considered should be a single task
: If done, all six steps
If undone, none of six is done.

Transaction: It is a logical unit of work that contain one or more SQL statement.
The result of all the statatement in transcation either gets completed successfully or if at any point any failure happens it gets roll back

ACID properties.

1. Atomicity
2. Consistensy
3. Isolation
4. Durability

Work-reading
Robust: Crisp & दृढ़ता
मजबूत

(a) Atomicity: Either all operations of transaction are reflected properly in DB, or none are

(b) Consistency:
- Integrity constraints must be maintained before and after transcation.
- DB must be consistent after transaction happens.

transaction → Success : new state
↳ Not-Success : old state recovery

Hence DB maintains
↳ old state
↳ Intermediate state

before	& after
3000	3000
money	money

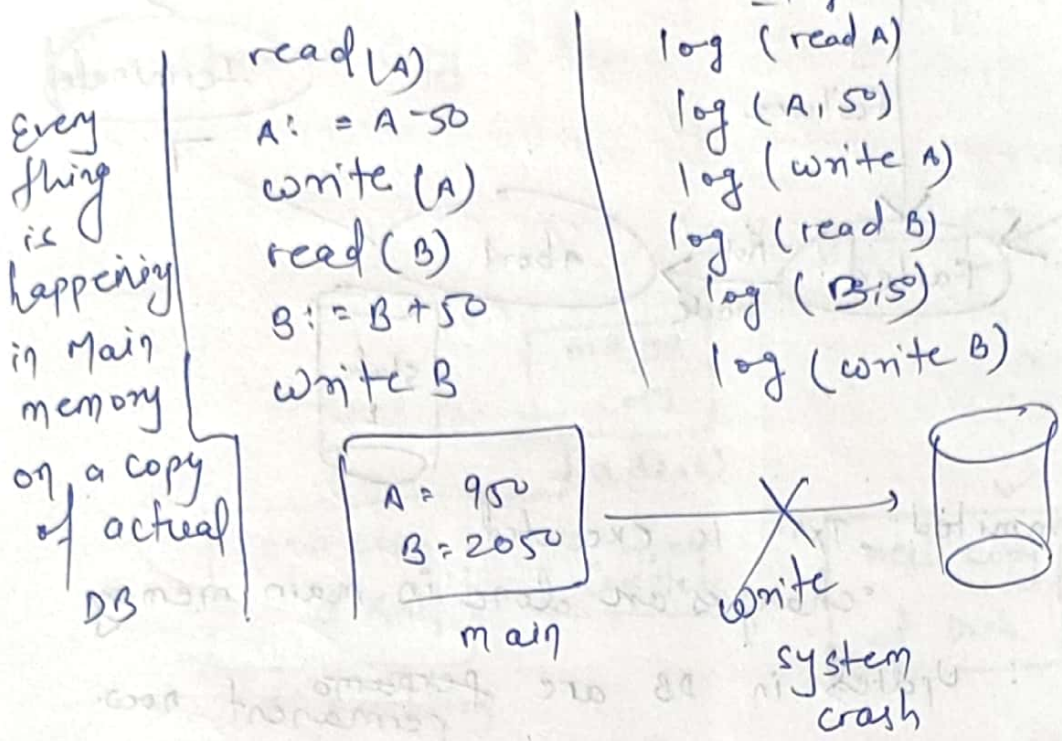
ⓒ Isolation
- It is achieved by sequential execution of the transactions.
पहले एक txn पूरा होगा फिर दूसरा शुरु होगा
एलॉजिक • transactions may occur concurrently but our DB should do them one by one to maintain consistency.

ⓓ Durability
After transaction completion, the changes made to database persist, even if there are system failures. (reflect, दिखना चाहिए)

for durability - recovery management component is present.
- After succesfull transaction, if system fails and DB is not updated then after restarting the system it will make changes in DB are persisting is achieved.

~~Transaction~~ - states

recovery - management - component

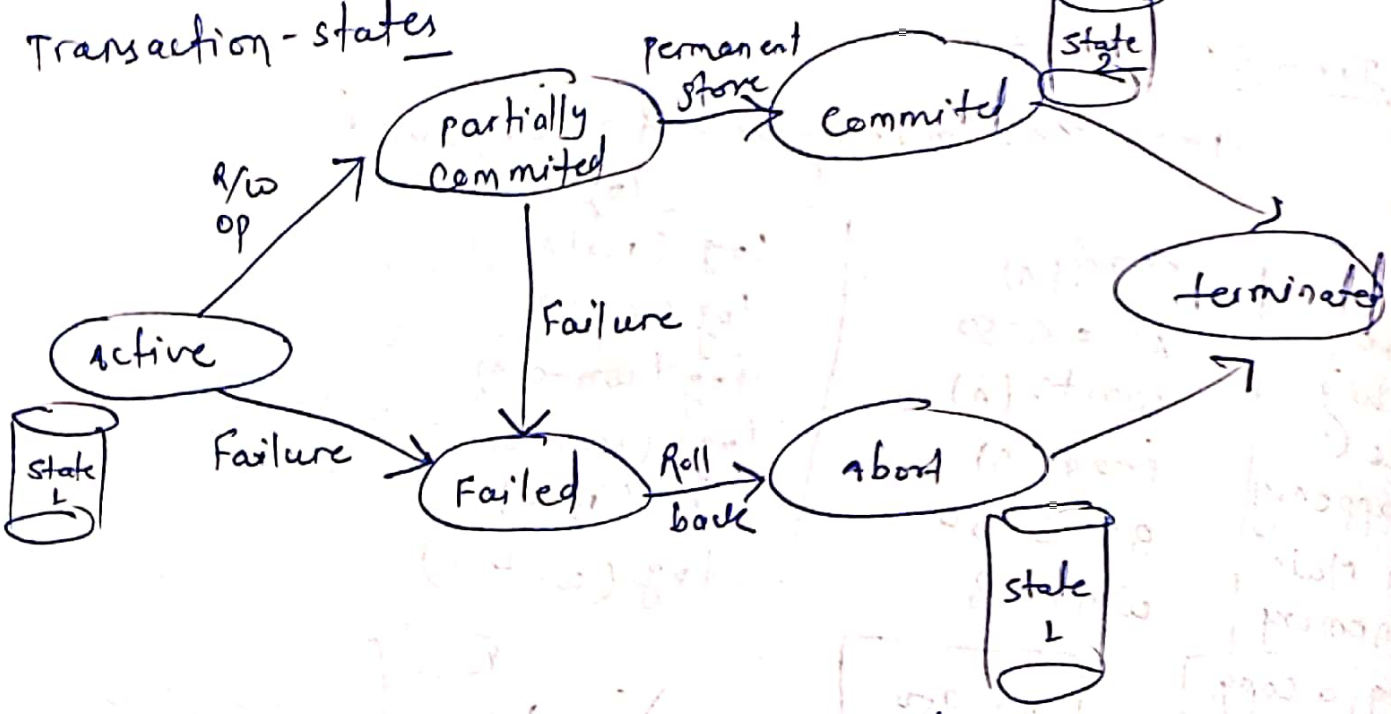


two methods for recovery

1st : We will continuously update the DB on each step

2nd : maintain logs and commit all change at a single time on DB.

Transaction - states



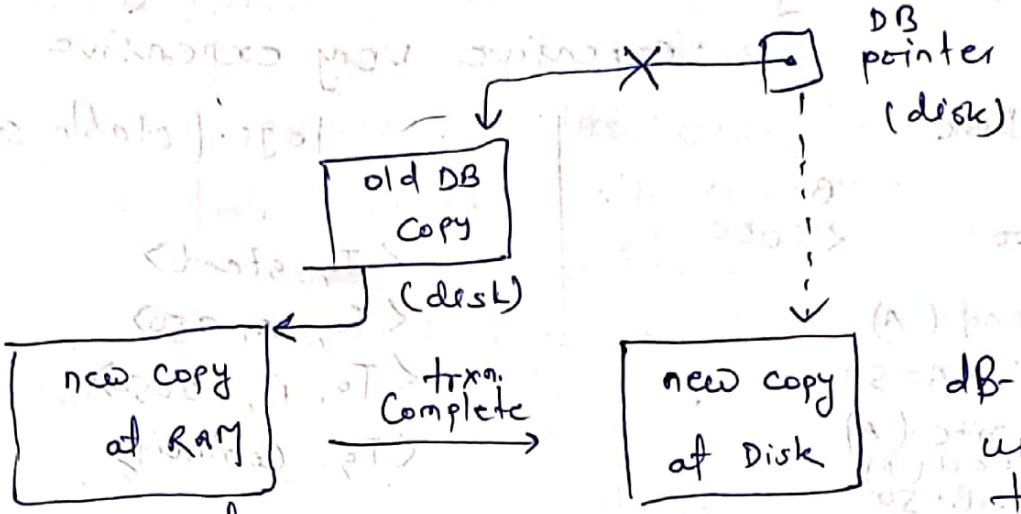
partially Commited: • Txn. is executed
• changes are done in main memory

Commited : Updtes in DB are ~~perform~~ permanent now.

Lecture: 13 Implementation of Atomicity and Durability

Durability: After completion of trxn the updates should persist in DB (database).

① shadow Copy scheme



→ trxn is said to be successful only if the dp-pointer point to new copy at disk.

→ It is inefficient as entire DB is copied for every transaction.

→ and it works on one transaction is assumption active at a time.

Imp ① DB pointer is atomic → change 0x02
→ not change 0x01

Log-Based Recovery

- we will maintain logs in some stable storage. So if system failure occurs, then it can be recovered from there.
- stable storage: • write o/p must be guaranteed
  • expensive very expensive

Disk

A	B
₹1000	₹2000

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

logs (stable storage)

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

• logs must be written before the step proceed

① Deferred DB modifications (late, delayed), ~~dTorfdI~~

one we write <T_0, commit>
⇒ transaction is successful and user gets message
⇒ But, what happens, if during execution of defter logs after <T_0, commit> failure occurs.
⇒ simply system will check for <T_0, commit> log. if found it will redo the changes.
⇒ and ~~atomicity~~ durability of data is ~~not~~ maintained (reflection)

② Immediate DB modification

- we will immediately updating the DB in active state.
- logs are maintained.
- Failure handling
  - Before completed, T aborted
    old value fields Undo the T
  - After completion <T₀, commit>
    new value fields Redo T having commit logs in the logs.

logs

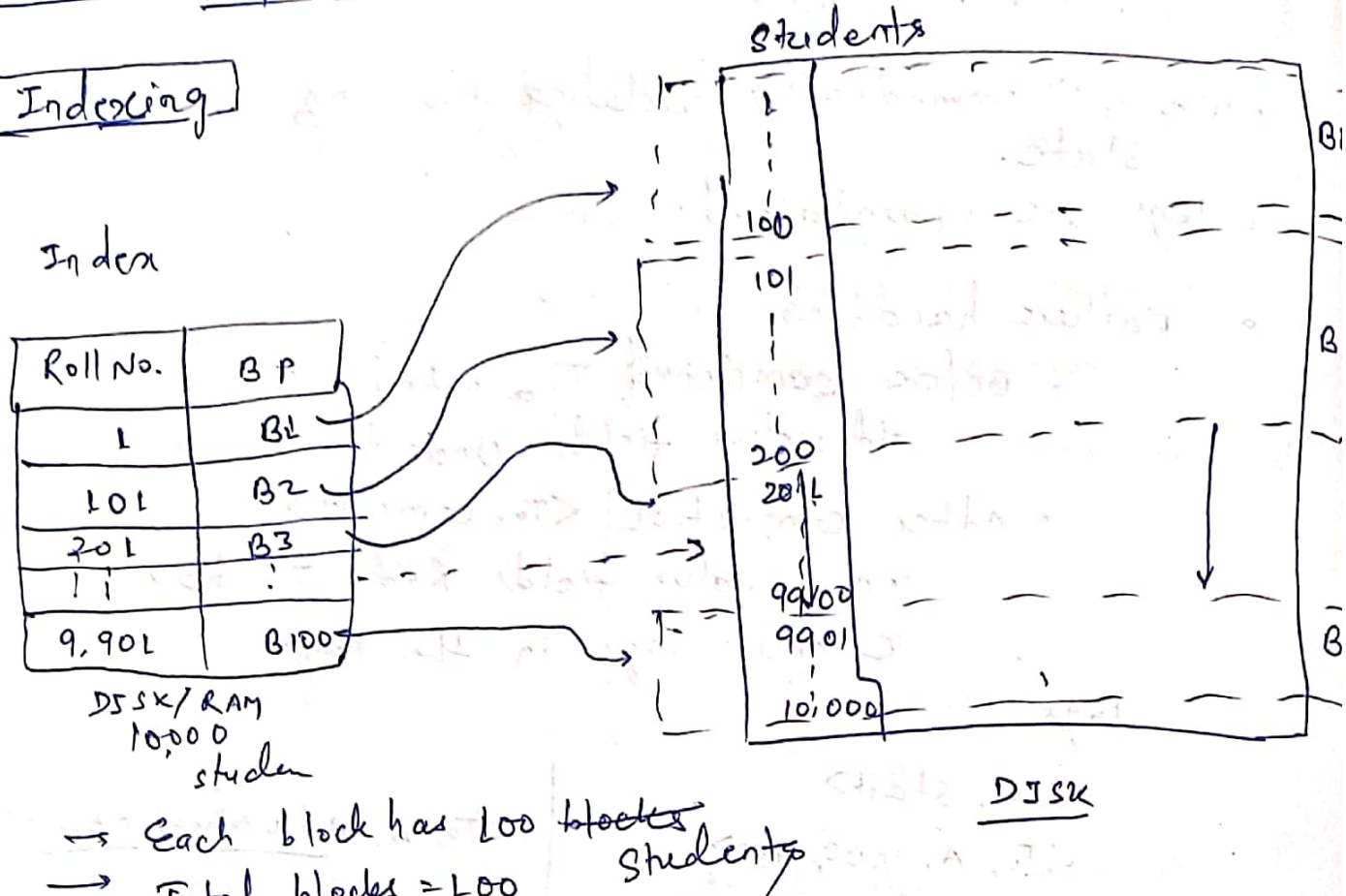
<T₀, start>
<T₀, A, 1000, 950>
<T₀, B, 2000, 2050>
<T₀, commit>

<T₀, A, old value, new value

③ checkpoints

Lec-14 : Performance Optimisation

Indexing



→ Each block has 100 ~~blocks~~ students
→ Total blocks = 100

Search key	Base pointer

BS :- Binary search

→ Indexing is used for very big data sets
→ B.S. also works well alone
→ but B.S. with indexing is a good approach

Find student: 843
→ Linear search ✗
→ Binary search ✗
→ 801 → B8 → 801 … 900 } Binary / linear search

Search key	Data Reference

primary key
or, Candidate key
or, Something else.

Types of indexing

1. Primary index (clustering index)
   - data file is sequentially ordered on any key/ other attributes that are capable of it.
   - Dense index and sparse index

Dense index
- It contains an index record for every search key value in the data file.

Sparse index
- An index record appears for only some of the search-key values.

primary indexing can be based on data file is sorted w.r.t primary key attribute or non-primary key attribute.

Sparse indexing:
- (a) Based on key-attribute (P.K/C.K)
  - ordering based on key attribute
  - Sparse indexing will work fine
    no. of entries = no. of blocks
- (b) Based on non-key attribute

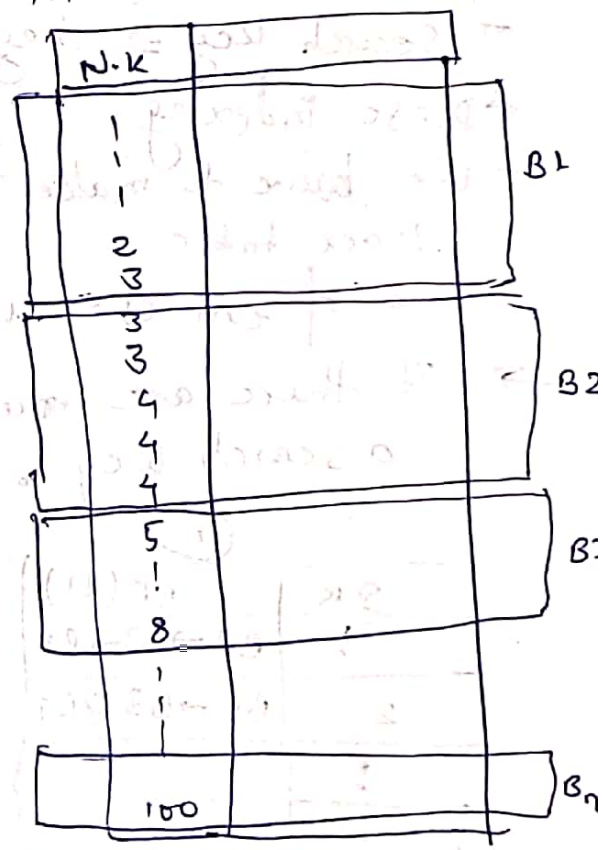
Dense Indexing for each unique value

Index

N.K	B.P
1	B1
2	B1
3	B1
4	B2
5	B3

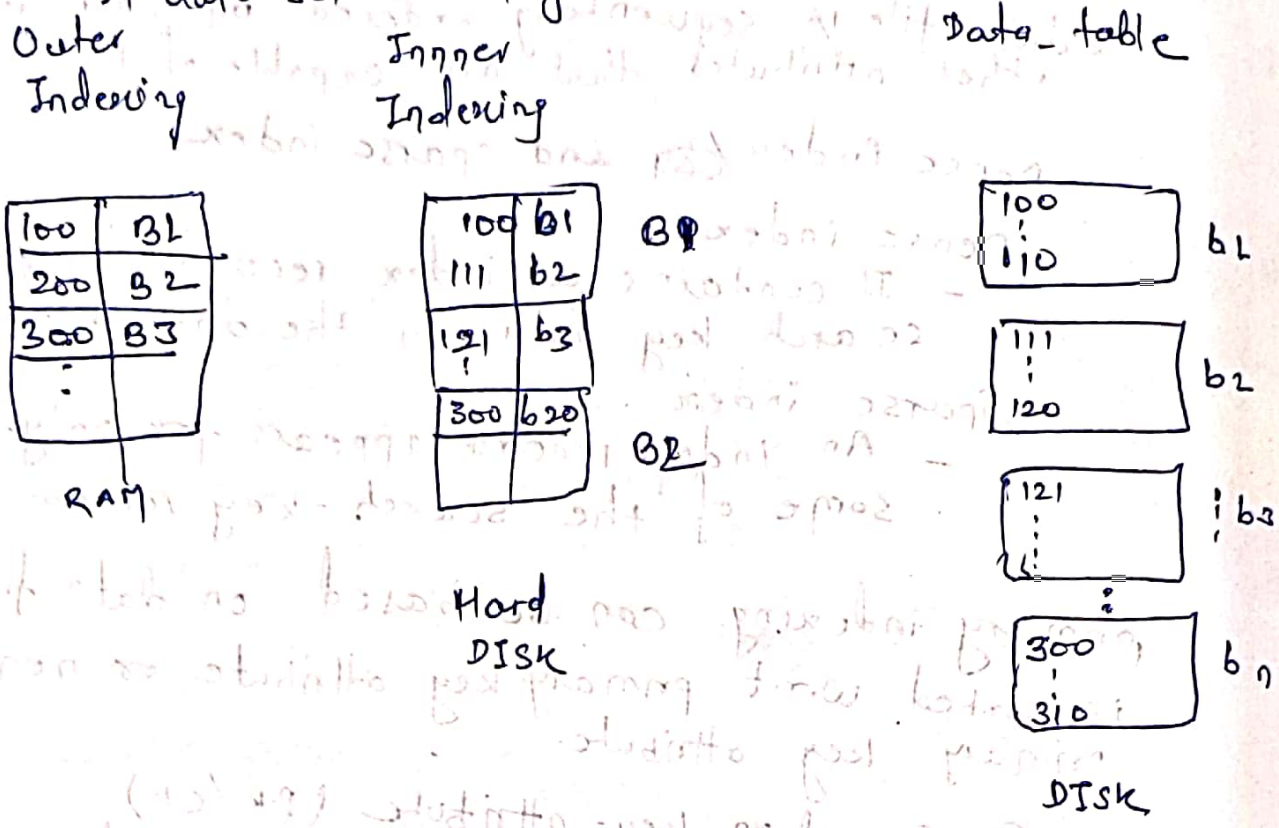
GROUP BY is usecase

no. of entries = no. of unique values of N.K attribute



© Multi-level indexing

→ If data set is huge.



2. Secondry-Indexing

- Applied on data files which are unsorted.
- I can not apply Binary search here.

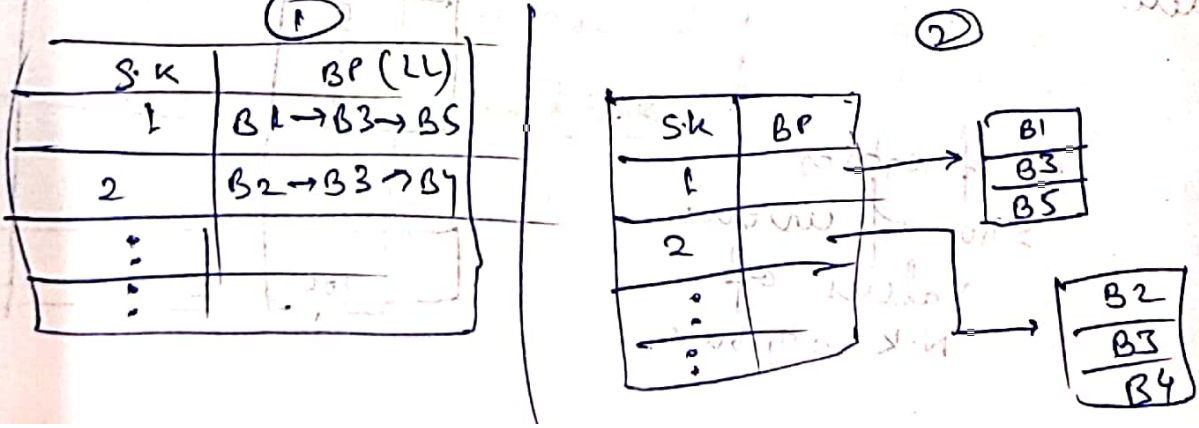
→ Search key = key / Non-key

→ Dense indexing

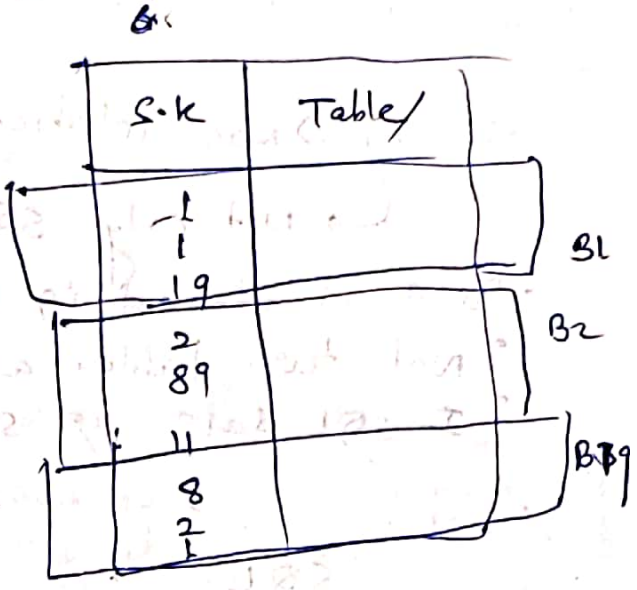
- We have to make entries for each search key in index table.

No. of entries = unique value of search-key

→ If there are multiple rows associated with a search key. How to handel



S.k	B.P
1	B1 → B19
2	B2 → B19



Method-① - Use linked list
Method-② - Use another table

Data file

- Since index table is sorted, I can apply binary search on index table
- fast accessing. I do not have to traverse data file linearly (unsorted).
- Cons: I have to do preprocessing
  Index table takes extra space.

Secondary Indexing is also called, non-clustering indexing.

Lecture: 15 : No SQL Models / DBs.

. No SQL ⇒ Non-relational model
↳ Not only SQL
- Data is stored in non-tabular form.
  - Not two tables are relational
  - In SQL data is structured / fixed schema.

SQL	NoSQL (Not only SQL)
• structured data	• structured data unstructured data semi-structured data
• Constraints	• No constraints
• Fixed schema	• Flexible schema.
• Vertical scaling is practial	• Vertical as well as horizontal scaling
• Tabular	• Non-tabular

- SQL is slowly accessible
- Consumes less memory
- But ☹ (No data redundancy)
  i.e. 1NF 2NF 3NF BCNF

Data-Modelling in SQL vs NoSQL (Not only SQL)

- JSON is an example of NoSQL data model.
- But the problem with NoSQL data model is
  ⇒ for a piece of information we have to import the whole file of that user even if they are not required.

Advantages of NoSQL

① Flexible schema (No need to store NULL as SQL)

② Horizontal scaling
↳ ~~Upscaling~~/upgrades

scaling

Vertical	Horizontal
→ (Hardware)	→ additional node
→ RAM	→ load is shared
→ CPU	→ efficient in terms of cost
Inside single system only	→ Scale out

→ Costly
→ upscaling

- we can apply horizontal scaling in SQL but, it will be extremely slow. It is not practical.
- Scaling horizontally is achieved by
  - sharding ↓ splitting large DBs to smaler, faster and managable sets.
  - and replical sets (10 server - same data)
    ↳ one goes down
    ↳ No problem

③ High-availability - more reliable

④ Easy read and insert operations but difficult to delete or modify operations.
The thumb rule of mongoDB is the data that is accessed together should be stored together.

⑤ Caching mechanism

⑥ No-SQL use case is more for cloud application

Use-case of NoSQL (Not only SQL)

1. Fast-paced agile development
2. Storage of unstructured and semi-structured data.
3. Huge volume of data
4. Requirement of scale-out i.e horizontal scaling
5. real-time streaming and micro-services

Misconception of NoSQL

- No SQL databases support ACID properties. ✗
  initially they don't support but now a days MongoDB will support them.
- Relationship data is best suited with relational databases.

- Types of NoSQL DB
  1. Key:value stores e.g. {id: "value"}
  2. Column-oriented / columnar / C-stores / wide-column use-case- analytics
  3. Document-Based stores
  4. Graph-Based stores
     e.g. Facebook

Disadvantages of NoSQL

1. Data Redundancy
2. Update and delete operation are costly.
3. All type of NoSQL data model doesn't fell all of your application needs.
4. Doesn't support ACID properties in general. However mongo-DB supports them.
5. Doesn't support data entry with consistency constraints.

Lec:16: Types of Databases

• SQL ko सबसे बड़ा Disadvantage:
  • Horizontal scalability
  • Incapable of cloud support (fast accessing)

- Relational databases
- Object oriented databases
- NoSQL databases
- Hierarical databases
- Network databases

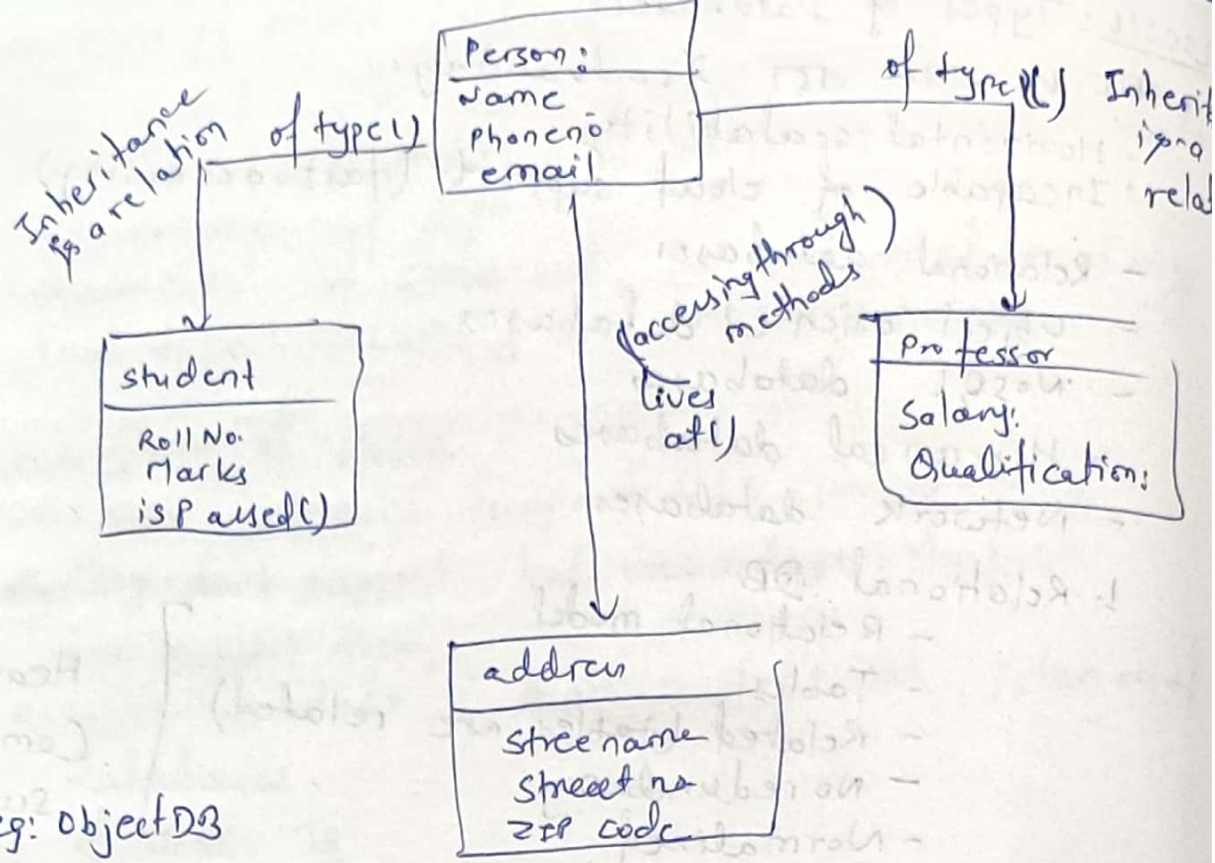
1. Relational DB
   - Relational model
   - Tables
   - Related (tables are related)
   - No redundancy
   - Normalized
   - Only vertical scaling
   - structured data

   ] Heavy Community support

2. Object Oriented DB
   class → attributes (characteristics)
   └→ Methods (Behaviour)
   objects are instance of class.
   - stores everything in form of object
   - object can have table, executable code
   - objects they communicate via methods
   - handles, which are equivalent to obj. ID.
   - object representation

   Use case:
   Sometimes DBs having multiple relations, so maintaing relationship between them can be tedious at times.
   (उबाऊ, थकाऊ, मांगा)

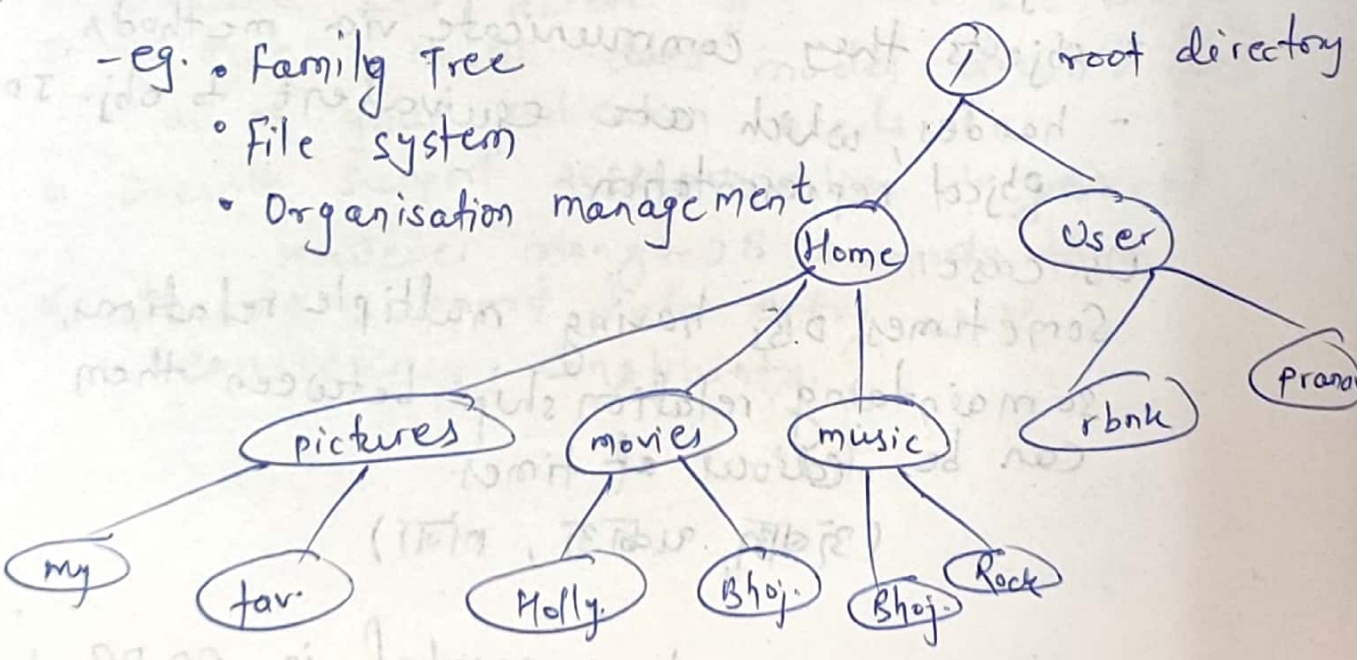
   - Views are not supported in OO DB. but



eg: ObjectDB

3. NoSQL databases
   - Non tabular
   - Schema free
   - Redundancy
   - Scalability (H & V) both
   - Handle huge data

4. Hierarical Databases
   - eg. • Family Tree
     • File system
     • Organisation management

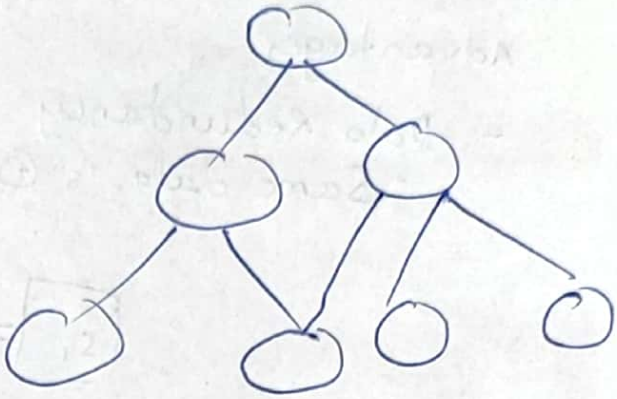


- easy to use structure
- disadvantage
  - Inflexible in nature
  - one to many relationship
  - No relationship between siblings.
  - only one parent / No multiple parent allowed.

eg: IBM IMS;

5. Network DB

- Extension of hierarical DB
- Multiple parents allowed
- M:N relationship
- Maintanance is tedious
- Graph
- No community support



Lecture:17 : clustering / Replication in DBMS

H.W: Content delivery network

- Database clustering (making replica sets) is process of combining more than one servers or instances connecting a single DB.
- Replicate the same dataset on different server.

Advantages

- Data Redundancy
  - Same data is ⊕rt on all three servers.

$S_1 = S_2 = S_3$

- Load balancing
  - Scalable database
  - High availability of data
- High availability

If S_1 → shutdown
  - → Maintainance
  - → attack
  - → power failure

S_2 & S_3 works as backup.

clustering Working:

- In this arcitecture, all requests are split with many computers.

Lecture: 18

→ Partitioning and sharding in DBMS.

Data = stored = DBMS

↳ Huge Amount of Data → Masses www hosted

⇓

Facebook/ Instagram/ Amazon

① Huge Data (issue: managability)

② No of request is high (issue: single system is not capable of handle the load)

⇓

Distributed Database

Database optimization

① Scale-up (hadrawware improvement)

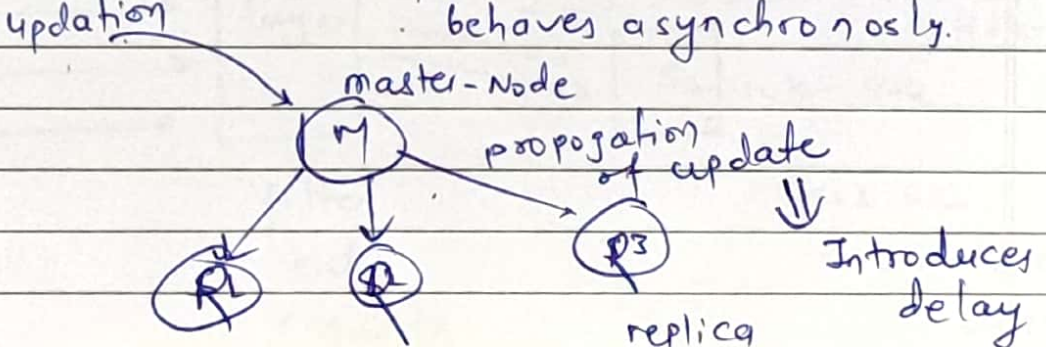
- expensive
- scaling up does not reduce responce time at that level, it have to.

scaling = 2x

responce time = $\frac{x}{2}$ } Not necessary

② Add-Replica set (clustering)

- Introduces redundancy
- Manages load
- Updation-problem. behaves asynchronosly.

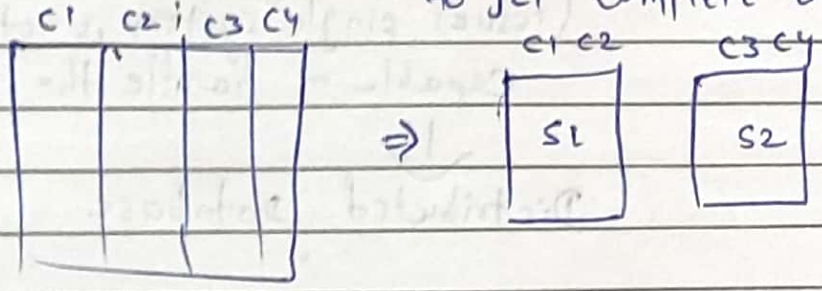


③ partitioning (scale-out)

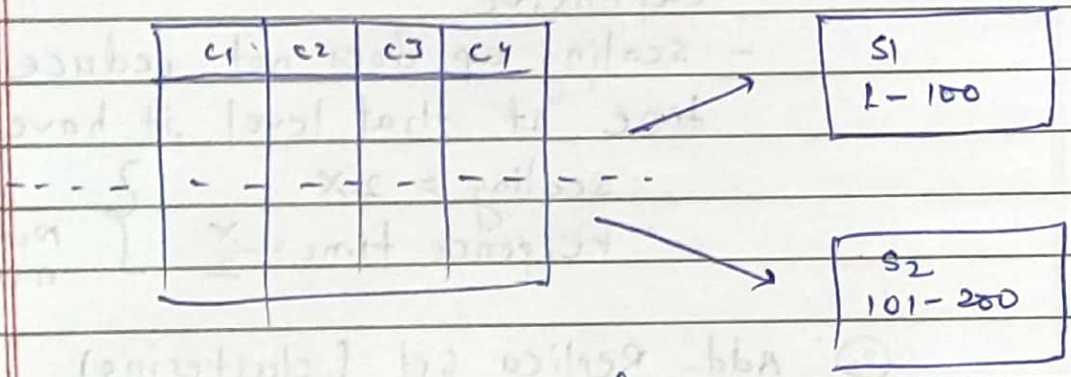
scale-out : Horizontal scaling
scale-up : Vertical scaling

- Vertical partitioning
- Horizontal partitioning

Vertical partitioning : needs to access multiple servers to get complete data.



Horizontal partitioning



Independent chunk of data tuples are stored in different servers.

Page No. :
Date : / /

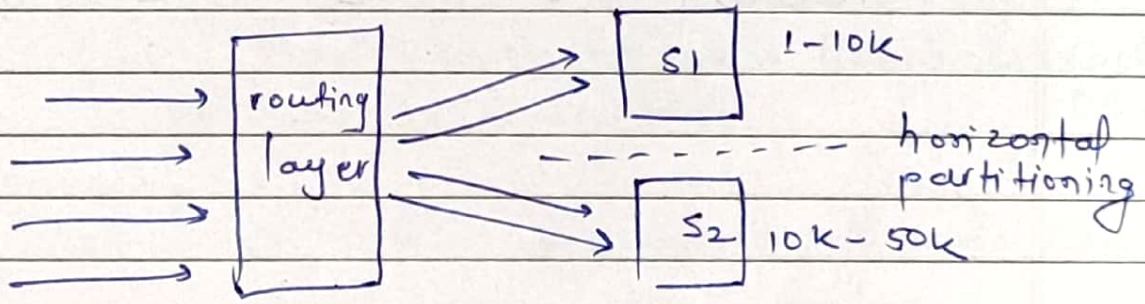
Q) When to use partitioning
- Data is huge
- Larger No of requests involves huge responce time

Advantages
- Parallelism
- Avolability ↑
- Performance ↑
- Managibility ↑
- reduce cost as scaling up is costly.

Distributed DB
A single logical DB, that is spread across multiple locations (servers) and logically interconnected by N/w.

Optimization techniques
- clustering
- partitioning
- Sharding

Sharding
- Technique to implement horizontal partitioning



filter out requests

S1 & S2 are independent units

Page No.:
Date: / /

advantages of sharding (horizontal partitioning)
- scalability
- Avalability

disadvantages
- Increases complexity by making routing layer
- Not well suited for analytical queries where, data of a column is required whole
- Non-uniformity can be introduced in different shards, if we will not use proper partition/sharding key.

Non-uniformity-
- shard 1 → 1000 units
- shard 2 → 10,000 units

again we stuck on same point as before

- To reduce non-uniformity we will reshard the table time to time.

Page No.:
Date: / /

Lecture: 19 Database scaling patterns

Step-by step manner,
when to use which scaling option.

A case-study
A cab booking app

Step: 1
request: 1 trip booking in 5 min
handle: A single machine, normal pc, laptop.

Step: 2
Request: 30 booking per minute

Problem: API latency ↑
DB performance ↓
Transaction facing deadlock, starvation, request failure.

handle:
(1) Performance optimization method
Query optimization & connection pool implementation
→ cache introduce
→ Introduce data redundancy
→ Use connection pool libraries: Cache DB connect
~~(2) scale your system~~

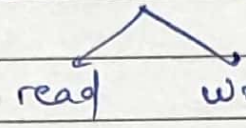
- Multiple application threads can use same DB connection.

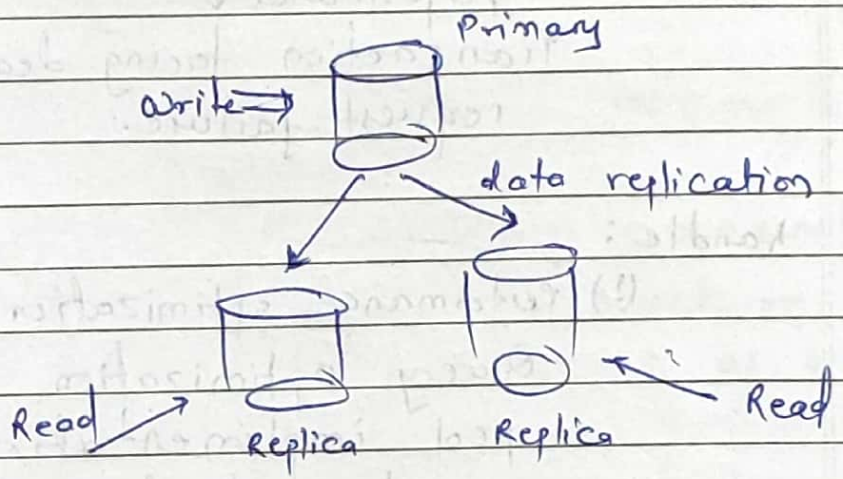
step: 3
request: 100 booking per minute

handle: vertical scaling
↳ upgraded your pc to the top of the line pc.

step: 4 request ↑ 300 booking per minute

handle: CQRS
Command Query Responsibility Segregate





there will be a lag between read and write operation but it is acceptable

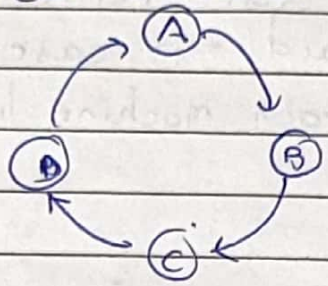
Page No. :
Date : / /

step: 5

request: 50 request per second

problem: primary is not able to handle all write requests.

handles: Multi. primary replication
- we can write at any node
- logical circular ring

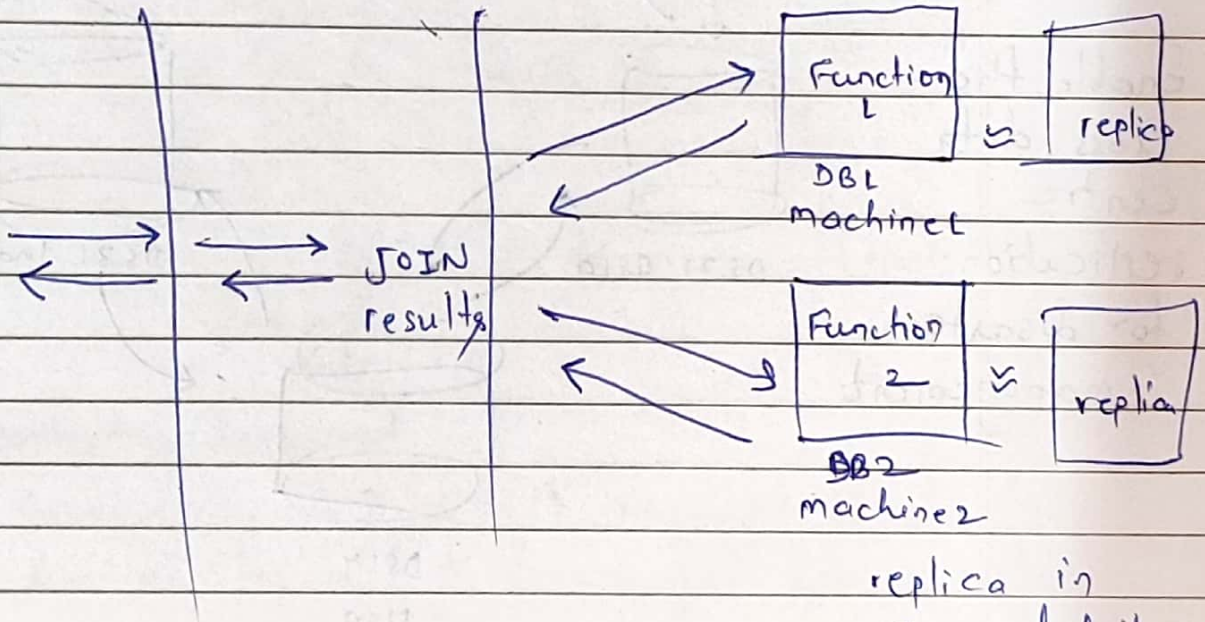


- read request is broadcast to all nodes whoever replies first gets sent.

step: 6

request: 50 request per second

handle: partitioning of data by functionality



replica in case of failure

Page No.:
Date: / /

step: 7 : Scaling to other country

problem: latency ↑ for foreign country customers

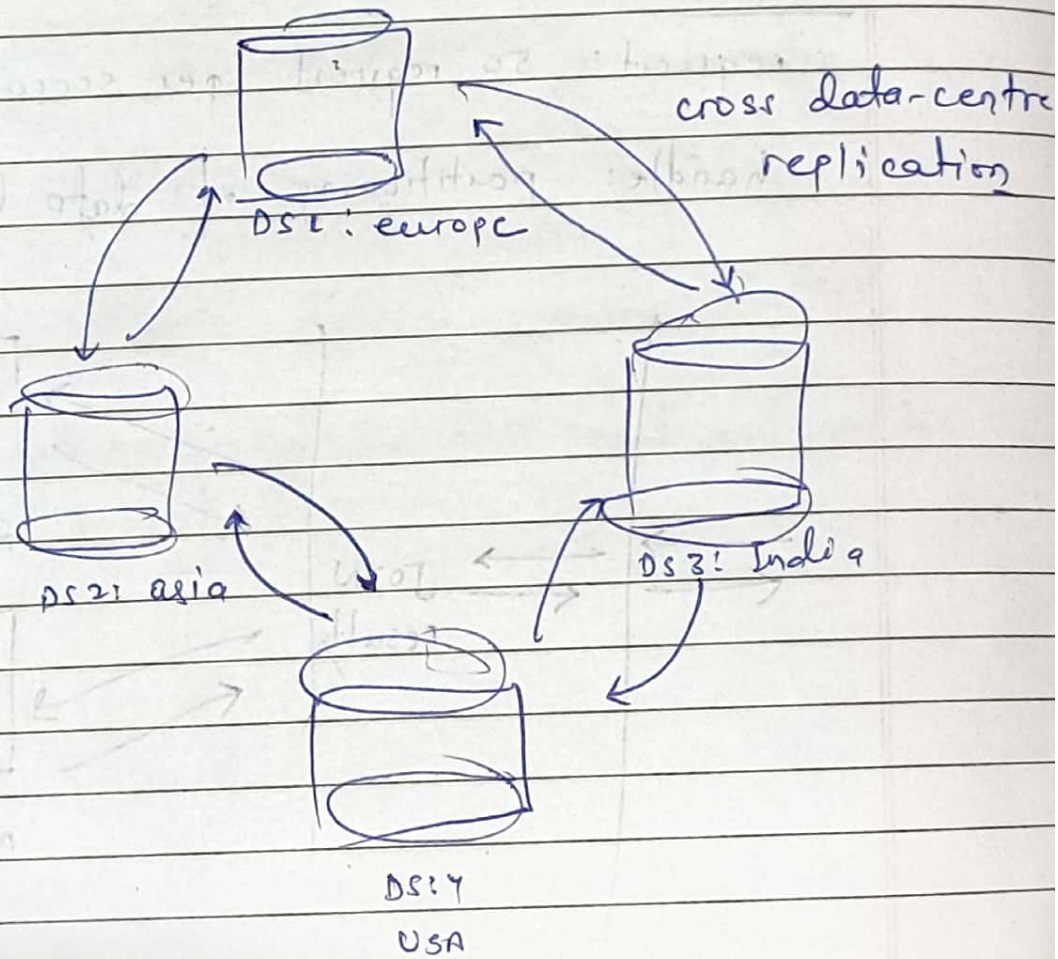
Solve: sharding : Horizontal partitioning
: set up so machine
: Also use replica sets for each shard, in case of system failure.

sharding: each machine holds a part of data.

step: 8 : Continent target
problem: high latency
handle: Data centre wise partition

Same continent request goes to nearest data centre.

enable the cross data centre replication for disaster management

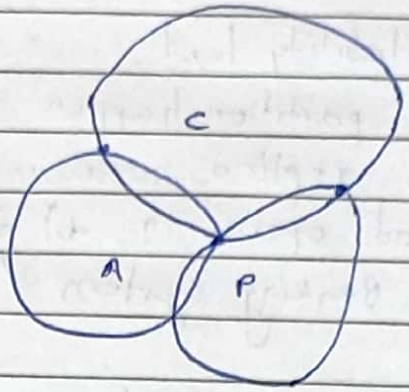


Lecture: 20 : CAP theorum

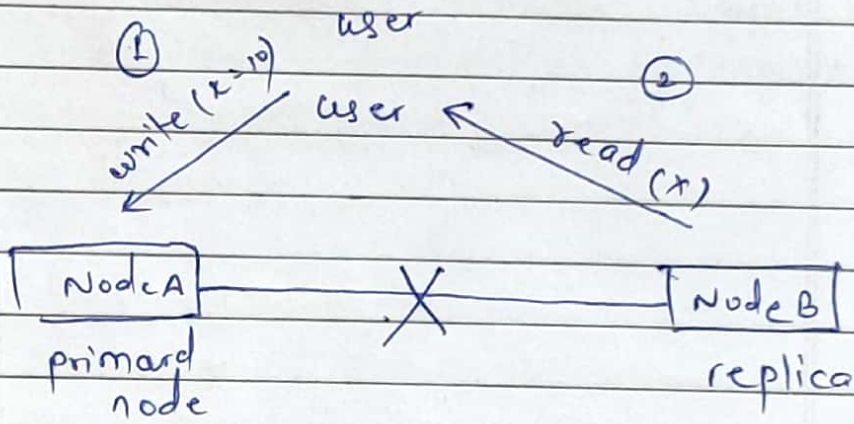
Page No. :
Date : / /

- For designing of efficient distributed storage CAP theorum
  - Consistency
  - Availability
  - Partition tolerance

- It states that



- The CAP theorum states that a distributed system can only provide two of three listed properties simultaneously.
- The theorum formalises the trade off between consistency and availability when there is a partition.



partioting happened

1. Consistency lost (both request should proceed)
2. Availability lost (if we fail one request)

CAP all three is achieved in a hypothetical DB of single node.
but wait CAP theorum is for distributed system

1. CA: partition tolerance
   DB on single Node
   DB is not distributed

2. CP: Availability lost
   ⇒ if partition happen we will turn off the replica nodes.
   ⇒ read operation को बंद कर देंगे
   uses: Banking system / mongo DB

3. AP: Consistency lost
   ⇒ If partition happens we do nothing.
   ⇒ Availibility high
   ⇒ use: facebook, social media, news page

Page No. :
Date : / /

→ BASE properties (HW)
→ ACID vs BASE properties.

BA: Basically available
S: soft state
E: Eventual consistency

} priority availibity and flexibility over strictness and immediate consistency

Basically available:
after system partition / partial failure the system gurantees that data is generally available.

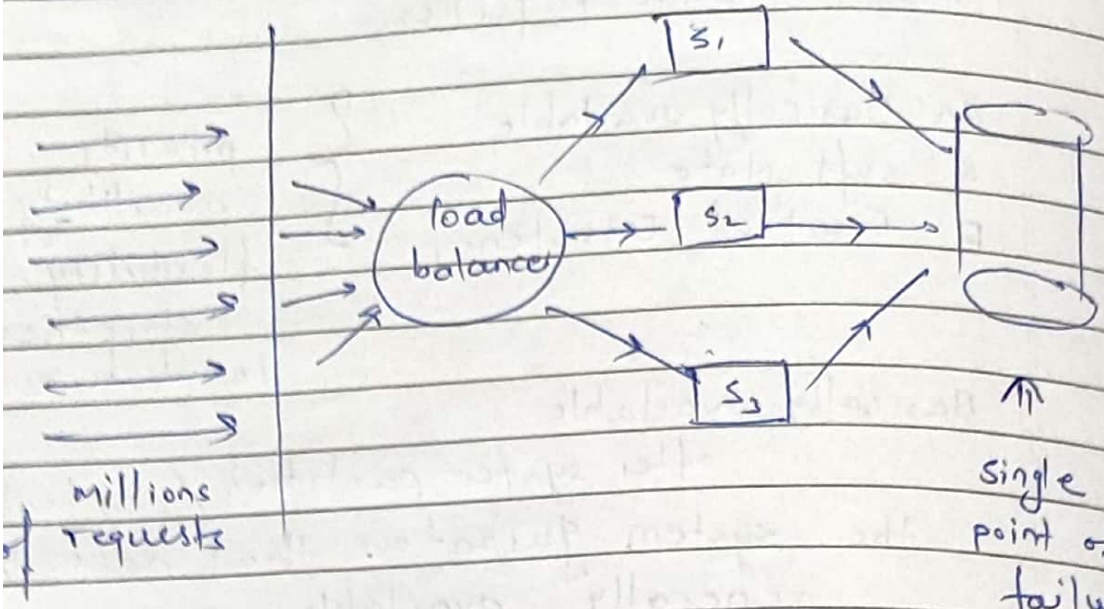
~~Soft state~~
Eventful Consistency:
temprory inconsistency may occur but the system will syncronize them given enough time.

Soft state:
different replicas of the same data might not be perfectly syncronised.

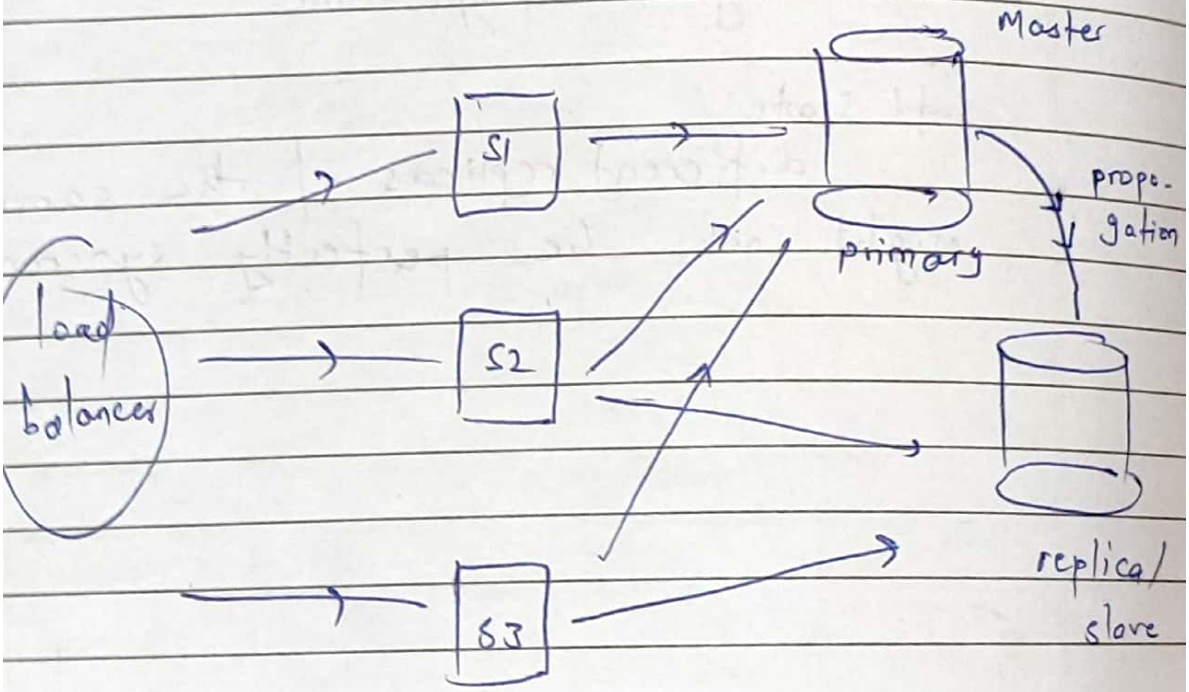
LCE:21 : Master - slave architecture

Page No.:
Date: / /

Normal - architecture



Master - slove architecture



Page No.:
Date: / /

Master Node: Handles all write operation
: Orginal DB
: latest DB
: primary DB
: latest information

example: patterns of lec: 19

Slave Node: Handles only read operation
: DB replication (to get updated)

Updation could be
→ synchronus: at the moment updation
eg: Banking a/c.
→ asynchronous: updation time to time (5sec)
eg: facebook likes
facebook Comments
zomato

Advantages M-S architecture
- as a backup
- scale out read operation
eg facebook
⇒ post ↓
reels scroll ↑
story likes ↑
- availability ↑
- Reliable ↑
- Latency ↓
- parallelism (read request) introduce

changes to be done

① update title in index.html
(remove Greatstack)