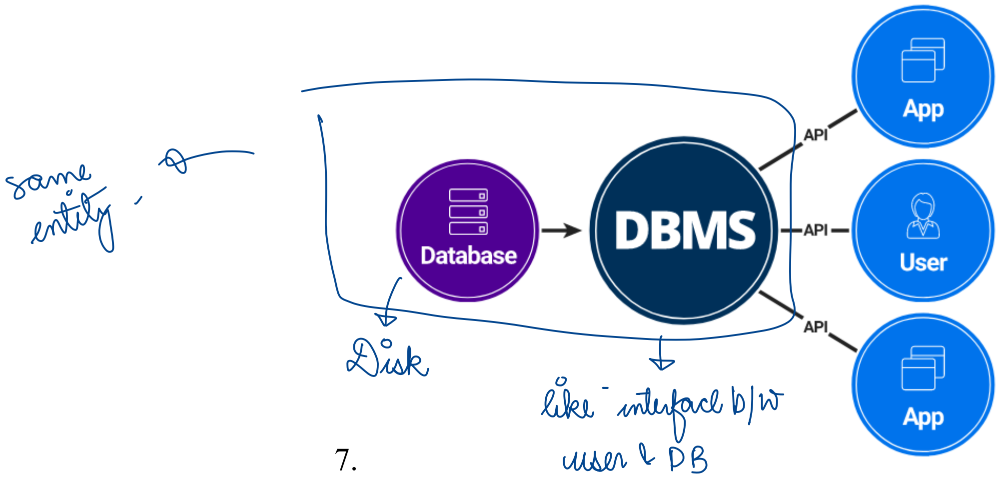
DBMS

7.



8. DBMS vs File Systems

a. **File-processing systems** has major **disadvantages**.

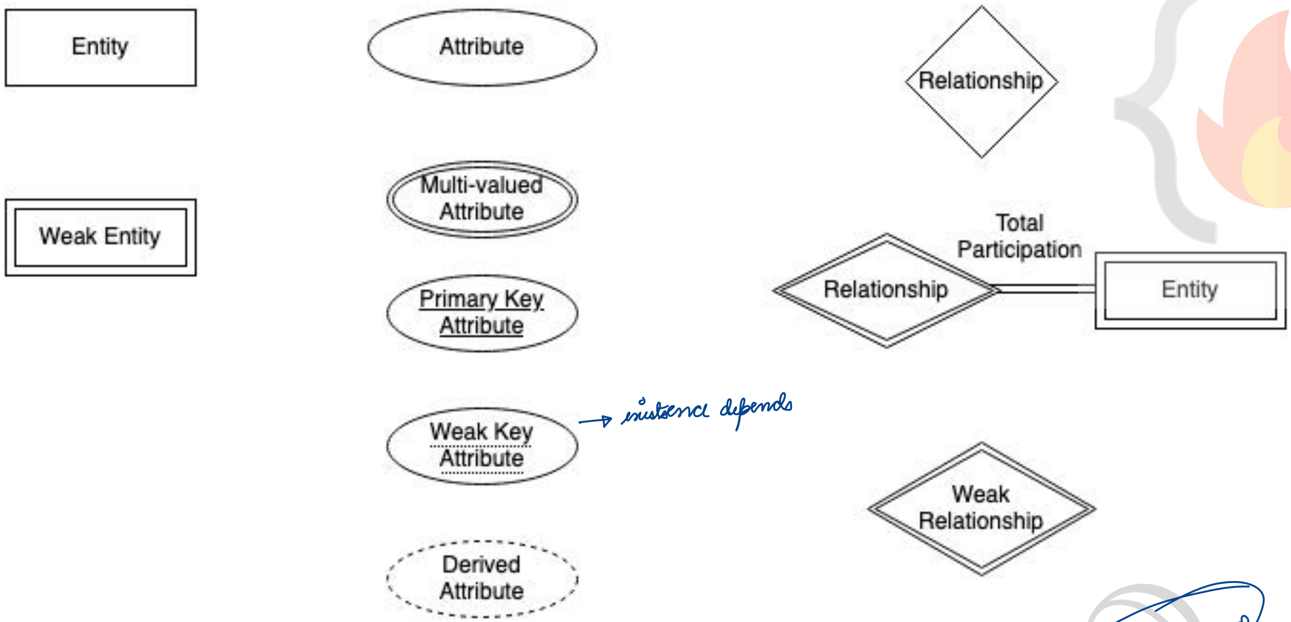
  i. Data Redundancy and inconsistency → may have same info under multiple files
  ii. Difficulty in accessing data
  iii. Data isolation
  iv. Integrity problems
  v. Atomicity problems → should complete.
  vi. Concurrent-access anomalies → accessing from multiple places at same time may cause prob.
  vii. Security problems → shouldn't be accessible by all.

all inbuilt in DBMS

major adv. in DBMS.

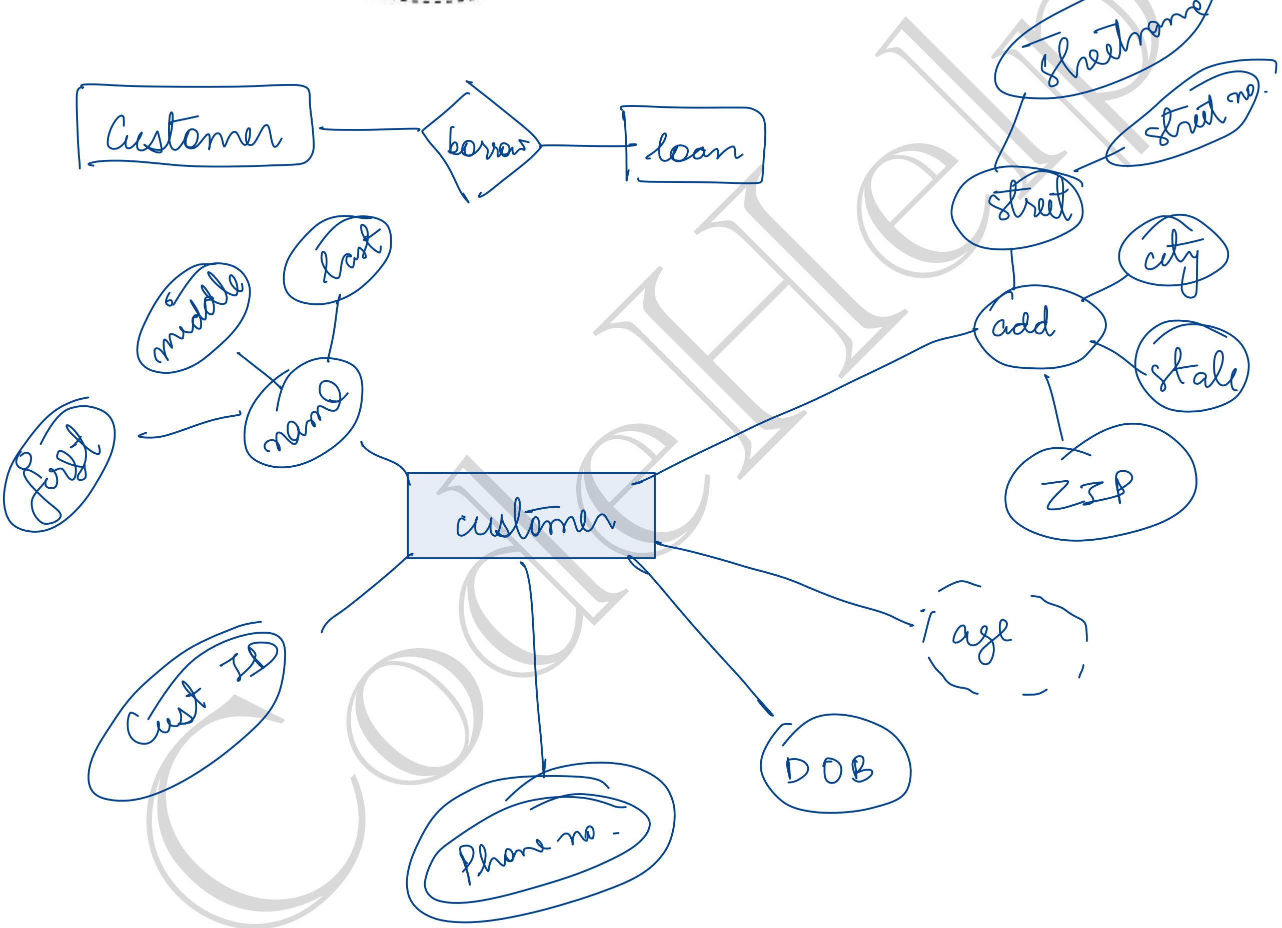
b. Above 7 are also the **Advantages of DBMS** (answer to "**Why to use DBMS?**")

Symbols used in ER Diagram



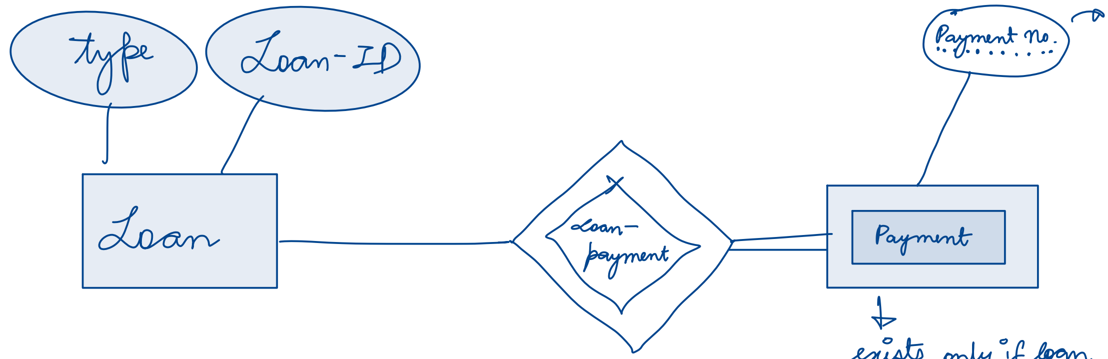
→ existence depends

eg:



eg.

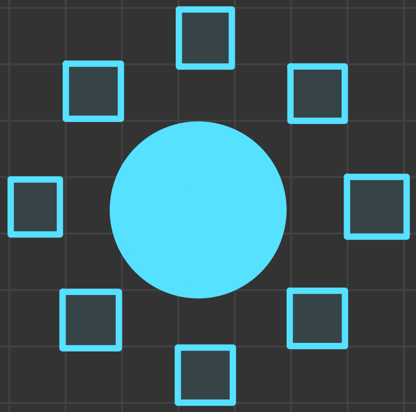
eg.



→ can't be uniquely identified.

↓ exists only if loan does

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

name

① Banking system — Branches, (name)

② Bank → customers.

③ Customers — accounts, , & take loan,

④ Customer associated with some banker.

associated

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 (generalise to) ⑦ payment (Loan) (weak entity)

② Attributes:-

① branch → name (→ primary Key), city, assets, liabilities

② Customer → cust_id (→ primary Key), 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

Key

derived attribute

single valued

multivalued,

④ Saving account → ~~acc-number~~, ~~balance~~, interest-rate

acc. no., balance, daily withdrawal limit.

PK

⑤ Current a/c → ~~acc-number~~, ~~balance~~, per transaction charges, overdraft-amount.

⑥ Generalized Entity "Account" → acc-no., balance

⑦ Loan → loan-number, amount

⑧ Weak Entity Payment → Payment no., date, amount.

③ Relⁿ & Constraints

① Customer borrow loan.

M : N

② Loan originated by branch.

N : 1

③ Loan loan-payment Payment.

1 : N

④ Customer deposit account

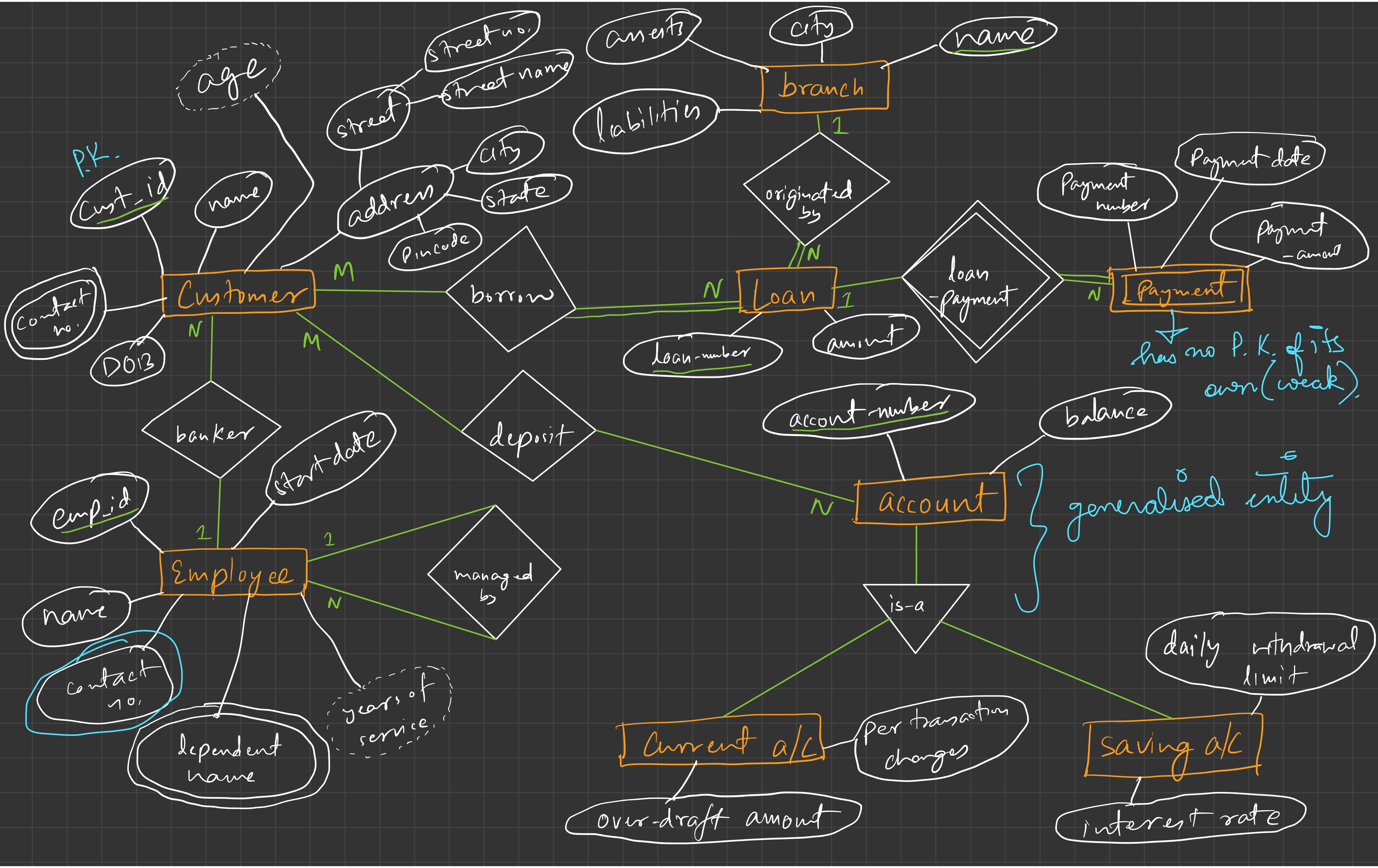
M : N

⑤ Customer banker Employee.

N : 1

⑥ Employee manage by Employee,

N : 1


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

Lec-6

★★ Practise ER diagrams.

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.

different as different users can

① identify Entity sets.

① user_profile ② user_post ③ post_comment

④ post_like

② Attributes + types.

① user_profile → Name, username (PK), email, Password, Contact no.

Name → Composite. DOB, age → derived.

email → multivalued, Contact no. → multivalued.

② user_post → post_id, text content, image, video,

image → multivaled, video → multivaled.

created_timestamp, modified_timestap.

③ post_comment → post_comment_id, text content, timestamp

④ post_like → post like_id, timestamp

③ Rel^n & Constraints.

① user_profile friendship user_profile.

M : N

② user_profile posts user_post.

1 : N

③ user_profile can post_like.

1 : N

=

④ user_profile comments post_comment.

1 : N

=

⑤ user_post has post_comment

1 : N

=

⑥ user_post has post_like

1 : N

=

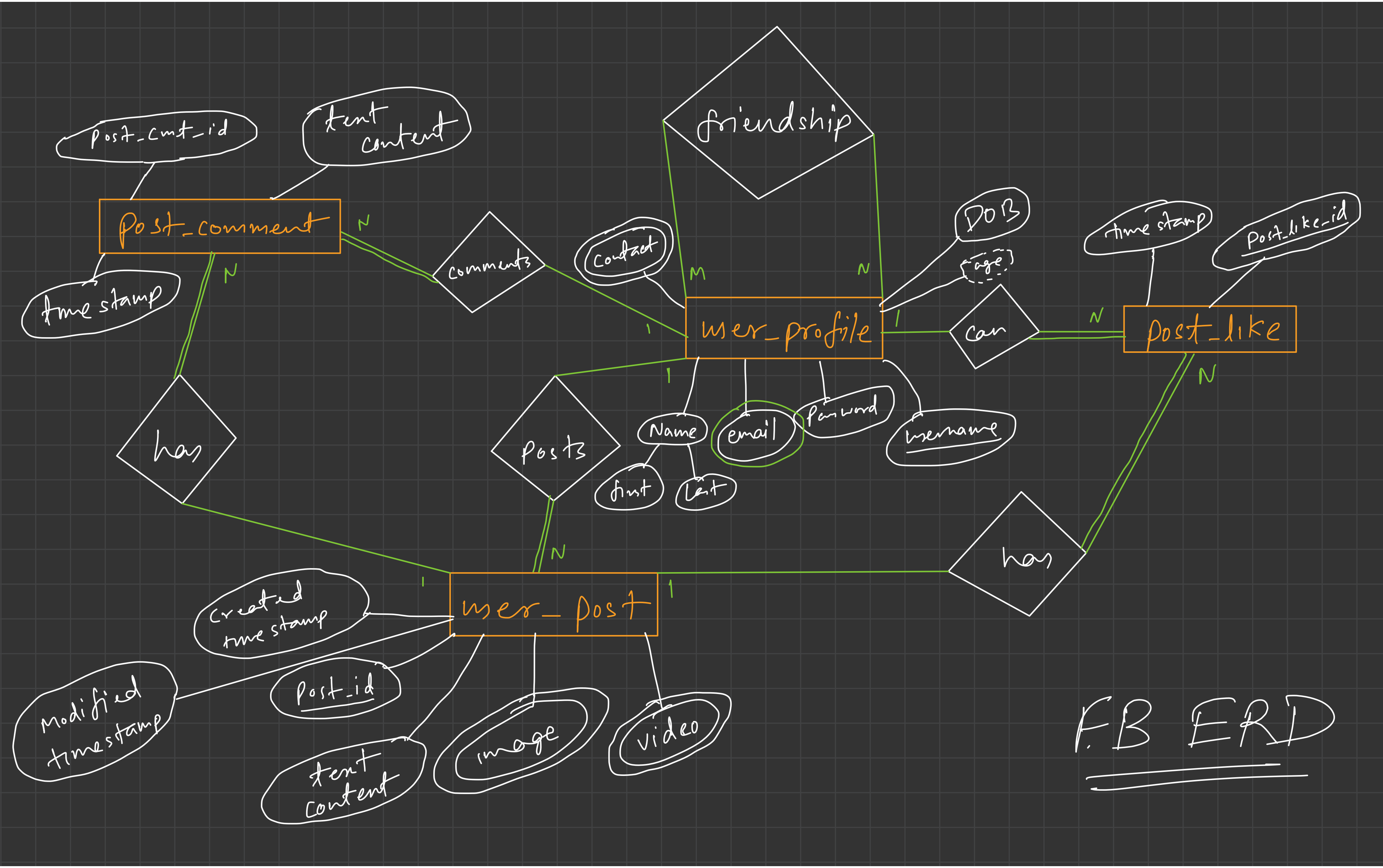

post_cmt_id
text content
post_comment
N
time stamp
N
comments
friendship
contact
M
N
DOB
age
time stamp
Post_like_id
1
user_profile
1
can
N
post_like
N
has
1
posts
Name
email
Password
username
first
last
N
1
user_post
1
has
created time stamp
Post_id
Modified timestamp
text content
image
video
F.B ERD

Table : manages (mg-id, emp-id, job-id, branch-id)

→ PK

1. Table of the relationship set is made.
2. Attributes includes primary keys of entity set and aggregation set's entities.
3. Also, add descriptive attribute if any on the relationship.

Loan →

Loan-no.	Amt.

weak en. Payment →

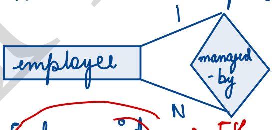
Loan no.	pay. no.	date	amt

PK

Loan no. → FK.

PK of strong

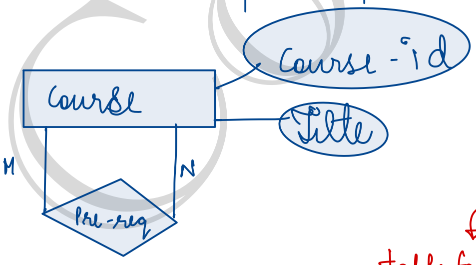
★ Unary relationship : we will add another attribute in employee table which will be FK.



emp-id	name	joining date	Emp-mg-id
200	..	..	205
202	..	..	205
205	..	..	NULL

Emp-mg-id → FK, PK of this table only.

★ M : N



① Course (id, Title)

② Prereq (id, pre-req-course-id)

table for relationship

id → FK

pre-req-course-id → FK.

→ PK

* 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})

9. **Advantages of Indexing**
    1. Faster access and retrieval of data.
    2. IO is less.
10. **Limitations of Indexing**
    1. Additional space to store index table
    2. Indexing Decrease performance in INSERT, DELETE, and UPDATE query.

8. SQL vs NoSQL

	SQL Databases	**NoSQL Databases**
Data Storage Model	Tables with fixed rows and columns	Document: JSON documents, Key-value: key-value pairs, Wide-column: tables with rows and dynamic columns, Graph: nodes and edges
Development History	Developed in the 1970s with a focus on reducing data duplication	Developed in the late 2000s with a focus on scaling and allowing for rapid application change driven by agile and DevOps practices.
Examples	Oracle, MySQL, Microsoft SQL Server, and PostgreSQL	Document: MongoDB and CouchDB, Key-value: Redis and DynamoDB, Wide-column: Cassandra and HBase, Graph: Neo4j and Amazon Neptune
Primary Purpose	General Purpose	Document: general purpose, Key-value: large amounts of data with simple lookup queries, Wide-column: large amounts of data with predictable query patterns, Graph: analyzing and traversing relationships between connected data
Schemas	Fixed	Flexible
Scaling	Vertical (Scale-up)	Horizontal (scale-out across commodity servers)
ACID Properties	Supported	Not Supported, except in DB like MongoDB etc.
JOINS	Typically Required	Typically not required
Data to object mapping	Required object-relational mapping	Many do not require ORMs. MongoDB documents map directly to data structures in most popular programming languages.

• structured	• struc. , unstruc.., semi-struc.
• fix schema	• flexible schema

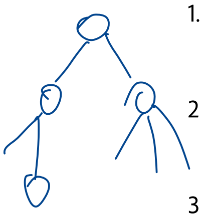
LEC-16: Types of Databases

1. **Relational Databases**
    1. Based on Relational Model.
    2. Relational databases are quite **popular**, even though it was a system designed in the 1970s. Also known as relational database management systems (RDBMS), relational databases commonly use **Structured Query Language (SQL)** for operations such as **creating**, **reading**, **updating**, and **deleting** data. Relational databases store information in **discrete tables**, which can be **JOIN**ed together by fields known as **foreign** keys. For example, you might have a User table which contains information about all your users, and **join** it to a Purchases table, which contains information about all the purchases they've made. MySQL, Microsoft SQL Server, and Oracle are types of relational databases.
    3. they are ubiquitous, having acquired a steady user base since the 1970s
    4. they are highly optimised for working with structured data.
    5. they provide a stronger guarantee of data normalisation
    6. they use a well-known querying language through SQL
    7. **Scalability issues** (Horizontal Scaling).
    8. Data become huge, system become more complex.

Class → inheritance → objects
→ attribrutes
→ methods.

→ stores as objects. → object → table, executable-code ;
→ object communicate via methods → handles → object ID

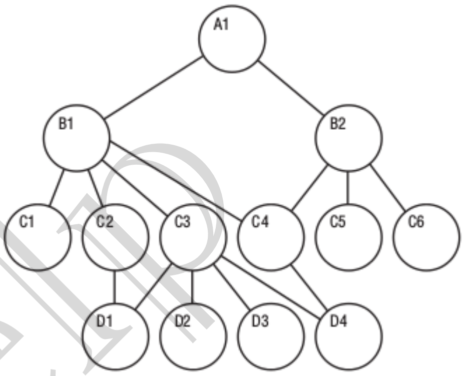
2. **Object Oriented Databases**
    1. The object-oriented data model, is based on the **object-oriented-programming paradigm,** which is now in wide use. **Inheritance**, **object-identity**, and **encapsulation** (information hiding), with methods to provide an interface to objects, are among the key concepts of object-oriented programming that have found applications in data modelling. The object-oriented data model also supports a rich type system, including structured and collection types. While inheritance and, to some extent, complex types are also present in the E-R model, encapsulation and object-identity distinguish the object-oriented data model from the E-R model.
    2. Sometimes the database can be very complex, having multiple relations. So, maintaining a relationship between them can be tedious at times.
        1. In Object-oriented databases data is treated as an object.
        2. All bits of information come in one instantly available object package instead of multiple tables.
    3. **Advantages**
        1. Data storage and retrieval is easy and quick.
        2. Can handle complex data relations and more variety of data types that standard relational databases.
        3. Relatively friendly to model the advance real world problems
        4. Works with functionality of OOPs and Object Oriented languages.
    4. **Disadvantages**
        1. High complexity causes performance issues like read, write, update and delete operations are slowed down.
        2. Not much of a community support as isn't widely adopted as relational databases.
        3. Does not support views like relational databases.
    5. e.g., ObjectDB, GemStone etc.
3. **NoSQL Databases**
    1. NoSQL databases (aka "not only SQL") are non-tabular databases and store data differently than relational tables. NoSQL databases come in a variety of types based on their data model. The main types are document, key-value, wide-column, and graph. They provide flexible schemas and scale easily with large amounts of data and high user loads.
    2. They are schema free.
    3. Data structures used are not tabular, they are more flexible, has the ability to adjust dynamically.
    4. Can handle huge amount of data (big data).
    5. Most of the NoSQL are open sources and has the capability of horizontal scaling.
    6. It just stores data in some format other than relational.
    7. Refer **LEC-15** notes…
4. **Hierarchical Databases**



    1. As the name suggests, the hierarchical database model is most appropriate for use cases in which the main focus of information gathering is based on a **concrete hierarchy**, such as several individual employees reporting to a single department at a company.
    2. The schema for hierarchical databases is defined by its **tree-like** organisation, in which there is typically a **root** "parent" directory of data stored as records that links to various other subdirectory branches, and each subdirectory branch, or child record, may link to various other subdirectory branches.
    3. The hierarchical database structure dictates that, while a parent record can have several child records, each child record can only have **one parent** record. Data within records is stored in the form of fields, and each field can only contain one value. Retrieving hierarchical data from a hierarchical database architecture requires traversing the entire tree, starting at the root node.
    4. Since the **disk storage system** is also inherently a hierarchical structure, these models can also be used as physical models.
    5. The key **advantage** of a hierarchical database is its ease of use. The one-to-many organisation of data makes traversing the database simple and fast, which is ideal for use cases such as website drop-down menus or computer folders in systems like

    Microsoft Windows OS. Due to the separation of the tables from physical storage structures, information can easily be added or deleted without affecting the entirety of the database. And most major programming languages offer functionality for reading tree structure databases.
    6. The major **disadvantage** of hierarchical databases is their inflexible nature. The one-to-many structure is not ideal for complex structures as it cannot describe relationships in which each child node has multiple parents nodes. Also the tree-like organisation of data requires top-to-bottom sequential searching, which is time consuming, and requires repetitive storage of data in multiple different entities, which can be redundant.
    7. e.g., IBM IMS. Info Mgt Sys.

5. **Network Databases**
    1. **Extension** of Hierarchical databases
    2. The child records are given the freedom to associate with multiple parent records.
    3. Organised in a **Graph** structure.
    4. Can handle complex relations.
    5. Maintenance is tedious.
    6. **M:N links** may cause slow retrieval.
    7. Not much web community support.
    8. e.g., Integrated Data Store (IDS), IDMS (Integrated Database Management System), Raima Database Manager, TurboIMAGE etc.



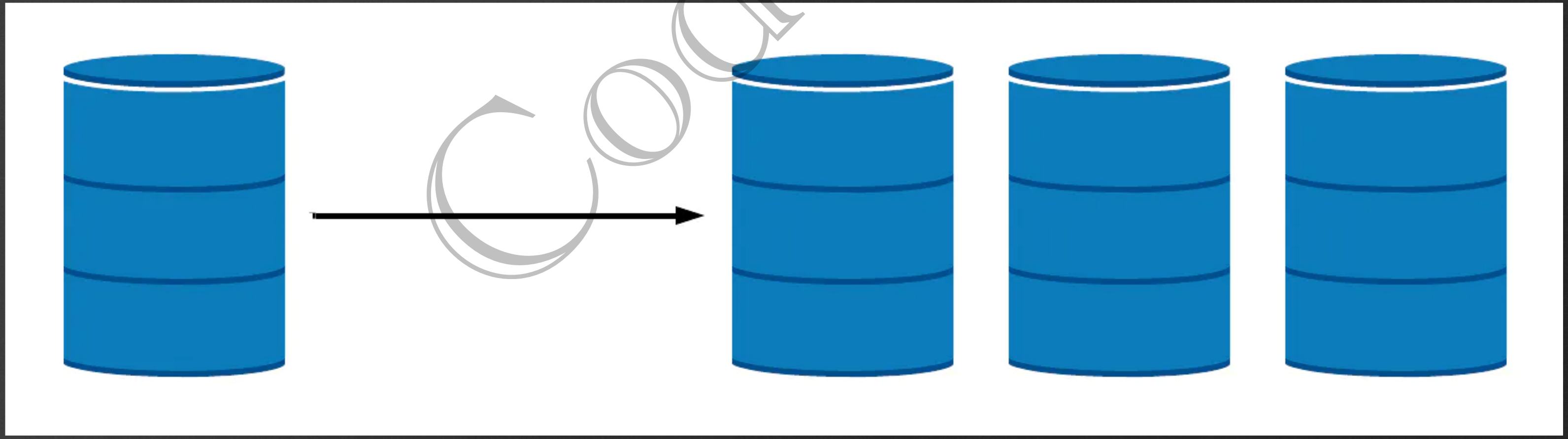
Database Scaling Patterns

Step by Step Scaling

- Lakshay

What will you learn?

- Step by Step manner, when to choose which Scaling option.
- Which Scaling option is feasible practically at the moment.



A Case Study

Cab Booking APP

- Tiny startup.
- ~10 customers onboard.
- A single small machine DB stores all customers, trips, locations, booking data, and customer trip history.
- ~1 trip booking in 5 mins.

Your App becoming famous, but…

The PROBLEM begins

- Requests scales upto 30 bookings per minute.
- Your tiny DB system has started performing poorly.
- API latency has increased a lot.
- Transactions facing Deadlock, Starvation, and frequent failure.
- Sluggish App experience.
- Customer dis-satisfaction.

Is there any solution?

- We have to apply some kind of performance optimisation measures.
- We might have to scale our system going forward.

Pattern 1

Query Optimisation & Connection Pool Implementation

- Cache frequently used non-dynamic data like, booking history, payment history, user profiles etc.
- Introduce Database Redundancy. (Or may be use NoSQL)
- Use connection pool libraries to **Cache DB connections.**

cache connection b/w client & server

- Multiple application threads can use same DB connection.
- Good optimisations as of now.
- Scaled the business to one more city, and now getting ~100 booking per minute.

Pattern 2

Vertical Scaling or Scale-up

- Upgrading our initial tiny machine.
- RAM by 2x and SSD by 3x etc.
- Scale up is pocket friendly till a point only.
- More you scale up, cost increases exponentially.
- Good Optimisation as of now.
- Business is growing, you decided to scale it to 3 more cities and now getting 300 booking per minute.

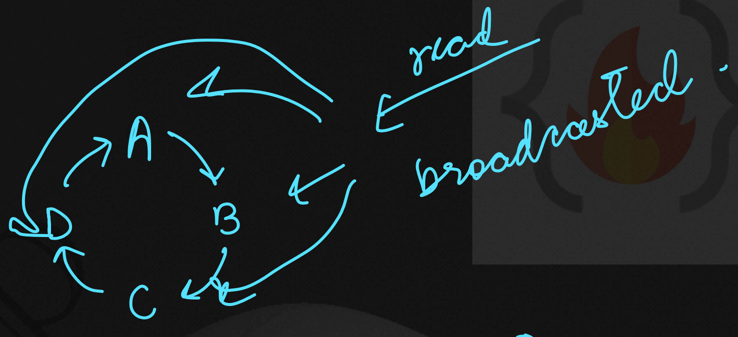
Pattern 3

Command Query Responsibility Segregation (CQRS)

- The scaled up big machine is not able to handle all read/write requests.
- Separate read/write operations physical machine wise.
- 2 more machines as replica to the primary machine.
- All read queries to replicas.
- All write queries to primary.
- Business is growing, you decided to scale it to 2 more cities.
- Primary is not able to handle all write requests.
- Lag between primary and replica is impacting user experience.

Pattern 4

Multi Primary Replication



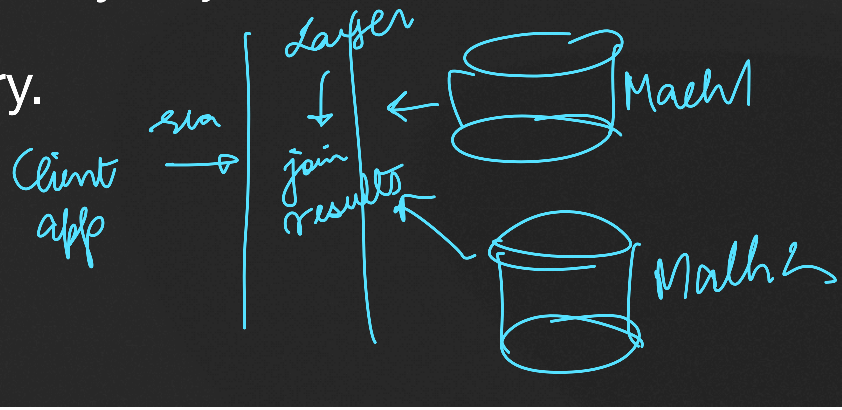
Any node can write

- Why not distribute write request to replica also?
- All machines can work as primary & replica.
- Multi primary configuration is a logical circular ring.
- Write data to any node.
- Read data from any node that replies to the broadcast first.
- You scale to 5 more cities & your system is in pain again. (~50 req/s)

Pattern 5

Partitioning of Data by Functionality

- What about separating the location tables in separate DB schema?
- What about putting that DB in separate machines with primary-replica or multi-primary configuration?
- Different DB can host data categorised by different functionality.
- Backend or application layer has to take responsibility to join the results.
- Planning to expand your business to other country.



Pattern 6

Horizontal Scaling or Scale-out

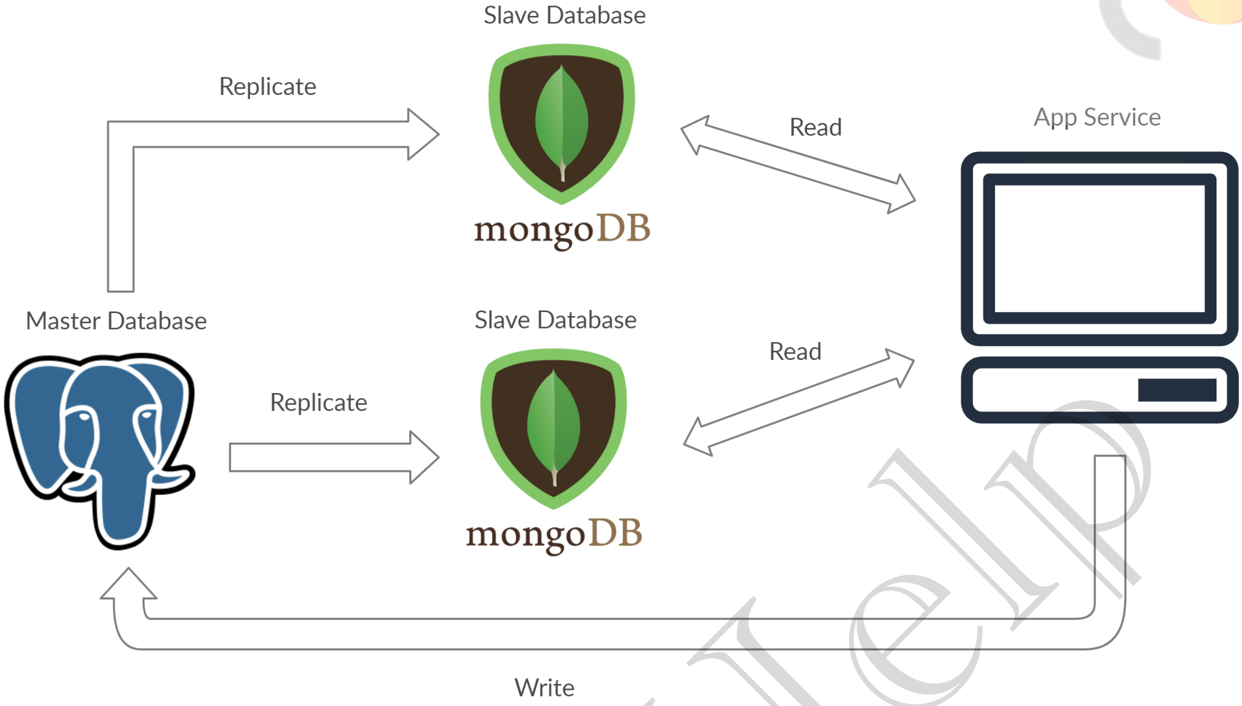
- Sharding - multiple shards.
- Allocate 50 machines - all having same DB schema - each machine just hold a part of data.
- Locality of data should be there. } shouldn't switch b/w machines baar-baar.
- Each machine can have their own replicas, may be used in failure recovery.
- Sharding is generally hard to apply. But “No Pain, No Gain”
- Scaling the business across continents.

Pattern 7

Data Centre Wise Partition

- Requests travelling across continents are having high latency.
- What about distributing traffic across data centres?
- Data centres across continents.
- Enable cross data centre replication which helps disaster recovery.
- This always maintain Availability of your system.
- Now, Plan for an IPO :p

LEC-21: The Master-Slave Database Concept



1. Master-Slave is a general way to optimise IO in a system where number of requests goes way high that a single DB server is not able to handle it efficiently.
2. Its a Pattern 3 in LEC-19 (Database Scaling Pattern). ***(Command Query Responsibility Segregation)*** (Replicas with one primary)
3. The true or latest data is kept in the Master DB thus write operations are directed there. Reading ops are done only from slaves. This architecture serves the purpose of safeguarding site ***eliability, availability, reduce latency etc .*** If a site receives a lot of traffic and the only available database is one master, it will be overloaded with reading and writing requests. Making the entire system slow for everyone on the site.
4. DB ***replication*** will take care of distributing data from Master machine to Slaves machines. This can be ***synchronous or asynchronous*** depending upon the system's need.

* Master Node → write operations
→ it has latest info

* Slave / replica → Read operaⁿ
\- DB replication

* Replication :

① Asynchronous { comments/likes)

② Synchronous (Banking)

Q. Slave gets update query?

1) ignore / never allow slave to accept.

2) Allow & make way to propagate to master.

↓

M-M Model → multi primary model (PATTERN-4)

* Advantages

1) Back-up (If M fail, read can happen)
2) Scale-out of read operations
3) Availability
4) Reliability
5) Lower latency