

## Agenda

- ① Introduction to Spring Data JPA
- ② Spring Data <-> Hibernate <-> JDBC =
- ③ Repository Pattern

④ Setting up

⑤ OOIN

⑥ Inheritance and Spring Data

→ Mapped Superclass

→ Single Table

→ Table Per Class

→ Joined Table

⑦ Creating entities

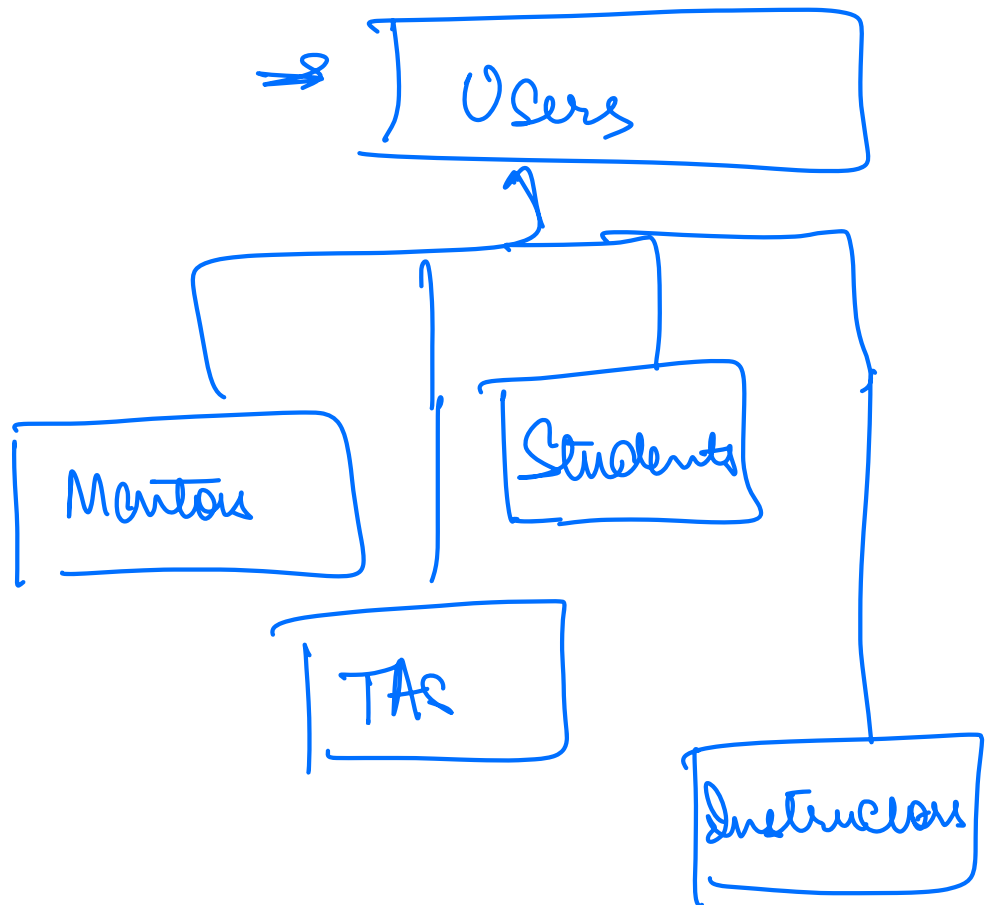
Will Start  
at  
9:10 PM

## Project Module and DB

① → Create tables

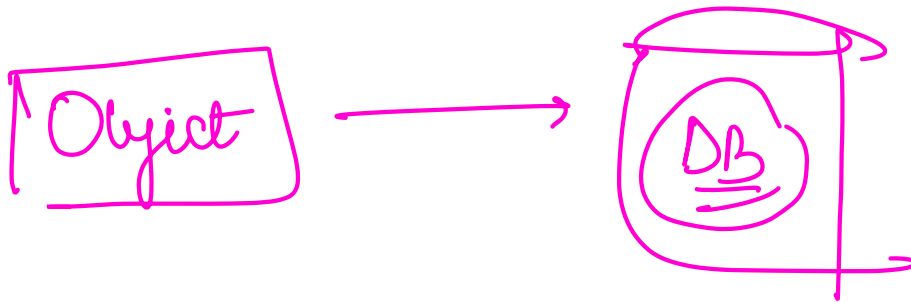
② → writing queries

③ → adv concepts (Schema migration)



# Introduction to Spring Data JPA

OOP  $\Rightarrow$  objects  $\Rightarrow$  persist



- ① SQL Query
- ② Connect to DB
- ③ Run the query
- ④ I get the rows.
- ⑤ Parse the rows. convert them to obj
- ⑥ Work on them

"  
name: Naman  
Age: 20  
"

OAM

libraries



Object Relational Mappers

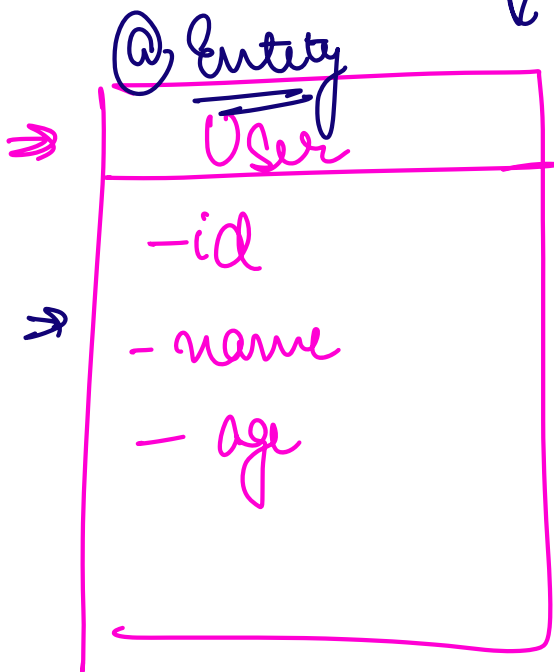
Provide you an easy to work way with DB and converting data into classes and obj.

→ write queries

→ convert results to queries

→ join.

methods / fn that make working with DB easier for an OOP lang

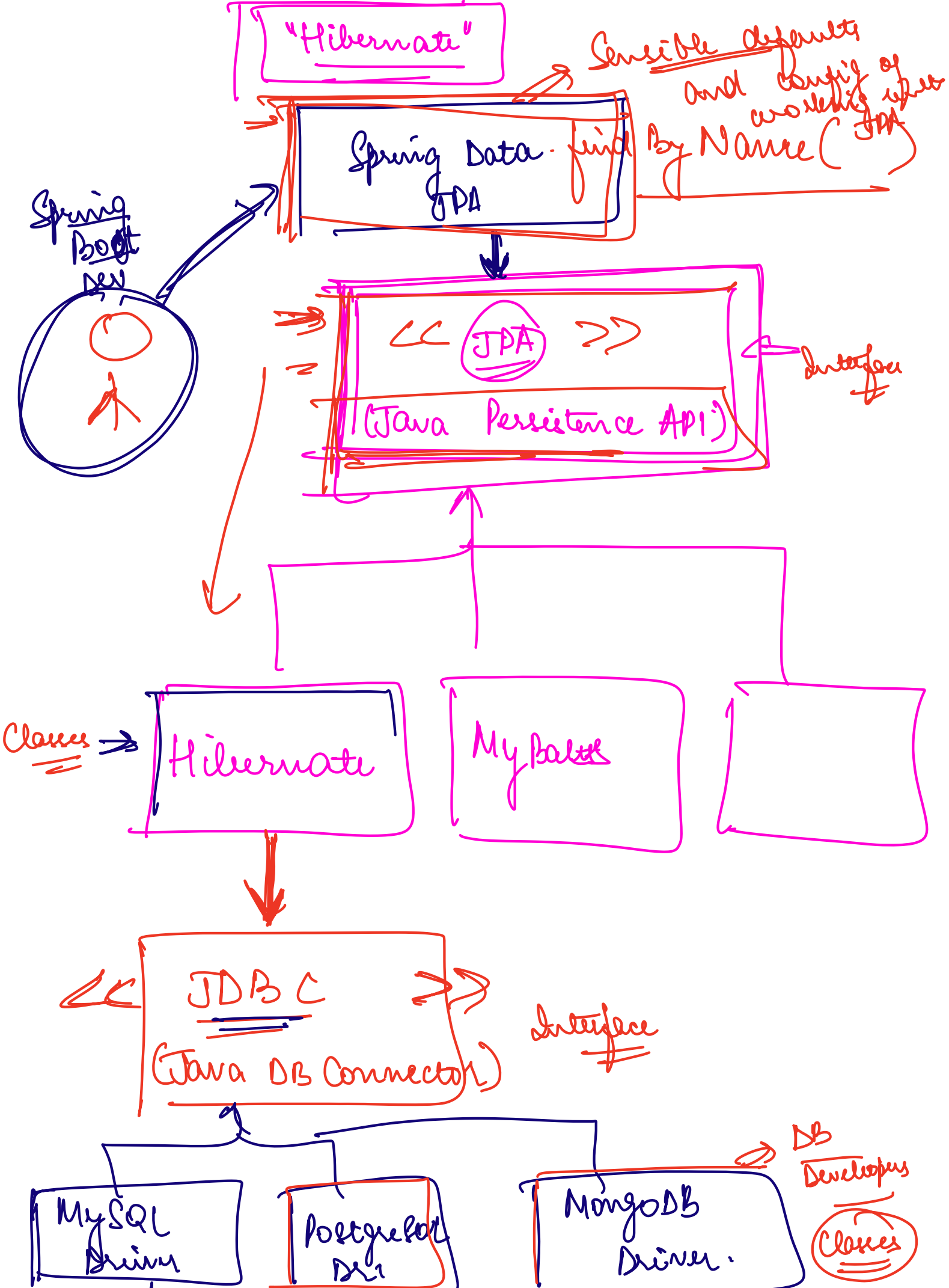


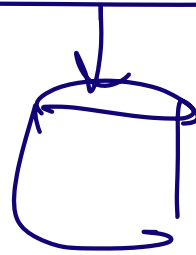
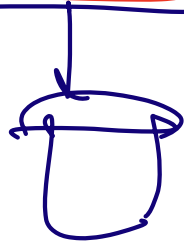
Users

id	name	age

Select\* from users where name = { }

User Repository . findByName (name);





JDBC jobbe = new JDBCconn(MySql)

find by Id (long id) {

jobbe.execute Query ( )

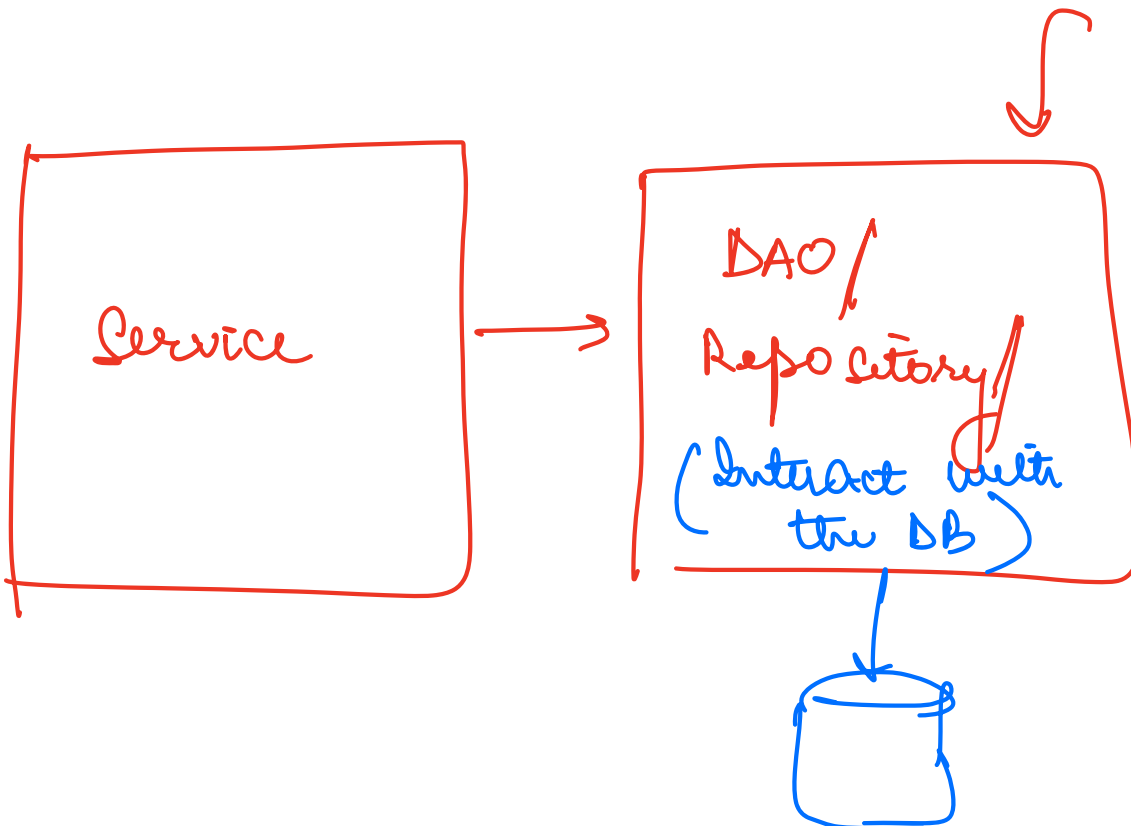
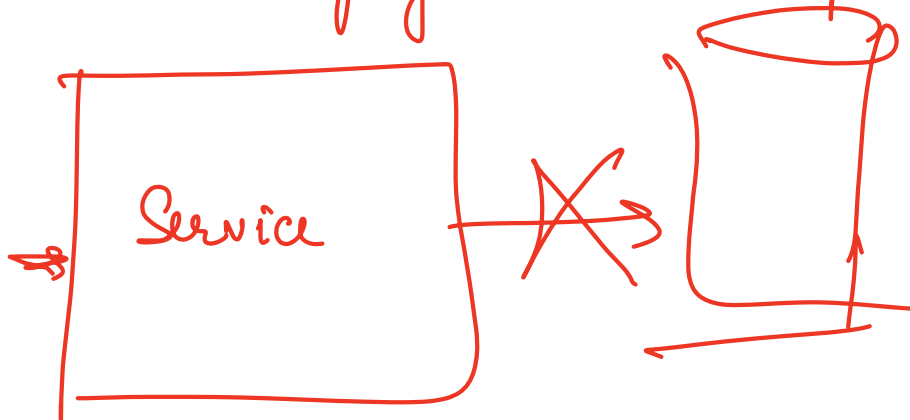
}

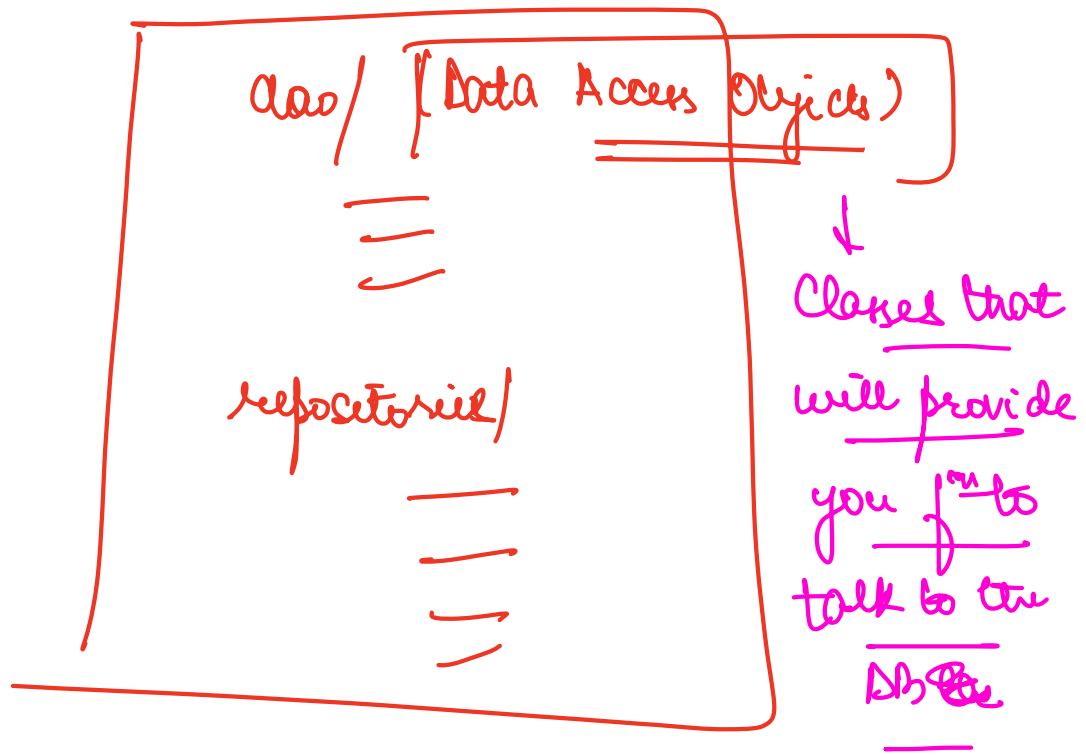
# 'Domain Driven Design'

## ↳ Repositories

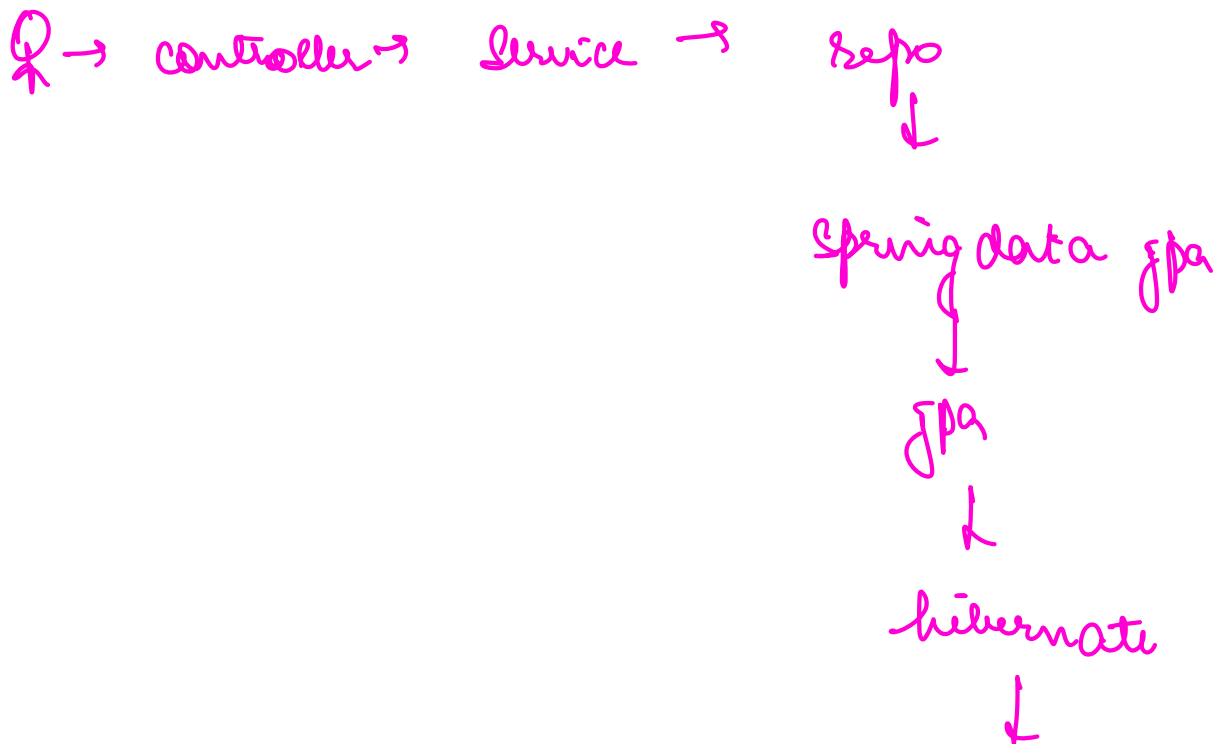
Repository Pattern: → Code to interact with persistence layer should be separate from the business logic  
→ Segregate the responsibility.

DAO






DAO  $\hat{=}$  Repositories  $\neq$  DPO



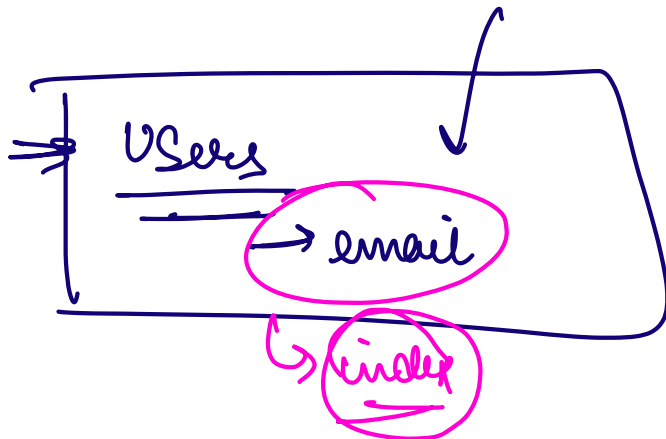


jabbe  
↓  
my eye → 

Break till 10:24 PM

VOID

~~DB~~



→ String pk are almost  
always a terrible idea

Integer/Long primary key

auto increment integral primary key

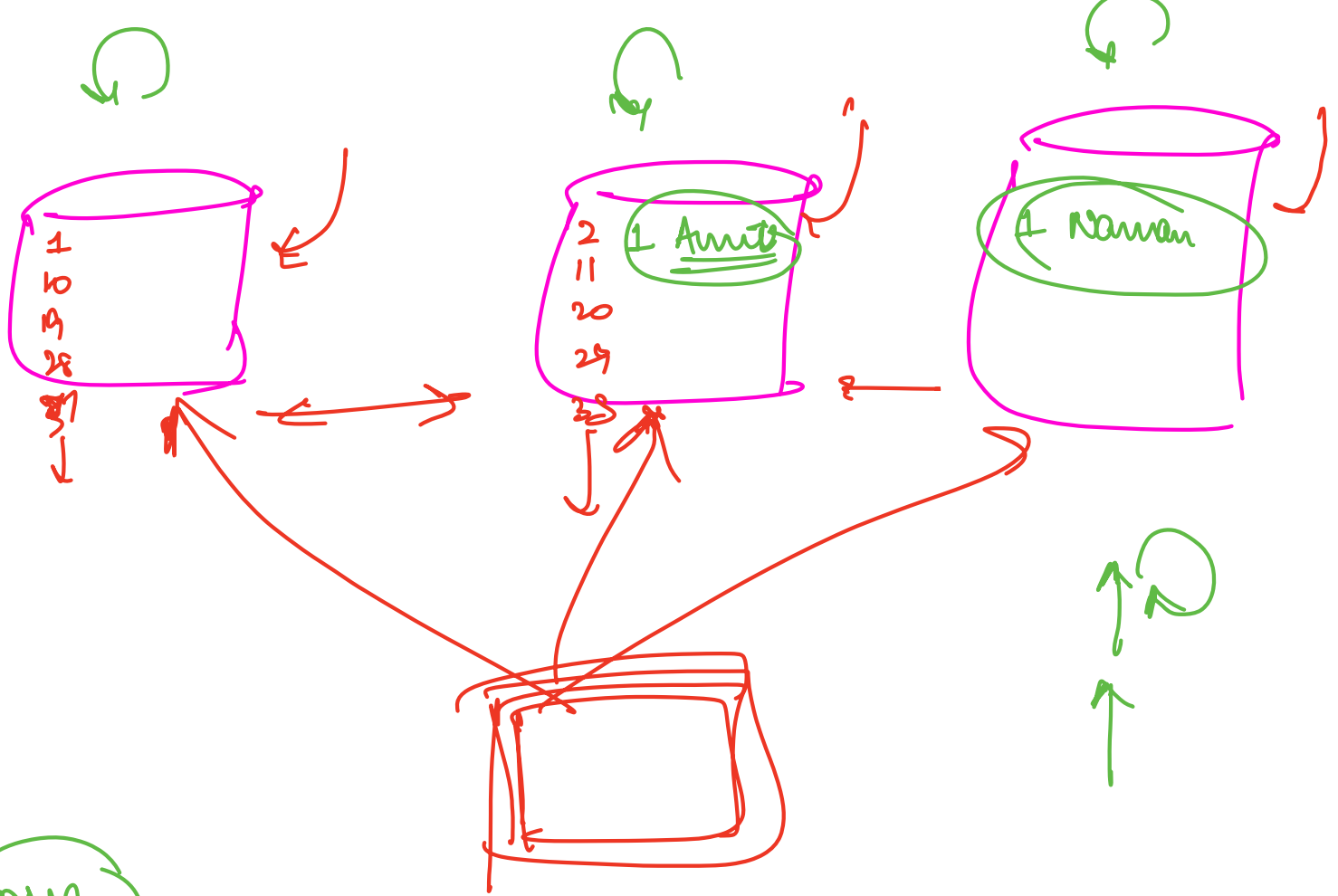
⇒ 1

⇒ 2

⇒ 3



100000



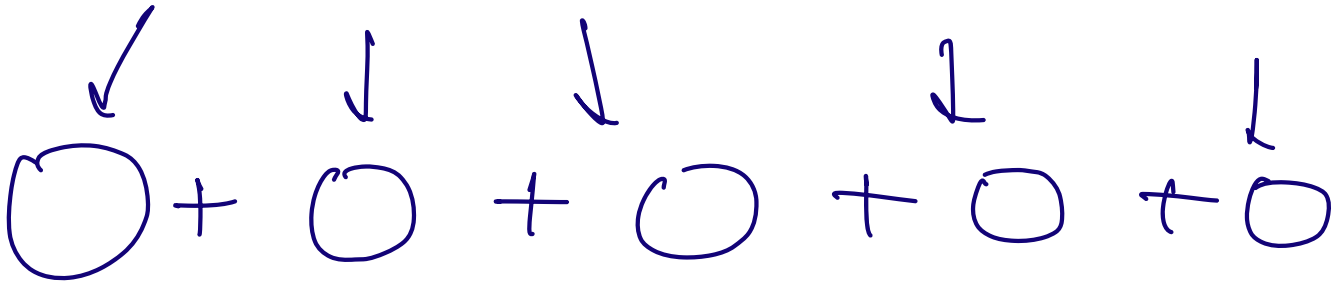
## CONS

- ① If DB is distributed, how to ensure diff DBs across diff shards.
- ② Only be able to store in a sequential manner



$$\left( \underline{\text{currentTimestamp}} + \underline{\text{ipAddr}} + \underline{\text{internetSpeed}} + \underline{\text{Ramt}} \right)$$

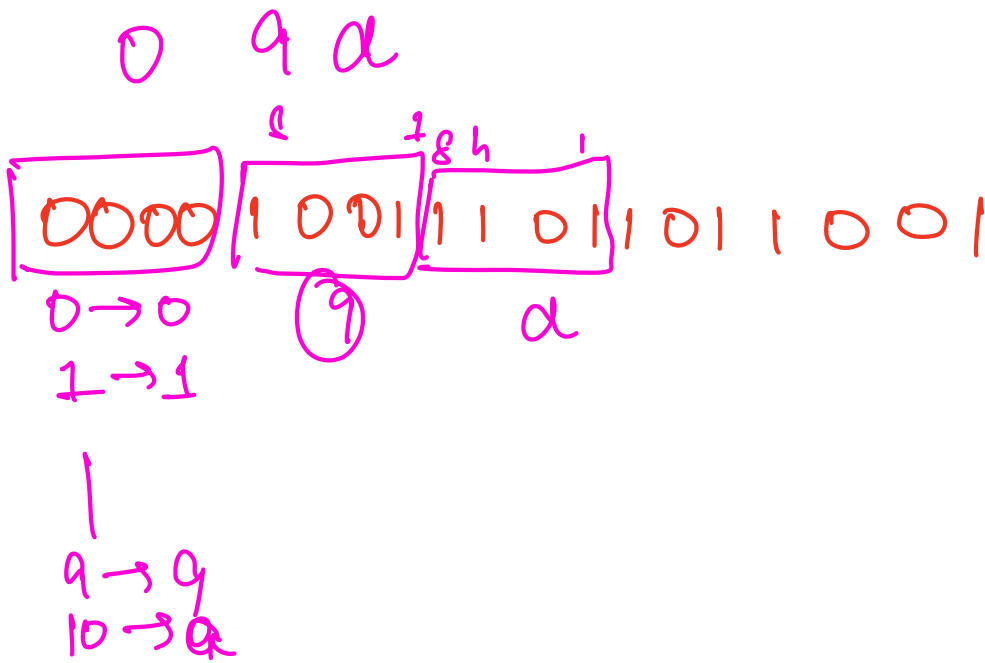
↑  
HDD + Ram + ...



HLD → Snowflake ID gen Algo

commit ID = commit timestamp + ...

UUID → universally unique identifier.



11 →  $\begin{matrix} b \\ c \\ a \end{matrix}$   
15 → f

Varchar

✓ | binary

$$\frac{128}{4} \Rightarrow \underline{\underline{32 \text{ chars}}} \Rightarrow \underline{\underline{32B}}$$

↑  
If UUID rep as  
str it will

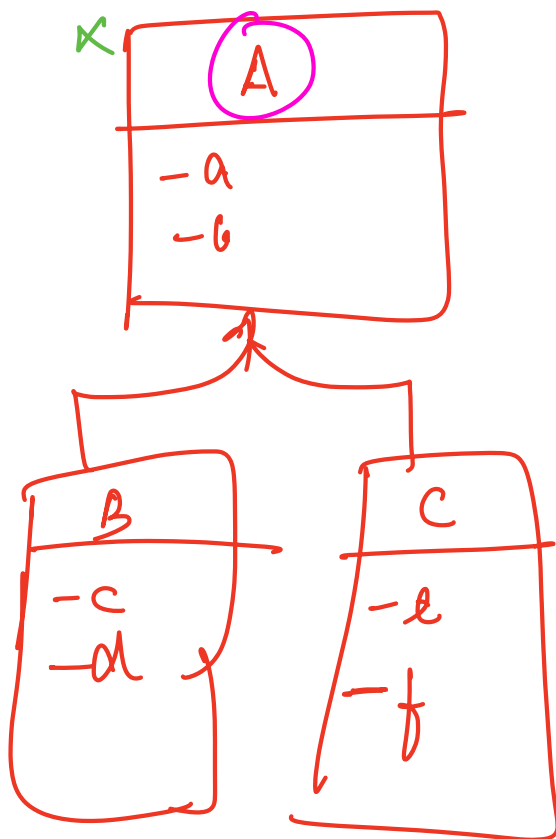
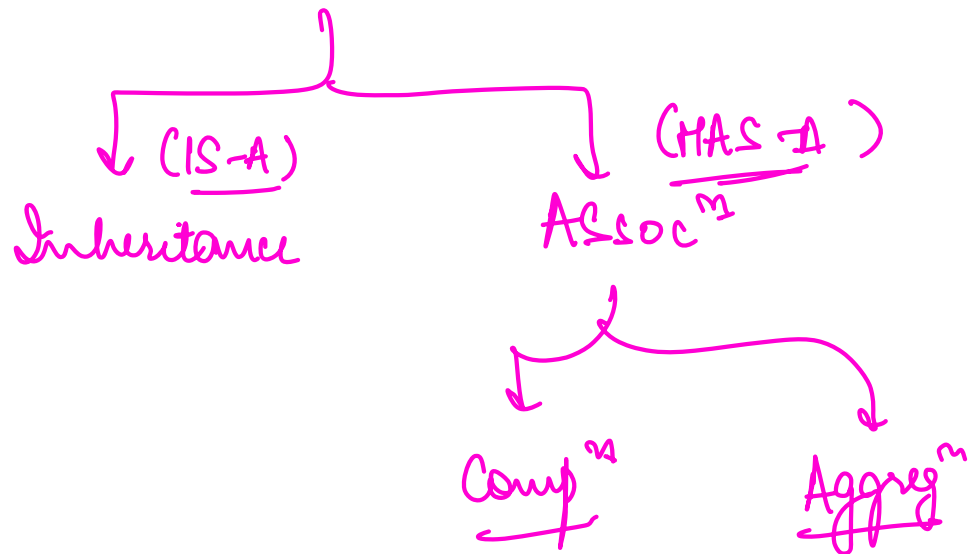
take 32B

$$\checkmark | 8 \Rightarrow \frac{128}{8} \Rightarrow \underline{\underline{16B}} \xrightarrow{\text{PK Sizer}} \underline{\underline{32B}}$$

H/W

→ UUID ✓4

→ UUID ✓7



## ① Mapped SuperClass

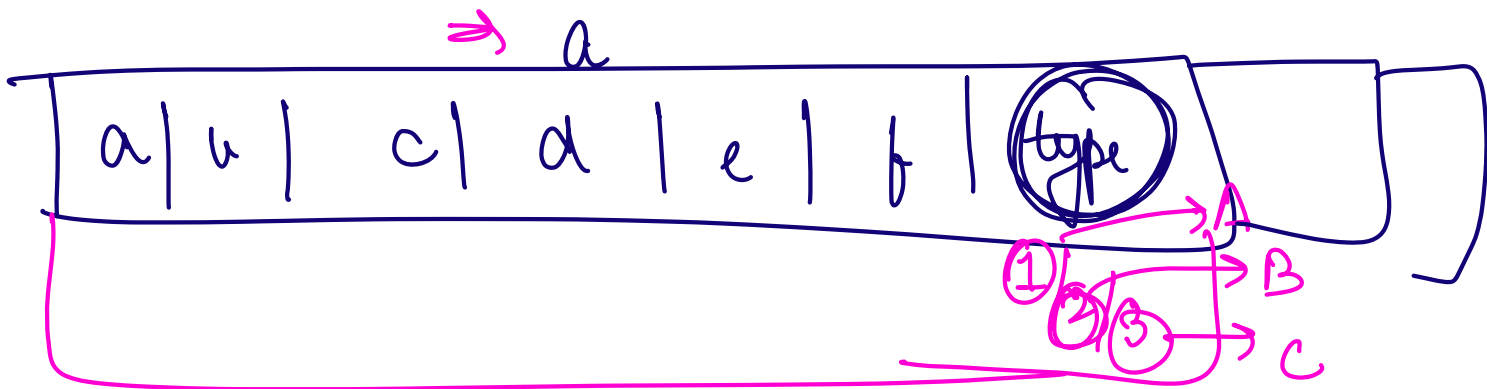
- { → No table for parent class
- { → all child classes will have a table
- { → Child tables will have attrs of parents

b			
a	b	c	d

c			
a	b	e	f

## ② Single Table

- Only one table
- Have col<sup>m</sup> of all of the children
- One new col<sup>m</sup> to identify type.  
(differentiator)



→ Very Bad Idea

→ Exists

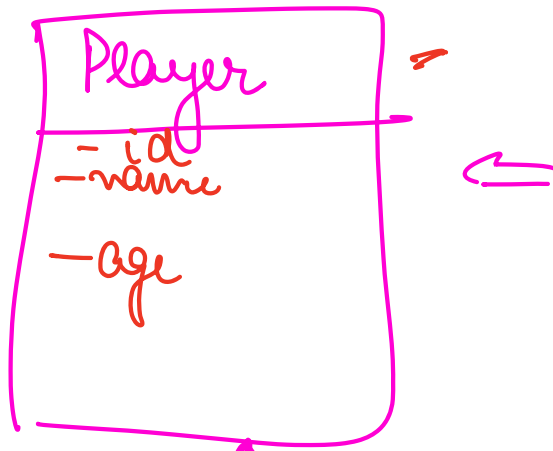
H/W

① Create a new SB

Project

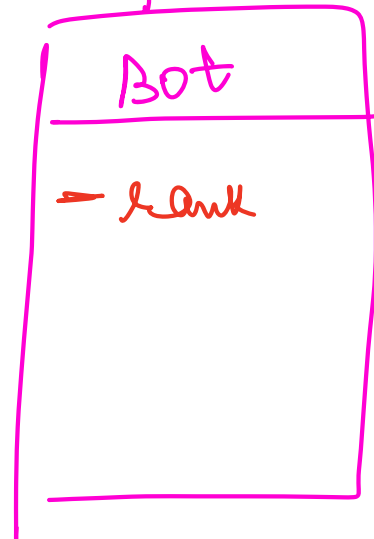


②



players

<u>id</u>	name	age



lots

<u>player-id</u>	rank

ST

players →

<u>id</u>	name	age	rank
-----------	------	-----	------

MS

⇒

<u>lots</u>	id	name	age	rank
-------------	----	------	-----	------