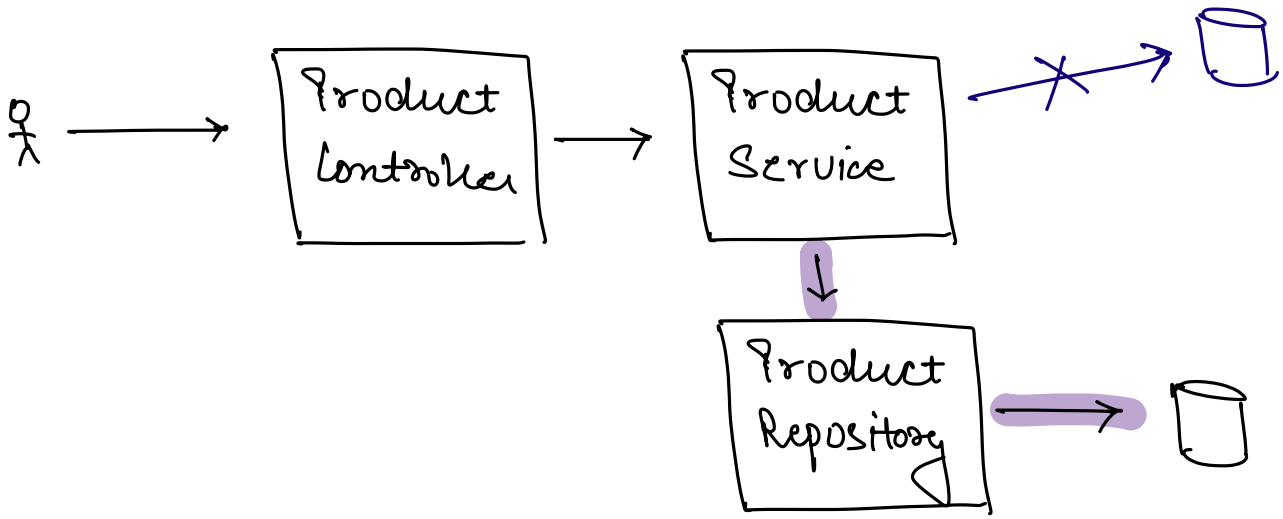
Agenda.

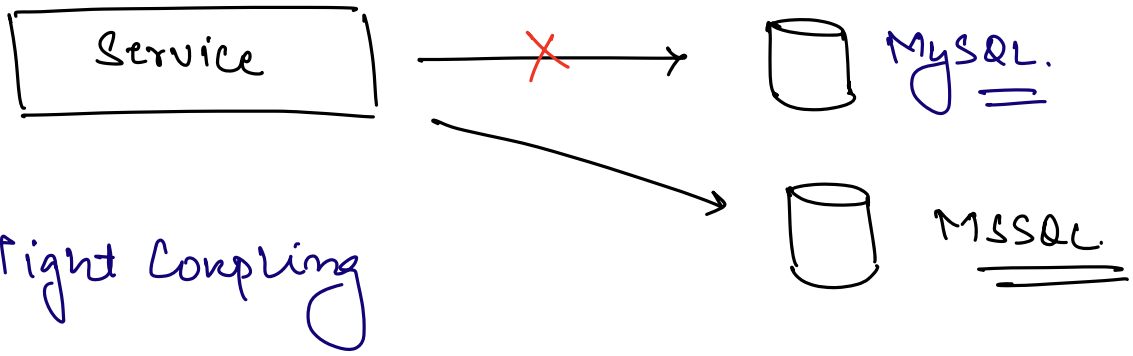
- → Repository Pattern
- → UUIDs
- → Representing Inheritance in DB.
- → CRUD operations on DB.

REPOSITORY PATTERN.

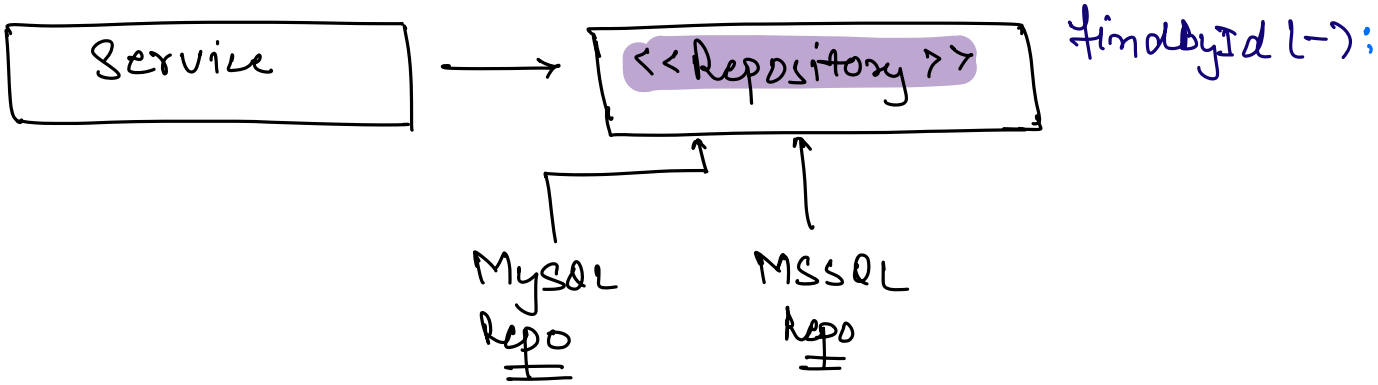


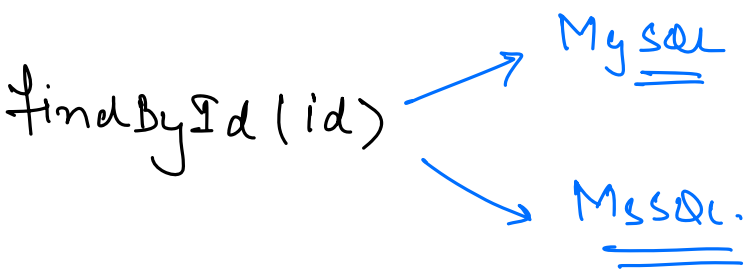
⇒ Code to interact with persistence layer (⇒ DB.) should be separate from Business logic.

⇒ We should a separate repository layer to interact with DB.

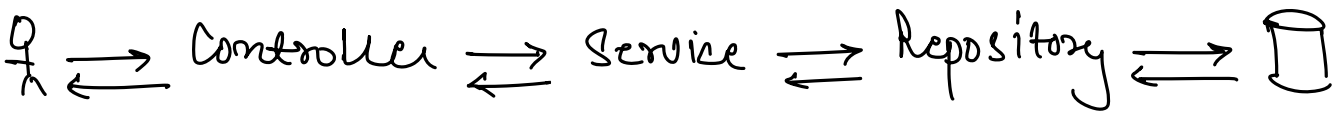


Tight Coupling



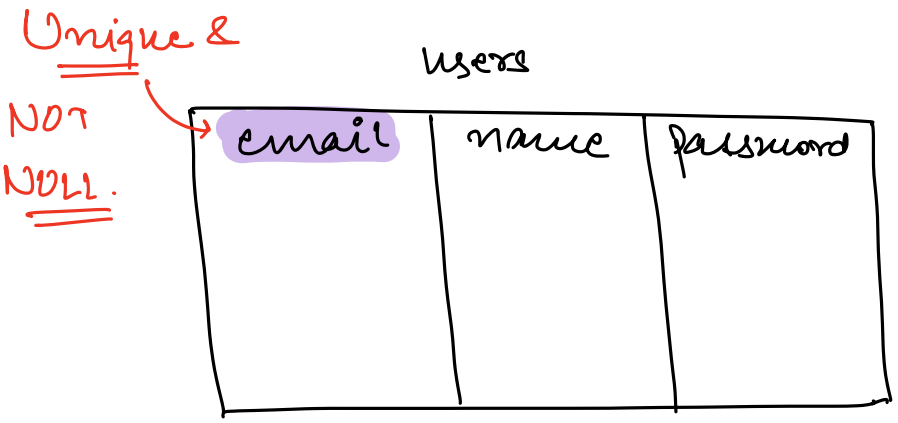


⇒ Loose Coupling



⇒ UUID : Universally Unique IDs.

Every table needs to have a primary key attribute to uniquely identify an entity.



We can consider email as a PK. ✓

Should we consider email as a PK ? ✗

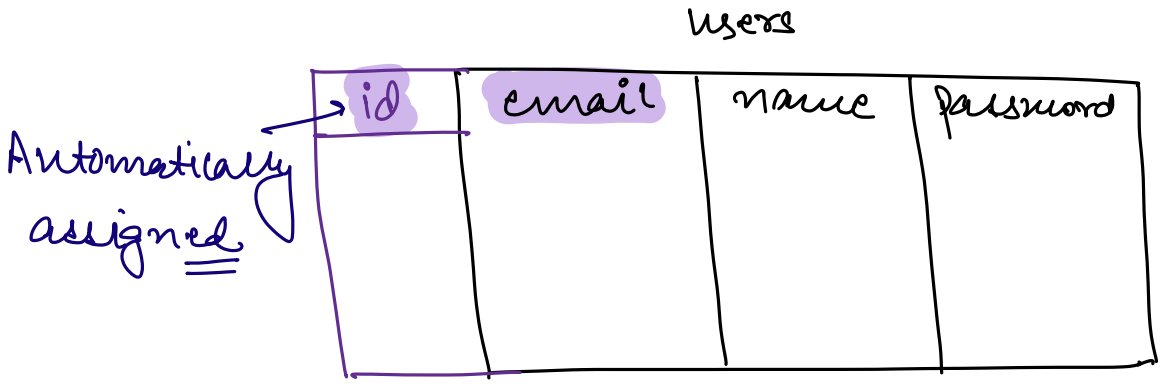
↳ NO.

I) String comparison is slower.

II) Email is a user attribute, and it can be changed.

III) Size of Index table would increase.

⇒ That's why user defined attributes shouldn't be considered as a PK column.



⇒ Datatype of Id :

Int : 4Bytes.

↳ Range : 2^{32}

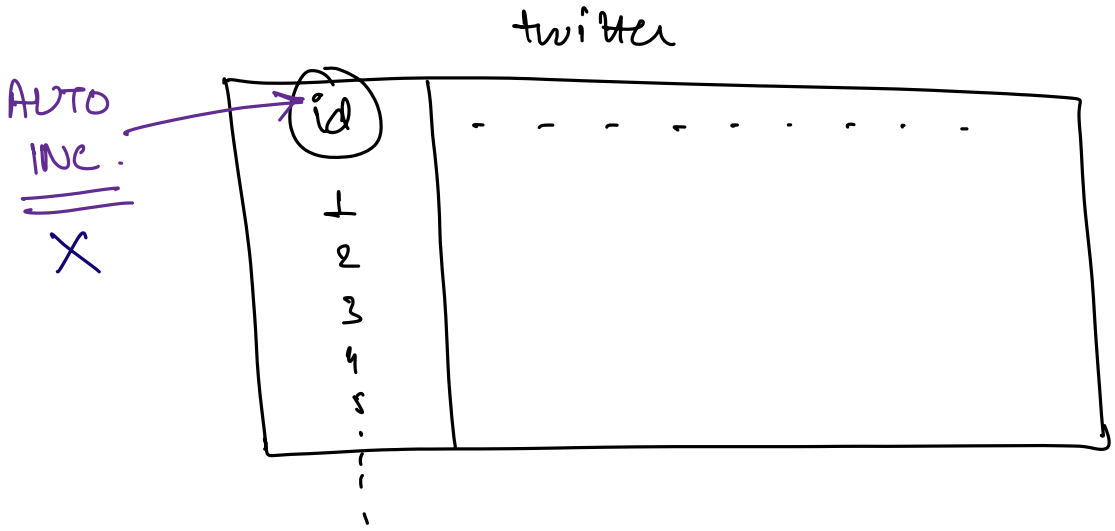
≈ 2B = 2×10^9.

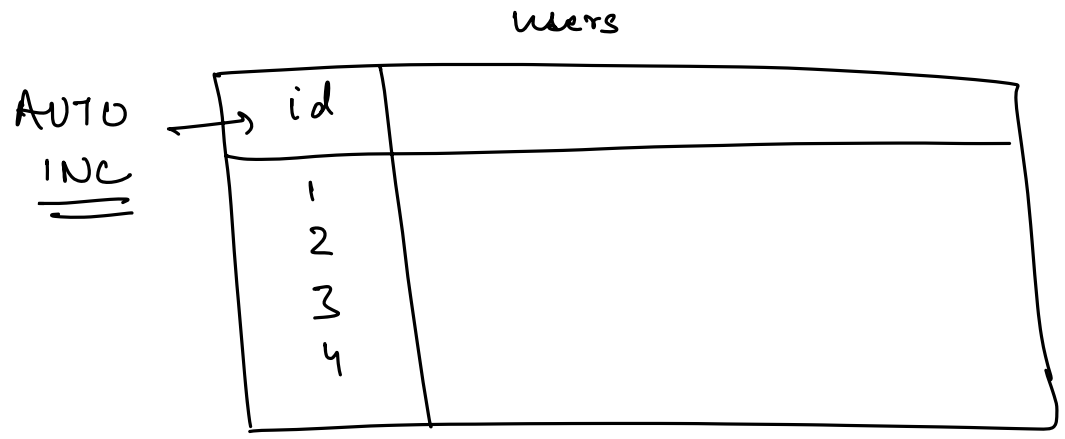
BigInt | long.

↳ 8B.

↳ 2^{64} ≈ 10^{18}

⇒ AUTO INCREMENT.

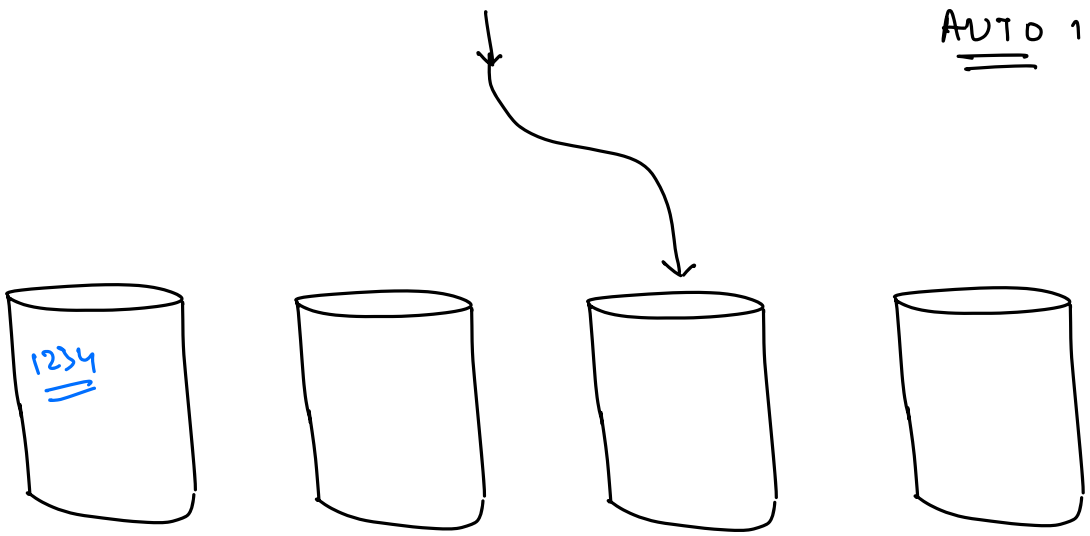




Scaler.~~com~~ | users | 1234 ~~5~~

1235

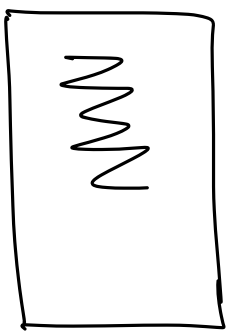
AUTO INC



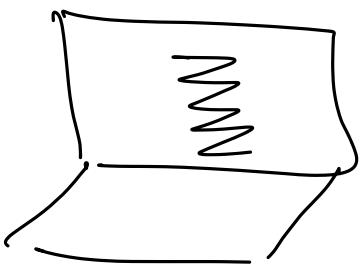
⇒ AUTO INCREMENT Ids won't work for distributed DB's.

Randomly generated unique Id, So that no one will be able to predict the Id of the next entry.

user_id + timestamp + m/c Id + Device info.



Mobile



Laptop.

Combination of multiple such parameters generates a Unique Id.

⇒ UUID : 128 bit Number

Range = 2^{128}

↓

> No. of atoms in the World.

UUID = fun (mac Id + timestamp + user_id + device_info)

= 10111100100000111 - - - - -

128 bits.

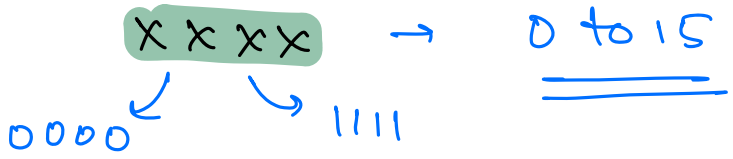
Binary.

⇒ Hexadecimal.

↳ Base 16.

0 → 0000
1
2
3
4
5
6
7
8
9

10 → A
11 → B
12 → C
13 → D
14 → E
15 → F

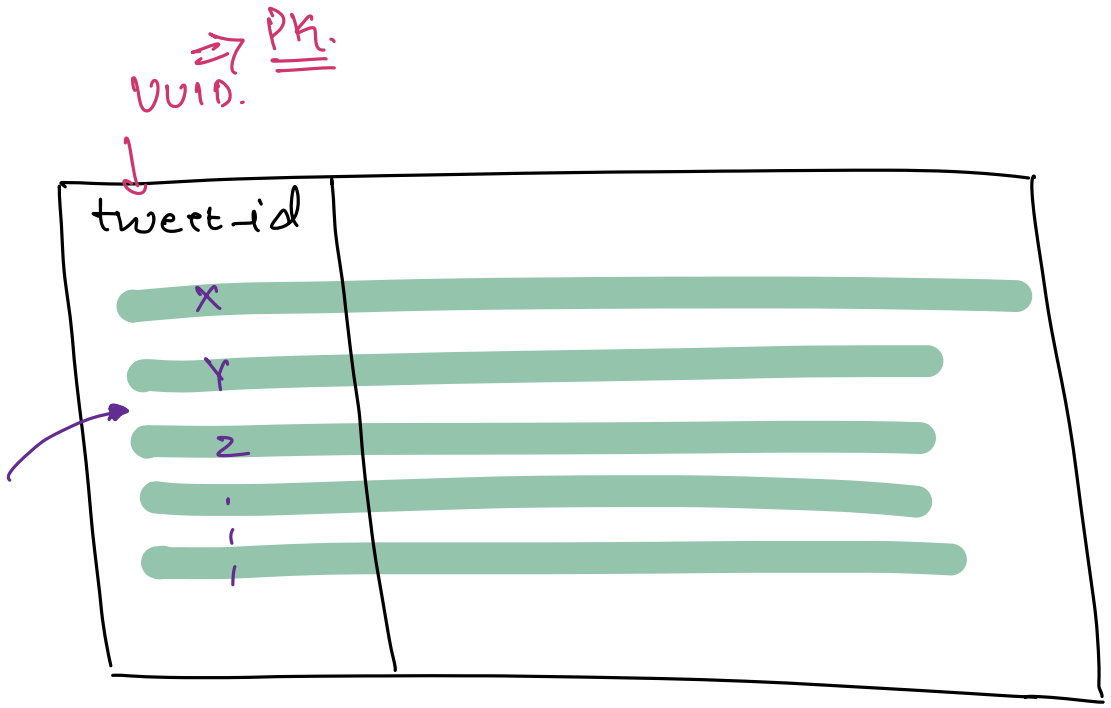


UUID : 1010 0010 1101 0010 0000 1111 - - -

A 2 D 2 0 f - - -

A2D20f - - -

$\frac{128}{4} = 32$



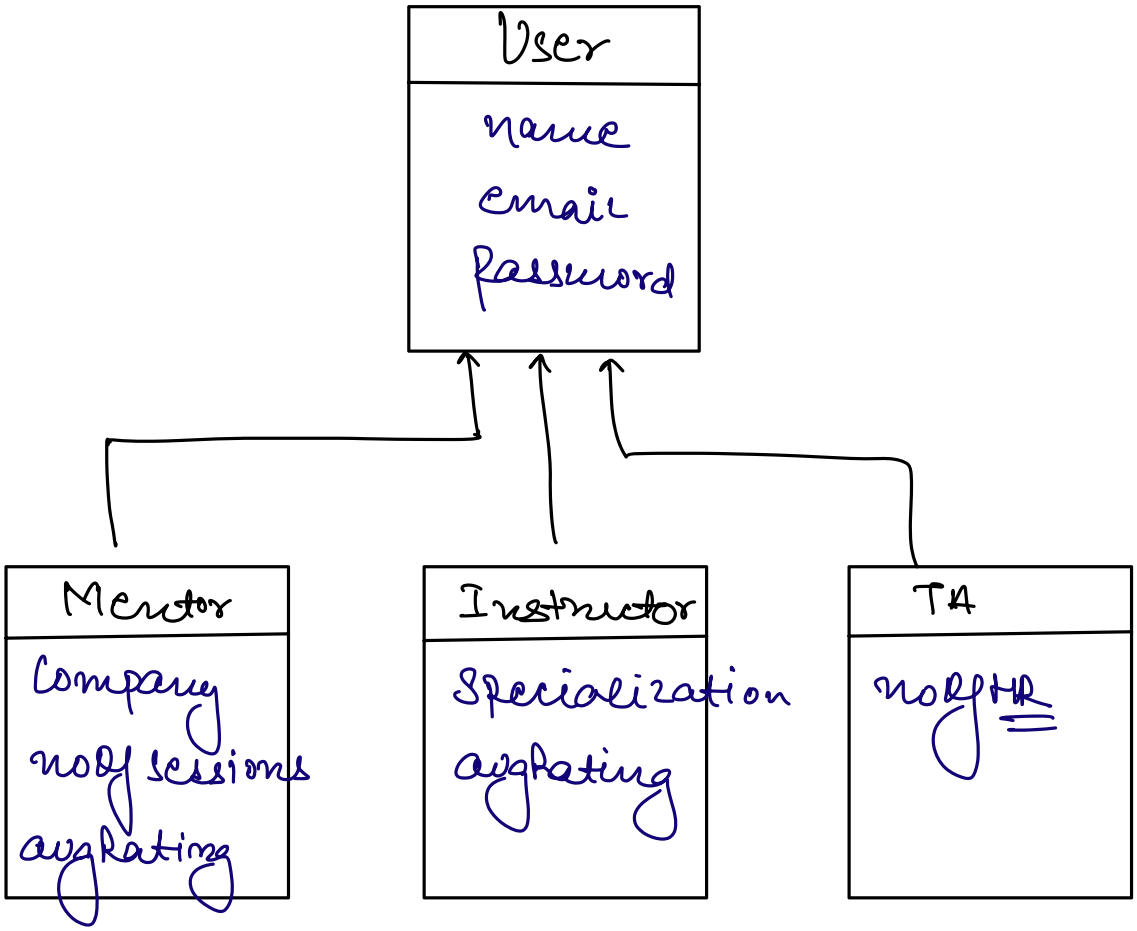
Y { newid < 2

⇒ Newly generated Id's are greater than the previously generated Id's.

↳ Timestamp.

⇒ UUID V7.

Representing Inheritance in DB.



1) Mapped Superclass.

→ When there's No object of parent class. (OR) parent class isn't a real entity.

→ No table for parent class.

→ Table for every child class with their own attributes and attributes of parent class.

mentors

name	email	password	company	no of sessions	avg rating

tas

name	email	password	no of HR.

instructors

name	email	password	Specialization	avg rating

Q. Get the email of all the users.

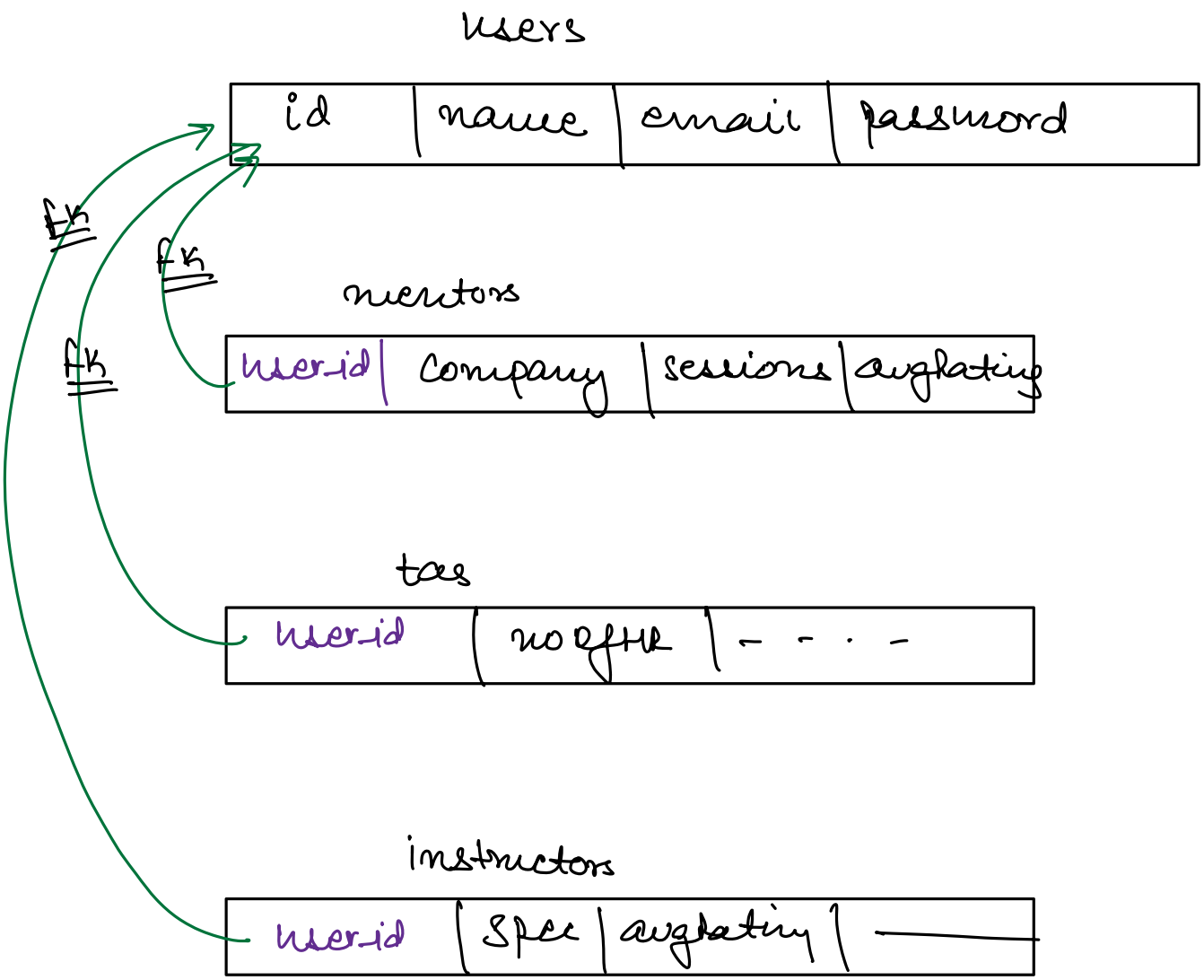
```
Select email from mentors
∪
Select email from instructors
∪
Select email from tas.
```

2) JOINED TABLE. ⇒ Most widely used

⇒ Every data of objects of parent class will be present in parent class table.

⇒ Table for each child with their own attributes.

⇒ Attrs of parent class will be retrieved from parent class table via foreign key.



Q. Give email of all the Instructors.

⇒ JOIN.

Q. Get the email of all the users.

```
select * from users.
```

3) Table Per Class.

⇒ Exactly same as Mapped Super class, only difference is here we create table for parent class as well.

⇒ Table for every child class will also be there with all the attrs.

users

id	name	email	password

↓ Only users data.

mentors

name	email	password	company	noOfSessions	avgRating

tas

name	email	password	no of HR.

⇒ REDUNDANCY.

instructors

name	email	password	Specializ^{n}	avgRating

Q. Give email of all the Instructors.

⇒ Select email from Instructors;

4) Single Table

↓

name	email	password	profession	avgrating	Special	noOfHR	type
—	—	—	NULL	4.8	HLD	NULL	

⇒ Too many NULLs

⇒ Sparse table.

⇒ Generally we have a type attribute to identify the type of user.