Agenda

1. UUID's
2. Inheritance in DB.

Break

8:00:00

8:10:00

8:11:00

UUID : Universally unique identifier

users

email	pass	phNo	name

Can email be PK => Yes

is it a good idea => NO.

Issues

1. Email id can change
2. Datatype of email id is String.

Sol": have a separate id column.

1. integer => 4 Bytes, 2^{32} = 2 Billion.

Sometimes not enough.

2. long => 8 Bytes, $2^{64} = 10^{18}$, mostly fine.

Problem 1: What if 'id' needs to be exposed in url.

ex:

flipkart.com /search / products/1 ; GET

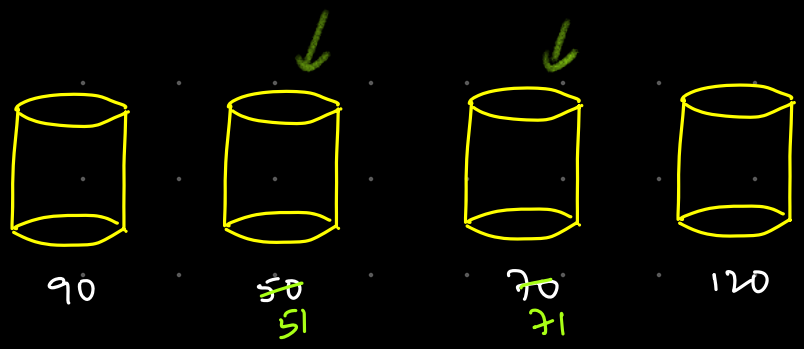
flipkart.com /search / products/2 ; GET

flipkart.com /search / products/3 ; GET

Data violation problem.

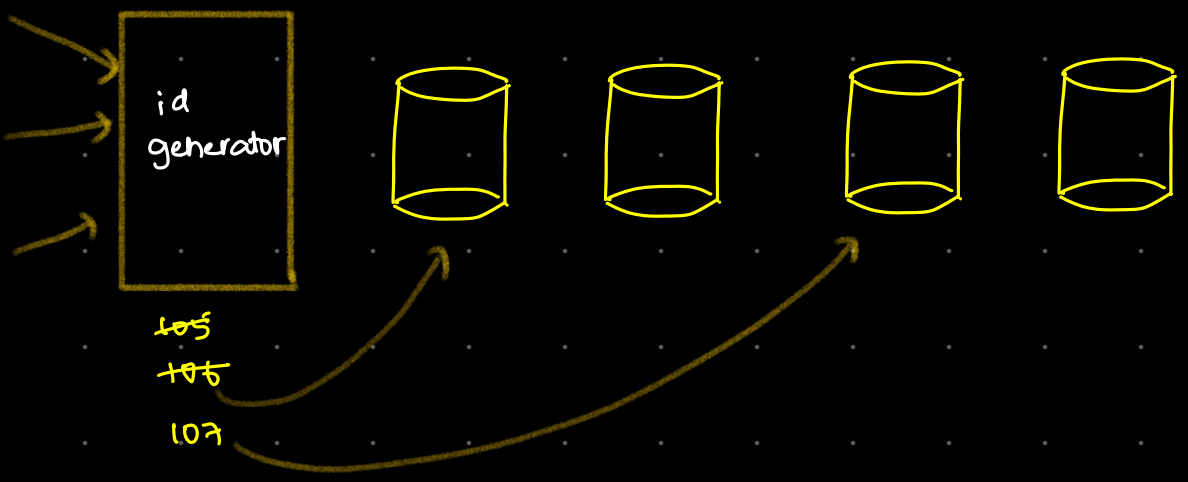
' If your id is not publicly visible, then it's still fine'.

Problem 2: What if we've distributed DB.



There can be a situation, where same id can be used.

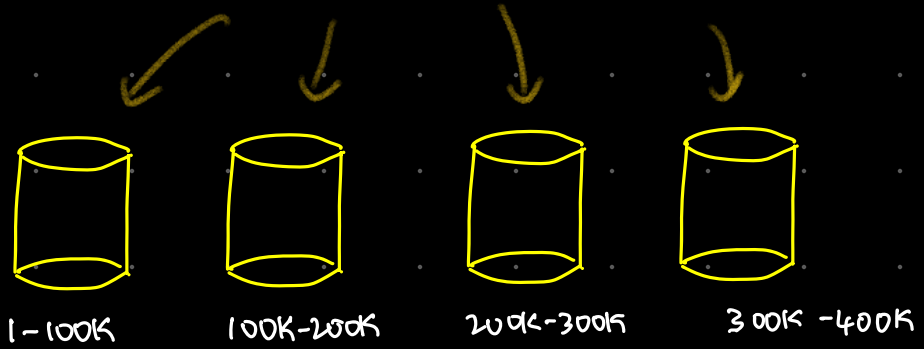
One possible sol^n:



Issues

1. Single point of failure
2. This will make things slow.

Better sol^n



"This will still make the id guessable".

For the id's, along with auto generated value,

machine-id + curr_time + user-id + ip + auto_inc

23.43.12.16 + 123456 + 1234 + 2.3.4 + 101

No conflicts can happen, also no one can guess my id.

This will be very big number.

UUID : 128 bit number

: not string; its a array representation of a number.

: created as a fn of multiple parameters.

128 bit

club 4 bits into 1 group. [0 - 15]

Total group = 128/4 = 32 bits

Max value → [15]

Min value → [0]

1 1 1 1 → 15

0 0 0 0 → 0

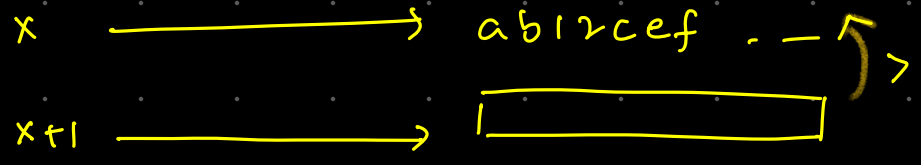
Hexadecimal number system.

7b50b2fe-c5b5-4649-801a-6d0ae99744b7

0111 1011 0101 - - - - -

128 bits.

Starting from UUID-V7, the UUID's generated later will always be bigger than the number generated before this.



email

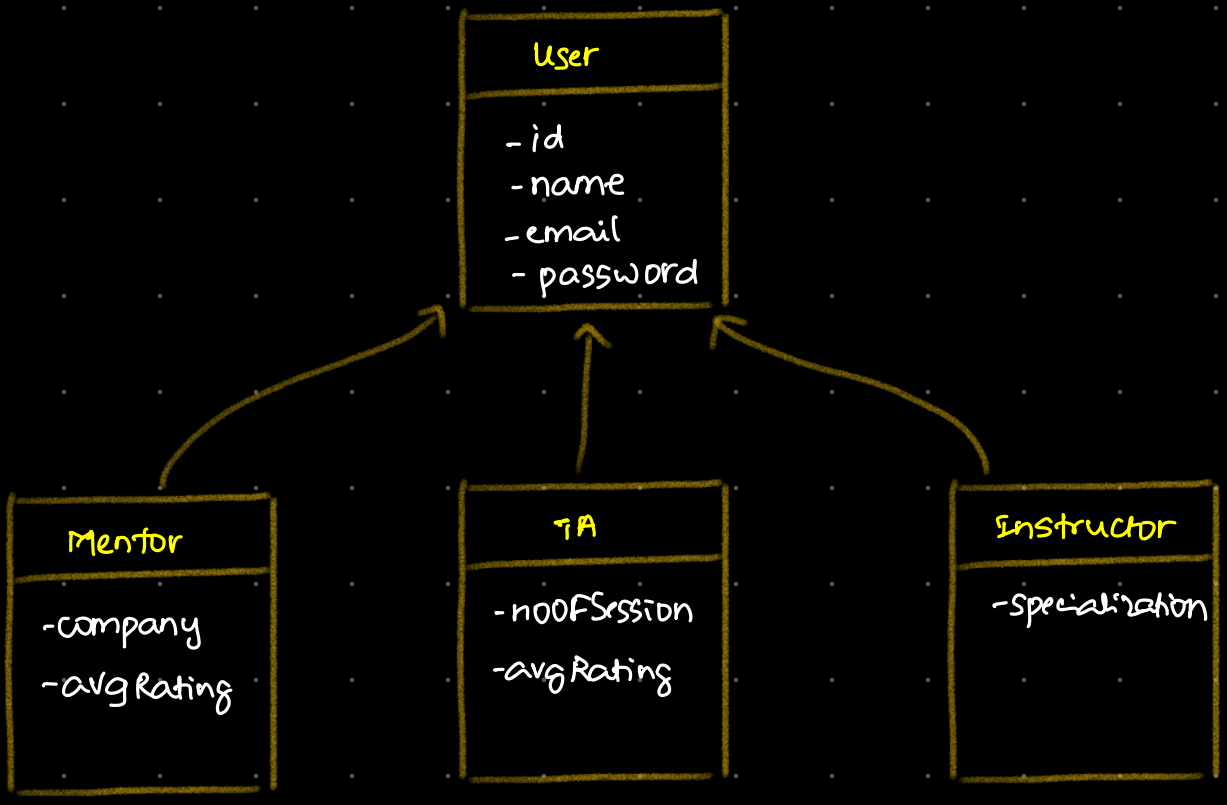
xyz@gmail.com
⇒
Renu@gmail.com

⇒ With this, there's a chance of restructuring entire tree.
& comparision of email ids are needed.

Ways to represent inheritance

- Mapped Superclass
- Joined Tables
- Table per class
- Single class

How to represent these classes in DB? --



1. Mapped Super Class. [No object of parent class].

Tables

Mentor

id	email	name	password	avgRating	company

TA

id	email	name	password	avgRating	noOfSessions

Instructor

id	email	name	password	specialization

Q: find all the users whose name starts with 'a'.

→ union on all 3 tables.

2. Joined Table

1. All the attributes of parent class will be in parent table
2. Each child table will have a foreign key reference to the parent table.

Tables

Mentor

user_id	avgRating	company

TA

user_id	avgRating	noOfSessions

Instructor

user_id	specialization

User

id	email	name	password

Q: Get email of every instructor.

Join on instructor and user table.

```
@Inheritance(strategy = InheritanceType.JOINED)
@Entity(name="jt-user")
class User
```

```
@PrimaryKeyJoinColumn(name="user_id")
@Entity(name="jt-mentor")
class Mentor extends User
```

Table Per class

Tables

Mentor

id	email	name	password	avgRating	company

TA

id	email	name	password	avgRating	noOfSessions

Instructor

id	email	name	password	specialization

User

id	email	name	password

Here, "mentors, tas, instructors" Data won't be present in "user" table.

The users who're not mentors, tas, instructors will be present in user table.

```
@Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
@Entity(name = "jt-user")
class User
```

```
@Entity(name = "jt-mentor")
class Mentor extends User
```

4. Single table

Create one table and keep the attributes of all parent and child classes as cols.

id	email	name	password	avgRating	company	avgRatingTA	noOfSessions	specialization	Type
1	K@gm	Keerti	abcd	4.8	Adobe	—	—	LLD	1 (Inst)
2	M@gm	Manoj	cdef	—	—	4.9	10	SQL	2 (Mentor)

```
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(
        name = "user_type",
        discriminatorType = DiscriminatorType.INTEGER)
@DiscriminatorValue("0")
@Entity(name = "jt-user")
class User

@DiscriminatorValue("1")
@Entity
class Mentor extends User

@DiscriminatorValue("2")
@Entity
class instructor extends User
```

H/W : Please demo all of this!. --