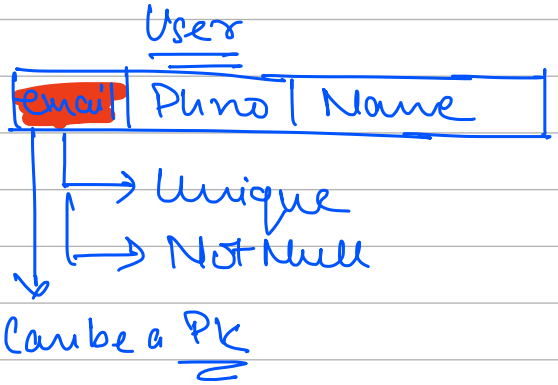
Agenda

1) UUIDs
2) Representing Inheritance in DB
   - Connect to DB
   - Create Tables

UUID Universally Unique ID

→ Every table in the database should have primary key

→ Primary key is required to be uniquely identifiable record in DB



① Email is user attribute & it can be changed
② String Attr
   - String comparisons are costly
③ By default the index gets created on PK
   - More Space
   - Write operations will be costly

→ An extra id column as PK

1) int
   - 4B — 32 bits
   - $2^{32} \approx 2 \times 10^9$
   - $\Rightarrow$ 2 Billion

FB = 2.9 Billion Active User

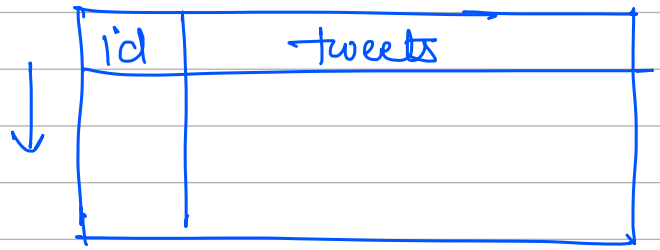
Scale

Amazon ≈ 1 Billion items
- 2 Billion events / week

2) BigInt / Long
   - 8 Bytes
   - 64 Bits
   - $\approx 10^{18}$

(Most frequently used)

Auto increment — x.com/tweets/(id)

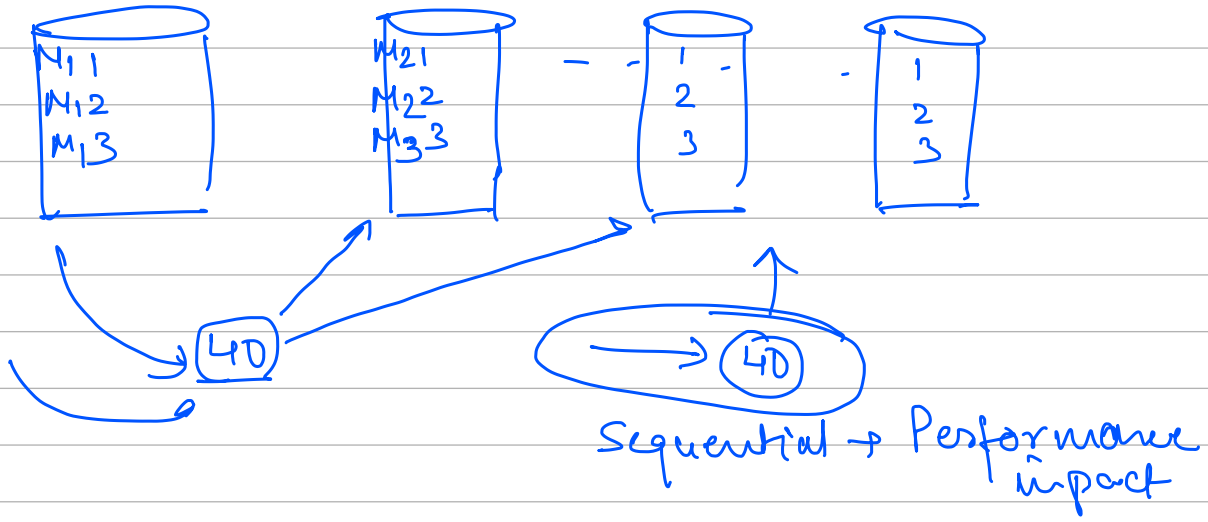


twitter, Scaler

Public API
www.scaler.com/profile/1 ②
↳ Nikhil

① If there are public Apis to fetch data auto increment is not good option

② If the database is distributed across multiple machines the auto increment doesn't work



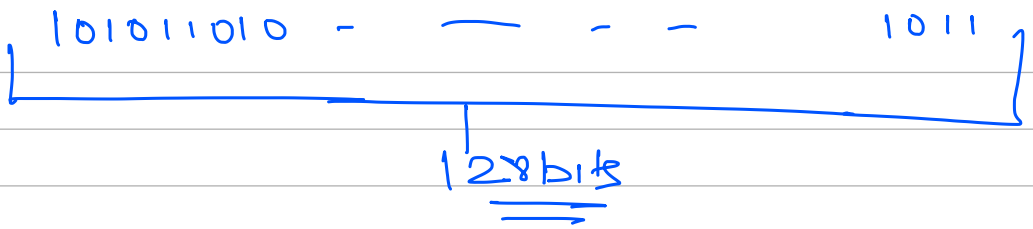
Sequential → Performance impact

Instead of Auto Increment we need some randomness.

Commit Id = func (mac id + timestamp + 7p + - - - - - - - -)

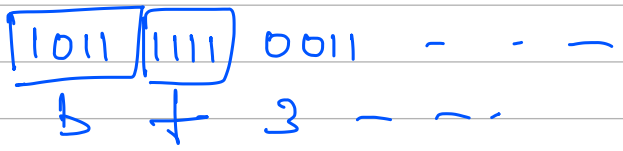
UUID = func (Machine id + user id + timestamp)

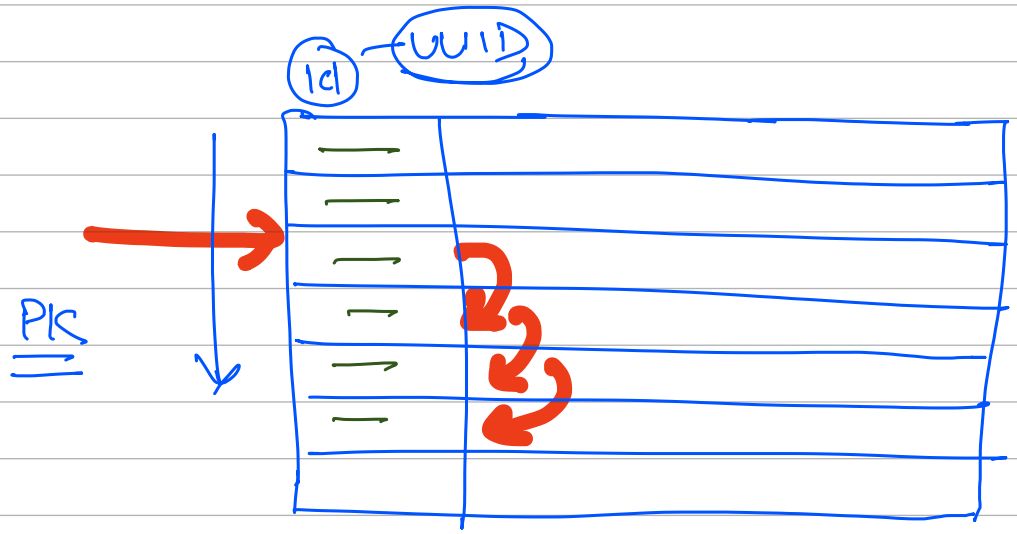
Complete Random = 128 Bits Number
≈ 2^{128} ≈ 10^{36}



Hexadecimal (16 Bits)

0000 – 0
0001 – 1
0010 – 2
0011 – 3
0100 – 4
0101 – 5
0110 – 6
0111 – 7
1000 – 8
1001 – 9
1010 – a
1011 – b
1100 – c
1101 – d
1110 – e
1111 – f

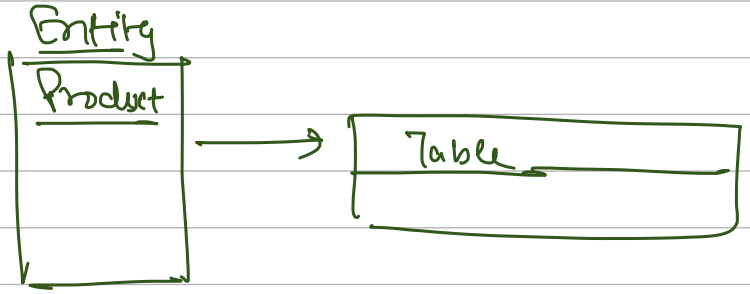




Somehow, along with randomness if UUID can maintain the new UUID will greater than previous UUID

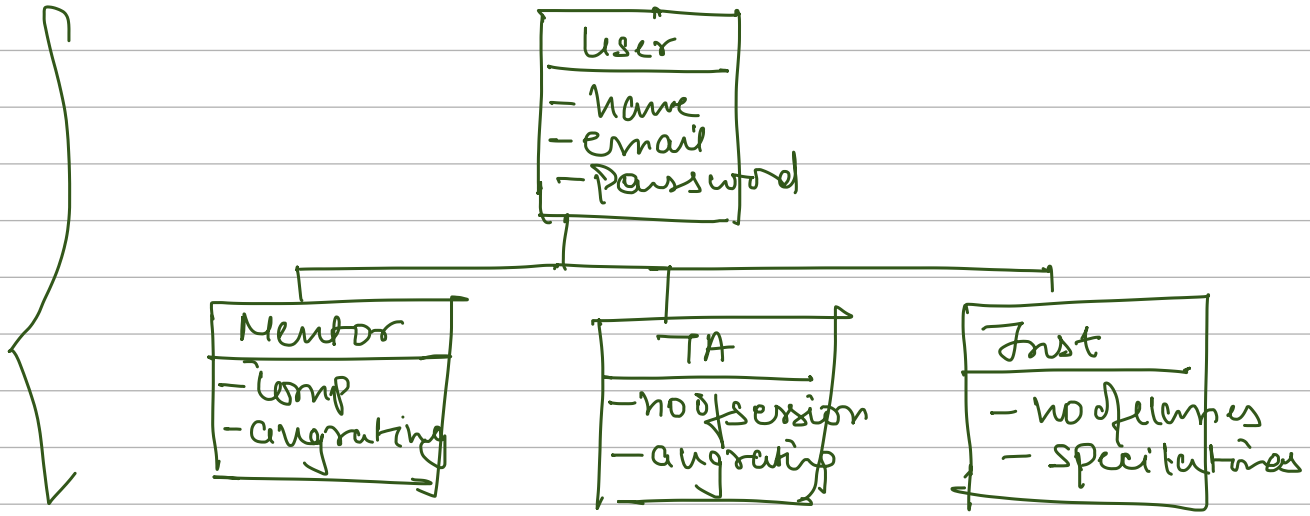
- Timestamp
  - Number of Milliseconds 01/01/1970
- UUID 7

Inheritance in DB



4 Ways to rep^n Inheritance in DB

1. Mapped Superclass
2. Joined Table
3. Table Per class
4. Singletable



→ Store the data of instructors, Ta & mentors in DB How?

① Mapped Super Class

→ When there is No Object for parent class
→ Parent class can be abstract

Approach

① No table for parent class
② One Table for every child class with all attributes of parent class

Mentor

Company Name	avgrating	Name	email	Password

TA

No of sessions	avgrating	name	email	Password

Inst

Specialization	no of class	name	email	Password

Q Get email of all users

Select email from Mentor →
UNION
Select email from TA →
UNION
Select email from Inst →

(2) Joined Tables (Best soln) 99%

→ Every data wrt objects of Parent class will store in Parent table

→ For each Sub class also we will create tables with only their attributes

→ We will get parent class attr in child class via fk

User

| id | name | email | Passwd |

Mentor

| compName | avgRating | User_id |

TA

| noOfSessions | avgRating | UserId |

Q Get email of all users

↳ Single Query

Select email from user

Q Get email of all Mentors

↳ Join Query

(3) Table Per Class

→ Exactly same as MappedSuperClass only diff. we will create table for parent Class.

→ Table for each class will have their own attributes as well as attrs from parent class

User

| name | email | Password |

Mentor

| Comp Name | avg Rating | email | name | Password |

TA

| noOfSessions | avgRating | name | email | Password |

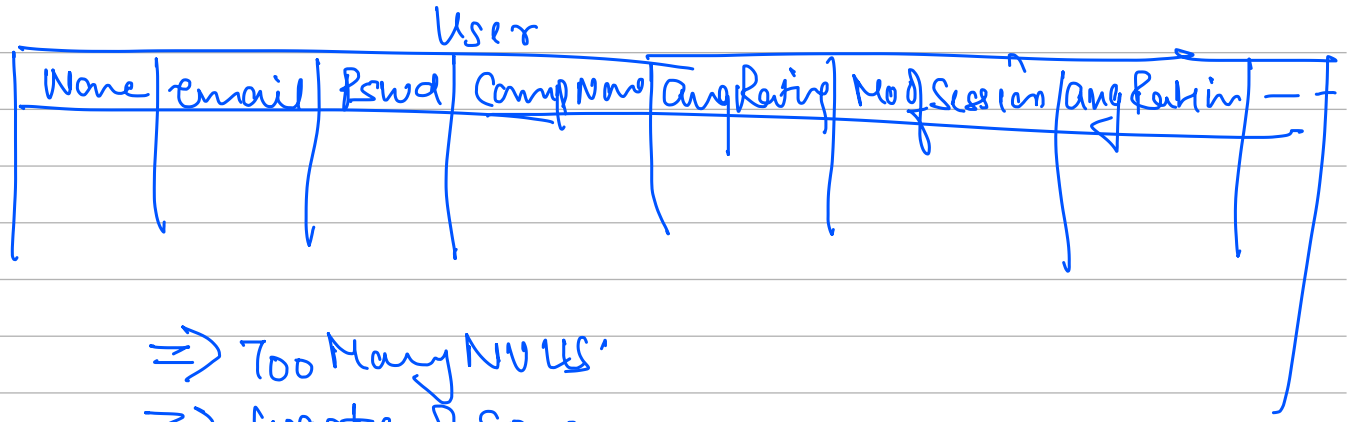
Note

① First define the Relationship in the Codebase

② Then identify the Query Pattern → Most freq. query to decide which out of 4 approach make sense

④ Single Table (Worst soln)

Create a single table with all attributes from all classes (Parent + Child)



⇒ Too Many NULLs
⇒ Waste of Space