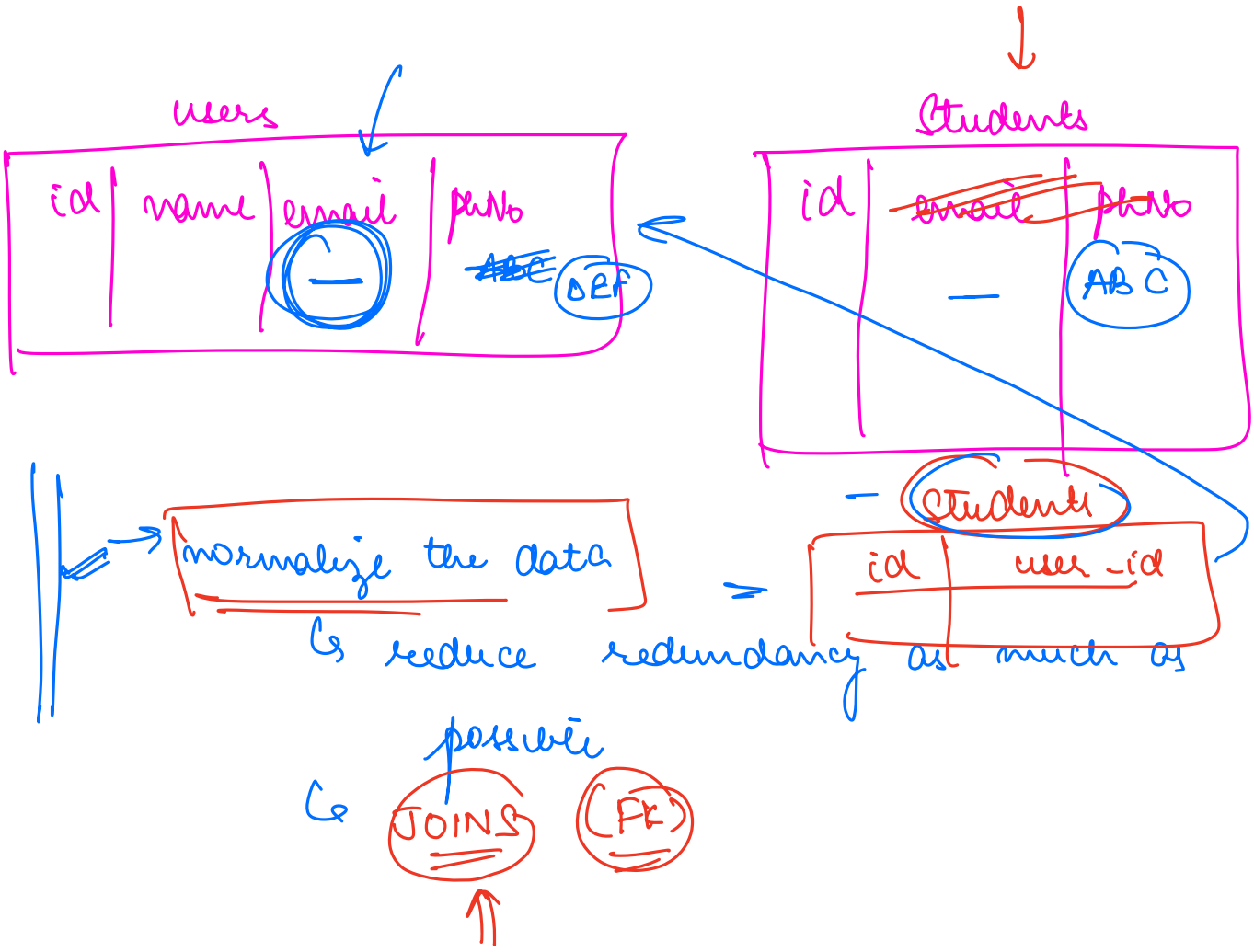
Agenda

1. Intro to NoSQL
   - → SQL
   - → Shortcomings
   - → NoSQL
2. How to decide sharding key

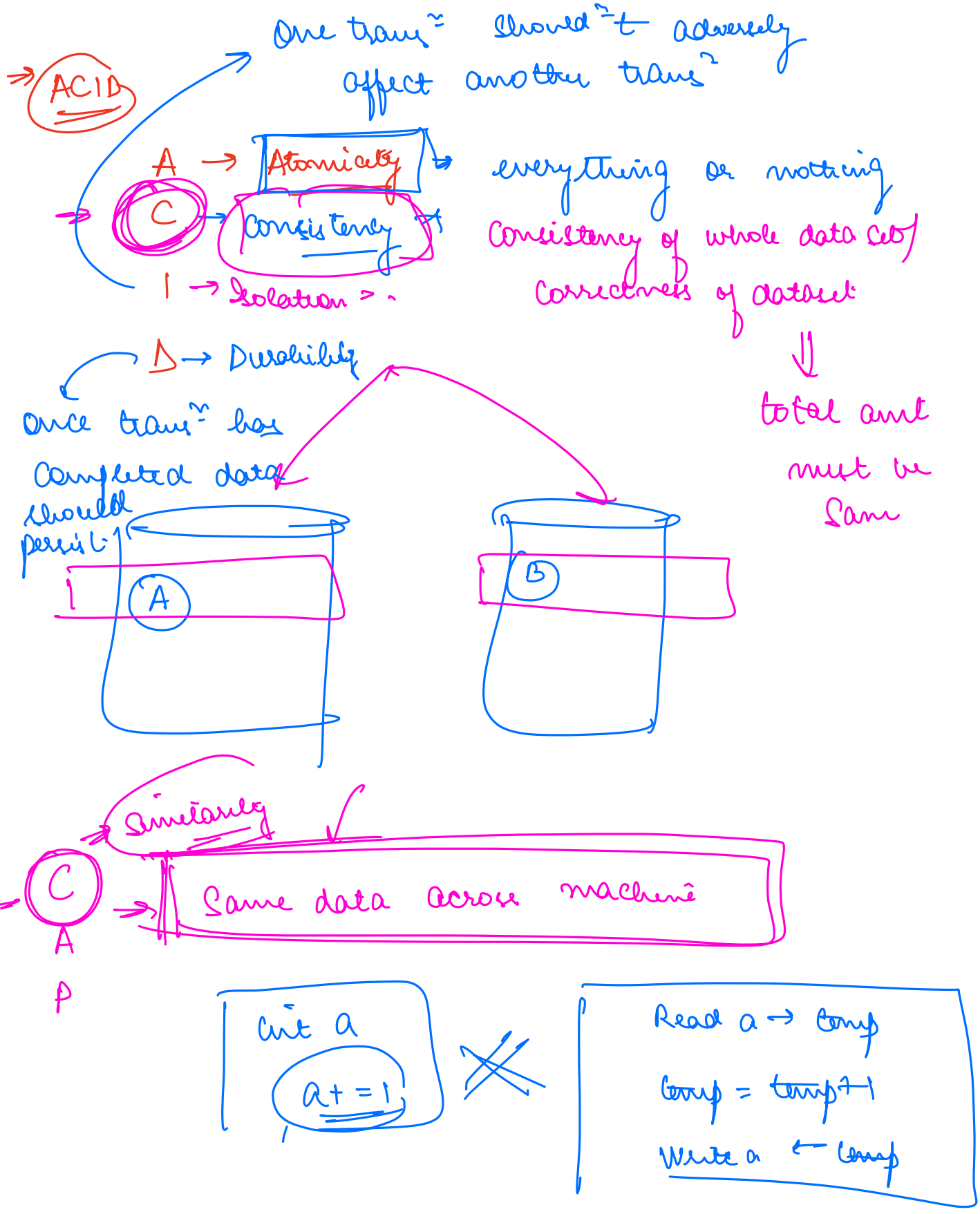
Features of SQL

→ ACID



⇒ If joins were slow. Then normalizing may

infact be bad.



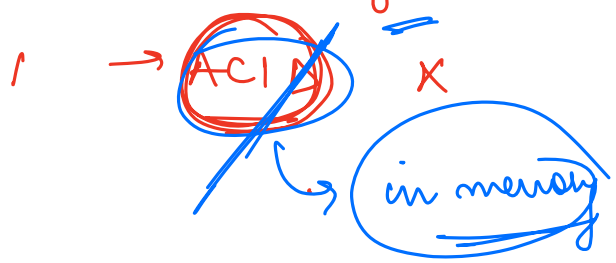
AtomicInteger a

a.increment()

Most NoSQL databases actually remove one of ACID to gain their properties

SQL → Normalization ✗

I → ACID ✗



Redis : NoSQL Database

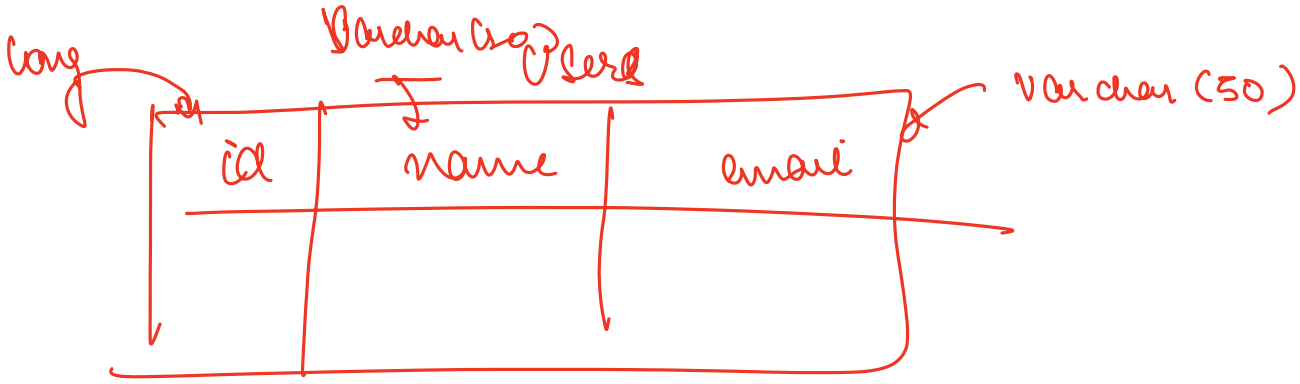
→ very fast write and read.

RAM >> SSD >> HardDisk

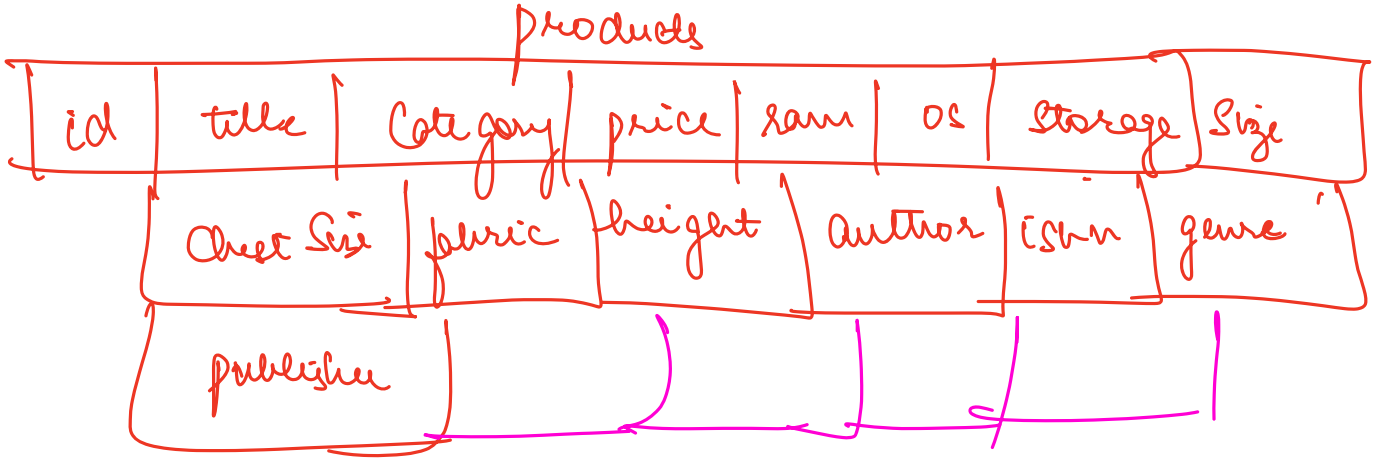
Structure

→ Defined Schema

→ defined set of attr & defined data types.



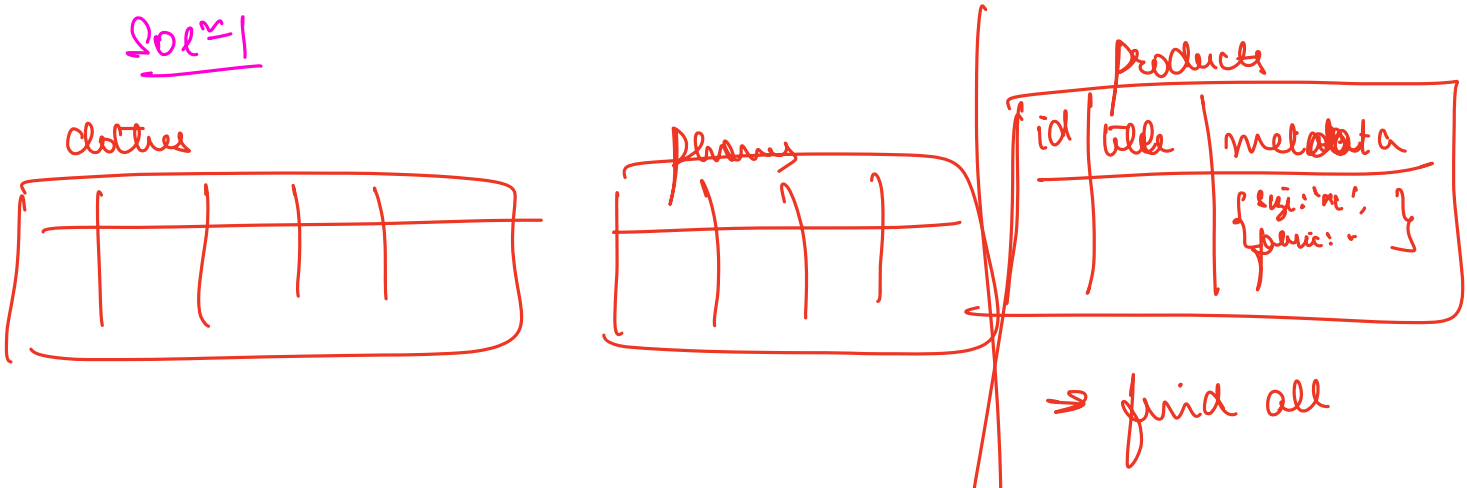
eCom website →

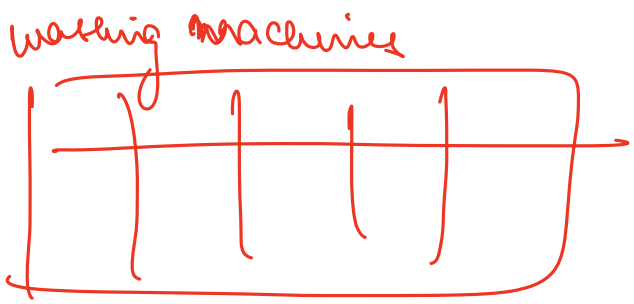


→ infinite # of attrs of products

→ if SQL ⇒ null values

Sol^n 1





25000 tables

→ SQL is optimized for large data in less # of tables V/S less data in more # tables

Clothes of all size.

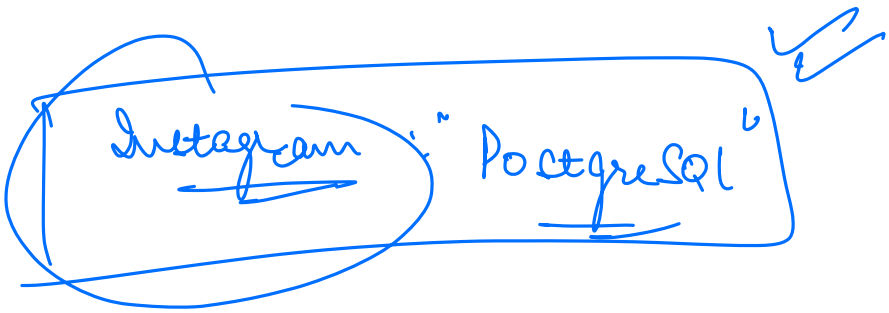
→ VVVV slow.

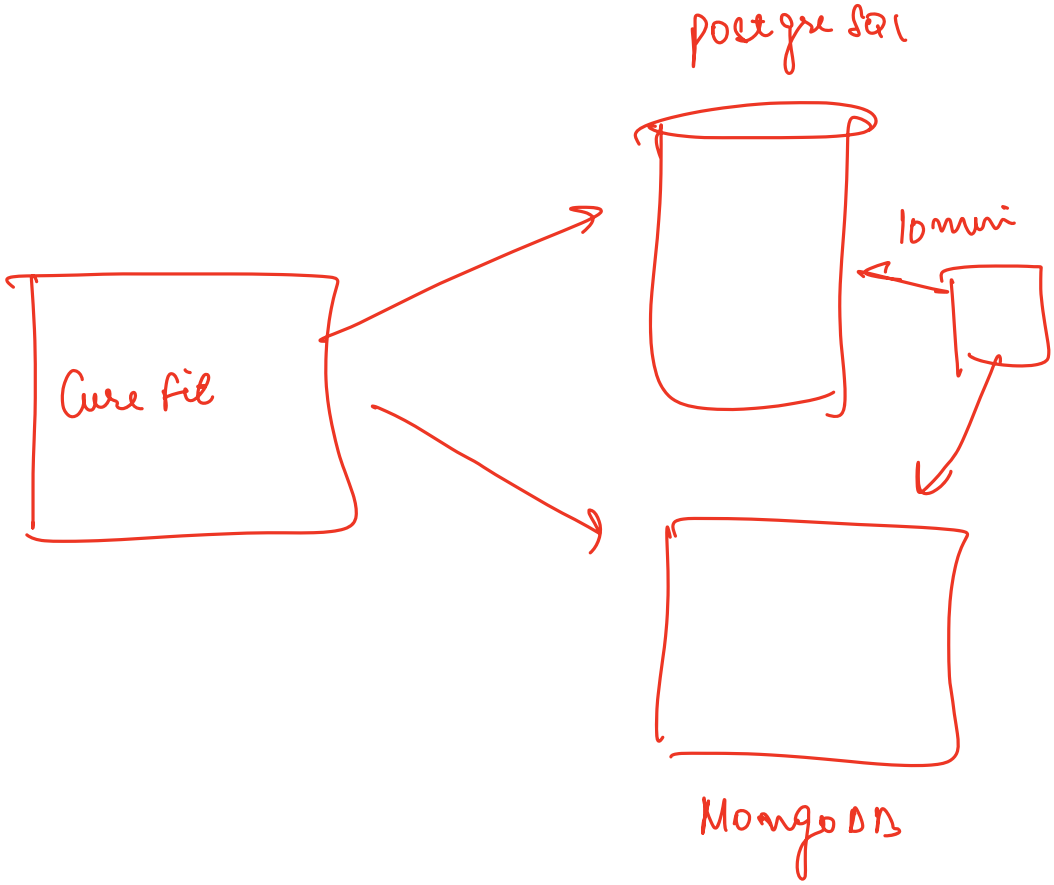
① Normalization
② ACID
③ Structured

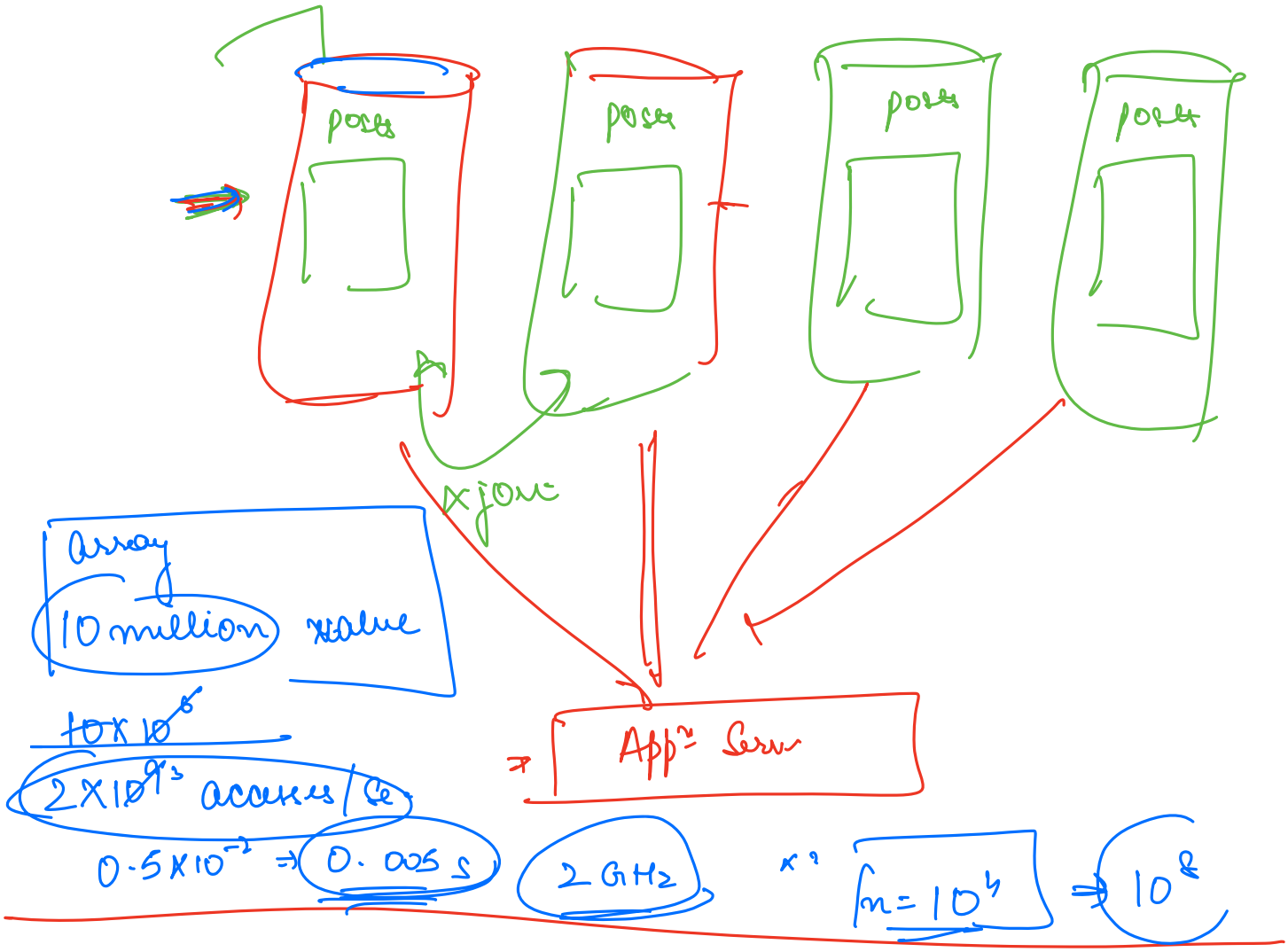
Efficiency V/S those.

NoSQL Database

- SQL DB work much better than what your data is
  - → evaluate SQL
  - → See if SQL poses some problems basis req
  - → If yes: evaluate alternatives





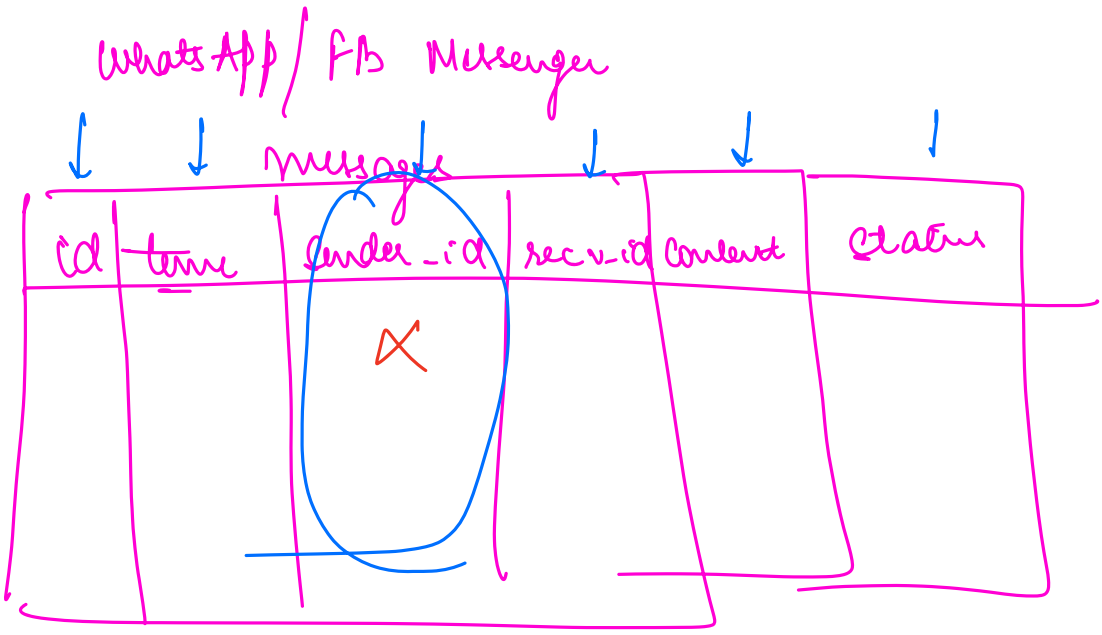


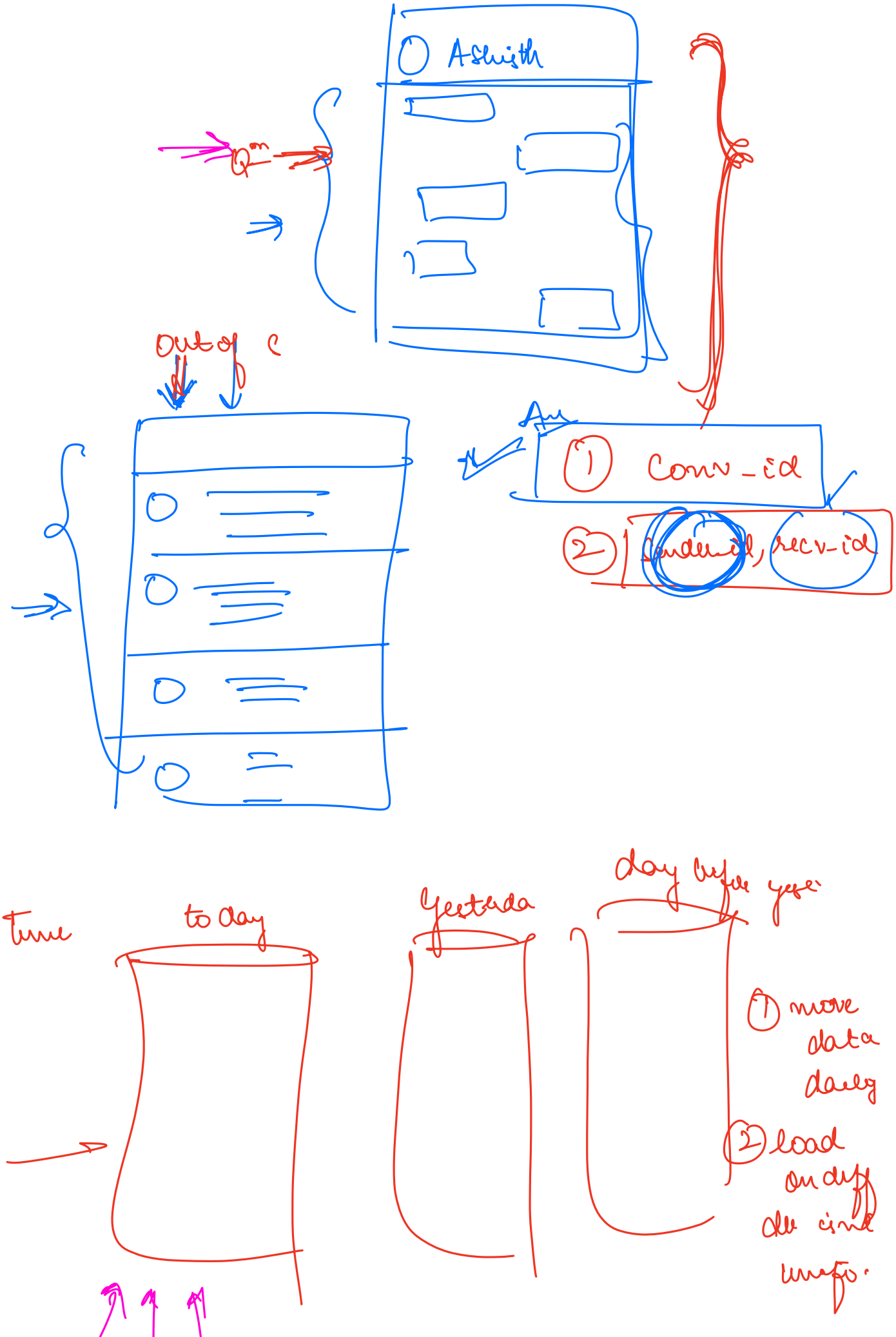
(1) What happens when size of data is huge.

- I will choose DB based on type of queries
- Size of data
  - Split data
    - Ideally split data in a way that SQL DB will work

⇒ How to choose sharding key

(1) Messenger apps




Ashish
Out of
Conv-id
sender-id, recv-id
time
today
Yesterday
day before yes
1 move data daily
2 load on diff dir and info.

Overloaded

→ Select * from tickets where train_id = 'ggg' →

HOW TO CHOOSE SHARDING KEY

hash(?)

⇒ creating too small Sharding key

1. makes our query fast (ideally only one db machine to go to)
2. load across shards should be uniform
3. don't choose too broad Sharding key

Conversations

id	user1	user2	lastMessageTime

h(a, b)

→ h(a*10^9 + b*10^6)

messages

id	conv_id	text	time

Bank System

Accounts

id	ownerName	branch	addressCity	type	balance

In Caccount-ids

Opⁿˢ

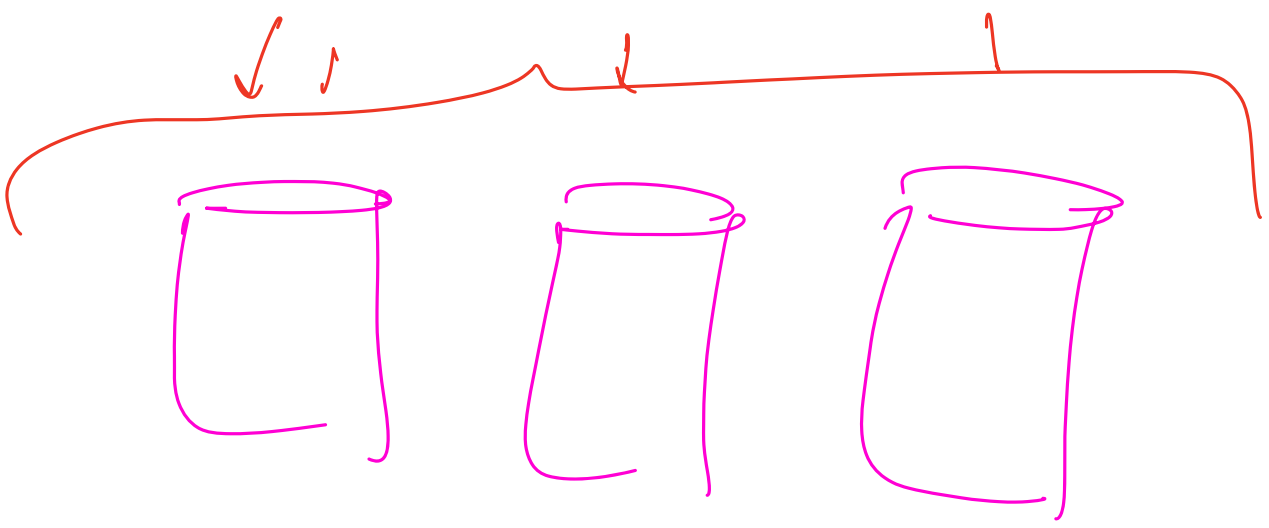
1. get balance of an account
2. get transaction history of an account

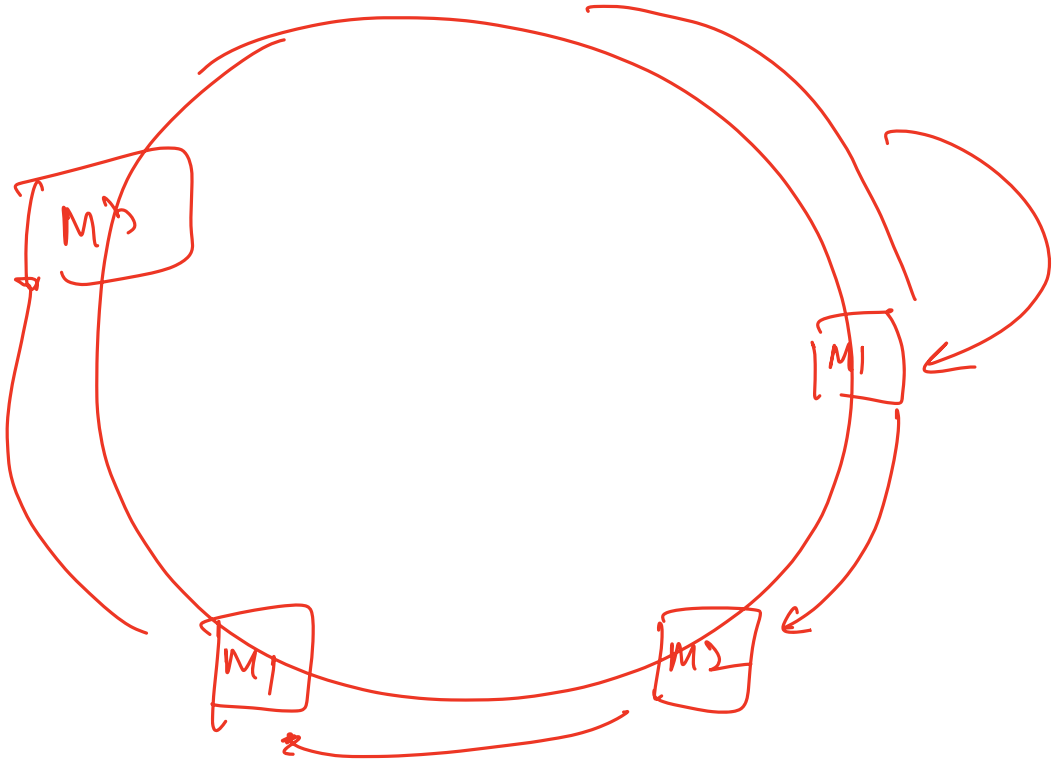
↓

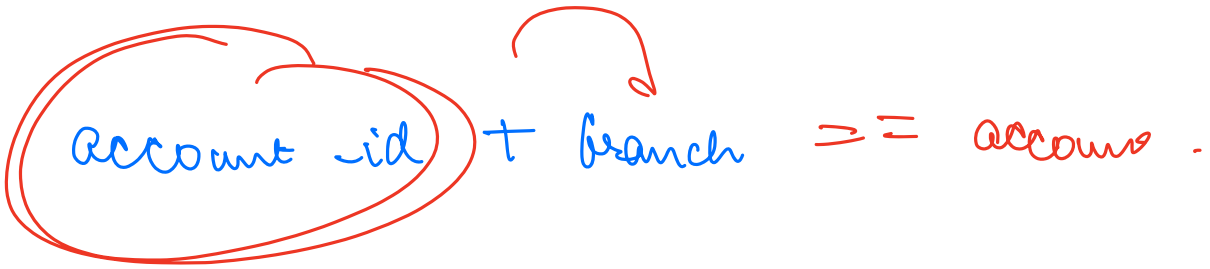
all trans^{n} of same accounts in same row

transaction

id	account-id	amount	type

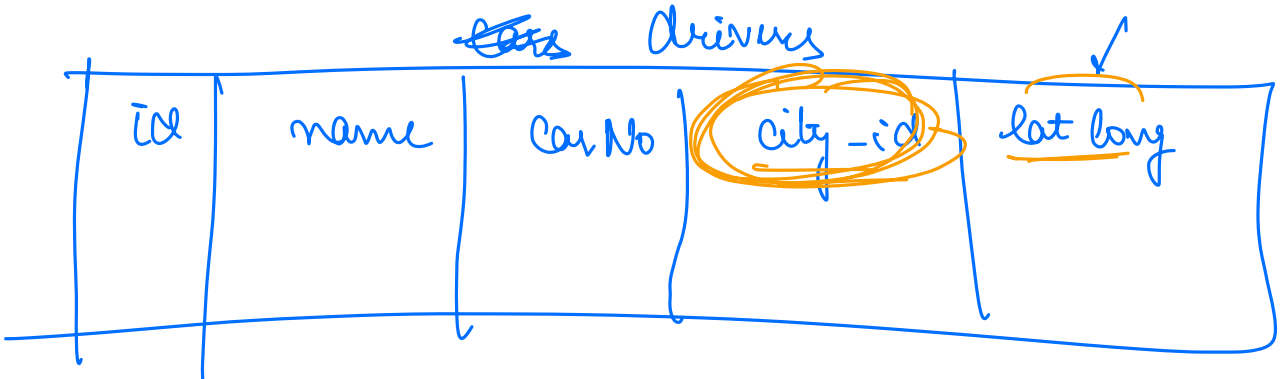




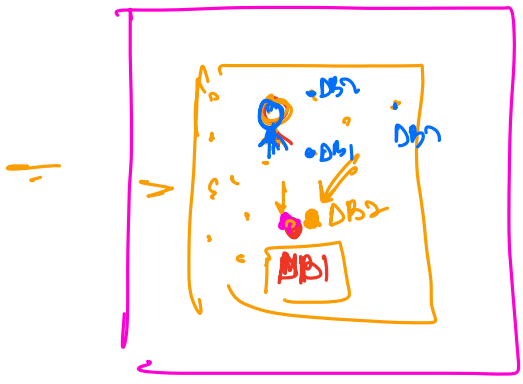


User

Req → people would want to search cars near to them.



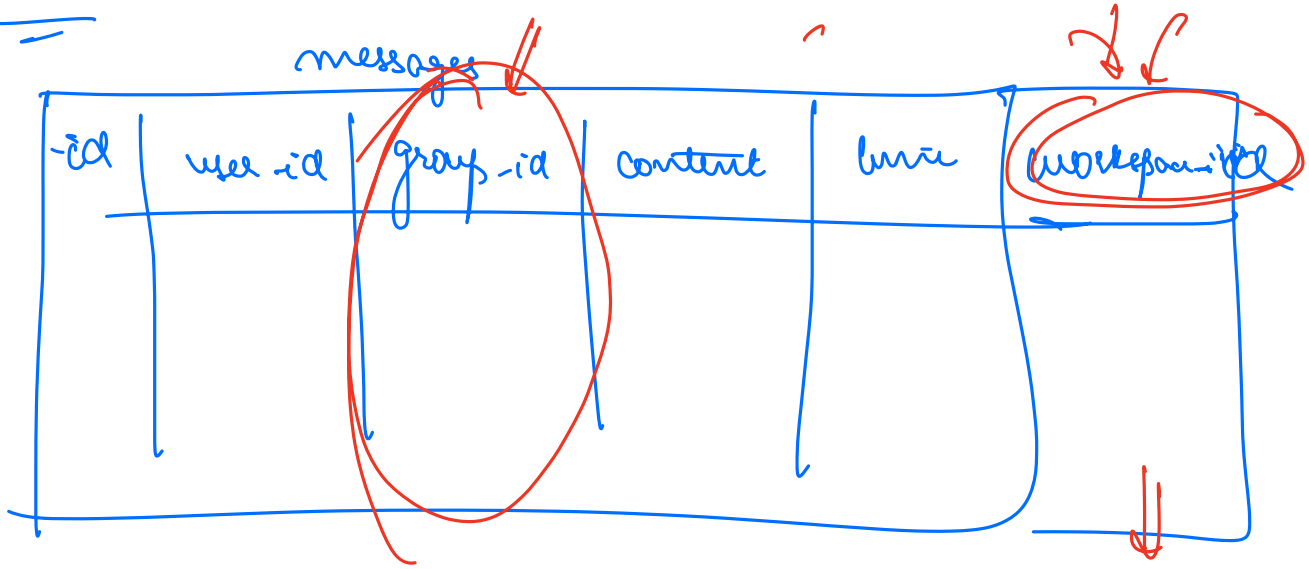
If you shard on (lat, long) → all cars with (lat, long) will be in same machine



⇒ ideally all cars nearby are in same machine

Google Maps / Uber → Tinder

Slack



too abstract
↓
too big

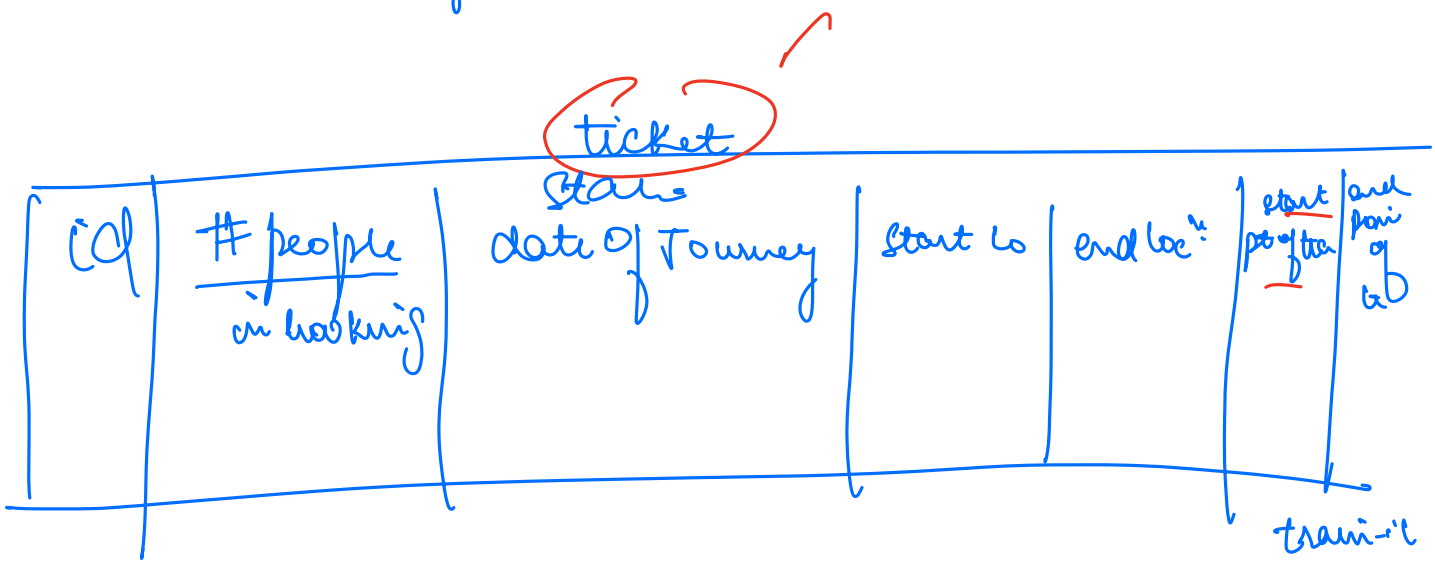
⇒ Sharding key should be user that

it doesn't overflow db in itself

IRCTC

(1) Same seat shouldn't be booked twice

(2) ⇒ = ⇒ { I should be able to get all bookings for a particular train to be able to assign



potential

(1) train_id + doj ≤ } Ans

(2) train_id

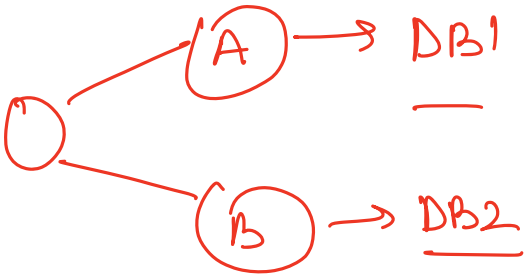
(3) start pt of Travel → too big

(4) ~~train_id + ticket_id~~ ~~# of people~~

= | train_id + ticket_id | ⇐

⇒ All tickets with same train_id and

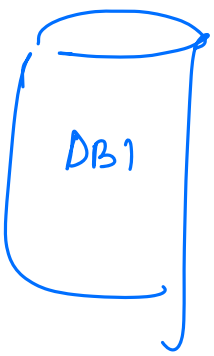
ticket id are in one machine



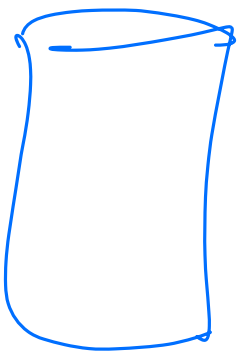
⇒ all tickets of same train to be together

→ train id

(1, 24th October)



(2, 24th October)

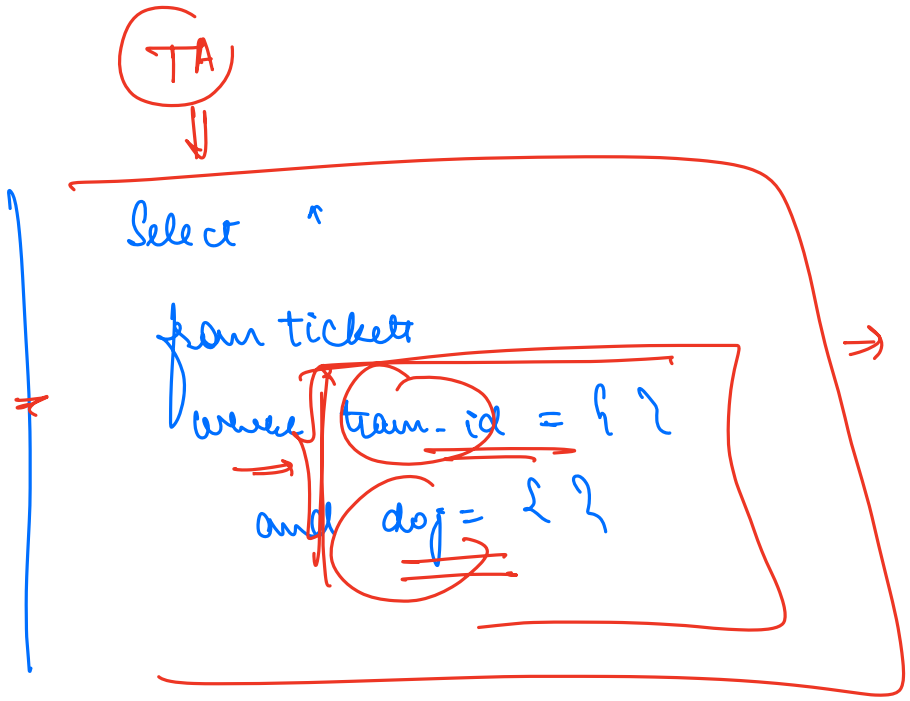


(1, 20th Jan)

AND

(2, 20th Jan)

hash(a, b) ⇒ hash((a * 10^8) % 10^7 + —)



→ Ideal key to get all such rows on same machine