NoSQL Databases

Any way to store data other than SQL

- → Document
- → Columnar / Column-Family
- → Key-Value Pairs
- → Graph DB
- → Vector DB

All storing data in a diff kind of way with diff pros and cons.

⇒ Relational DB may not be always good
↳ databases other than relational
⇒ NoSQL

Agenda

1. Types of NoSQL Databases.
   - (i) Key Value Pairs
   - (ii) Document DB
   - (iii) Columnar DB.
2. Case Studies: Situation and you have to tell what NoSQL Type

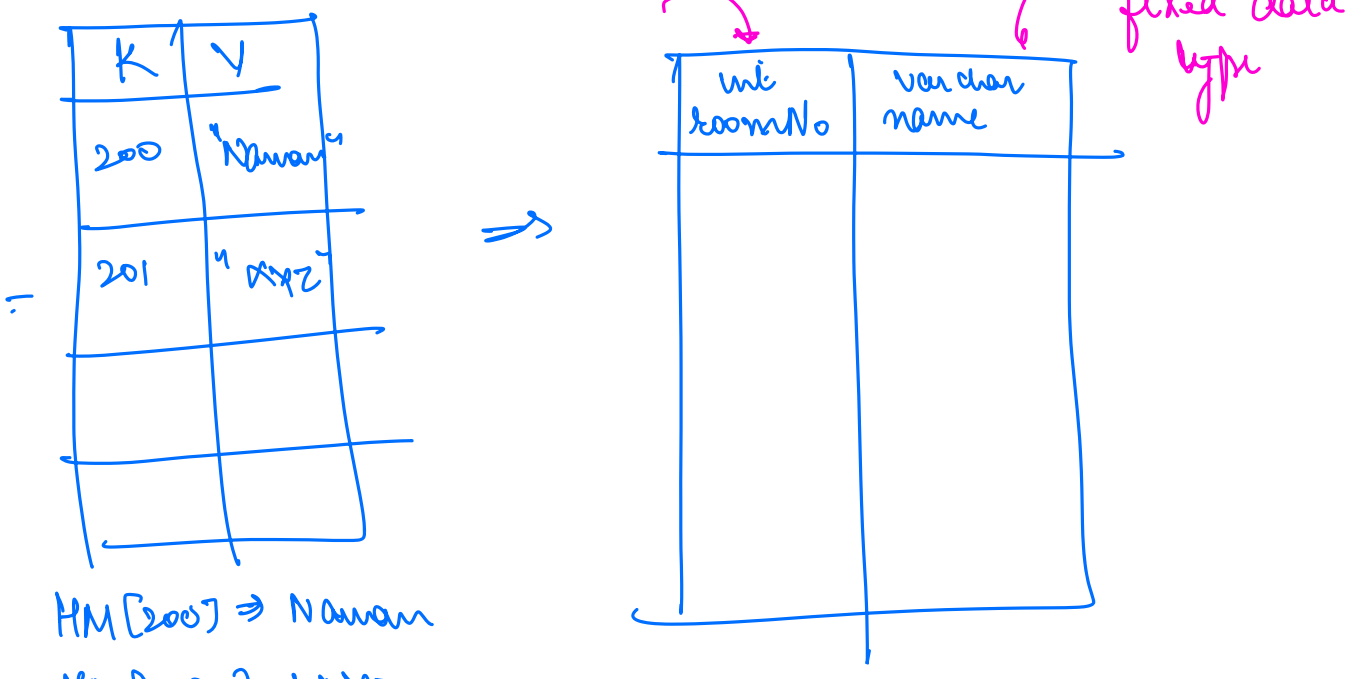
will be best for it

Next Class ⇒ How diff types of NoSQL internally work.

---

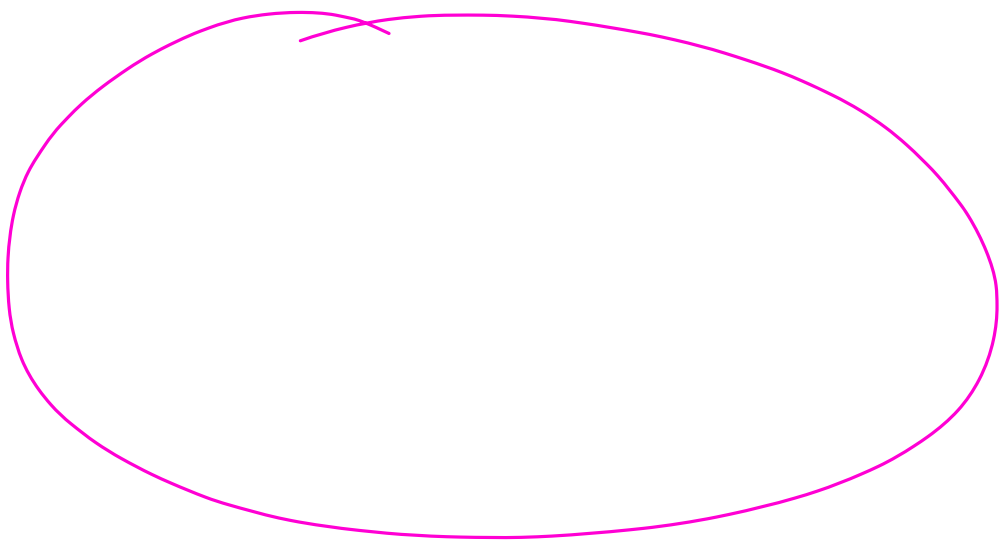
① Key Value Pair

→ HashMaps ⇒ (K,V) pair

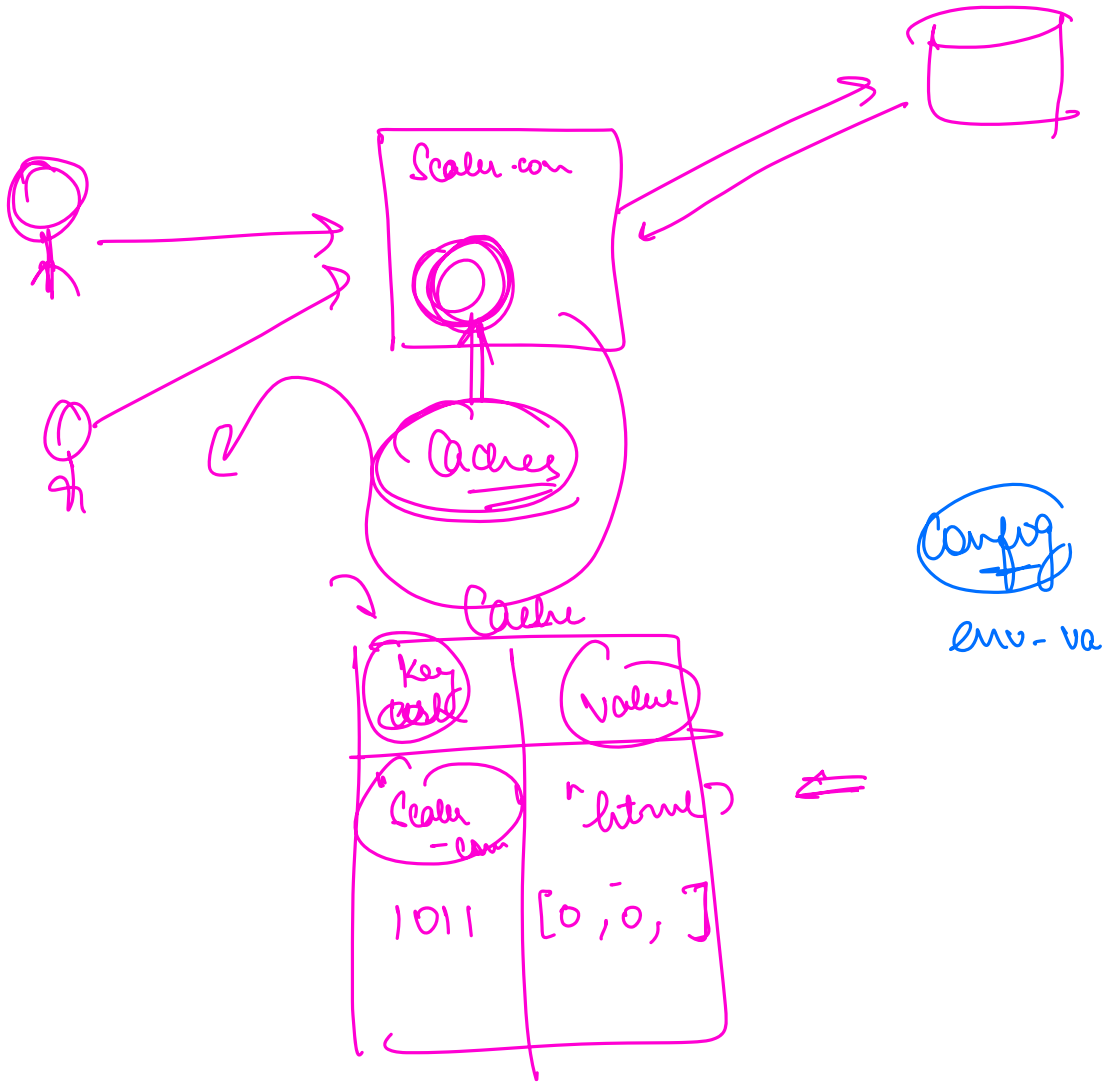


HM[200] ⇒ Naman

HM[201] ⇒ XYZ



⇒ typically used when I dont have a fixed data type for values.

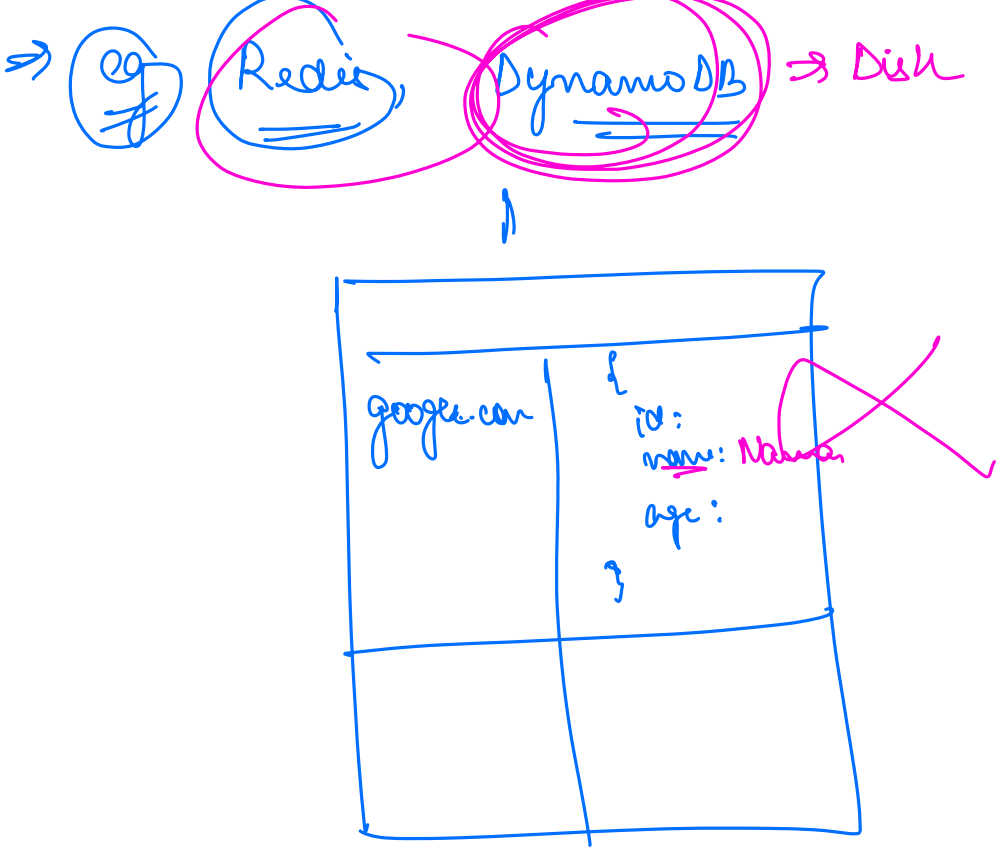


⇒ When all the queries are on keys.

=> { . get ( ) } =>

= { . set ( ) }

work fast in SQL



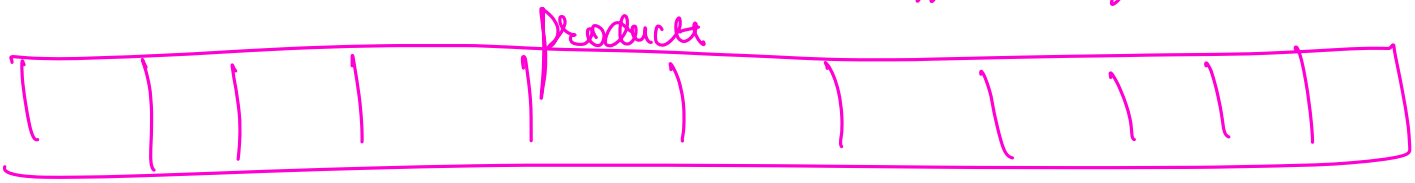
Type of NoSQL DBs is decided by what type of f^n it optimizes for.

Side Project: You can build every type of DBs using a key value DB

Document DB

ecom ⇒ Schema of products is not well defined

⇒ diff product can have diff set of attr.



⇒ sparse table

⇒ waste a lot of storage

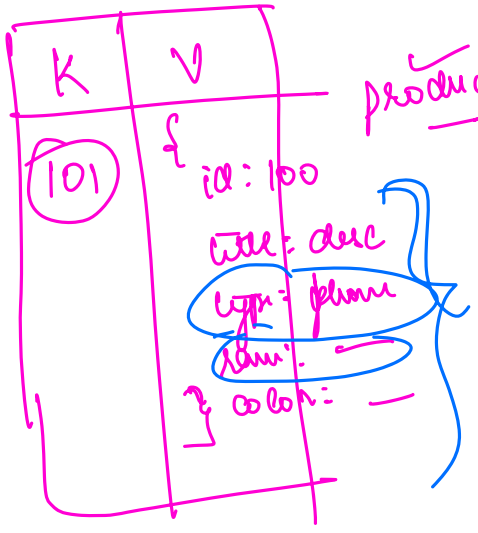
Problem Statement

⇒ unstructured data

⇒ every entity can have its own set of attr.

① Ability to store unstructured data

Key / Value



② ability to query on attributes of unstructured data.

→ Document DB

↳ can think of them as k-v pair with an addition f^n of querying on attributes of every entity

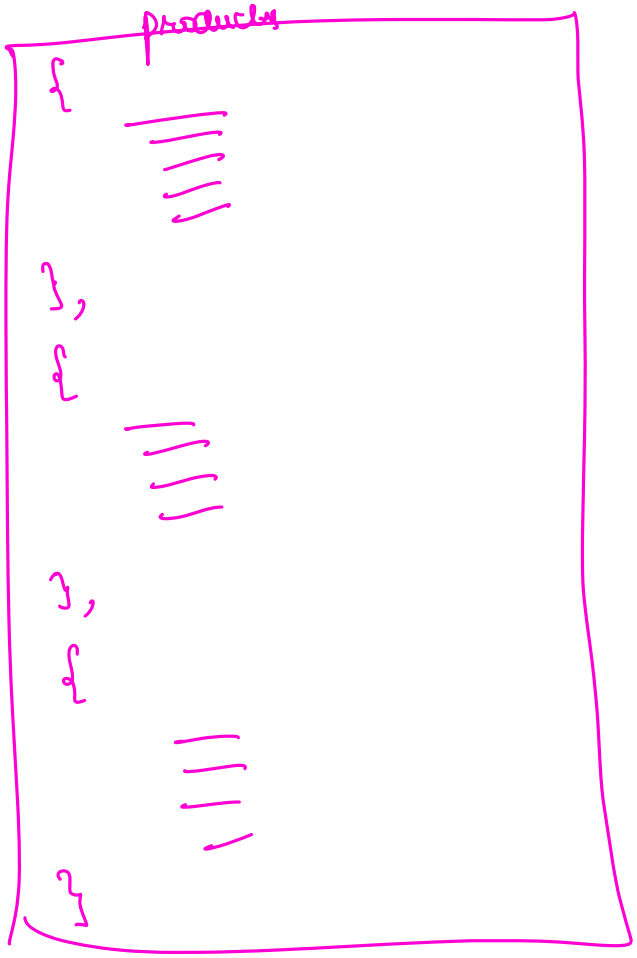
↳ Mongo DB, Elastic Search

① k-v pairs where values are jsons

② every entity is a json with whatever attrs it needs to have

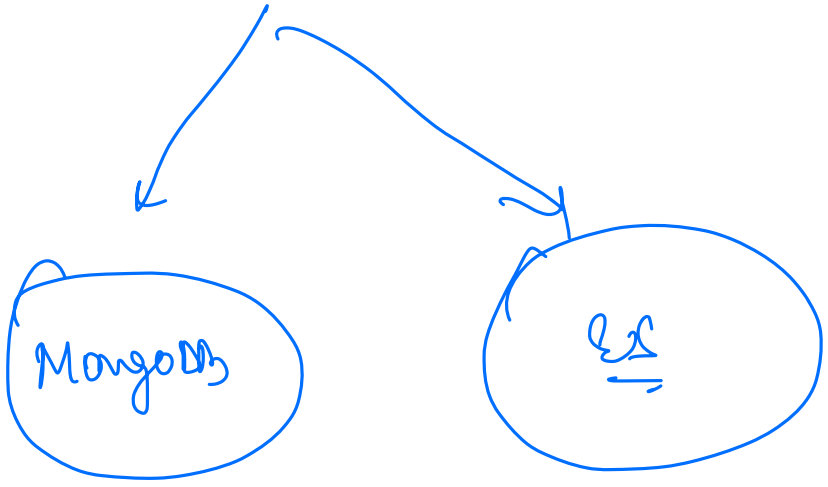
{
id:
title: —,
desc: —,
fabric: —,
size: —,
color
}

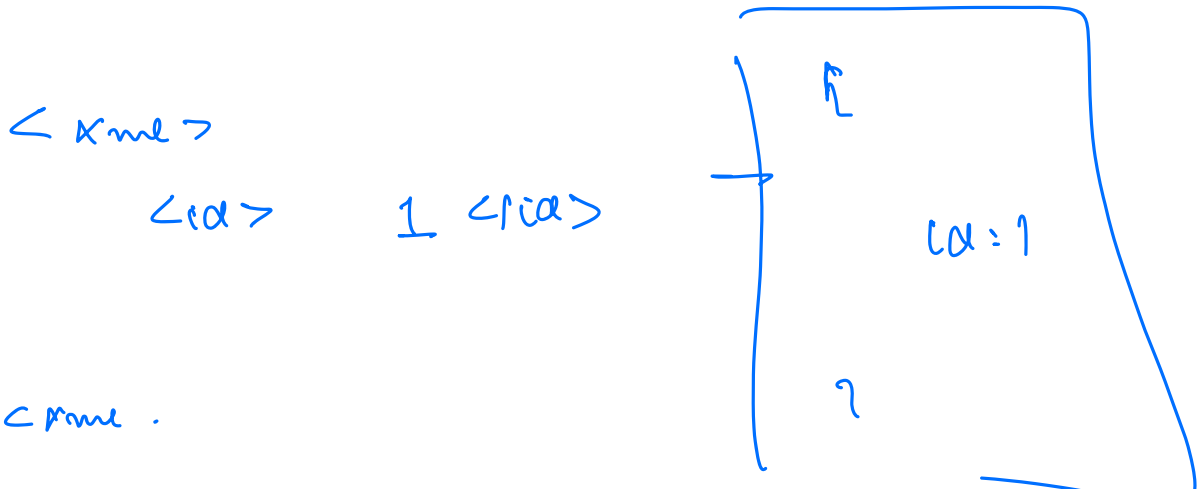
{
id:
title:
desc:
size:
OS:
brand,
}



SOLR

Inverted index





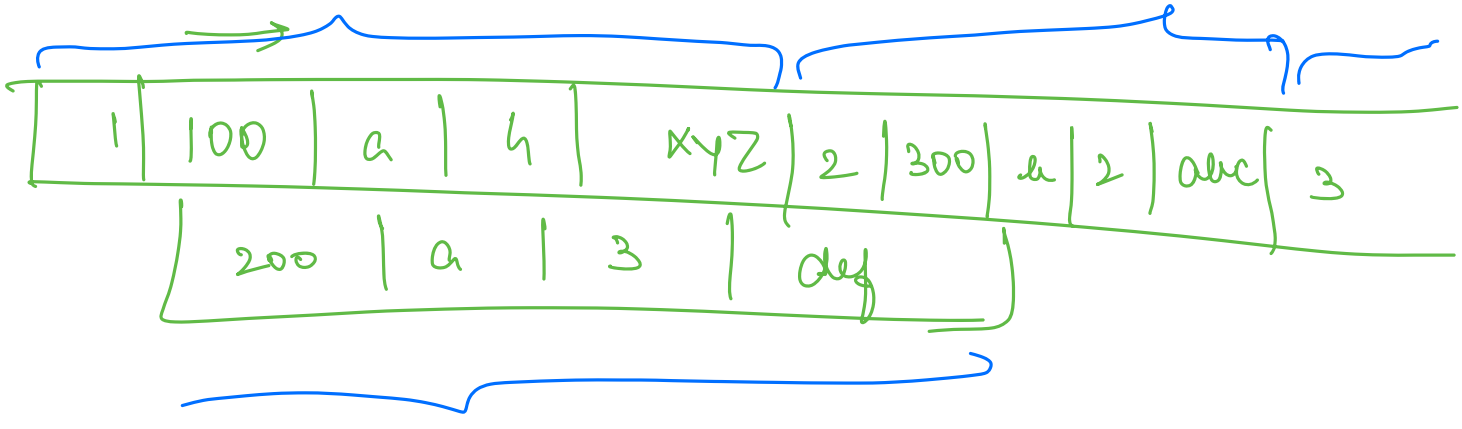
Columnar DBs / Column Family

Orders

id	amount	user-id	# of products	date
1	100	a	4	XYZ
2	300	b	2	abc
3	200	a	3	def

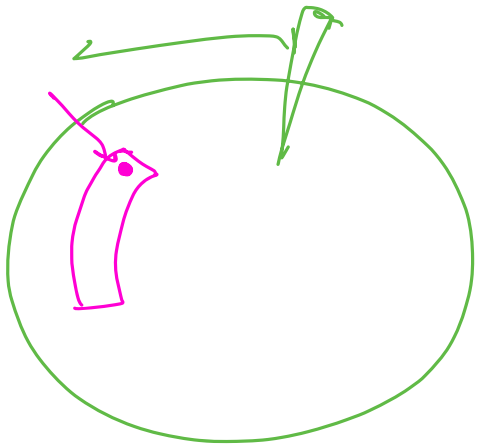
1. Select * from orders..
2. Select * from orders where amount > 100
3. Select sum(amount) where user-id = -
4. Select sum(amount)
5. Select avg(amount) group by date;

How will SQL internally store data

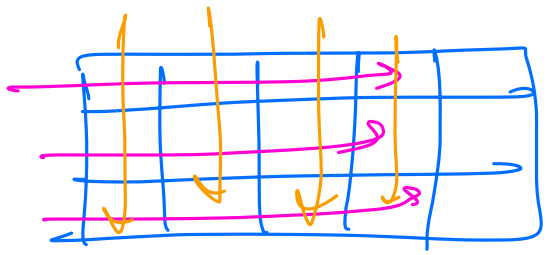


⇒ indexes reduce # input/output from disk

disk accesses are slow



→ when you access a particular data on disk, nearby data is also fetched to mem



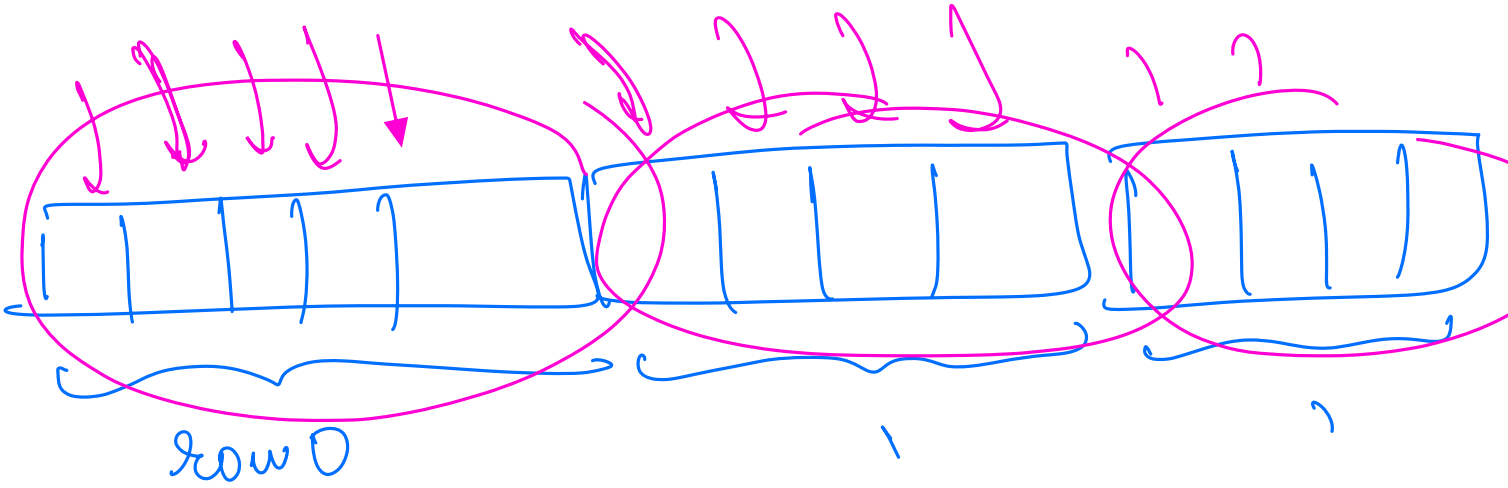
a[][]

⇒
```
for (int i=0; i<n; ++i) {
    for (int j=0; j<n; ++j) {
        print a[i][j]
    }
}
```

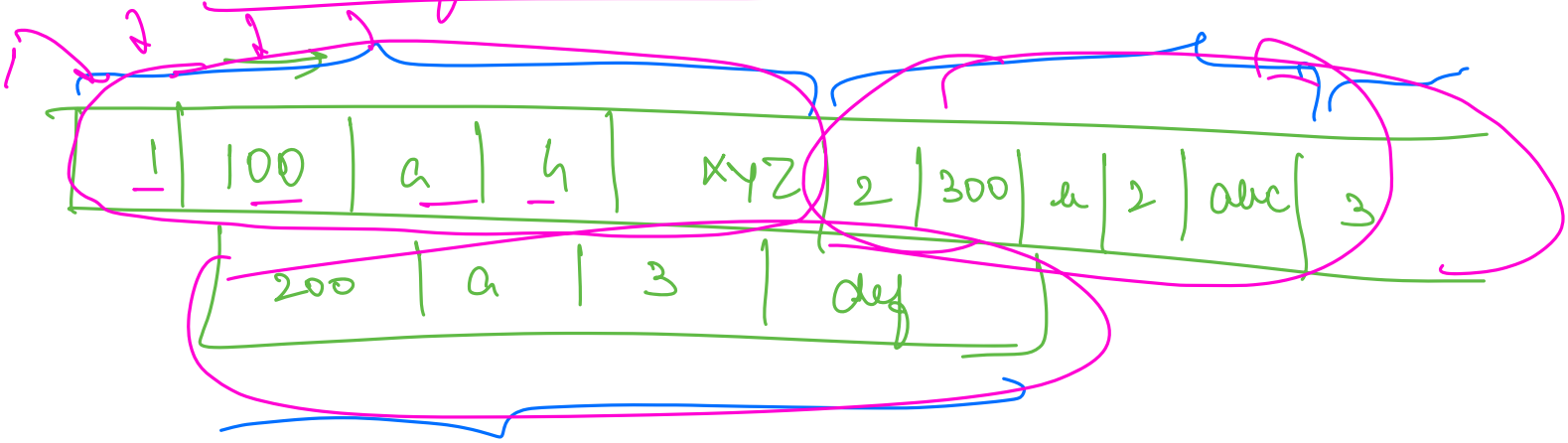
(1)

```
for (int j=0; j<n; ++j) {
    for (int i=0; i<n; ++i) {
        print(a[i][j])
    }
}
```

(2)


row 0

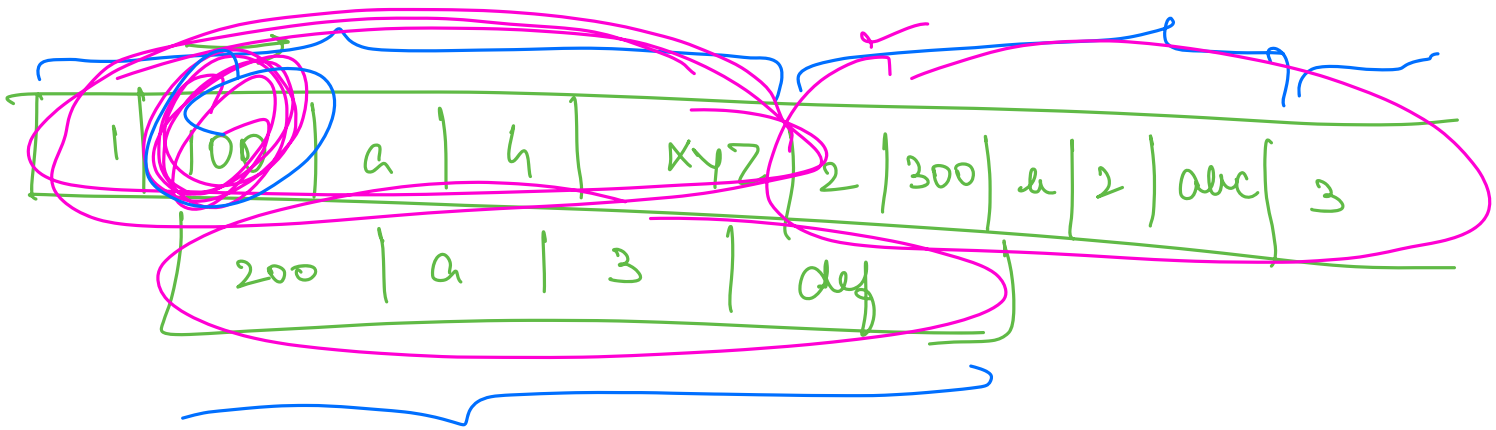
Select * from order



1	100	a	4	XYZ
2	300	a	2	abc
3	200	a	3	def

⇒ all data I ever fetched from disk was needed.

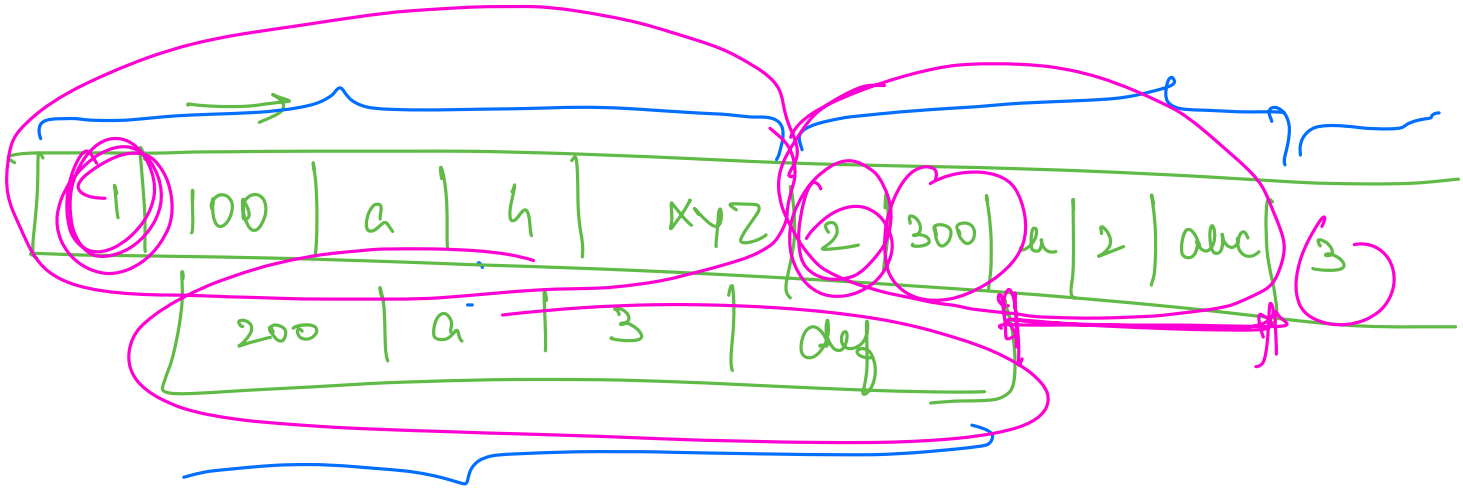
Select * from orders where Amount > 100



2 | 300 | u | 2 | abc | $

3 | 200 | a | 3 | def

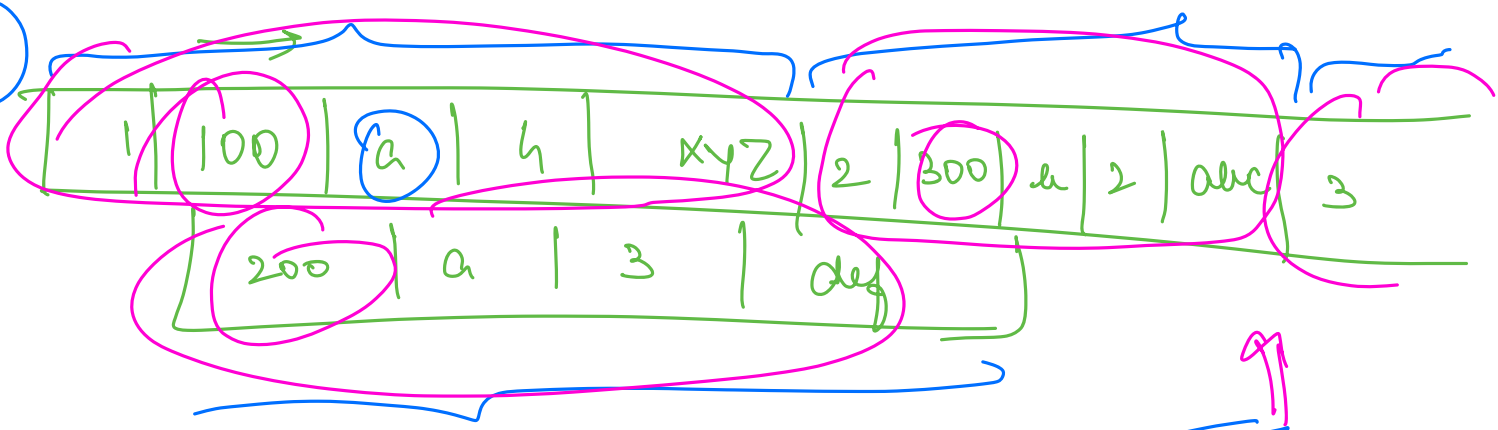
⇒ I did some useless disk access but fine, it was okay.



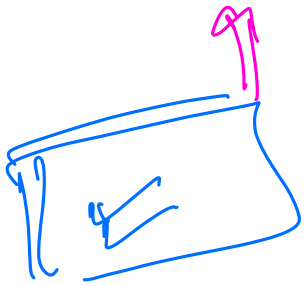
Select sum(amount) where user_id = 2

⇒ did I end up fetching a lot of cells that I never needed.

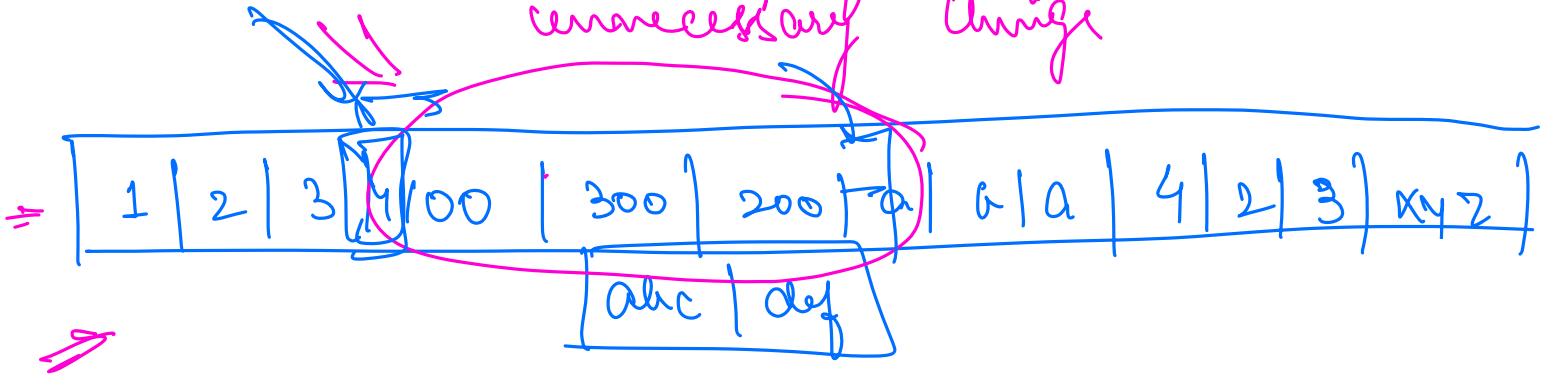
4



✓ Select sum (amount)

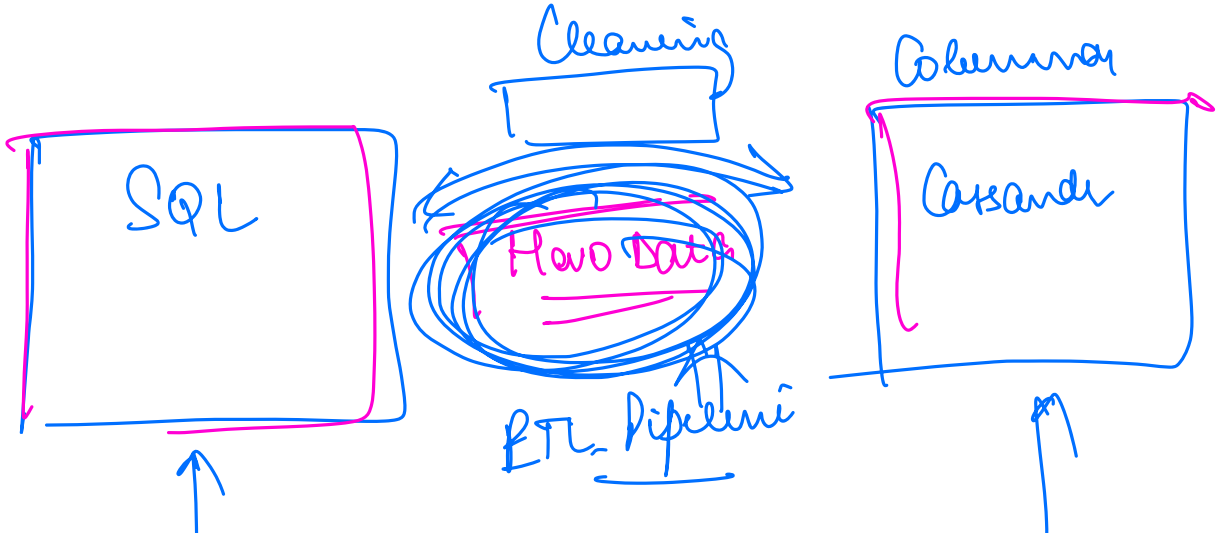


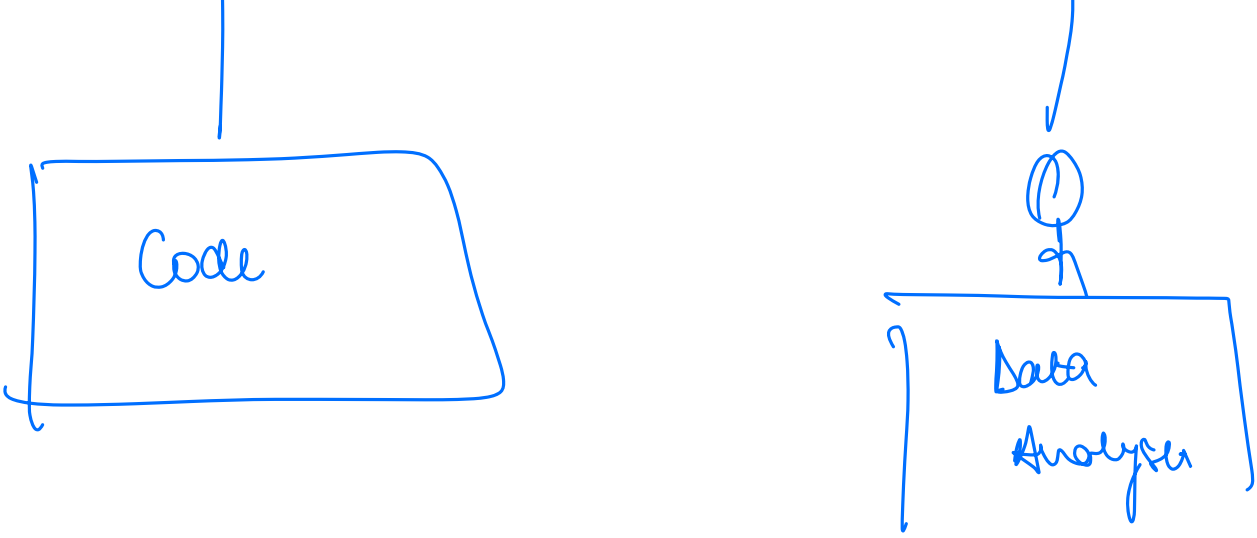
→ I ended up fetching even more unnecessary things



→ Much less disk access!!

→ data analysis





⇒ Databases that store data as col^n instead of rows are called columnar DB / column family

Column families

→ data is structured

→ queries are on value of few columns / agg queries

Eg: Cassandra, HBase

Break till 10:30

10:50

→ Other optimizations with Col^n family

→ Neo4j, Timeseries DB

→ Case Studies

↳ Uber

"aaaaaaaaaaccccaaaaaaaaaaa"

⇒ "6a2a3c2c6a2a"

(i) Imagine each character like a DB row

⇒ Can't compact. Because no 2 rows in the DB are completely same.

Products

id	title	Category	price
1	A	P	100
2	B	P	120
3	C	cloth	140

⇒ I will store things stored by cat

100 100.

[1|2|3] A|B|C [P|P] C | 100 | 120 | 140

[1|2|3] A|B|C [2P] C | 100 | 120 | 140

100 MB → 90 MB

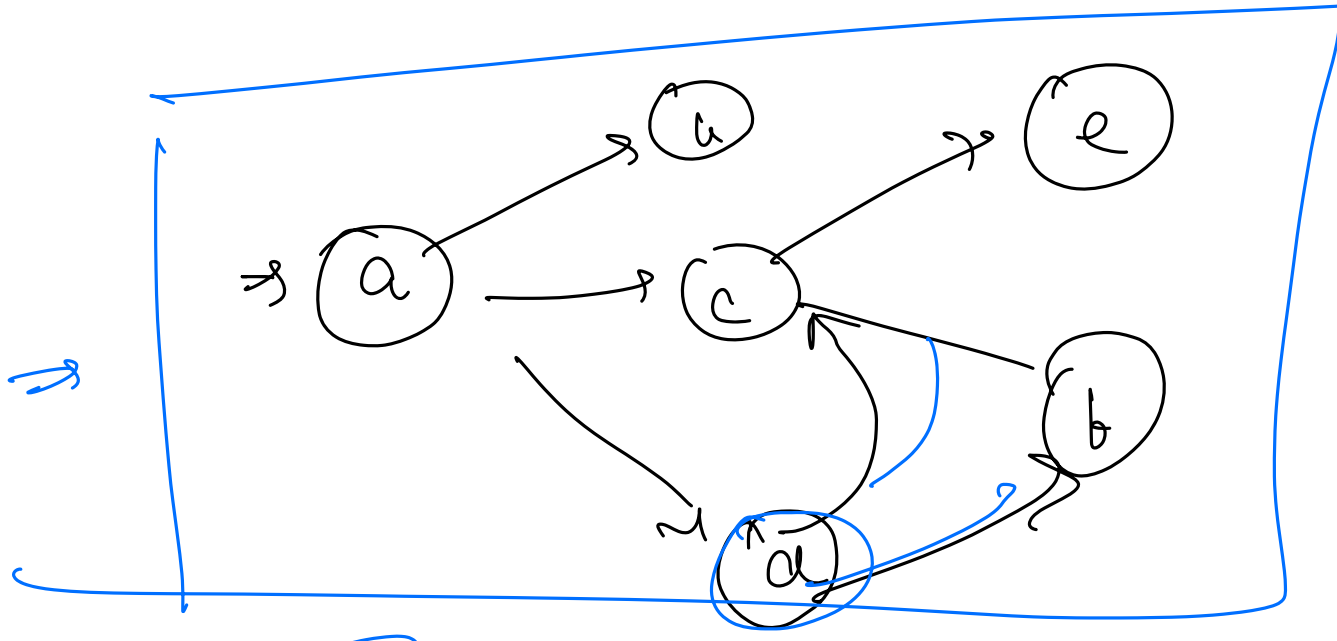
100 GB → 90 GB

Rel^n b/w entities

→ Social graph

Graph DB

→ Neo4J



(1) Recommendation

Time Series DB = query after multi time ranges

0-10 → 100

10-20 → 20

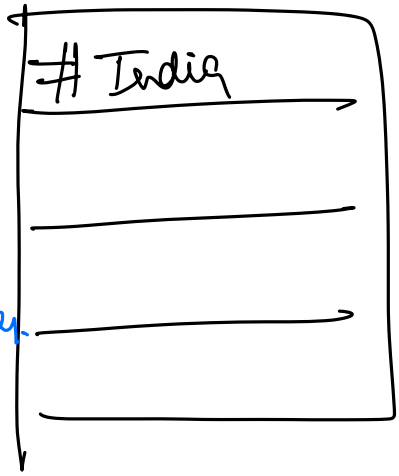
20-30 → 0

30-40 → 5

(1) Twitter Hash Tags

a:) for each hash tag, store recent tweets of that

b:) fetching tweets incrementally



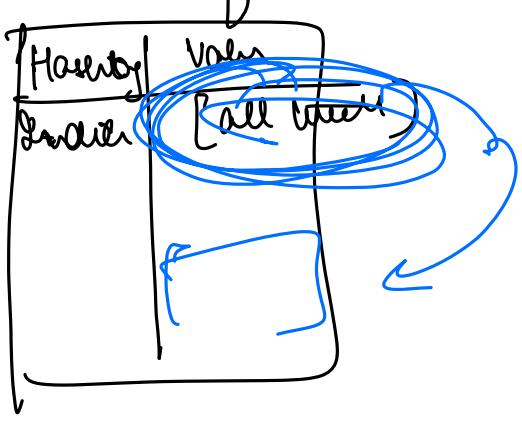
10

10

10

relational → No. A lot of write queries.

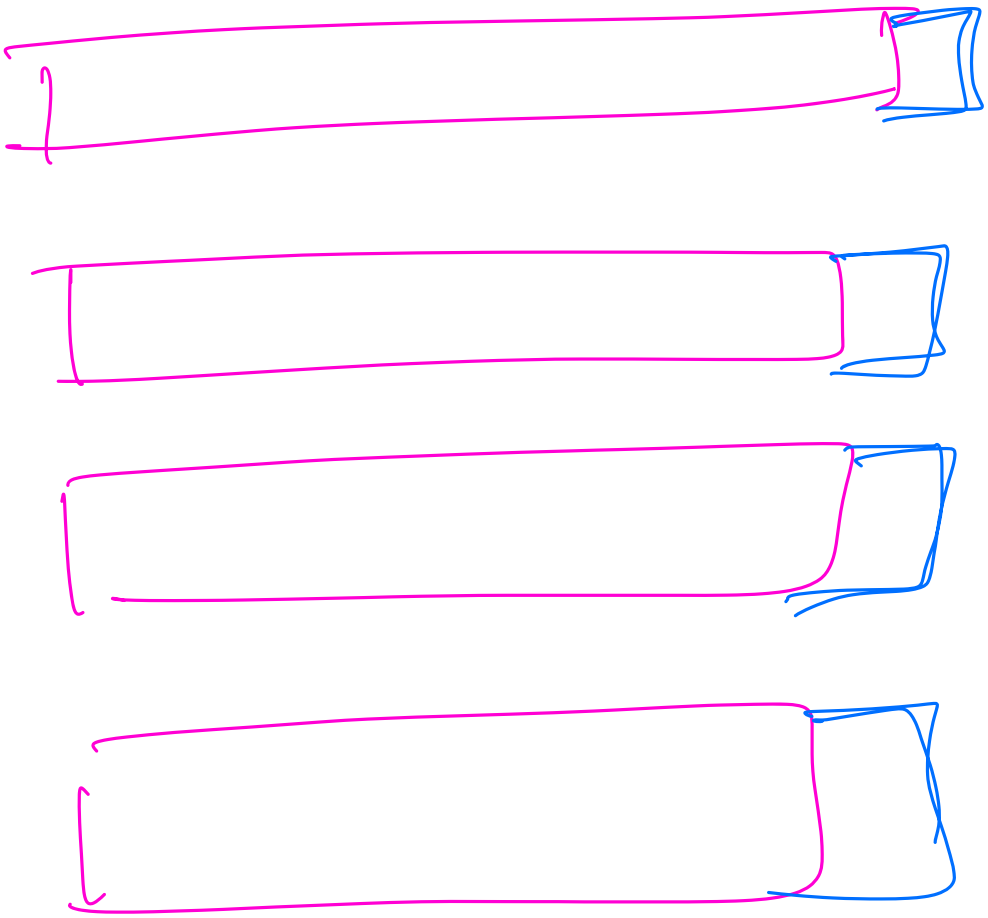
key-value →



No. Doesn't allow query within val

Document ⇒ No. Because data is structured

Column-family ⇒ if writes are append only they can handle very fast.



Live Score of a Match (current live score)

101 = 12/8

- History
  - TimeSeries / Col^m Family
- No History
  - Key-value

Current Loc^n of an Uber Driver

- Key-value (if history not req)
- Columnar / Time Series (if History)
- Geo spatial DB
  - get me drivers in 2 km of (lat, long)

↳ get Nearest driver