

## Agenda

- ↳ Common types of NoSQL databases
- ↳ When to choose what
- ↳ How are shards managed in NoSQL
  - ↳ replication
  - ↳ multi-master

SQL → Pros & Cons

↓                      ↓  
Schema ~~~~~ Schema might be too limiting  
ACID ~~~~~ Handling loses ACID across shards

NoSQL

Key-Value

Document DB

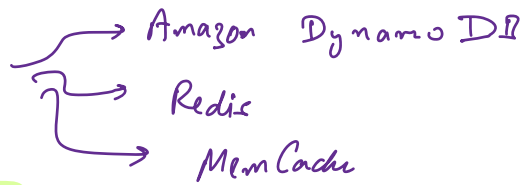
Column Family

Graph DB

↓  
Knowledge management  
ML Pipelines

File storage

## Key-Value DB



## Completely Schemedless

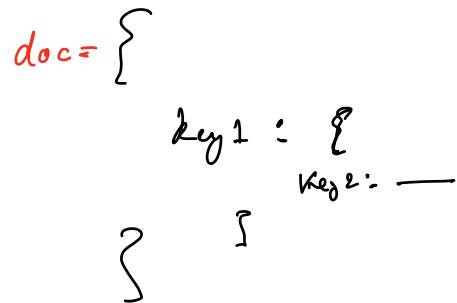


Query is only possible  
by the key

↳ get(key) → value

↳ set(key, value)

↳ has(key) → boolean



doc[key1][key2]

## Index

find all keys whose value > 10

X slow  
∴ no way to  
index values

## Document Database

MongoDB  
DocuDB  
Elastic Search

They are JSON stores.

documents uuid  
"abc2d - d2dd - 261a - bfggha" : {

```
{
  'key1' : 'Hello',
  'key2' : 20,
  'key3' : [1, 2, 'H'],
  'key4' : {
    'key5' : 'Hello'
  }
}
```

⇒ JSON  
object

}

Amazon — Products

```
{
  uuid : {
    'type' : 'laptop',
    'hdd' : '80GB',
    'RAM' : '—',
  }
}
```

```
{
  uuid : {
    'type' : 't-shirt',
    'color' : 'white',
    'neck size' : '—'
  }
}
```

```
void: {  
  'hello': 'bye'  
}
```

## Query

get (void) → json → entire document  
set (void, json) → set the entire document


get All Match ( { 'key1': 'laptop' } ) → automatically indexed (mostly)

in MongoDB → add indexes manually.

indices → B<sup>+</sup> trees

index on 'key1'

void	value
<del>_____</del>	<del>_____</del>
<del>_____</del>	<del>_____</del>
<del>_____</del>	<del>_____</del>



# Column Family Databases

→ Cassandra  
→ HBase

row-id → sharded based on rowid.

rowid = 'Vadhirajai'

Posts	
ts	val

Comments	
timestamp	val
t2	v2
t3	v3
...	...

↑ Data Type  
↑ string

↓ Sorted by  
timestamp desc

Messages	
ts	val

← Column Families

rowid = 'Akhil'

Posts is  
not there for  
Akhil

Posts

Messages

## Query

① fetch Most Recent (rowid, col family, 10)  
Akhil Message 10

② fetch By Prefix (rowid, colfamily, 'sanabhi')

Akhil → Sandesh

Rohid → Akhil

Find all recent messages  
Akhil has received  
from Most Recent

Received - Rohan	
9:53	<u>Sandesh</u> : Hello
9:58	<u>Sandesh</u> : Hi
1:00	Anurag: Hey

Find all recent mess in the Akhil/Sandesh conv.

Rohid → Sandesh

Sent - ~	
9:53	Akhil: hello

Tricks

↳ efficient Prefix  
matches

↓  
Sorted list

↳ Saves space

1

## Twitter's Hash Tag data Storage

Context:

→ #India → 10 Billion Tweets

↳ for a hashtag → top 'K' most recent ✓  
10, 20, 50 most popular ✓

↳ Scroll → loads more tweets for next tag. ✓

→ paginated data

Relational DB  
SQL ?

Can't handle  
Scale!!

manually  
sharding

~~Key-Val~~

key: Hashtag  
value: all tweets  
with this  
hashtag.

get(#India)

~~Docset~~

uuid: ==  
{  
  hashtag: '#India',  
  tweets: [  
    {ts, id, msg},  
    {ts, id, msg},  
    '  
  ]  
}

uuid: ==  
{  
  hashtag: '#Scam',  
  tweets: {  
    ==  
  }  
}

uuid: ==  
{  
  hashtag: '#India',  
  ts: [ ]  
  msgs: [ ]  
  popularity: [ ]  
}

✓ Column

Rowid - hashtag

top tweets	
ts	value
	tweet

Popular	
ts	value
	#hilo - msg

prefix matrix

100000 - Hi  
→ 1000 - Hilo  
→ 100 - Hilo

100000 ✓  
100000  
100000

~~Graph~~

sorted  
string  
order

```

get All Match (
  {
    category: 'Dinner',
    ts: < 30 min
  }
  {
    likes: > 10000
  }
)
  
```

① Popular tricks.

table

<u>c1</u>	<u>c2</u>	<u>c3</u>	<u>c4</u>

Indices on all column

select \* from table

where c1 > 20

c2 < 100

c3 = —

c4 > —

At max 1 index will be used.

c1	1	2	3	4	5	6	7
c2	a	b	c	x	y	w	f

→ index

c1	3	2	1	7	6	4	5
c2	a	b	c	f	w	x	y

→ index

find me all vals

c1 b/w (3, 5)  
c2 b/w (f, x)



Rowid → hashtag

tweets	
timestamp	value
10:40	Msg id. Hi → id
10:39	
10:10	

#Dinli
=====
Pop
Recent

Sorted  
Index

Recent → fetch Most Recent  
10  
Sorted based  
on time stamp.

Popular	
fr	Like msg r
	500 - Hi - id
	1 - Hey - id
	1000 - Hello -

Popular → top 10 values

20 results per page

1000 results → 50 pages

T1 → 1000 <sup>2</sup>/<sub>2</sub> likes → 2nd.

T2 → 1001 likes → top

60 mins → Reindex with updated counts

11:00

②

## Live Match Scores

Context

→ given a recent match, you have to show the ongoing score information.

→ is the data for a given match → large  
→ small ✓

→ how frequently is the data being updated?

→ v. high → 100 updates/sec

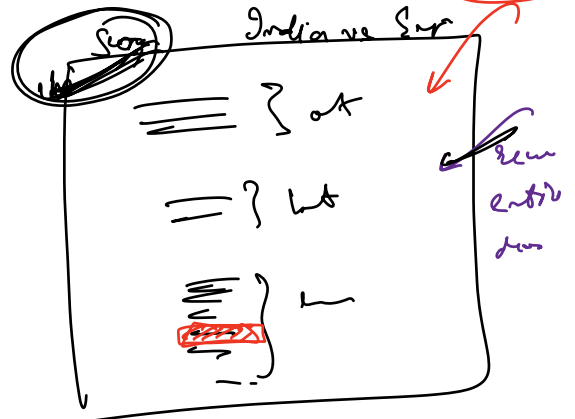
→ high → 1-10 updates/sec

→ low → < 1 update/sec

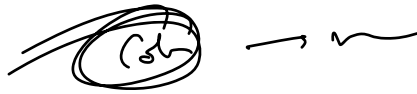
→ v. low → < 1 update/min

→ reads → r. high → 1000 reads/sec ✓

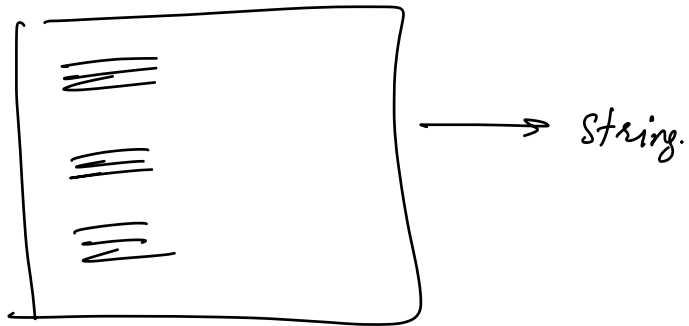
Key-Value Store



COVID - run



DNA



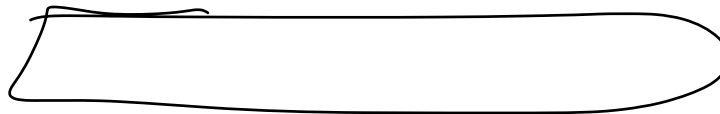
India vs Png  
2022

! 'Nob 9 on : 10 \m 1 \rightarrow 5 \m 2 \rightarrow 10 \m 3 \rightarrow 2 \m  
& p \rightarrow Sah \m 2m — — —'

Ind vs P  
2022



Ind vs P  
2022



3

Uber

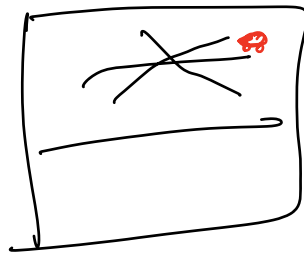
Context:

↳ need to track the live location of cabs.

↳ given a cab → where is it?

Not doing nearest cab query. ✗ → later

① Only want latest location



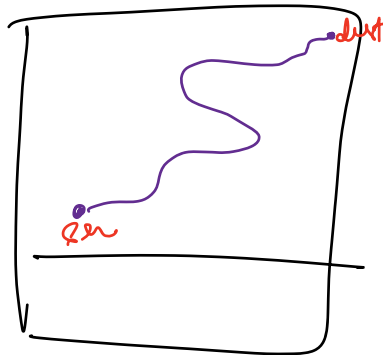
→ current cab location

key-value

key: Cabid

value: location.

② in a ride → route got taken



→ keep track of past location info.

Column database

Row-id: trip-id

Location	
12	12
	⋮

③ track driver.

locid : driver-id

## Doubts

group-chat app

→ Relation DB ?

small data ✓

big data ✗

key-value ✗

group-id :  $\begin{bmatrix} \text{ans} \\ \text{mess} \end{bmatrix}$

docu db ✗

↳ { group-id : [ ] } ✗

↳ { id : —  
group-id : —  
≡ } ✗

fetch all mess (

{ group-id : 123 , timestamp > past 30 min }

)

Column



row-id: group-id

messages	
ts	vals
	<u>msg-id-ct</u>

not  
rew

get all m → gr

10

↑ 10 ltr

Stack over

/ reddit

↳ question  
↳ user  
↳ comment  
↳ votes  
↳ reports

MySQL

user 1 ———+ question

user 1 ———+ comment

user 1 ———+ votes

ques 1 ———+ votes

user 1 ———+ c

c 1 ———+ v

v 1 ———+ R

c 1 ———+ R

$U' \rightarrow *R$

Score  $\rightarrow$  High Read fr

$\rightarrow$  adv. feat  $\rightarrow$

stemming / lematization /

word vector

title  $\rightarrow$  row w

$\rightarrow$   $\rightarrow$   $\rightarrow$   $\rightarrow$

ACID



Consistency



CAP



Consistency

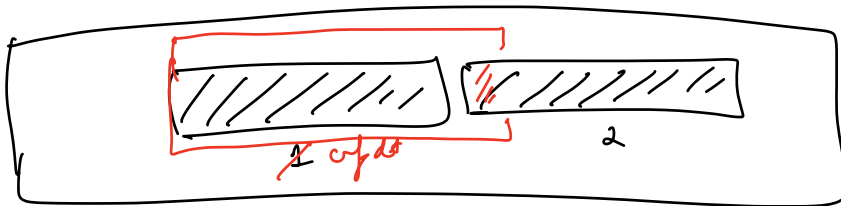
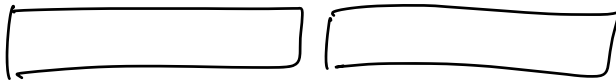


NoSQL  $\rightarrow$  not clac

distributed

transaction

acq. to



SQL

NoSQL

① primitive data types → int / bool / float / string

'\_' 'true' '\_'

② Array (dict) / sets

→ immutable / mutable

Postgres / MySQL

Redis

Mongo / Elasticsearch

Cassandra / Hbase

UUID v4

↳ randomly generated id

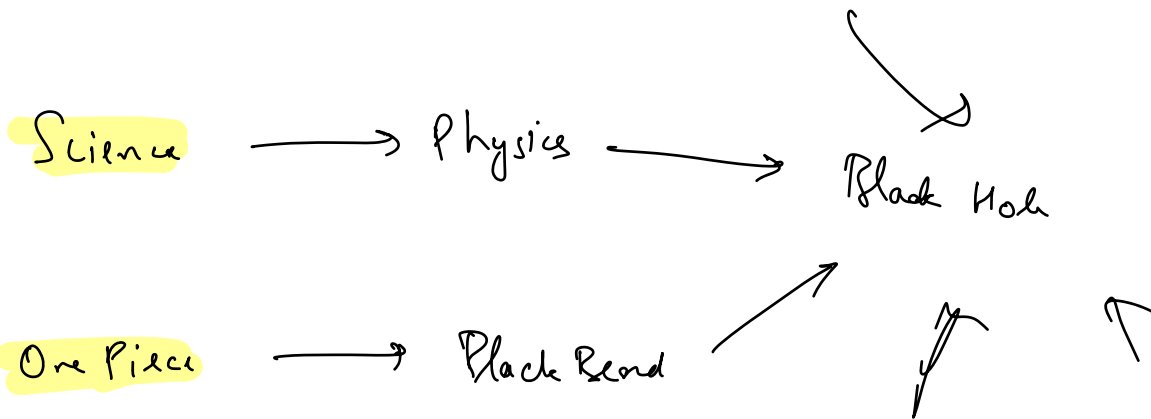
↳ 1 billion prop  
each cannot 1 billion  
objects

↳ chance 1 collision  
< 50%

given a graph → find the shortest path b/w  
two nodes



↳ Mind.wiki



fits in all corner childn of the two nodes.

Mind.wiki