Agenda.

- → SQL vs NoSQL.
- → How to choose sharding key?
- → Types of NoSQL Databases.

⇒ NoSQL DB.

NoSQL ≠ Don't use SQL.

NoSQL ↳ Not Only SQL.

⇒ Don't Jump on NoSQL DB directly

⇒ Our de-facto choice should always be Relational or SQL DB.

⇒ Move to NoSQL iff we are able to justify that why SQL DB won't work?

NoSQL.

↳ BASE.

- ↳ Basically Available : High Availability.
- ↳ Soft State :
- ↳ Eventually Consistent.

Horizontally Scalable
$\hookrightarrow$ SHARDING.

⇒ NoSQL DBs supports sharding automatically, they are built with Horizontally scale in mind.

⇒ But SQL DBs requires manual sharding, there's No inbuilt support for sharding.

Denormalization.

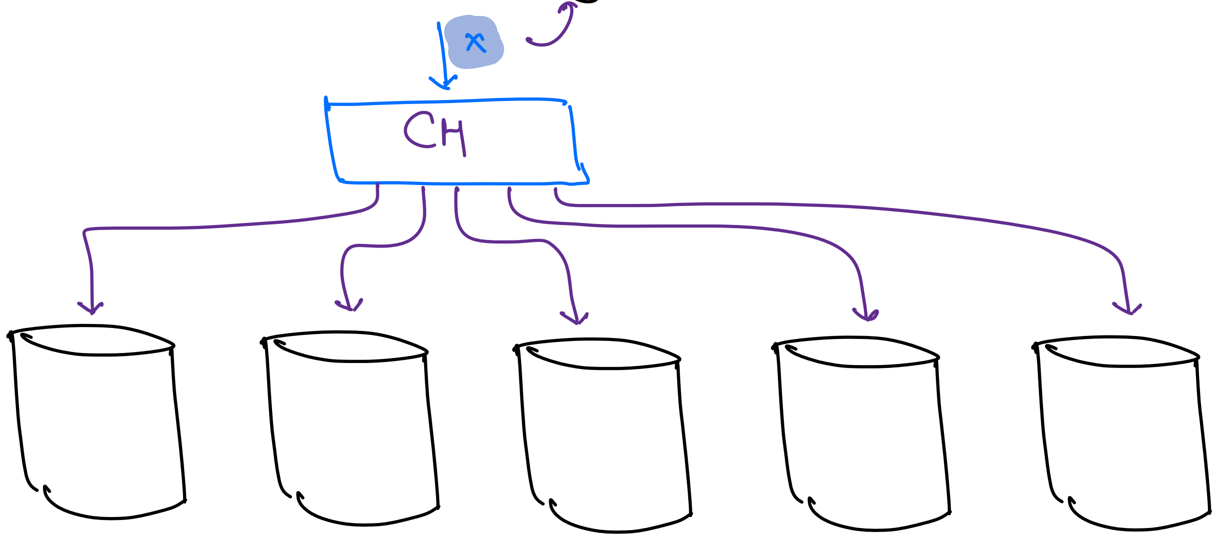
⇒ SQL DB discourages denormalization & redundancy.

NoSQL DB

→ Anyways we need to show denormalized data to the user then why can't we store this data itsely in the DB.

⇒ Weakness of NoSQL.

How to Choose Sharding Key ?



Sharding Key decides how our data|query gets distributed across various DB servers.

⇒ Sharding Key vs Primary key

Key used to uniquely identify the row.

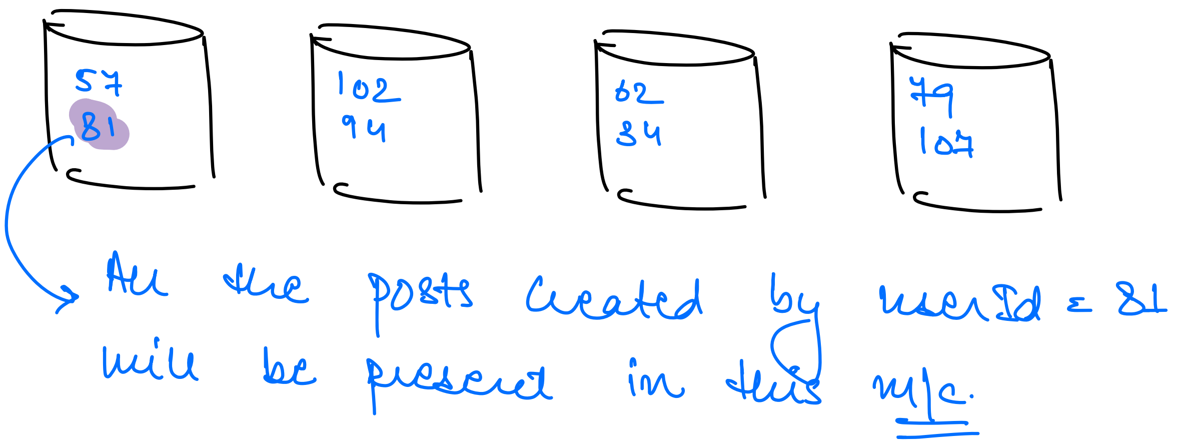
Sharding & Primary keys may or may not be same.

Properties of a Good Sharding key

1) Equal load distribution.

FB posts DB.

↳ userId : Sharding key


57
81
102
94
62
34
79
107
All the posts created by userId = 81 will be present in this m/c.

⇒ All the posts from a particular userId goes to the same m/c.

2) High Cardinality.

age ∈ [0-150] ✗

userId ⇒ 64B.

3) Part of the every request.

4) No fan out.

⇒ Most frequent queries shouldn't hit lot of servers. (It should as minimum no. of servers as possible).

5) Immutability

⇒ Ideally sharding key shouldn't change because we'll have to do lot of data migration if sharding key changes.

⇒ Banking Example

Users can have accounts across multiple cities.

Most frequent operations.

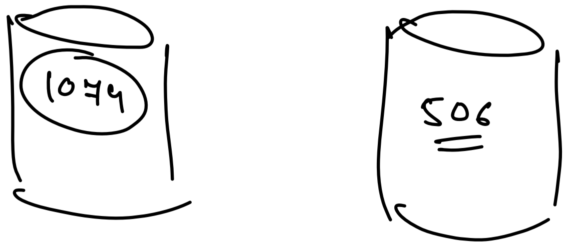
1) Balance query (userId)
2) fetch transaction history (userId)
3) fetch list of accounts of a user. (userId)
4) Create new transactions. (sender-id, receiver-id, amount)

City_id.

txn_id.

↳ Not in the request.

userId.



Balance query ⇒ 1

fetch transaction history ⇒ 1

fetch list of accounts of a user. ⇒ 1

Create new transactions. ⇒ 2

timestamp.

↳ Never choose timestamp as sharding key.

→ Unequal load distribution

→ Not part of the request.

Ride Booking (Uber)

Most frequent use case is to search for nearby drivers.

↳ input : location of the user.

driver_id.

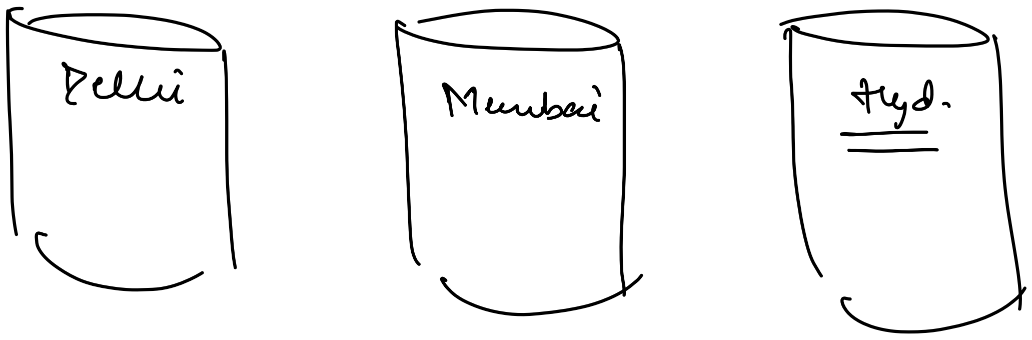
↳ Not part of the request.

user_id

↳ ✗

city_id

↳ Single shard query.



IRCTC.

↳ Prevent double booking of a seat.

↳ Handle the peak traffic during tatkal booking (~20-50x) book_ticket(userId, trainId, seats, date of journey, ----)

ticketId.

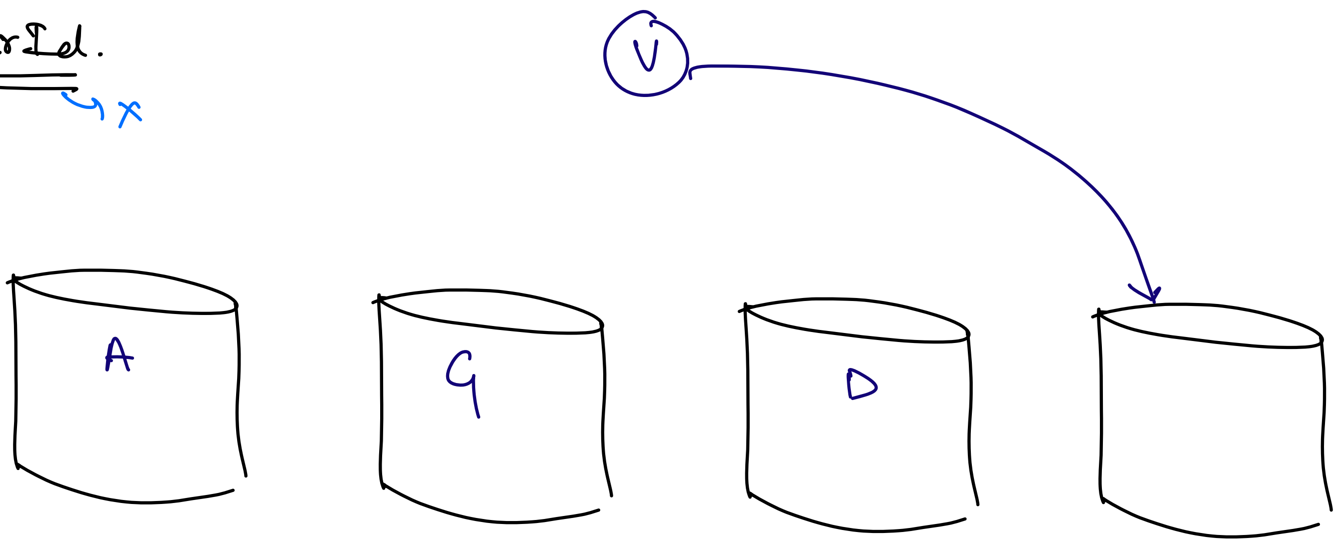
↳ Not part of the request.

dateoftravel.

↳ X

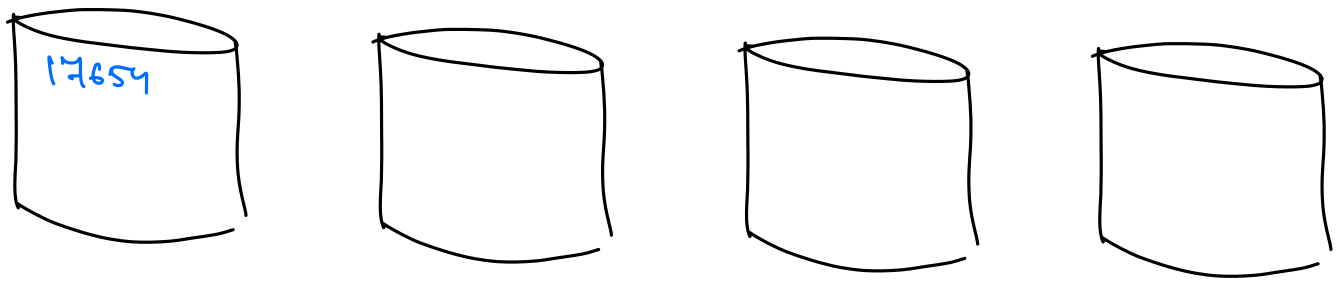
userId.

↳ x



trainId.

↳ ✓



——— * ———