LEC 19 | Database Scaling patterns

Aim: Step by step manner, when to choose which scaling option. which scaling option is feasible practically at the moment.

* Case study.
  - Cab booking App
  - Two startup
  - ~ 10 customers onboard.
  - a single small machine DB stores all customers, trips, locations booking data and customer trip history.
  - ~ 1 trip booking in 5 min

* Suppose app became popular & 10 booking / min ho rahi

Then feedback aaya ki app slow chl rha h pta kiya th problem API Latency mil rha

* Your app becoming famous, but... the problem begins.

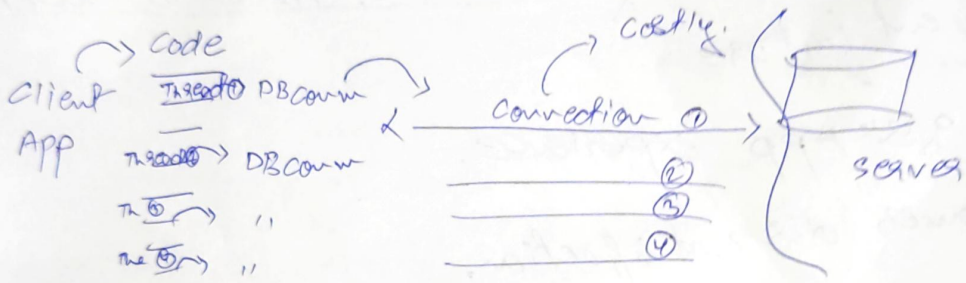
- Req. scales upto 30 booking per minute.
- Your fancy DB system has started performing poorly.
- API latency has increased a lot.
- Transactions facing Deadlock, starvation and frequent failure.
- Sluggish App experience.
- Customer dissatisfaction.

* Is there any solution?

- We have to apply some kind of performance optimisation measures, we might have to scale our system going forward.

* Pattern 1. | Query optimisation & connection pool implementation

- Cache frequently used non-dynamic data like booking history, payment history, user profile etc.
- Introduce DB redundancy (or may be use NoSQL)
  => less response time
- use connection pool libraries to cache DB connection



Jb bhi koi connection create hota h ye costly hota h in terms of time complexity. So, it better ki, ~~baar baar new~~ code me jb bhi DB conn. ki call ho th baar baar new connection establish na karo so its better ki ek connection ka pool bana ke rakh lo connection ①, ②, ③, ④

as & when bhi connection h ye inhi threads ko baar baar reuse kr le

=> Jo bhi performance ka issue aa rha h Connection create krne me wo gyab ho jaega.

Jo bhi library ko use kr rhe h, un sab library me connection pool ki libraries aati h un ko use kr ke cache DB connection.

- multiple application threads can use same DB connection.
& Good optimisation as of now
- Scaled the business to one more city, and now getting ~100 booking/minute

And then phir se wahi problems wapas aagye
Ib booking/min increase hua tb.

Now,

* Pattern (2) vertical scaling & scale-up.
- upgrading our initial hwy machine.
- RAM by 2x and SSD by 3x etc.
- scale up is pocket friendly till a point only.
- More you scale up, cost increase exponentially.
- Good optimisation as of now
& then chlo as growth hui phir
- Business is growing, you decided to scale it to 3 more cities and now getting 300 booking per minute.

& then phir se problems aagyi lekin ab scale up th nhi kr ske kyuki us wqt th bhut paisa jata h th ab ek better option dhkna hoga.

* Pattern 3 | Command Query Responsibility Segregation / CQRS

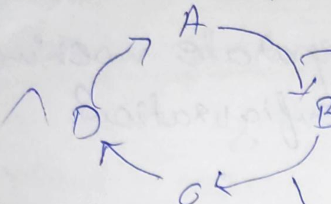
- The scaled up big machines is not able to handle all read/write requests.
- Separate read/write operations physical machine wise

- All read queries to replicas. ⇒ Read mai lag but write lekin consistent rhe
- All write queries to primary
- Business is growing, you decided to scale it to 2 more cities.
- Primary is not able to handle all write requests.
- Lag b/w primary and replica is impacting user experience

ab agar or scale krtch th Pattern 3 bhi hoga tb.
next Pattern

* Pattern ④ / Multi primary replication

- why not distribute write req. to replica also?
- All machines can work as primary & replicas.



④ B jo A se replicate karega

C jo B se replicate karega

Ab agar koi bhi write req aai th ye kisi bhi random node ko de dega (any node will handle it).

i.e agar N req. aai th N req distribute hojaengi th fir thoi thoi time pe replication chl hi rhi h

ar read karne keliye read ko broadcast karenge

i.e charo node wae se phile reply kiya usko chli jaegi

- Multi primary configuration is a logical circular ring.
- write to any Node.
- ~~write data to any node.~~
- Read data from any node that replies to the broadcast fast

- How scale to 5 more cities & your system is in pain again.
  ~ 50 req/s

  then

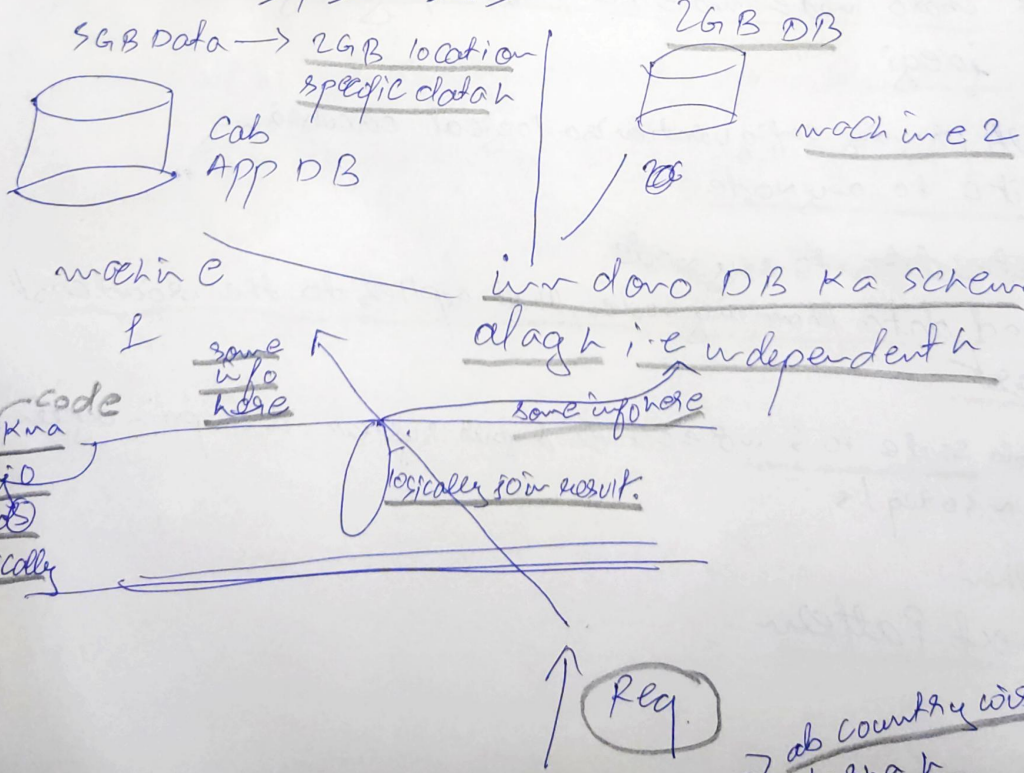
  next Pattern

* Pattern (5) Partitioning of Data Functionality

- What about separating the location tables in separate DB schema?
- What about putting the DB in separate machine with primary replica & multi primary configuration?

→ ie jo phle repo mai separate DB schema bana h usko separate machine mai daal do. with Prim rep & multi confi

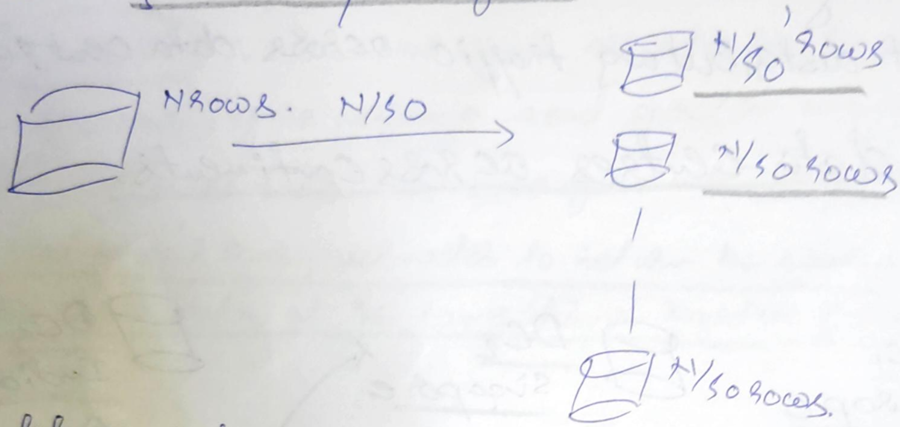
- Diff DB can host data categorised by different functionality.
- Backend application layer has to take responsibility to join the results.
- ~~Planning to expand your~~



* Pattern ⑥ | Horizontal Scaling or scale out

- Sharding – multiple shards.

Allocate 50 machines – all having same DB schema – each machine just hold a part of data



- Locality of data should be there.
  Matlab each machine mai local data hona chahiye taki machine mai switch na karna pade.

- Each machine can have their own replica, maybe used in failure recovery.

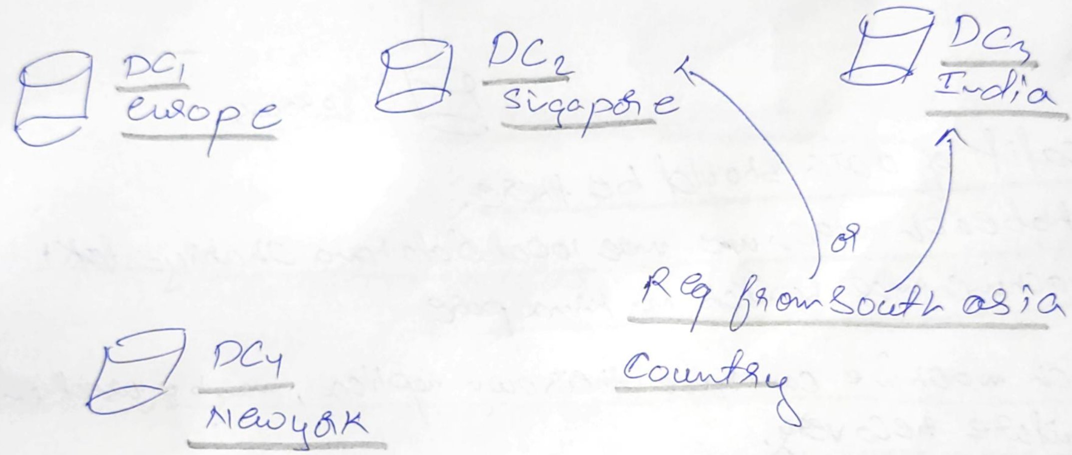
- Sharding is generally hard to apply { No pain No Gain }

{ Now scaling the business across continents.
then again jaha server h woo thik h lekin bahar log probs no areth ab next pattern.

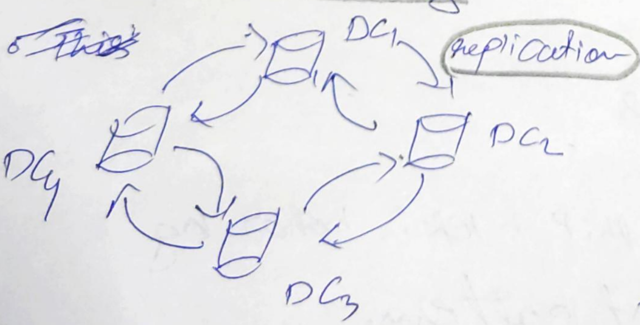
★ Pattern 7 | Data centre wise partition

- Requests travelling across continents are having high latency.
- What about distributing traffic across data centres?

⇒ Create data centres across continents.



- Enables cross data centre replication which helps disaster recovery.



Yani ki kuch kuch time mae inn DC aapas mae replication krte rhenge data ki

⇒ agar europe ka DC nhi chlta for some reason the public of europe's req wo Sigapor & India bhej skte h

yes, latency will come but availability will be high

But jb DC europe sahi ho jaega then again normal