Agenda.

→ SQL (vs) NOSQL DBs.

↓ Relational ↳ Non Relational

SQL. : Structured Query Language

⇒ RDBMS : Relational DBMS.

↳ MySQL | Postgre | MSSQL | - - - -

↳ Stores data in the form of tables.

↳ tables are related to each other.

⇒ Well defined & Strong schema

students

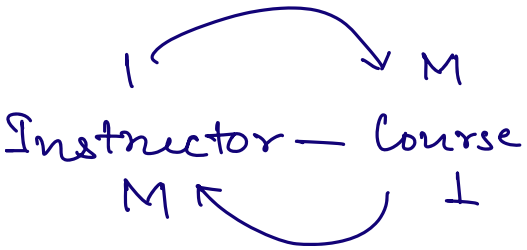
id	name	email	.....	

instructors

id	name	email	.....
10	Deepak	—	—

Courses

id	name



⇒ M:M.

instructor_courses

instructor_id	course_id

⇒ ACID.

Atomicity ✓

Consistency

Isolation → Isolation levels.

Durability → Doesn't protect us from HDD failures, we need to create Replicas.
= HDD

ACID Consistency ≠ CAP Consistency.
=

DB constraints are enforced.
- Non/Null
- fk
- ≡

⇒ Normalisation.

Student_Courses

StudentId	Course
1	HLD
2	LLD
3	DSA
4	LLD
2	HLD
2	HLD
1	LLD
=	=

Students

st_id	name	email ....

HLD → System Design

→ Redundant.

Students

st_id	name	email ....

Courses

id	name
1	DSA
2	LLD
3	HLD
=	=

Student_Courses

StudentId	Course_id
1	2
2	1
3	3
4	1
2	2
2	1
1	1
=	1

→ JOINS.

⇒ Scaling is a challenge in SQL DB.

⇒ All the properties of SQL DB we have seen are applicable at low scale.

Single DB m/c.

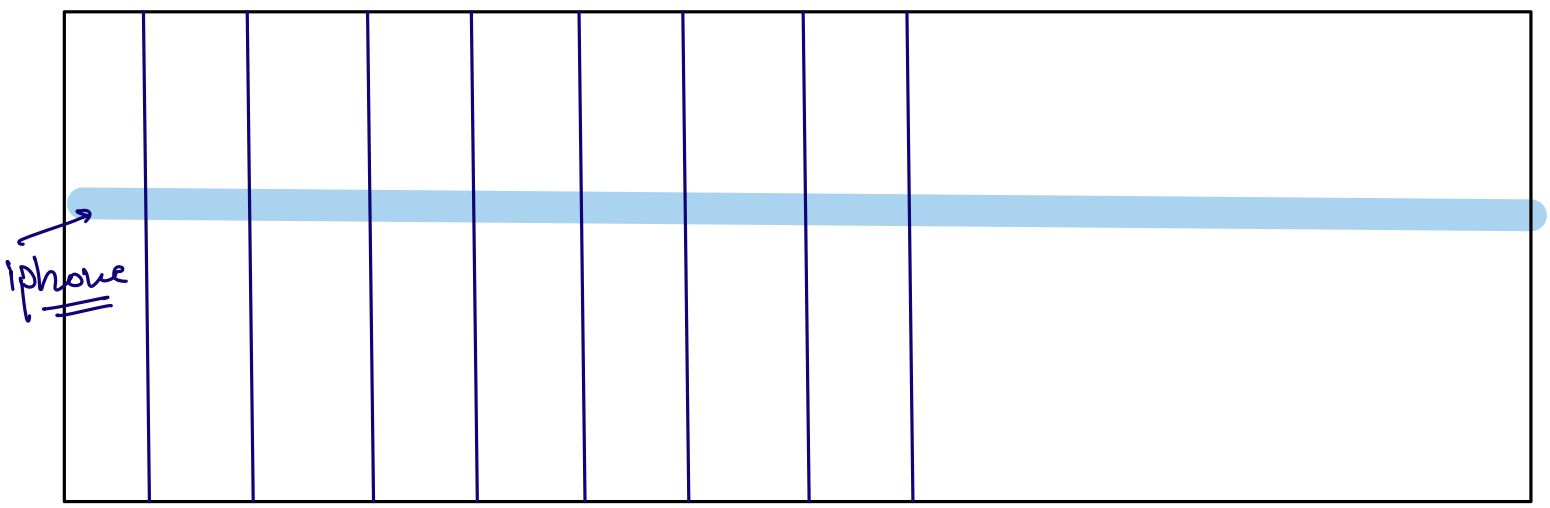
⇒ Amazon

↳ Products. ⇒ No well defined schema.

↓

⇒ 10,000 + Categories.

I)



10000 x 10 = 100,000

↑ # of Categories

↘ avg attr each Category

⇒ Huge space wastage.

II)

products : id, title, desc, ---

mobiles : product_id, --------

laptops : product_id, --------

tshirts : product_id, --------

10K+ tables.

→ JOINS

⇒ NOSQL.

↳ Design for very high Scale.

⇒ Sharding nullifies ACID.

BASE.

→ Basically Available

→ Soft State : Txn can be in soft state for some time.

→ Eventually Consistent.

Scalability.

- SQL DBs requires manual sharding.
- NoSQL DBs provides inbuilt support for sharding.

Denormalization & Redundant Data.

⇒ SHARDING KEY

Primary key : Uniquely identifies the row.

Sharding key : How to distribute the data.

⇒ How to Choose a Sharding key?

1. Equal Data distribution.
2. High Cardinality.
   - Count of possible values.

age ∈ [0, 110]

→ 111 values.

user_id ⇒ 16 B

⇒ 2^{64}

3. Should be part of every request.

4. No fan Outs.
   - Most frequent queries should only hit 1 (or atmost 2/3) DB m/c.

5. Immutable.

Ex-1 : Banking
- User can have multiple accounts across multiple cities.

Most frequent queries.

1) balance(user_id)
2) fetchTxnHistory(user_id, accountId)
3) fetchAllAccounts(user_id)
4) transfer(senderid, rec_id, amount)
   - senderid

→ location ≡ city ✗

→ user_id

→ accountId ✗

→ txnId ✗

→ timestamp. ✗

Uber

↳ getNearestDrivers (location)

→ CityId ✓

IRCTC.

→ Prevent double booking.

→ Tatkal traffic. → 50x traffic.

→ ticketId ✗

→ dateOfTravel.

↳ Timestamp ⇒ NO.

→ userId. → Jan Ont.

trainId

———— * ————