

Last class e covered CAP theorm, master-Slave..

Cap theorm says, in real world system you have to take care 3 things. Consistency: you should be reading the value of latest write…

It’s a spectrum… strong, eventual, weak consistency..(what you write may never visible to user)

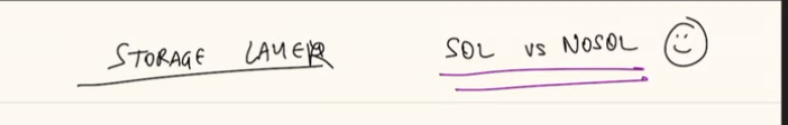
Availableilty: should be available, even the read is not latest version/ previous value.

Partioning Network: if network partition happens, that point you either choose consistency or choose availability.

Storage layer concept: replication..

Spanner: if u want to build a system which has very little chance of NP. Then most case me it will be consistant and available. Google spanner example. He worked on it. When network partition happens.

Today we will talk about different type of storage. Go deeper on storage layer..



Today we will form base on it. When to use SQL or no sql.

Sql is basically is most dominant DB in world for decades. Most mature technology. Despite all the noise around on no-sql still sql is most important.

By design sql is single machine DB. If you have

Back in the day, the data we needed to store was not so big. And hardware easily supported.. then hardware improved.. and data increased. SQL started from a time where data storage was not an issue. As sql emerged as single machine database. It doesn’t have any portioning in it. Banks, ticket booking, or financial data will have some component of SQL for sure. Zeodah etc..

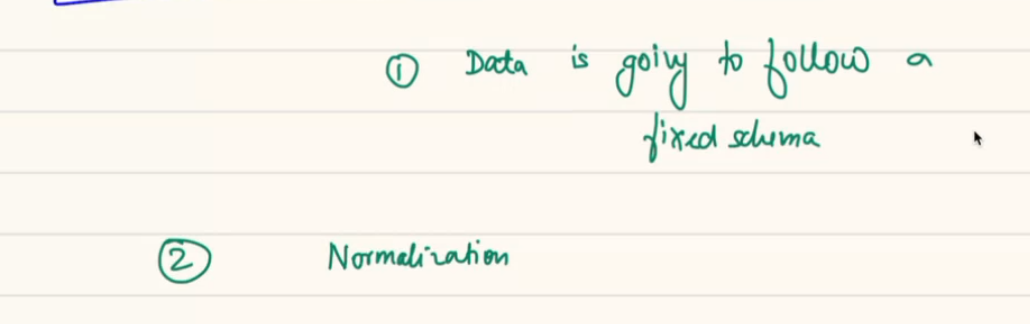
How can we make SQL a multi machine DB. By design SQL is a single machin DB.

SQL wants you to create Tables, table is collection of data which are logically related. Columns are fixed/ rigid. Column will store quality or attributes.. Every row will be one Data Point.

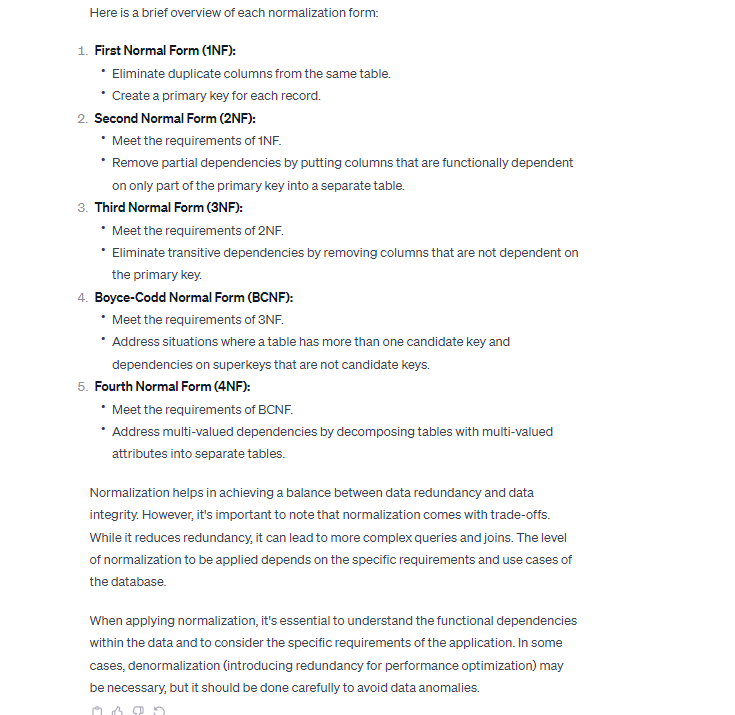
With such setup there are 2 things we should be cognizant about. SQL me data is going to follow a fixed schema. If student table has 5 attributes then each row/ std will have values of 5 values/ columns/ attributes we will fill.. structure needs to be fixed. Every new entry should follow same pattern. Of course we can add or delete a column. We can do it. But adding or deleting is costly process. Entire data need to be reqritten. So lets agree SQL db wont allow every data point to come in different format/ structure.

We require fix schema for underlying data.

GivenSQl wants to store data in single machine. SQL is a big fan of NORMALIZATION.

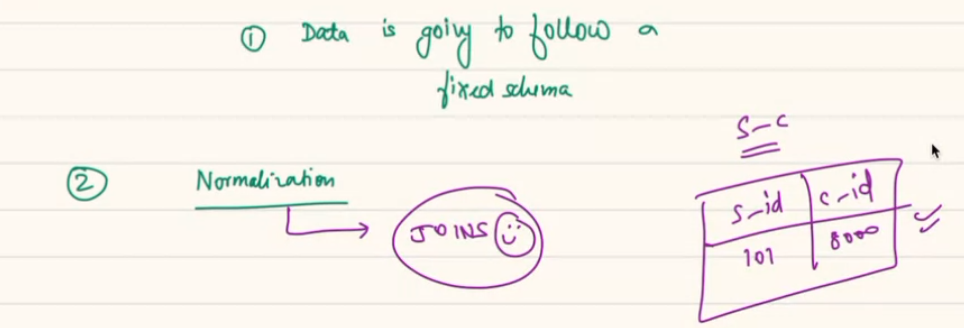


***Normalization is a database design technique that organizes data in a relational database to reduce redundancy and improve data integrity. The normalization process involves dividing large tables into smaller tables and defining relationships between them. This helps eliminate data anomalies and ensures that data is stored efficiently. The most commonly used normalization forms are First Normal Form (1NF), Second Normal Form (2NF), Third Normal Form (3NF), Boyce-Codd Normal Form (BCNF), and Fourth Normal Form (4NF).***

******

If you have a sql dB, it motivates you not to store in duplicate manner . you will break the table down such a way that there is no duplication of data. For e.g. student can take 10 diff courses, diff courses, teachers raher than taking everything in same table, create diff table and add only id. Store relation in smaller format and all data placed differently. Normalized to reduce redundnecy. It directly yields to implement join. The more you normalize and make data lerss redundant and make joins. Even join effects performance, its greate to operate in single machin. As 2 tables are in same machine DB will perform join and give you data. Normalization is possible when you do joins.

Normalization is only possible when you do joins.



SQL also places regulation and SQL gives you ACID guarentees.

If you execute a query it will either execute 100% or not completely. Means Atomic. Your transaction can be atomic.

C- Consistency: not same as CAP theorm. Here it means your DB state remains COnsistant before and after trasnactioon. Once transaction is done your DB will be in same state. Your data wont effect by some % etc. unless n until;l course Id is present in course table then only….

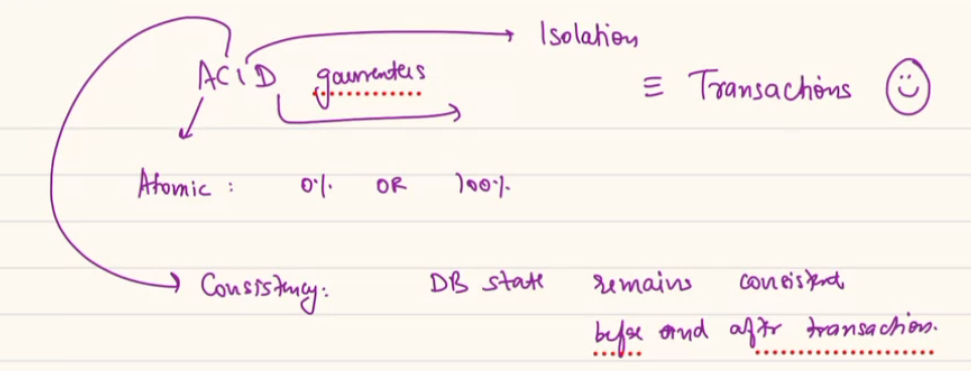
There is a bank, A, B users. Each has 5 cr. If A 🡪 1 cr to B. B should have 6 cr. A is 4. Here before and after the transaction the total amount is 10. Its not that one transaction is done, another is not happening. Across transaction your data is going to be consistant.

Isolation: you have underlying data. Multiple . you can set up SIOLATION level such a way that no transation is going to intervine another transaction. No transaction is going to have effect on another transaction. 1 T is not going to intervine another one.

Durability: your Data once commited. The data is never going to be lost. Data needs to be durable inside the machine. SQL gives you these ACID.

Data can be rolled back. But without you rolled back data is not going to rolled back automatically.

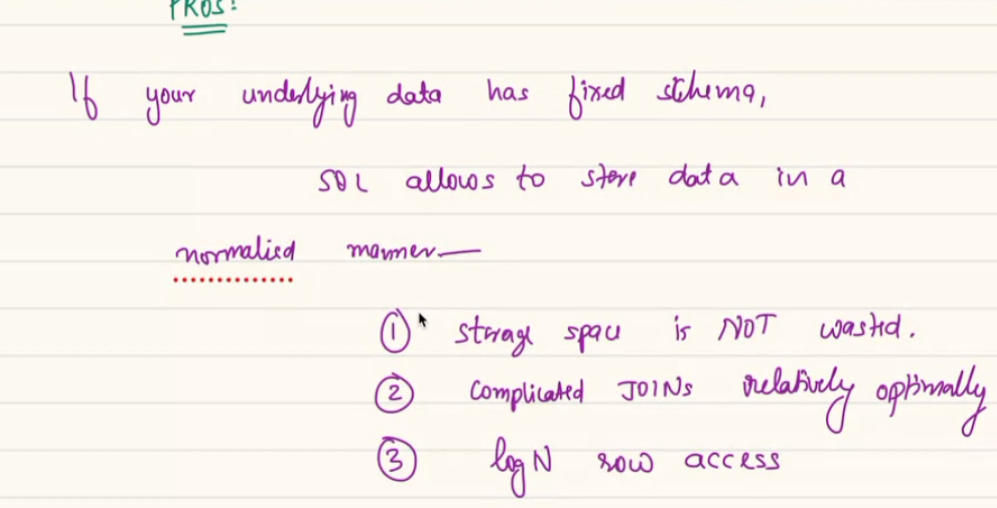
Weak consistency: your data may or may not…



ACID has 4 ppties. Across it mean, when you run a SQL T, that T will happen is a manner that none of these fundamental ppty will be compromised.

PRO:

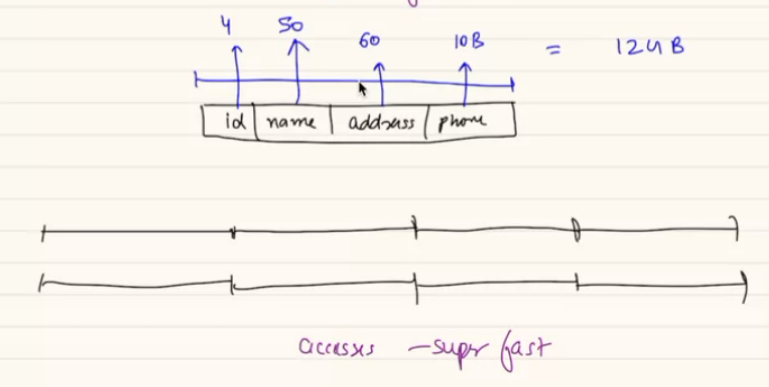
1. Your storage is not wasted. When you use normalization. As SQL is one MC storing nicely is good thing.
2. It can allow you to perform complicated join relatively optimally. As joins are costly. But given it’s a single machine, you can join data and view data very optimally. You can run complicated join easily.
3. Garentee of LogN row access. N is no of rows. Read B/ B+ trees.



Con/ Shortcoming:

For SQL we know before hand a row can cost me this much in bytes.. 124Bytes.

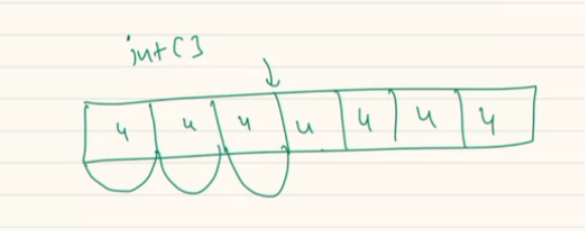
If we row 100 rows then. If you want to go to a row, you can access very quickly. You can recall the array Data structure.



BQ: LogN time access even if there is no index on column ? Doesn't that mean sql db will search every row hence O(N) ? to a value. But for a row you can access easily. Using row id.. but if you want to search by any col, then indexing needed.

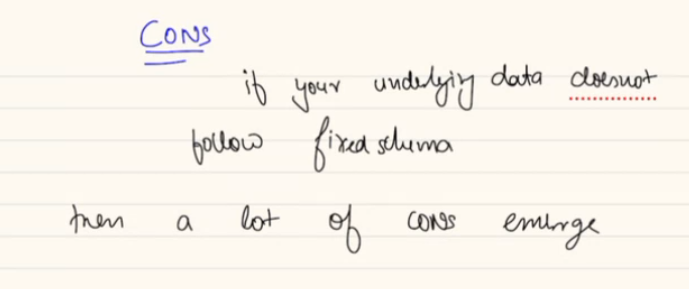
If you want to add a new column in table. It takes 10 bytes. We wil have reqrite the entire data.. modifying the structure is costly.

The way SQL stores internally. It considers every row has a fix size. Say an array each element is 4 byte. To go to 4th elem. You go to 16th byte in HD.



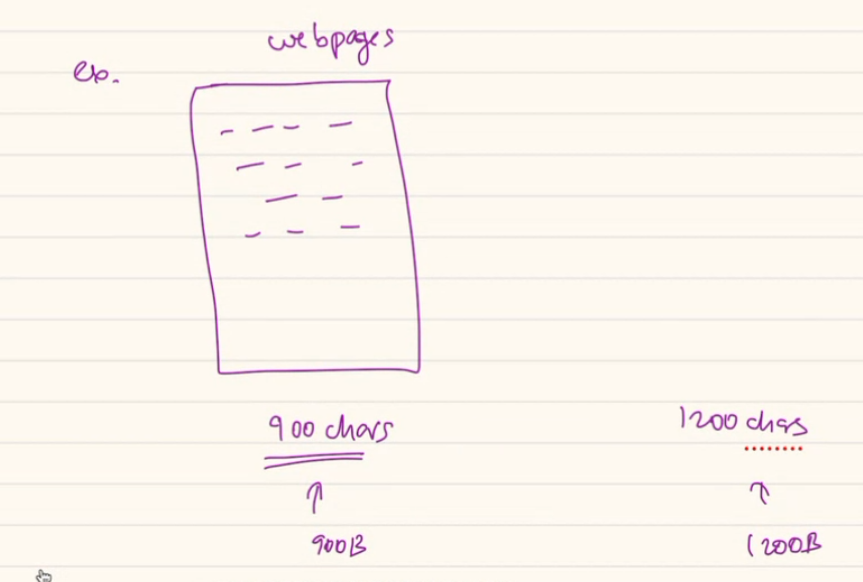
But if you add a new row.. the contiguous is lost. I wil re write. So add and remove is taf jon, resource allocation. You cannot keep adding removing. Downtime will be there.

So this same adv gives disadv.. if your data is such you have problem.

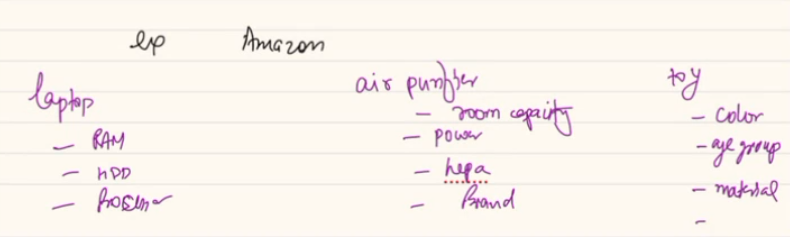


If u try to store Webpage: a webpage will have image, word, paragraph,

Say a page has 900 char = 900 bytes. Other poage = 1200.. given no limit how big a page can be you have to decide upper limit. So each col has to reserve this size.. sql stops performing well when underlying data is big.

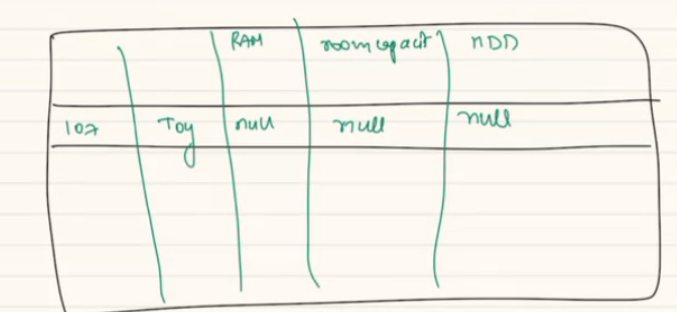


Amazon has many product. Attribute of each prod will be different.



So when you try to create a sql product table. You will have 1000 of column as acresss product you will have many attributes you will have to track and many will be null.

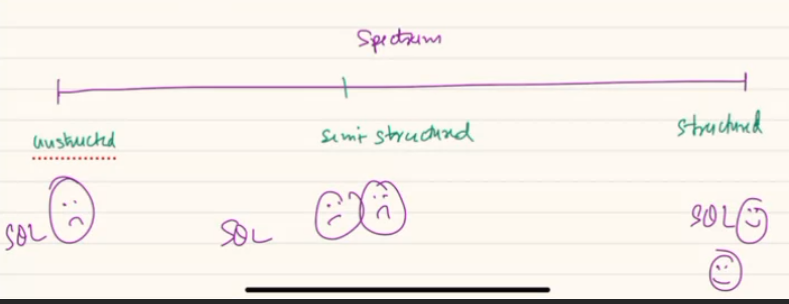
A toy null for RAM. Null of HDD a toy. Same way..



Even if yu create table for each product then ho wmany table we wil create. So when data is not structured SQL looses its Advantage.

Lets look at a spectrum.

If your underlying data is very structured, semi-structured. And unstructured.



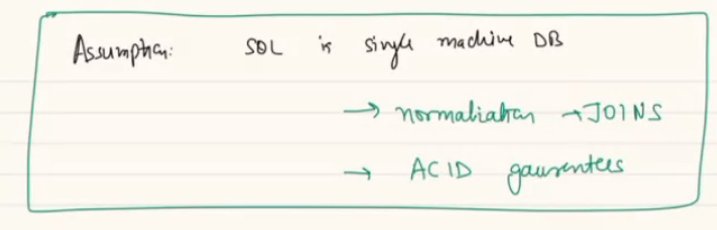
***Lets look at a thing which is going to be misunderstood to many people in the world…***

People says SQL is good if you have less data. But if you have big data sql is not good. He wil share the video is whatsapp.

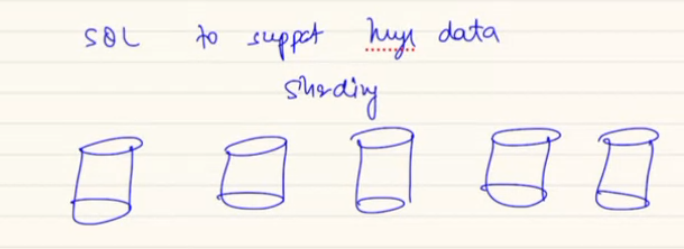
One thing, SQL is build with an assumption that everyting is going to stored/fit inside same machine. Only because of the assumption that SQL is single machine db. SQL gave you these features like normalizations,,, as SQL asum you will have everyting in same machine.

ACID ppty guarentee.

If a machine can store 4TB data then you cant store more data in here.



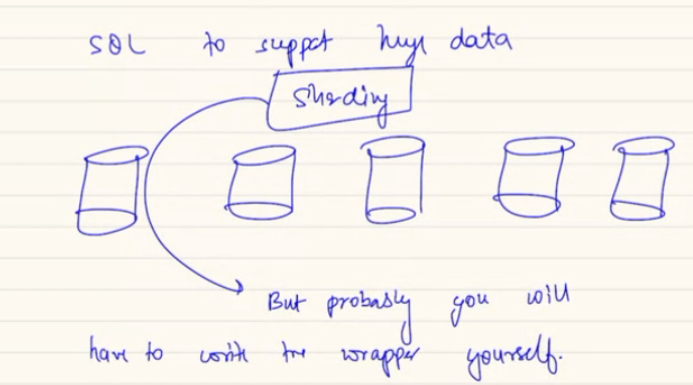
SQL can big data. For that you can SHARDING. Rather then employing one machine you can add more machines together in a cluster then SQL will be able to Scale.



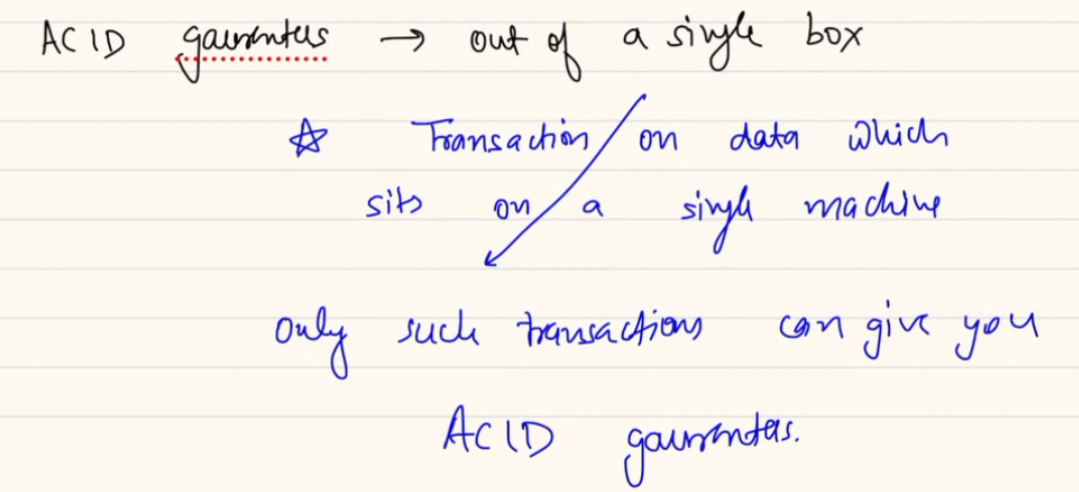
Usually SQL machine do not come with an configuration a sto run as sharding. To use you have to work hard and write Wrapper, write reverse proxy etc... Sql do not come in build for sharding..

Usually you have to write these things.. but you can get from open sorce library.

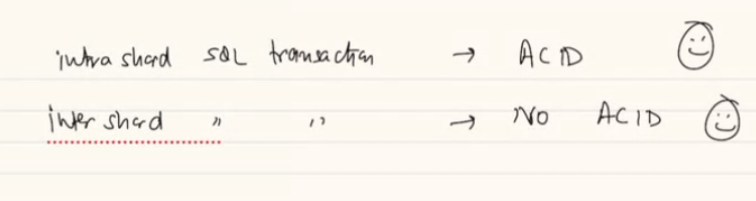
SQL can be sharded but you have to do heavy lifting.. so we have to add a load balancer between server and database. To support huge data in SQL you have to do SHARDING.



2. the ACID ppties you were thinking, they only come out of a single box. If you have partition across multiple machine. Data is spread, and you runa transaction, you cant get ACID ppties. Which means the transaction of data which sits on a single machine. Only such transaction can give you ACID guarenties.

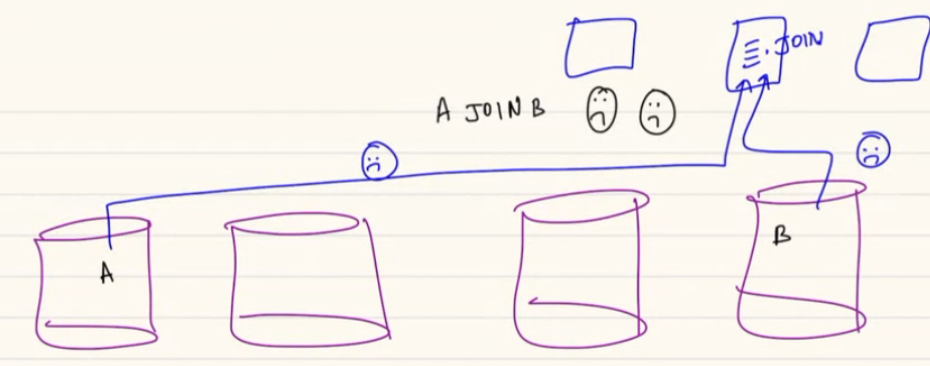


Means if you are sharding you have to be sure the sharding key, that if you want ACID ppties that data should be a intra shard queries. Inter shard queries cannot give you CID guarentee.



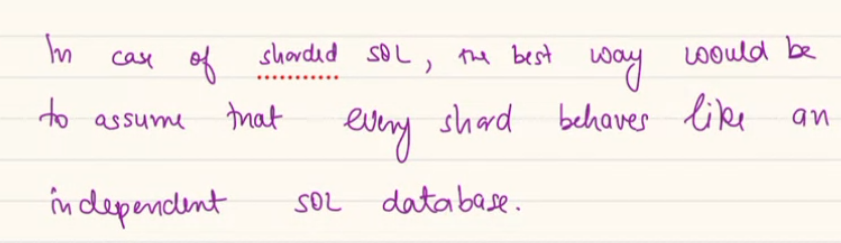
You have to shard table in a manner. The queries you want in acid . those data should sit in same machine.

I have student 107, 11, 200, 400, 600 in diff machine. Same way data of course of 1000, 2000, 3000 in diff machine. If I run a query which will have a transaction which included 400 600s 1000c.. then I can get ACID.



Atomicity: a transaction wither will go completely or not at all.. but diff machine me one is up diff is not. Then you have to deploy something then will have to do 2 phase commits.

Whenever you have a sharded SQL assume every shard is acting as a independent SQL DB as its OWN. Given a single can give ACID so one SHARD can give you ACID ppties.

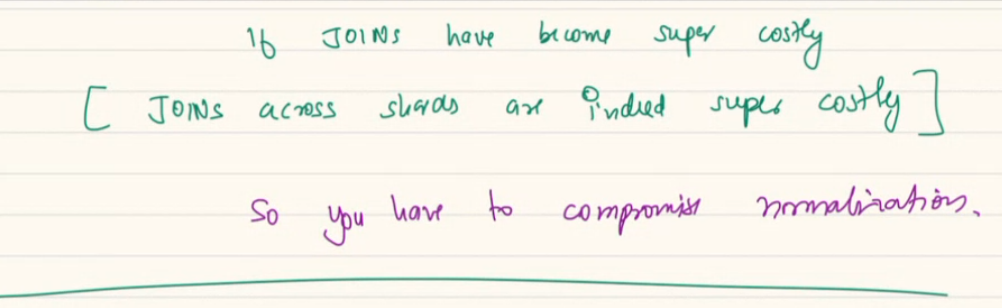


If you have a Sharded machine. You Normalizes, and data is siting in different machines. You have to copy data from A to app server B to app server then in APP server you perform join. We do have DB links. Its costly(slow). We cant do regular join in DB link. We can do some cases not on every query.

Read DB link… over network you will talk multiple machine. Requires temporary data copying.

If joins are costly you have to compromise normalization.. you should start duplicating the data.

e.g: in last class, facebook news feed. Is A is friend of B, we put B is also a friend of A. so I can keep data in same table. You have to compromise normalization as you had sharded your DB. This is Denormalization..



Normalization is good, but in certain case you have to prefer denormalized data as its best to use.

one benefit of normalization is that we have to update in less places. But at times of reading/ joins you have to do costly action. You copy data in diff places so your read is easier.

In reality people study and learn as SQL is totally different concept/ entities all together. But concept is remains same only the diff what are the decisions the DB designers have taken.

SQL: structured

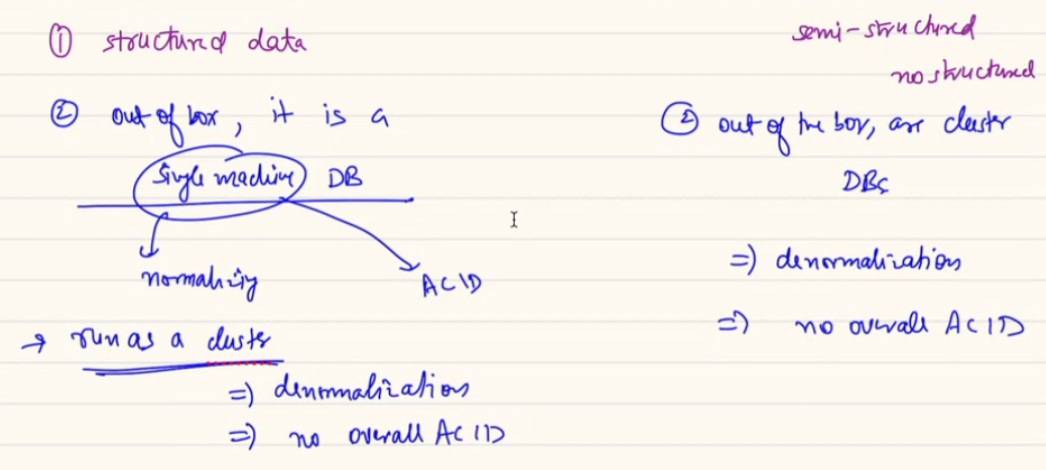
No-SQL: semi or no structured. No sql = any db that doesn’t follow sql design..

Sql: out of box it is a single machine db.

No-sql: out of box are cluster db. Based on assumption, they will have multiple machines together. They will run across machines in a cluster.

SQL: if I have normalized. And run as cluster I will have to denormalized. If sql has to run across machine we will do denormalization. Doesn’t give ACID guarentee.

No-sql: no over ACID.



In reality: sql and no-sql is itself a spectrum. Means there is no boundary that

These days in sql we store JSON(key-value) this is not a write a way of storing as JSON can have many value. Means you are trying to use SQL as a no sql. Which means SQL , no sql is a spectrum. Depending how you want to use it.

Storing json in sql is not purest way of using SQL.



Why people do that?

The database designer are in business of making money. If I create a DB I want o sell. So I wil make so many configuration. So my sales team has easy way to sell it to user.

If my team has 10 engineer. They know SQL. But if I have JSON to store. I wont ask them to learn mongo. Start up uses mongo even if they don’t need to. May be tech lead knew mongo so they are using mongo.. there is a cost to it. We as software architect should realize. If you use a DB in a manner the way it was not meant to. Its not optimal.

Facebook uses SQl, no sql, redis, cassendra. They use collection of db together. Small company don’t want to spend, keep use same DB in diff ways which is not best way to use.

By design sql is single machine DB.

With normalization comes joins, joins in diff mchine is costly.

No SQL INTRO:

Anything which is not SQL. There are diff kind of..

1st kind: key value pair (redis, memcached): you have a DB, no underlying structure. But they just follow a simple key value pair. Like MAP, hashmap. This datastrucure in scale.

How do u decide which k will go to which Shard. You can use CH here. Sharding key will be the kwy of data itself.

J