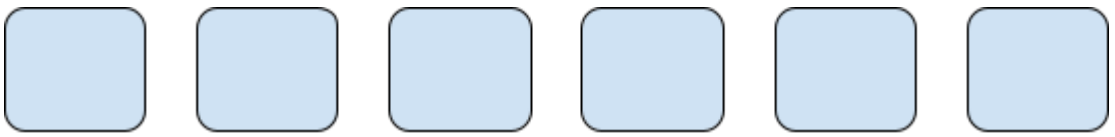
Sharding

Sharding

While we are horizontally scaling our system into multiple systems, we need to make replicas of the database as well, but let's consider a bigger system where we have tons of users and their data like social network or banking system. We can't replicate all the data on all servers as an optimum approach. This is going to cost so much and wouldn't be helpful.

So, In such scenarios, we split up a database into parts, such that one part of the database is stored in one server and the other part is stored in another server on the basis of any logical thought process.



Shards

Does Sharding works in all cases? No, Sometimes if any server goes down we don't have replicas of shards, In that case we have replicas of db present as saviour to take over following master slave architecture.

Example- Let's consider a large table where we have all student's information for an year, Here we have Name, adm no., results, personal information etc. Now thought process comes and we are accessing the Result page only once or twice in a year, So do

we need to store result for each student in same table? No, we don't need to access result every time, So we can split table by columns and can store another table named result somewhere else. This is a general use case of Sharding with vertical partitioning.

Types of Sharding-

1. **Key-Based Sharding-** In key based sharding we basically partition table on the basis of a shard key. Shard key is not primary key it's basically the key which is pointing to feature of table. A primary key can be shard key but vice versa is not true.
   We need to be careful in choosing shard key as it can lead to inconsistency if user changes that data frequently, like cityName, User can change cityName at any time, but it's also significant in case of Tinder like application where we want to put such people together.
   For example- We have a user table having users detail who are visiting our website in last one year. Now we can choose userIds as shard key and make a hashfunction to map user detail to any shards based on shard value.

UserId(shard key)	Name	Address	Login time
101	ABC	--	
102	PQR	--	
103	XYZ	--	

   After mapping we will have a shard value corresponding to every key and that row will be stored in that shard only. Hash Function will always return same shard value, so our code will be consistent and it knows which user's detail is placed where. It's also known as algorithmic sharding where we aren't saving meta info of tables but our algorithm know about it.
   Advantage- We can evenly distribute data in all shards so that load is same on all shards.
   Disadvantage- Let's suppose, data is increased and we have to make a new shard but our hash function is returning same shard values, so in that case we have to change hash function and data needs to be migrate to new shards.

2. **Range-Based Sharding-** We apply range based sharding when we want to split data in terms of a definite range on any field like date range month wise or quarterly.
   We create partitions of data based upon that date range. So that Entries of that date will be present in shard for made for that range only.

For Ex- We want to store transaction quarterly so we can made a shard for each quarter and txn date falling in particular quarter will be stored in shard made for that quarter

Txn Id	Date	Shard
101	21/03/2021	Shard 2
102	01/04/2021	Shard 3
103	31/12/2021	Shard 1

3. **Directory-Based Sharding(Dynamic Sharding)-** We apply directory based sharding when we have a fixed number of entries in any column(contienent, countries, states, zones). We basically store each zone in a new shards and map them in a lookup table using which we can check which shard contains entries of which zone.
Example: Govt. data storage, Social Network websites, MNC having services in many countries.

User Id	Name	Country
340	ABC	India
341	DEF	US
342	PQR	Canada

User Table

Country	Shard
India	Shard 1
US	Shard 2
Canada	Shard 3

Look-Up Table