What is database sharding?

By Avash Mitra

How would you make your relational databases scalable and available and what are the trade-offs that you should keep in mind?

- Database sharding is the process of dividing the data into partitions which can then be stored in multiple database instances.
- It uses some key to partition the data. This key is an attribute of the data that you are storing.

Working

Suppose, there are 1000 users in your database and you have 5 database servers. You want to shard the data on userID. So you can partition the data in the following manner

- userID 000–199 -> database 1.
- userID 200–399 -> database 2.
- userID 400–599 -> database 3.
- userID 600–799 -> database 4.
- userID 800–999 -> database 5.

Now if the userID 546 wants to perform read/write operations, he will only connect to database instance-3. And since there are only 200 userIDs, query processing will be fast.

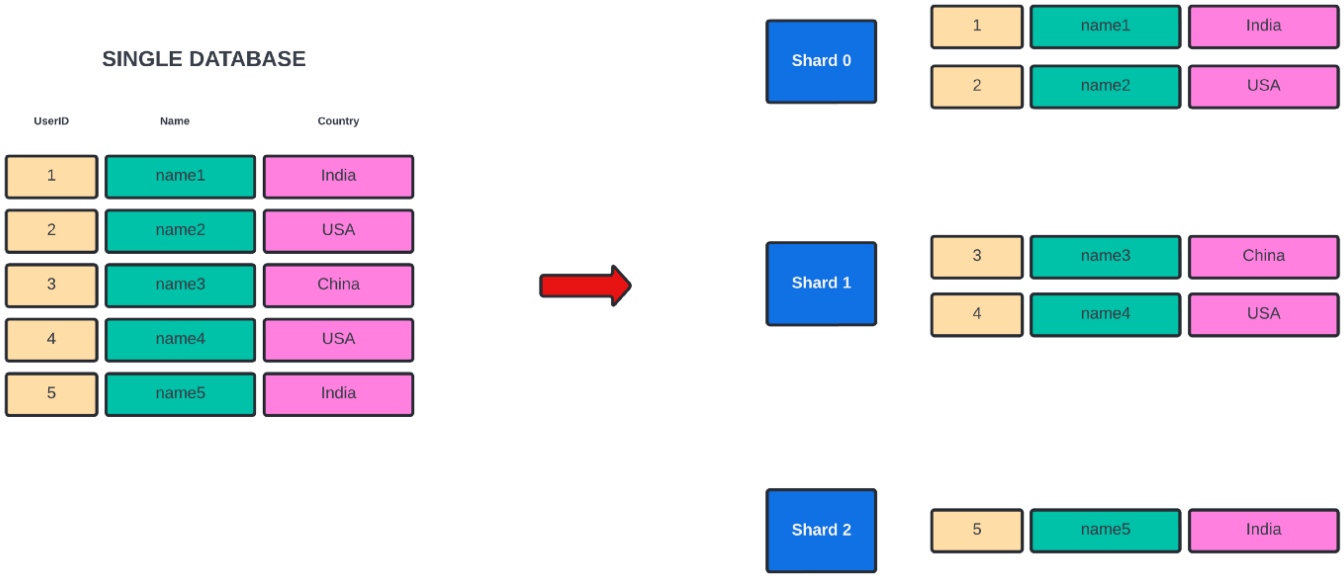
Note: This is an example of range-based sharding.

Types of Sharding Architectures

Range Based Sharding

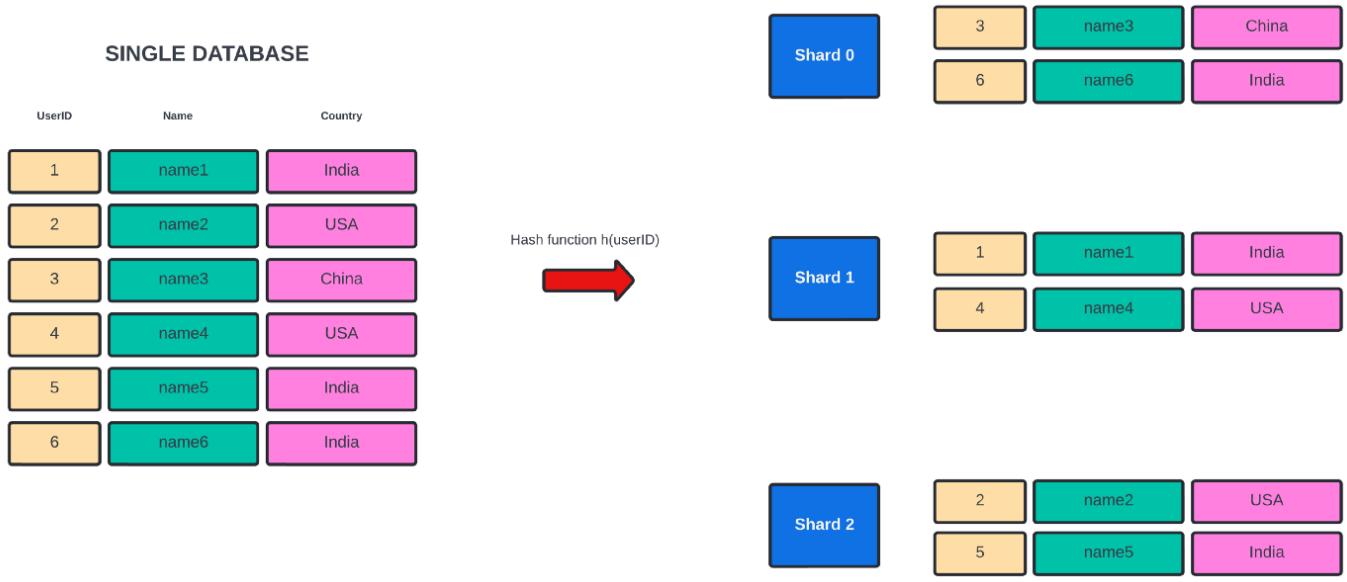
- In this method, we partition the data based on the ranges of the key.
- It is very easy to implement

- Data may not be evenly distributed across shards.
- In the example below there are 5 tuples and 3 shards.



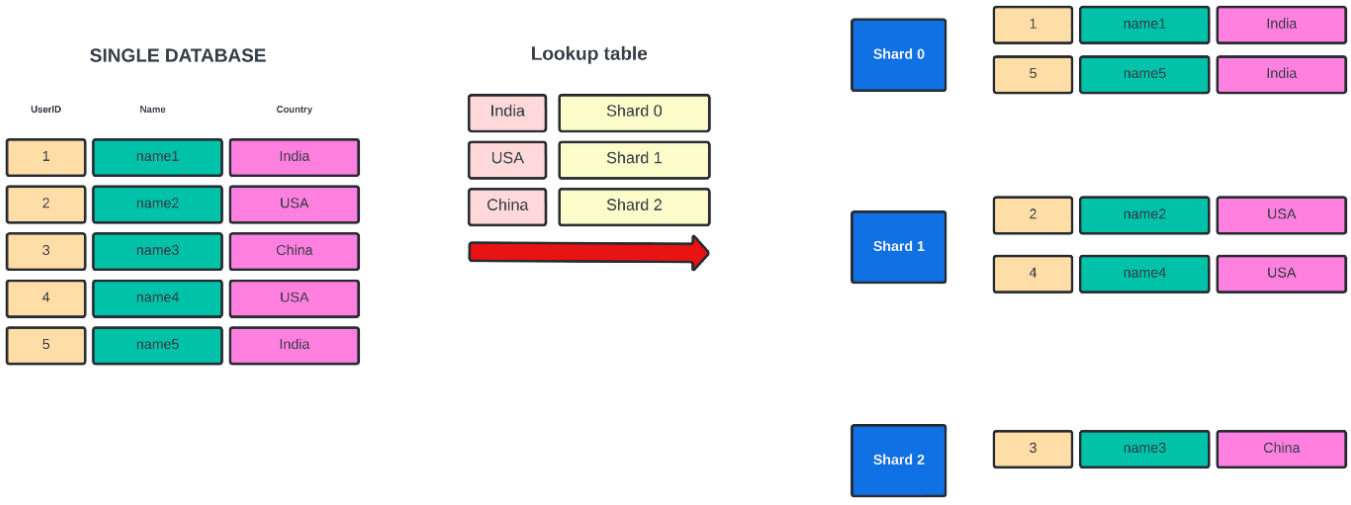
Key Based Sharding / Hash-Based Sharding

- In this method, we generate a hash value of the key (Here key is one of the attributes of the data). This hash value determines the shard we will use to store the data.
- Using the simple hash function to distribute data can cause skewed distribution. To overcome this we can use Consistent Hashing.
- In the example below there are 6 tuples and 3 shards. We have used a simple hash function `h(x) = x%3`



Directory-Based Sharding

- In this method, we create a lookup table that uses a shared key to check which shard holds which data. The lookup maps each key to the shard.
- It is more flexible than range and key-based sharding.
- The lookup table is a single point of failure.



Difference between Horizontal Partitioning and Sharding

- In Horizontal Partitioning, we split the table into multiple tables in the same database instance whereas in sharding we split the table into multiple tables across multiple database instances.

- In Horizontal partitioning, we use the same database instance so the names of the partitioned tables have to be different. In sharding, since the tables are stored in different database instances, table names can be the same.

Advantages of Sharding

- High availability Even if one shard crashes, other shards are still functioning and can still process queries. So the database as a whole remains partially functional.
- Provides security Users can only access certain shards. So you can implement different access control mechanisms on different shards.
- Faster query processing Since the size of the dataset in each server is small the size of the index is also small. This results in faster query processing.
- Increase read and write throughput Both read and write capacity increases as long as operations are done on one shard.
- High Scalability Partitioning the data and storing them in different shards provides scalability in terms of data and memory (because it spreads the load on multiple machines memory usage in each shard is less and the network bandwidth does not saturate)

Disadvantages of Sharding

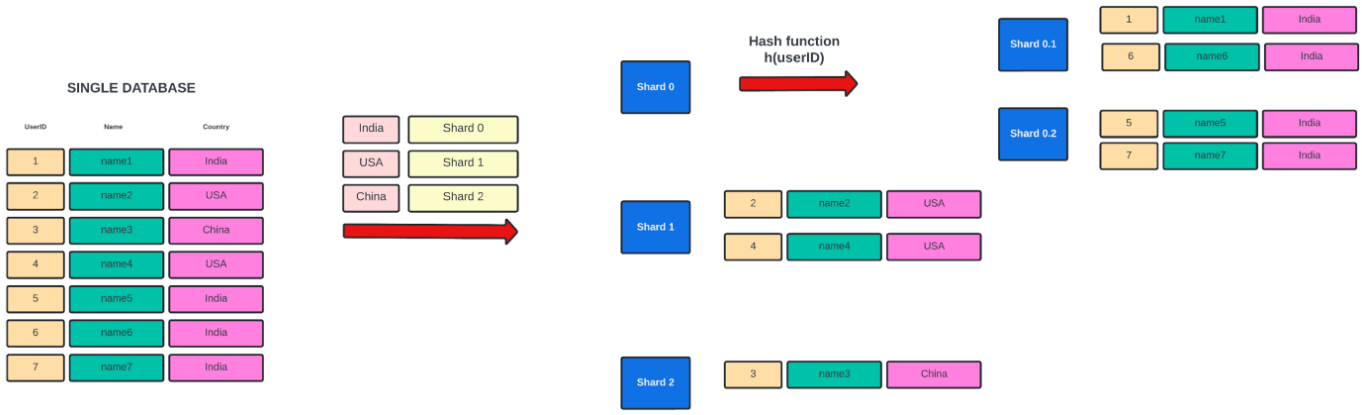
- Complexity The server has to know how to route a query to the appropriate shard. If we add the code for finding the shard in the server, it makes the server more complex.
- Transactions and Rollback You cannot process queries for the two different tables present in different shards. So transactions across shards are not possible. And therefore rollbacks are also not possible.
- Joins across shards If we want to join two tables from two different shards, then the query needs to go to two different shards, pull out the data and join the data across the network. This is a very expensive operation.
- Infrastructure cost Sharding requires more machines and computing power over a single database server. If there is no proper optimization then the increase in cost can be significant.

Hierarchal Sharding

It is very difficult to increase/ decrease the number of shards. So you can only have a fixed number of shards.

Since the number of shards is fixed, one of the shards might grow too big. To solve this issue we can do sharding on the large shard. Every shard has a manager who maps the request to the correct mini-shard. It is known as hierarchal sharding.

In the example below, on the first level, we have 3 shards and we use directory-based sharding. We again partition Shard 0 and use key-based sharding.



Master-Slave architecture for High Availability

If we want to query data from a shard even if the database instance goes offline, we can use master-slave architecture.

In master-slave architecture, there are multiple slaves which are copying the master. Whenever there is a written request it is always on the master and whenever there is a read request it is distributed evenly across the slaves. In case the master fails, slaves choose one master among themselves.