Database Sharding & Partitioning Q&A

Q1: What is the key difference between horizontal partitioning (sharding) and vertical partitioning in a database?

Answer:
Sharding is basically the method of distributing data across multiple machines whereas partitioning is splitting the subset of data within the same instance. Horizontal partitioning is row-level partition of data, and vertical partitioning is at the column level, so in case of vertical we might need to do join operations if data is split.

Q2: What are some sharding key selection criteria (things to consider when choosing a sharding key)?

Answer:
While choosing a sharding key we should make sure that data gets uniformly distributed (so no shard becomes hot), queries should be efficient and match with how app queries data, cross-shard queries should be minimized, and it should also allow scaling later without too much reshuffling. Also the key should be stable and not keep changing. For example, in a social media app `user_id` is better than `country` because `country` can cause imbalance.

Q3: In a messaging app (like WhatsApp), would `message_id` or `user_id` be a better sharding key? Why?

Answer:
`user_id` would be a better sharding key because then all the messages of a particular user will stay in the same shard, so queries like getting all chats of a user will only hit one shard. If I use `message_id` then messages will be spread randomly and fetching all messages of one user will require checking multiple shards which is expensive.

Q4: If some users (like celebrities) generate way more traffic, making their shard hot, what strategies can handle this hotspot problem?

Answer:
If one shard gets hot because of celebrity users then we can break their data into smaller parts and spread across multiple shards, or use some key modifier so one user's data doesn't sit on one shard only. We can also use caching like Redis for heavy read traffic. Another way is to

design fan-out models where instead of directly storing everything under the celebrity, we push or pre-compute updates for followers.

Q5: Suppose DB is sharded by `user_id`. Query: `SELECT COUNT(*) FROM users WHERE age BETWEEN 18 AND 25;`. What's the challenge and solution?

Answer:
Since users are distributed across shards, a query on age (which is not the shard key) will have to go to all shards and then results need to be combined, which is expensive. One way to solve this is scatter-gather queries, or we can precompute counts and store in cache like Redis so it is fast. Another option is to keep summary tables or use OLAP systems for such analytical queries.

Q6: What's the difference between range-based sharding and hash-based sharding? Which avoids hotspots?

Answer:
In range-based sharding we divide data into ranges, like `user_id 1–1M` in one shard and `1M+1–2M` in another. This is good for range queries but can create hotspots if many values fall in one range. In hash-based sharding, we take a hash of the key (like `hash(user_id) % N`) and distribute it, so data spreads evenly and hotspots are avoided, but range queries become costly since you have to check all shards.

Q7: What challenges arise when migrating from a monolithic DB to a sharded architecture?

Answer:
If we are moving from monolithic to sharding, then first the app has to become shard-aware or even move towards microservices where different services can have their own DBs. Then we need to carefully choose the sharding key to keep latency low and throughput high. We also have to decide how many shards to start with and how to scale them later. Data migration is also a challenge because moving existing data into shards without downtime is tough. Also queries that were simple before can become cross-shard now, and even things like backups and monitoring get more complicated.

Q8: What's the difference between sharding and replication? Can they be used together?

Answer:
Replication means we keep the same data in multiple servers, usually one server is for writes and others are read replicas. This improves read scalability and availability but doesn't reduce the size of data on each server. Sharding means splitting the data itself into parts across servers, so each shard has different data and this helps in scaling writes and storage. Both can be used together, for example each shard can have its own replicas.

Q9: With hash-based sharding using `user_id % num_shards`, what problem arises if shards grow from 4 to 8, and how does consistent hashing help?

Answer:
If we are using `user_id % num_shards`, then when the number of shards changes (say from 4 to 8), almost all data will have to be reshuffled because the modulo changes. This is very expensive. Consistent hashing helps because it maps data on a ring and when a new shard is added only a small portion of data needs to move, so scaling becomes much easier.
