igotanoffer.com

Sharding: system design interview concepts (7 of 9)

12-15 minutes

If you want to succeed in system design interviews for a tech role, then you'll probably need to understand sharding and when to use it within the context of a larger system.

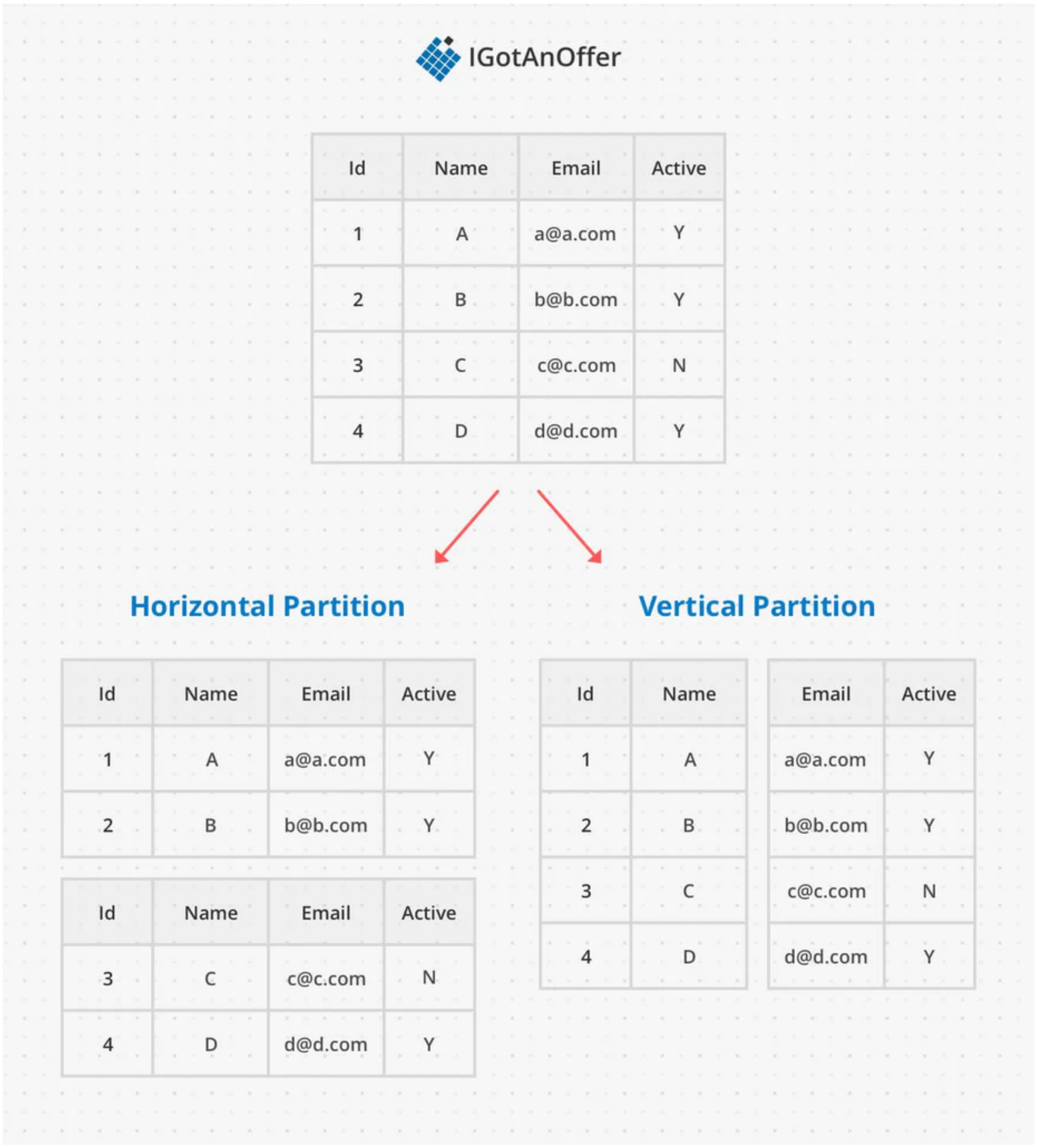
Sharding is essentially the horizontal scaling of a database system that is accomplished by breaking the database up into smaller "shards", which are separate database servers that all contain a subset of the overall dataset.

This is just a high-level definition, so to fill in the gaps we've put together the below guide that will walk you through sharding and how you should approach it during a system design interview. Here's what we'll cover:

1. Sharding Basics

Modern software services increasingly collect and use more data than can fit on a single machine. The capacity of the database server can be increased, but eventually runs into a physical limit. The alternative is splitting up data across a cluster of database servers.

Database sharding splits up data in a particular way so that data access patterns can remain as efficient as possible. A **shard** is a **horizontal partition**, meaning the database table is split up by drawing a horizontal line between rows. This is in contrast to a **vertical partition**, where partitions are made between columns.



Id	Name	Email	Active
1	A	a@a.com	Y
2	B	b@b.com	Y
3	C	c@c.com	N
4	D	d@d.com	Y

Horizontal Partition

Id	Name	Email	Active
1	A	a@a.com	Y
2	B	b@b.com	Y

Id	Name	Email	Active
3	C	c@c.com	N
4	D	d@d.com	Y

Vertical Partition

Id	Name
1	A
2	B
3	C
4	D

Email	Active
a@a.com	Y
b@b.com	Y
c@c.com	N
d@d.com	Y

The goal of sharding is to increase database capacity, but another effect is that the database can handle more traffic because any one server in the cluster only has to respond to a fraction of the total requests.

There are a couple of key features a sharded architecture needs to be efficient. We'll go over how it's implemented with shard keys, and the importance of denormalizing data. Then we'll cover when to use and when not to use sharding. Let's jump in.

1.1 How sharding works

Each row is assigned a **shard key** that maps it to the **logical shard** it can be found on. More than one logical shard can be located on the same **physical shard**, but a logical shard can't be split between physical shards.

When creating a sharded architecture, the goal is to have many small shards to create an even distribution of data across the nodes. This prevents **hotspots** from overwhelming any one of the nodes and produces fast response times for all nodes.

Sharding can be implemented either at the application level or the database level. At this point, most databases support sharded architectures, with the notable exception of PostgresQL, one of the top relational databases.

1.2 Denormalization

When implementing a sharded database architecture, you need to take special consideration about how the relational model spans shards. Relational data models are optimized for data with many cross-table relationships. If shards aren't isolated, the cluster will spend a lot of time on multi-shard queries and upholding multi-shard consistency.

This process of making sure there aren't relational constraints between different shards is called **denormalization**. Denormalization is achieved by duplicating data (which adds write complexity) and grouping data so that it can be accessed in a single row. Notably, denormalized data is different from **non-normalized** data where relationships between data are unknown rather than located on a single table.

As an example, let’s think about where to store data for messages between two friends. Both users want fast responses, but what if their user records are split up onto different shards? We can store the messages in both places. This increases write time slightly, because it means the data is written twice. But it also ensures the message data is fast to look up regardless which friend is checking their messages.

If denormalization isn’t possible (multi-shard queries are necessary,) then the service needs to consider the tradeoffs between consistency and availability. You can read more about this tradeoff and the CAP theorem in our article on databases.

1.3 When to use sharding

The benefits of a sharded architecture, versus other kinds of database partitioning are:

- Leveraging average hardware instead of high end machines
- Quickly scaling by adding more shards
- Better performance because each machine is under less load

Sharding is particularly useful when a single database server:

- Can’t hold all the data
- Can't compute all the query responses fast enough
- Can’t handle the number of concurrent connections

You might also need sharding when you need to maintain distinct geographic regions, even if the above compute constraints *haven't* been hit. Either your service will be faster when the data servers are physically closer to the users, or there's legislation about data location and usage in one of the countries your service operates in.

1.4 When not to use sharding

The disadvantages of database sharding are all about complexity. The queries become more complex because they have to somehow get the correct shard key, and need to be aware of avoiding multi-shard queries.

If the shards can't be entirely isolated, you need to implement eventual consistency for duplicated data or upholding relational constraints. The implementation and deployment of your database get a lot more complex, as do failovers, backups, and other kinds of maintenance. Essentially - you should only use database sharding when you absolutely have to.

2. Approaches to Sharding

The different approaches to sharded architectures are based on how the shard key is assigned. Regardless of what they're derived from, shard keys need to be unique *across* shards, so their values need to be coordinated. This leads to a tradeoff between a centralized "name server" that can dynamically optimize logical shards for performance, and a predetermined distributed algorithm that is faster to compute.

Shard keys are derived from some invariant feature of the data that utilizes business logic to optimize for the most common queries. Common choices are tenant ids, location, and timestamps. Custom configuration can also tune the performance of a sharded architecture based on usage patterns, actual shard sizes, etc.

Here are the three most common approaches to sharding you should know about when designing systems.

2.1 Range

Data can be assigned to a shard based on what "range" it falls into. For example, a database with sequential time-based data like log history could shard based on month ranges. One big advantage of range-based shard keys is they make sequential access patterns very fast, because data that is "close" in the given range will be on the same shard.

One downside to ranges is the balance of data can be unpredictable. For example, an e-commerce company might have a lot more orders in December because of holiday shopping, so the shard with the December range could get overwhelmed while the other shards aren't doing much.

2.2 Hash

To address the issue of imbalanced shards, data can be distributed based on a hash of part of the data. An effective hash function will randomize and distribute incoming data to prevent any access patterns that could overwhelm a node. For example, the profile pages of celebrities get substantially more traffic than the average user, so a hash function can be used to randomly distribute these celebrity users and prevent hotspots.

One major advantage of a hash-based sharded architecture is the shard key can be computed by any server that knows the hash function, so there's no centralized point of failure.

One big downside of hashing is adding shards can require a lot of overhead, depending on the implementation. **Consistent hashing** limits this by guaranteeing a minimum amount of data will have to be moved when a new node is added.

2.3 Name Node

The last approach is any implementation that uses a central "name node" to coordinate the mapping of data to shard keys. What's nice about this approach is it makes the business logic very clear and easy to update based on usage patterns.

For example, an important client might want to guarantee a precise location distribution of their data on your service. You can base the shard key on tennant it and have a clear mapping in the name node that assigns their tenant to shards in their desired geographic regions and is easy to update if their needs change.

The main downside is a name node is a central point of failure, and additional care is needed to implement good replication and failovers for that node. This approach also adds an additional step of consulting the name node in the lookup process, which can slow down database operations.

3. Example sharding questions

The questions asked in system design interviews tend to begin with a broad problem or goal, so it's unlikely that you'll get an interview question entirely about sharding.

However, you may be asked to solve a problem where sharding will be an important part of the solution. As a result, what you really need to know is WHEN you should bring it up and how you should approach it.

To help you with this, we've compiled the below list of sample system design questions. Sharding is relevant for all of the below questions.

- Design Dropbox (Read the answer)
- Design Pastebin (Read the answer)
- Design a web crawler (Read the answer)
- Design Flickr (Read about Flickr's actual architecture)

4. System design interview preparation

Sharding is just one piece of system design. And to succeed on system design interviews, you'll need to familiarize yourself with a few other concepts and practice how you communicate your answers.

It's best to take a systematic approach to make the most of your practice time, and we recommend the steps below. For extra tips, take a look at our article: 19 system design interview tips from FAANG ex-interviewers.

4.1 Learn the concepts

There is a base level of knowledge required to be able to speak intelligently about system design. To help you get this foundational knowledge (or to refresh your memory), we've published a full series of articles like this one, which cover the primary concepts that you'll need to know:

- Network protocols and proxies

- Databases
- Latency, throughput, and availability
- Load balancing
- Leader election
- Caching
- Sharding
- Polling, SSE, and WebSockets
- Queues and pub-sub

We'd encourage you to begin your preparation by reviewing the above concepts and by studying our system design interview prep guide, which covers a step-by-step method for answering system design questions. Once you're familiar with the basics, you should begin practicing with example questions.

4.2 Practice by yourself or with peers

Next, you'll want to get some practice with system design questions. You can start with the examples listed above, or with our list of 31 example questions.

We'd recommend that you start by interviewing yourself out loud. You should play both the role of the interviewer and the candidate, asking and answering questions. This will help you develop your communication skills and your process for breaking down questions.

We would also strongly recommend that you practice solving system design questions with a peer interviewing you. A great place to start is to practice with friends or family if you can. If you don't have anyone in your network who can interview you, then you might want to check out our our system design mock interview peer group.

4.3 Practice with ex-interviewers

Practicing with peers can be a great help, and it's usually free. But, at some point you'll start noticing that the feedback you are getting from peers isn't helping you that much anymore. Once you reach that stage, we recommend practicing with ex-interviewers from top tech companies.

If you know someone who has experience running interviews at Facebook, Google, or another big tech company, then that's fantastic. But for most of us, it's tough to find the right connections to make this happen. And it might also be difficult to practice multiple hours with that person unless you know them really well.

Here's the good news. We've already made the connections for you. We’ve created a coaching service where you can practice system design interviews 1-on-1 with ex-interviewers from leading tech companies. Learn more and start scheduling sessions today.

Learn more about system design interviews

This is just one of 9 concept guides that we've published about system design interviews. Check out all of our system design articles on our Tech blog.