

What is Sharding?

Understanding the purpose of sharding

- What happens if your setup grows beyond the capacity of a single machine? What if you want to run so many transactions that one service is simply not able to keep up? Let's assume you have millions of users and tons of thousand want to perform certain task at the very same thing.

- Clearly at some point you can not buy servers that are big enough to handle infinite load, anymore. It is simply impossible to run a Facebook or google like application on a single box.
- At some point you have to come up with a scalability strategy that serves your needs. This is when sharding comes into place.

- **Sharding is a method of splitting and storing a single logical dataset in multiple databases.** By distributing the data among multiple machines, a cluster of database systems can store larger dataset and handle additional requests.
- Sharding is necessary if a dataset is too large to be stored in a single database. Moreover, many sharding strategies allow additional machines to be added. Sharding allows a database cluster to scale along with its data and traffic growth.

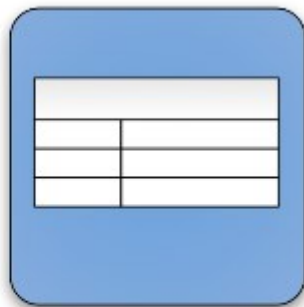
- Sharding is also referred as **horizontal partitioning**.
- The distinction **horizontal** vs **vertical** comes from the traditional tabular view of a database. A database can be split vertically — storing different tables & columns in a separate database, or horizontally — storing rows of a same table in multiple database nodes.

Horizontal Partitioning

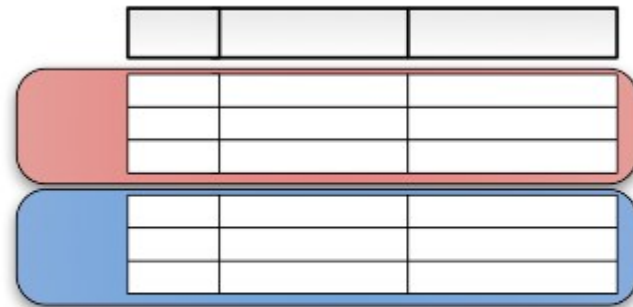


A diagram illustrating vertical partitioning. It consists of a red rounded rectangle containing a table with 3 columns and 6 rows. The top row is shaded light gray, and the bottom row is white.

Vertical



A diagram illustrating vertical partitioning. It consists of a blue rounded rectangle containing a table with 2 columns and 4 rows. The top row is shaded light gray, and the bottom row is white.



A diagram illustrating horizontal partitioning. It shows three stacked rounded rectangles. The top rectangle is light gray with 3 columns and 1 row. The middle rectangle is red with 3 columns and 3 rows. The bottom rectangle is blue with 3 columns and 3 rows. The top row of the middle rectangle is shaded light gray, and the bottom row of the bottom rectangle is white.

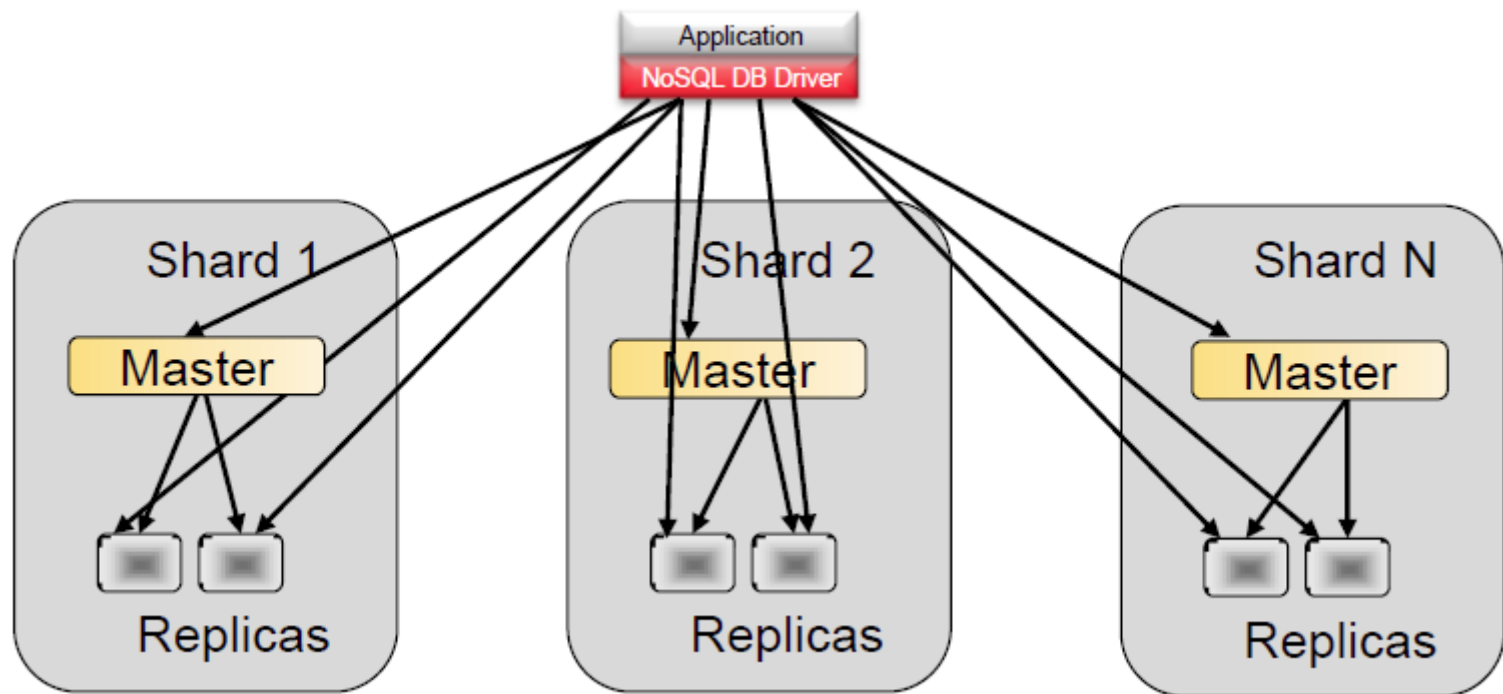
Horizontal

- Vertical partitioning is very domain specific. You draw a logical split within your application data, storing them in different databases. It is almost always implemented at the **application level**—a piece of code routing reads and writes to a designated database.

- In contrast, sharding splits a homogeneous type of data into multiple databases. You can see that such an algorithm is easily generalizable. That's why sharding can be implemented at either the application or **database level**.

- In many databases, sharding is a first-class concept, and the database knows how to store and retrieve data within a cluster.
- Almost all modern databases are natively sharded. Cassandra, HBase, HDFS, and MongoDB are popular distributed databases. Notable examples of non-sharded modern databases are Sqlite, Redis, Memcached, and Zookeeper.

Shard (database architecture)



- There exist various strategies to distribute data into multiple databases. Each strategy has pros and cons depending on various assumptions a strategy makes.
- It is crucial to understand these assumptions and limitations. Operations may need to search through many databases to find the requested data. These are called **cross-partition operations** and they tend to be inefficient. **Hotspots** are another common problem — having uneven distribution of data and operations. Hotspots largely counteract the benefits of sharding.

Before you start: you may not need to shard!

- Sharding adds additional programming and operational complexity to your application. You lose the convenience of accessing the application's data in a single location. Managing multiple servers adds operational challenges. Before beginning, one must see whether sharding can be avoided or deferred.

- **Get a more expensive machine:**

Storage capacity is growing at the speed of Moore's law. From Amazon, you can get a server with 6.4 TB of SDD, 244 GB of RAM and 32 cores. Even in 2013, Stack overflow runs on a single MS SQL server.

- If your application is bound by read performance, you can add **caches** or database **replicas**. They provide additional read capacity without heavily modifying your application.

- **Vertically partition by functionality:**

Binary Large Object (**BLOB**) tend to occupy large amounts of space and are isolated within your application. Storing files in S3 can reduce storage burden. Other functionalities such as full text search, tagging, and analytics are best done by separate databases.

- Not everything may need to be sharded. Often times, only few tables occupy a majority of the disk space. Very little is gained by sharding small tables with hundreds of rows.

Driving Principles

- Compare the pros and cons of each sharding strategy, use the following principles:
- **How the data is read**—Databases are used to store and retrieve data. If we do not need to read data at all, no need to use sharding. If we only need to batch process the data once in a while, we can append to a single file and periodically scan through them. Data retrieval requirements heavily influence the sharding strategy.

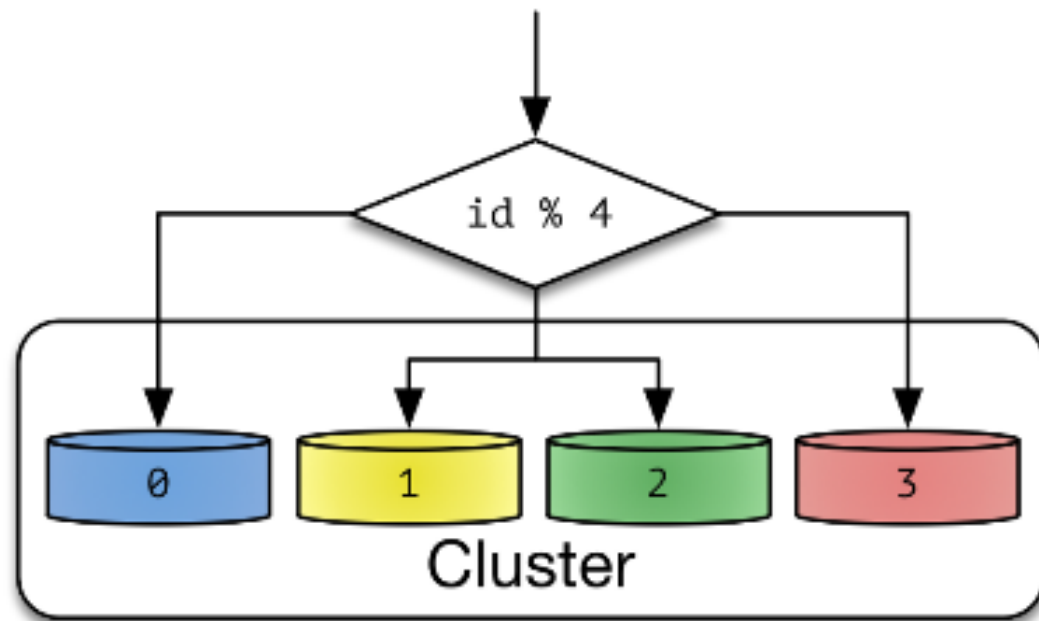
Driving Principles

- **How the data is distributed** — Once you have a cluster of machines acting together, it is important to ensure that data and work is evenly distributed. Uneven load causes storage and performance hotspots. Some databases redistribute data dynamically, while others expect clients to evenly distribute and access data.

- Once sharding is employed, **redistributing data** is an important problem. Once your database is sharded, it is likely that the data is growing rapidly. Adding an additional node becomes a regular routine. It may require changes in configuration and moving large amounts of data between nodes. It adds both performance and operational burden.

Case 1 — Algorithmic Sharding

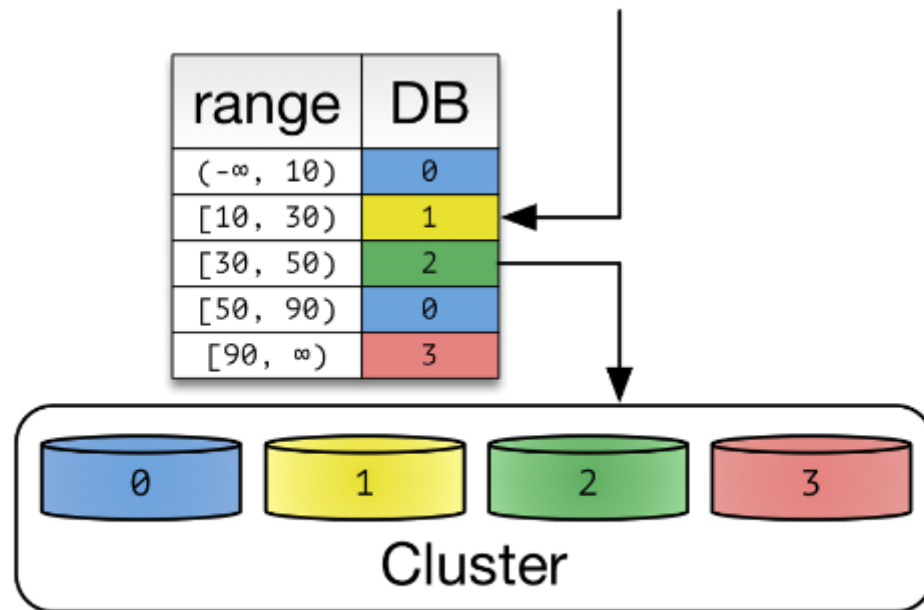
- One way to categorize sharding is algorithmic versus dynamic.
- In algorithmic sharding, the client can determine a given partition's database without any help. In dynamic sharding, a separate locator service tracks the partitions amongst the nodes.



An algorithmically sharded database, with a simple sharding function

- Algorithmically sharded databases use a sharding function (*partition_key*) \rightarrow *database_id* to locate data. A simple sharding function may be “*hash(key) % NUM_DB*”.
- Reads are performed within a single database as long as a partition key is given. Queries without a partition key require searching every database node. Non-partitioned queries do not scale with respect to the size of cluster, thus they are discouraged.

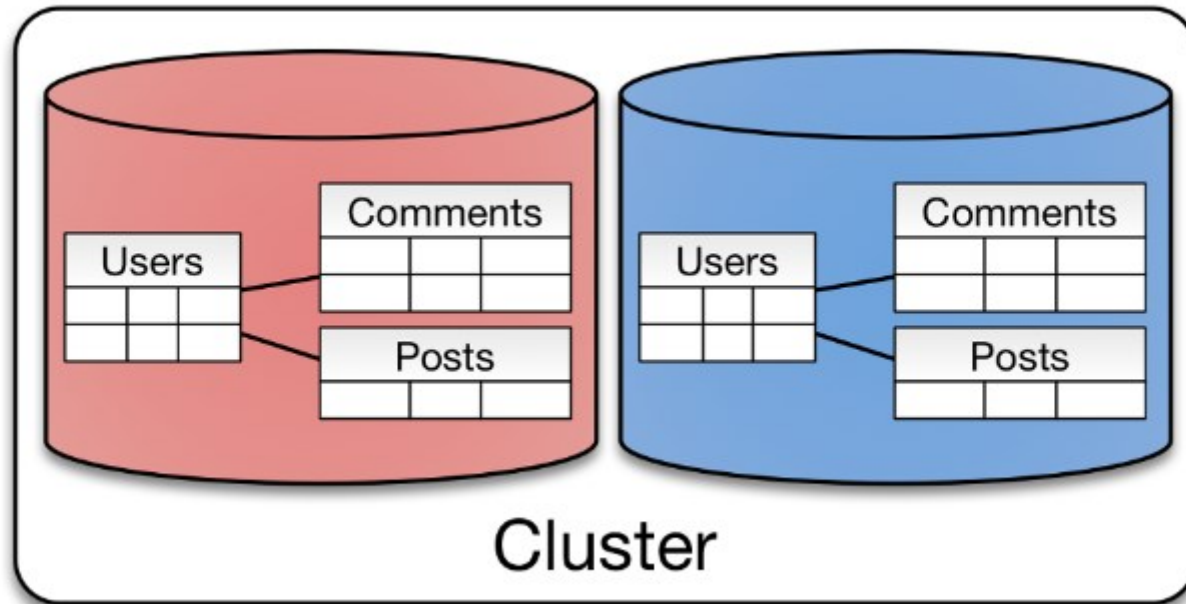
Case 2— Dynamic Sharding



A dynamic sharding scheme using range based partitioning.

- In dynamic sharding, an external **locator service** determines the location of entries. It can be implemented in multiple ways.

Case 3 — Entity Groups



- Many databases have expressive querying and manipulation capabilities. Traditional RDBMS features such as joins, indexes and transactions reduce complexity for an application.

- The concept of entity groups is very simple. Store related entities in the same partition to provide additional capabilities within a single partition. Specifically:
- Queries within a single physical shard are efficient.

- Stronger consistency can be achieved within a shard.
- This is a popular approach to shard a relational database. In a typical web application data is naturally isolated per user. Partitioning by user gives scalability of sharding while retaining most of its flexibility. It normally starts off as a simple company-specific solution, where resharding operations are done manually by developers.

- Entity groups can be implemented either algorithmically or dynamically. They are usually implemented dynamically since the total size per group can vary greatly.
- Other than sharded RDBMS solutions, **Google Megastore** is an example of such a system. Megastore is publicly exposed via Google App Engine's Datastore API.

Case 4 — Hierarchical keys & Column-Oriented Databases

row	columns				
1	a=...	b=...	c=...	d=...	
2	x=...				
3	y=...				b=...
4	d=...				
5	a=...				c=...
6					

- This model has been popular since mid 2000s. The restriction given by hierarchical keys allows databases to implement data-agnostic sharding mechanisms and efficient storage engines. Meanwhile, hierarchical keys are expressive enough to represent sophisticated relationships. Column-oriented databases can model a problem such as time series efficiently.

- Column-oriented databases can be sharded either algorithmically or dynamically. With small and numerous small partitions, they have constraints similar to key-value stores. Otherwise, dynamic sharding is more suitable.
- The term *column database* is losing popularity. Both HBase and Cassandra once marketed themselves as column databases, but not anymore.

Conclusion Remarks

- Replication is a crucial concept in distributed databases to ensure durability and availability. Replication can be performed agnostic to sharding or tightly coupled to the sharding strategies.

Conclusion Remarks

- The details behind data redistribution are important. Ensuring both the data and locators are in sync while the data is being moved is a hard problem. Many techniques make a tradeoff between consistency, availability, and performance.

Conclusion Remarks

- None of this is magic. Everything follows logically once you consider how the data is stored and retrieved.
- Cross-partition queries are inefficient, and many sharding schemes attempt to minimize the number of cross-partition operations.
- On the other hand, partitions need to be granular enough to evenly distribute the load amongst nodes. Finding the right balance can be tricky.
- Of course, the best solution in software engineering is avoiding the problem altogether. As stated before, there are many successful websites operating without sharding. See if you can defer or avoid the problem altogether.