Release: 20250117

Topics we cover

Sharding

Horizontal vs. Vertical Scaling

Components of a Sharded Cluster

Shard Keys

How Sharding works

Sharding

Sharding is a mechanism for partitioning larger collections across multiple servers

- Enables bringing unlimited hardware to a scaling problem
- Allows parallel processing for some tasks
- Allows enforcement of geographical data location

Sharding is utilized for horizontal scaling.

Sharding vs. Replication

Replication is to ensure resilience

Sharding is about bringing more capability

Many baby ducks is resilience, some don't survive (HA). Wolves form a pack of adults because a group is more capable than one, they divide up the task of hunting between them working together.

- In some NoSQL systems Scaling and High Availability (HA) use the same mechanism - In MongoDB, they do not
- Replication is used for High Availability
- Sharding is for Scaling Out large workloads.
- Sharding without HA is wrong
  - Increased risk of something breaking not reduced
  - Higher cost and time to repair

Resources used by a database

CPU Cycles	Document DBs often use less CPU as not joining
RAM	Acts as a cache and much faster access to data than disk
	Ideally should hold all your frequently accessed data
Disk Reads and Writes	Provides long term storage but slowly
	Measured in: Disk capacity, throughput, operations per second
Network	Relevant for data moving about

It is essential to understand how the database uses different resources.
What resource is it that is limiting performance, what one will run out first?

Vertical vs. Horizontal scaling

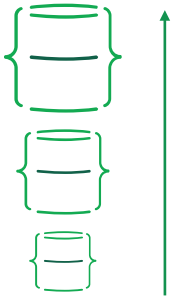
There is a scaling limit for a single server

With horizontal scaling, there is no limit

Vertical vs. Horizontal scaling

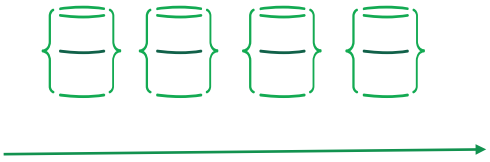
Vertical Scaling

Increase size of instances (RAM, CPU etc)



Horizontal Scaling

Add more instances (small or large)



Limitations of Vertical Scaling

Adding more hardware is called vertical scaling

Limits to how much machine's resources can be increased:

- Architecture: Max number of cores/RAM
- Provider: Cloud instance sizes
- Cost: Price isn't linear once things get large

- Vertical scaling has limits that often prevent the ability to scale easily
- Few places are happy to provision a server with 128 cores and 4TB of RAM

Case Study

In 2014 a fitness app company had a MySQL server with 750GB of RAM

- Expanding to 4TB of RAM as they grew was disproportionately expensive
- Decided to changed to 4 MongoDB shards, each with 512 GB
- Now they have expanded beyond this significantly

Example of how application growth can take advantage of horizontal scaling

When to shard?

Sharding is a solution to Big Data problems

Sharding is needed when resource limitations require horizontal scale.

How to tell if you need to Shard

- Resource are maxed out but schema and code are already optimized
- Need to increase overall throughput or increase in-memory cache/storage with affordable costs
- Need to meet a backup restore time (RTO) target and need parallelism to do so

Sharding is required when the data exceeds the capacity of your system's RAM or Disk IOPs.
On a separate note, Atlas provides a hard limit of 4TB per shard (applicable for clusters which are upto M700 instances, higher level of instances have higher capacity).
We recommend using these guidelines:

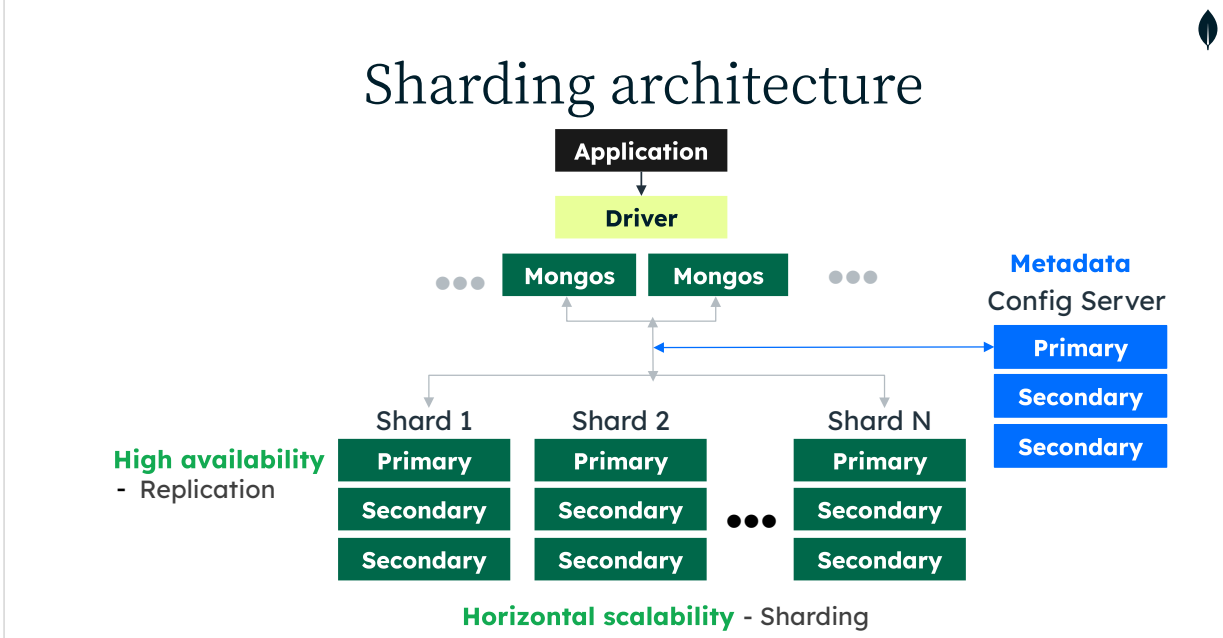
- **Load Requirements**: Sharding enables increased overall throughput through distributing connection and command load.
- **Memory Requirements**: Sharding enables collections to gain access to a larger in-memory cache for fast command execution.
- **Storage/IO Requirements**: Sharding enables collections to gain access to a larger storage capacity, which can satisfy frequently accessed large databases as well as infrequently accessed databases.
  In addition to additional capacity, sharding for additional storage purposes gains the benefit of increased throughput for single and cross-shard commands, as well as faster time to backup/restore and faster time to initial sync.

Why understanding sharding now?

Might not need to shard now but schema design decisions taken today matter in sharding

Start planning for sharding on day one

It is important to understand the concept of sharding so that you can plan for future growth.

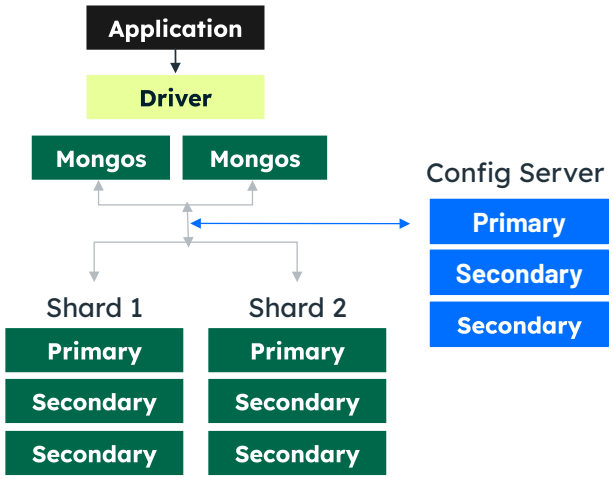


Sharding has a number of moving parts - each of which should be planned and sized. Sharding is about solving big problems - you can do a huge amount of processing before you need to go there so don't be afraid of the complexity - remember that you may be managing a 200 server cluster ultimately and that means you are doing something that has real world value.

Sharding uses more hardware

Sharding adds:

- A config (metadata) RS (3 nodes)
- Ideally at least 2 query routers
- Several Replica sets holding data

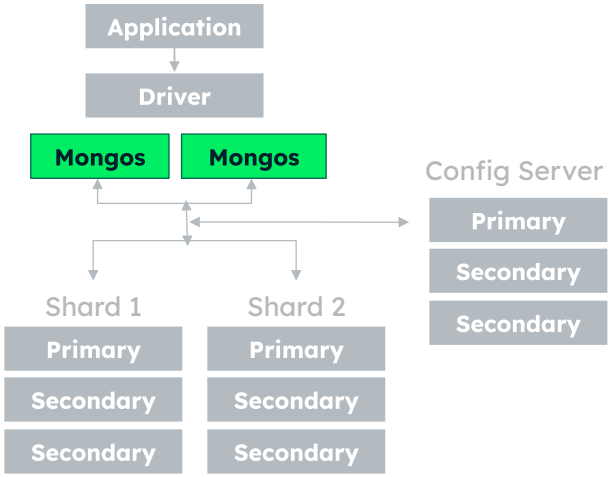


- The smallest production cluster should be 11 nodes.
- 2 Replica Sets to hold data
- 1 Replica set to hold metadata
- 2 Routers to avoid a single point of failure
- Sometimes these are combined - e.g., a router on a data server.

mongos

mongos is like a proxy server you connect to in sharding

- Behaves like a MongoDB database but routes requests instead
- Only sends work where it needs to
- No local storage needed
- Mongos should be hosted alongside the application



mongos routes requests (replaces mongod from standalone / replica set)

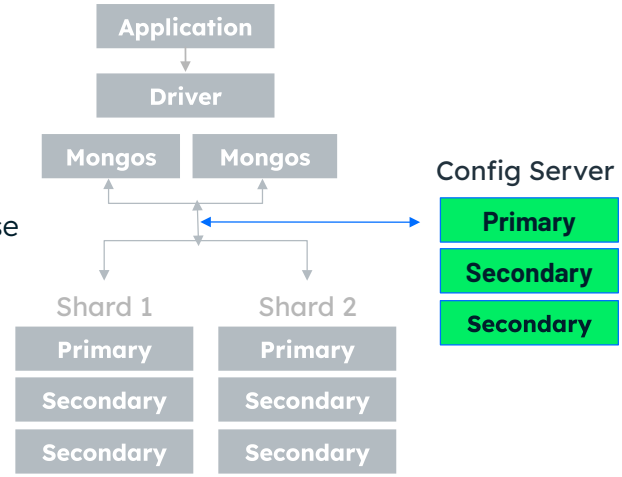
mongos will balance the data by monitoring data distribution across different shards and coordinate the data migration.

Configuration servers

Config Replica Set (Config server)

- Holds metadata about users, partitioning, and data locations
- Metadata store in 'config' database
- Accessible via mongos
- Does balancing of the data
- Can be embedded to save costs

Always a replica set - Why?



Configuration servers hold metadata about users, partitioning, location of ranges of data.
From MongoDB 8.0, you can deploy a sharded cluster with an embedded config server. This means one of the shards in your cluster will be a user data shard along with a config data shard.

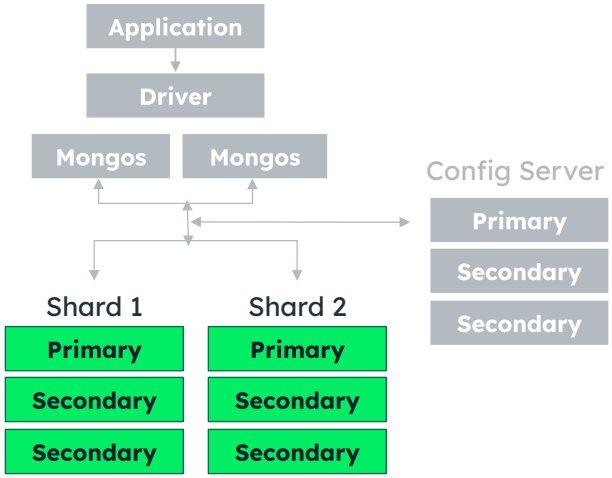
- This shard is called the Config shard and plays the role of a user data shard and a config server replica set. This way, resources of the shard will be shared between the two roles.
- If the config shard is expected to have a high read and write workload, it is possible that the config server operations may be starved of resources. In such cases, we recommend using a dedicated config server.
- To configure a dedicated config server to run as a config shard, run the `transitionFromDedicatedConfigServer` command.

Shard servers

Each shard

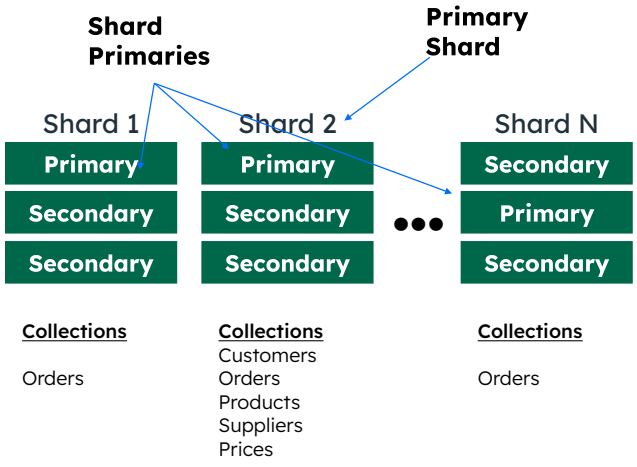
- Must be a replica set (RS)
- Holds a subset of your largest collections

The scaling benefits of sharding starts with two or more shards.



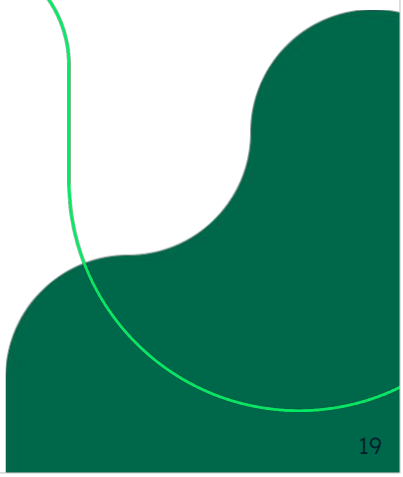
Each shard is a replica set.
We do not want a shard to be unavailable; hence each shard must be a replica set.
Each additional server statistically increases the chance of any one server failing, by using replica set, a server failing doesn't break the system.

Primary Shards vs. Shard Primaries



A Shard Primary is whatever node is the current Primary in the Replica Set representing that Shard.
The Confusingly similar term "Primary Shard" is the shard , for each database, designated when sharding was enabled to hold any unsharded collections in that database,
The shard primary is also the shard used for any other tasks that cannot be done across all the shards.

Splitting collections



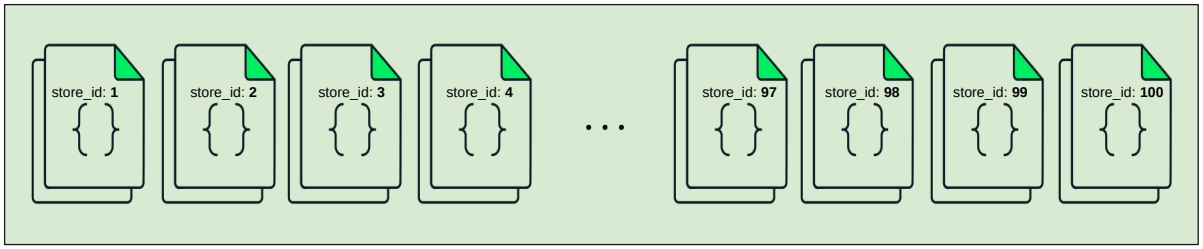
So how to split a collection and store it's parts in different shards?

Let us understand this with an example.
Let us say there is a brick and mortar retail store chain called 100-stores that sell grocery across various suburbs of London. They have 100 stores in London. The POS application stores the transactions in a collection in a MongoDB replica set. The collection is becoming very big and all optimization/vertical scaling options have been exhausted.

We need a way to split this large collection and distribute it across multiple shards in order to horizontally scale the cluster.

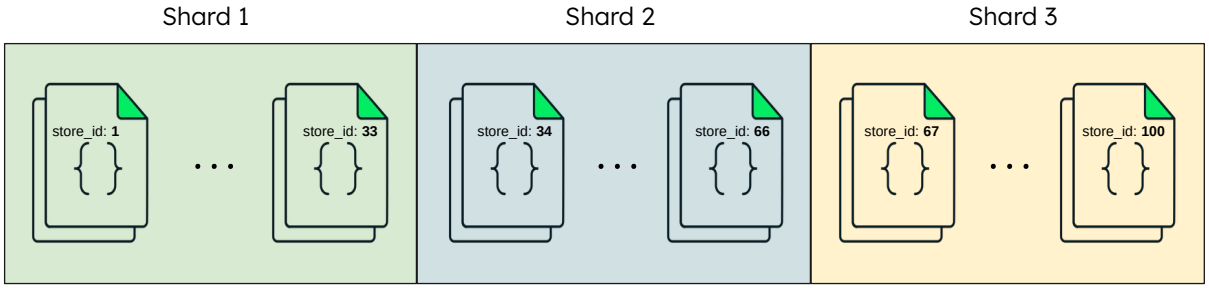
This is done by sharding the cluster and then choosing a shard key.

How to split a large collection?



The collection contains one document per purchase, and contains fields such as store_id, transaction_date, etc.
Since there are 100 stores, the store_id field contains values ranging from 1 to 100.

Shard key



- Shard key: Field(s) chosen to partition data
- Key range(s): Range of key values that are present in a shard

The documents in the collection can be partitioned using the store_id field and distributed to different shards.
Ex. Documents with store_id values between 1 and 33 can be assigned to Shard 1, between 34 and 66 can be assigned to to Shard 2 and the remaining to Shard 3.

In this example the store _id is called the "Shard Key". Shard keys can be composite. In this example we could choose to composite shard key such as (store_id, transaction_date) for partitioning the data.

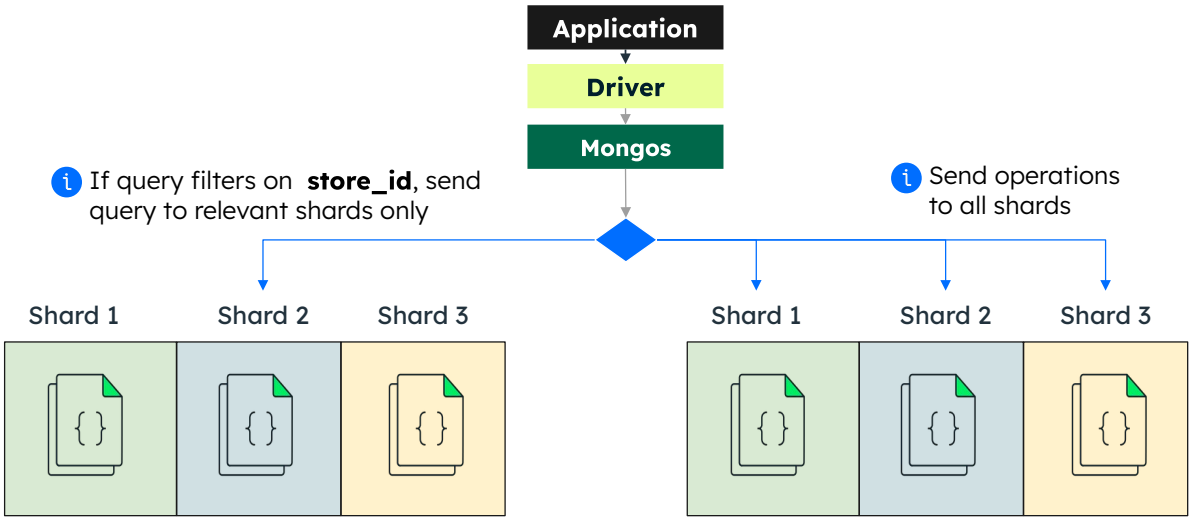
While distributing the data to different shards we have multiple goals:

1. The data should be uniformly distributed to the various shards
    a. What happens if some stores do more business than others?
2. Query load should be uniformly distributed to different shards
    a. What happens if some stores do more business than others?
3. A shard becoming unavailable should have a minimal impact on the queries
    a. How many stores will be affected if Shard 1 goes down?
4. As much as possible a single query should be processed on a small number of shards to keep the query response time low
    a. How many shards contain information related to Store 1?

Thus choosing a good shard key is very important.

A badly chosen shard key can adversely impact query performance and prevent us from realising the horizontal scaling benefits of sharding.

Sharding- distributing operations



- The query is sent to mongos by the driver
- IF query contains shard key (or at least first field of the key)
  - Lookup where data with that key is stored
  - Send query to those replica sets to run as normal
  - When results come back stream to the client.

  ELSE
  - Send query to all servers.
  - When results come back stream to the client.
- Concepts regarding reading from secondaries and read concerns still apply and the sharded replica sets are queried appropriately.
- If sorting required, do merge-sort of results as they come back

Sharded operations

When having a good shard key

- Most operations target a single shard or a few shards
- Individual users mostly hit a single shard
- If a shard is down, only some operations fail

More efficient than every query going to all shards

- Less work done
- Lower average latency

An operation that cannot target specific shards is called "broadcast" or "scatter gather"
If a shard is down - all operations targeting it fail completely

- All scatter gather queries fail
- Ideally, only a subset of users are affected by a shard being down

A good shard key will target few shards, preventing a “scatter gather”

Choosing Shard Keys

Rules for choosing a good shard key

- Included in most of your queries
- Reasonably high cardinality
- Ideally no more than 128MB of data to share a shard key value
- Co-locates data you wish to retrieve together

Effects of choosing a bad shard key

- Queries that do not filter on the shard key turn into “broadcast” queries
- Low cardinality shard keys can lead to uneven distribution of data
- When collections are sharded on different keys a query that needs to fetch related information from a different collection ($lookup) can slowdown due to network overhead of fetching data from different shards

A shard key will determine how your data is partitioned.

If the cardinality is too low or too many items have the same value then you will get 'Jumbo" ranges - which cannot be split and balanced.

An example of a jumbo key range is **{ sk: 1 } ->{ sk: 2 }.** All documents in this key range have the shard key value of 1. This range cannot be further split and if the size of this range grows above the configured balancing range size then the range becomes a jumbo key range.

Prior to v6 the key range was referred to as a chunk and it could be up to 64MB in size. From v6 onwards the term chunk has been replaced with the term key range.

Choosing shard keys in practice

Only shard large collections

In large collections, users work with subsets of the data

- User data - shard by a compound key including user and other fields, for example: bank accounts + transaction_time or games + session_id
- Departmental data - shard by a compound key involving department or branch and other common fields
- Where there are no clear subsets and the queries do not heavily rely on Joins ($lookup), shard for parallelism
  For instance, in an analytic data store, add a random value for the shard key

NOTE: Using a hash (random value) for a shard key may help in balancing the data across the shards, however it is very unlikely for queries to filter on random hashed shard keys. This will turn all queries into broadcast queries.

The choice of a shard key should be considered carefully! It cannot be changed without dropping and reloading your data.
Once it's done, it's hard to undo;

Changing shard keys

Changes on shard key (based on version)

- Before 4.4, Cannot change the shard key afterwards
- Version 4.4+, Can add another field to refine the key
- Version 5.0+, Can change the shard key and reorganize the data

The choice of a shard key should be considered carefully! It cannot be changed without dropping and reloading your data.
Once it's done, it's hard to undo;

Exercise - Choosing Shard Keys

What fields could be used as shard keys in the following scenarios and why?

- A web-based email service like Gmail
- A government database of businesses and their directors
- A stock database for a national chain of electronic stores
- A customer accessible product catalog like Amazon

There is no one correct answer. Please explain your thinking behind choosing the shard keys. What assumptions were made about data distribution, query patterns etc.?

One possible solution and explanation is provided at the end the slide.

Quiz Time!

#1. Which component is used to access a sharded cluster?

A	Mongos	B	Config Server	C	Any Shard
D	Primary Shard	E	Mongod		

Answer in the next slide.

#1. Which component is used to access a sharded cluster?

A	Mongos	B	Config Server	C	Any Shard
D	Primary Shard	E	Mongod		

Answer in the next slide.

#2. Which of these are attributes of a well chosen Shard Key?

A	Has a low cardinality	B	Keeps commonly accessed data on a single shard	C	Minimises the impact on users when a shard is down
D	Distributes all newly inserted documents across shards evenly	E	Targets most queries to a subset of shards		

Answer in the next slide.

#2. Which of these are attributes of a well chosen Shard Key?

A	Has a low cardinality	B	Keeps commonly accessed data on a single shard	C	Minimises the impact on users when a shard is down
D	Distributes all newly inserted documents across shards evenly	E	Targets most queries to a subset of shards		

Recap

Sharding is a mechanism of partitioning large collections across multiple servers

Choosing the shard key well is important

Atlas provides a hard limit of 4TB per shard.

Plan for sharding while designing your Schema

Exercise Answers

Exercise: Choosing Shard Keys

- A web-based email service like Gmail
  - Email_id + mail_timestamp
- A government database of businesses and their directors
  - Entity_business_name
- A stock database for a national chain of electronic stores
  - store_id + sku
- A customer accessible product catalog like Amazon
  - Delivered_to_country + sku

We should not only understand the data distribution (cardinality, histogram etc) but also know the query pattern in order to pick a good shard key.

A web-based email service like Gmail:
A compound shard key containing email_id and mail_ time_stamp is chosen over a simple shard key (email_id) because the size of all documents per email_id is very likely to be greater than 128MB. In this case creating a simple shard key will lead to the formation of "Jumbo key ranges"

A government database of businesses and their directors
If most queries try to search for a business entity in order to find the names of the directors then entity_business_name is a good shard key.

A stock database for a national chain of electronic stores
If we assume that the application is used by store managers to track the inventory of their store then most queries are likely to be store specific. So a compound shard key including store_id and sku could be a good shard key. Sku is added to increase cardinality of the shard key.

A customer accessible product catalog like Amazon
Global e-commerce company like Amazon will maintain product catalogue per country. Most people who look up products will be interested in those that can be delivered to their country. Queries are likely to include a filter on delivered_to_country field. The catalog is also likely to be very big so a low cardinality simple shard key (delivered_to_country) is unlikely to be enough. Hence, a compound shard key containing the country and sku maybe better.