ACID & Indexes

What is Acid:

In MongoDB, ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure reliable transaction processing within the database. Here's a breakdown of each property in the context of MongoDB

Atomicity:

Atomicity guarantees that all of the commands that make up a transaction are treated as a single unit and either succeed or fail together. This is important as in the case of an unwanted event, like a crash or power outage, we can be sure of the state of the database. The transaction would have either completed successfully or been rolled back if any part of the transaction failed. If we continue with the above example, money is deducted from the source and if any anomaly occurs, the changes are discarded and the transaction fails

Consistency:

Enforcing Rules: MongoDB ensures that the database remains in a consistent state before and after a transaction. All constraints, such as unique indexes, are maintained throughout the operation. Any transaction must bring the database from one valid state to another, preserving database invariants.

Isolation:

Isolation ensures that all transactions run in an isolated environment. That enables running transactions concurrently because transactions don't interfere with each other. For example, let's say that our account balance is $200. Two transactions for a $100 withdrawal start at the same time. The transactions run in isolation which guarantees that when they both complete, we'll have a balance of $0 instead of $100

Durability:

Durability guarantees that once the transaction completes and changes are written to the database, they are persisted. This ensures that data within the system will persist even in the case of system failures like crashes or power outages. The ACID characteristics of transactions are what allow developers to perform complex, coordinated updates and sleep well at night knowing that their data is consistent and safely stored

Indexes in MongoDB

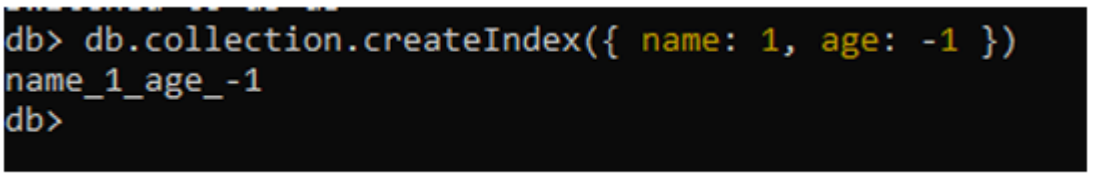
Indexes are special data structures that store a small portion of the data set in an easy-to traverse form. They improve query performance and enable efficient execution of various operations.

Types of Indexes:

1.**Single Field Indexes:** Indexes on a single field.
2.**Compound Indexes:** Indexes on multiple fields. Multikey
3.**Indexes:** Indexes on array fields, creating an index key for each array element.

4.**Geospatial Indexes:** Indexes supporting geospatial queries.
5.**Text Indexes**: Indexes supporting text search on string content.
6.**Hashed Indexes**: Indexes with hashed values of the indexed field.

The default name for an index is the concatenation of the indexed keys and each key's direction in the index (1 or -1) using underscores as a separator. For example, an index created on

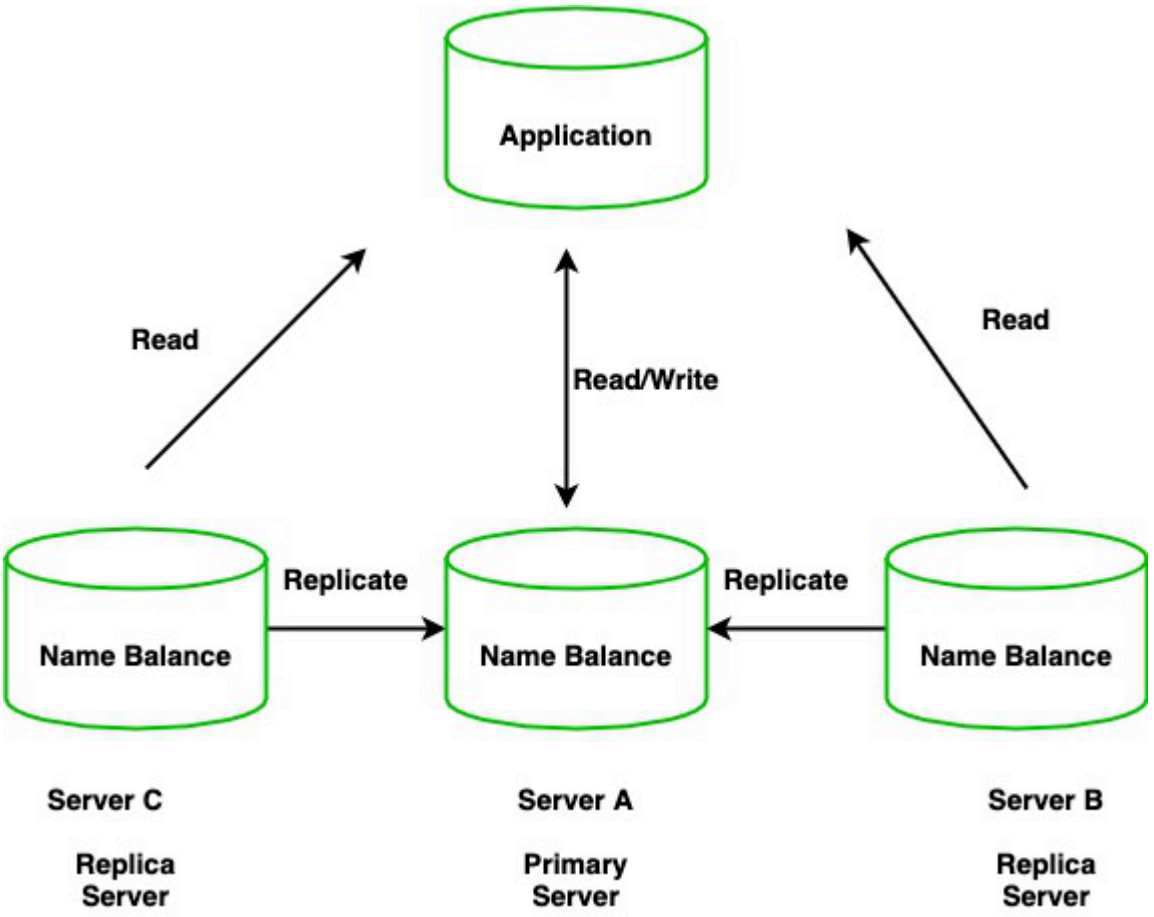


You cannot rename an index once created. Instead, you must with a new name. drop and recreate the index Incorporating indexes into your MongoDB collections is a fundamental practice for improving query efficiency and overall database performance. By carefully designing and maintaining indexes, developers can ensure that their applications remain responsive and capable of handling large volumes of data and complex queries. Understanding the different types of indexes and their appropriate use cases is essential for making informed decisions about index implementation and optimization

Sharding and replication

Replication in MongoDB

Replication is the method of duplication of data across multiple servers in MongoDB.



Replication **increases redundancy** and **data availability** with multiple copies of data on different database servers. So, it will increase the performance of reading scaling.

The set of servers that maintain the same copy of data is known as **replica servers** or **MongoDB instances**.

Sharding in MySQL

Sharding is a method for distributing large **collection**(dataset) and allocating it across multiple servers. MongoDB uses sharding to help deployment with very big data sets and high volume operations.

Sharding combines more devices to carry data extension and the needs of read and write operations.

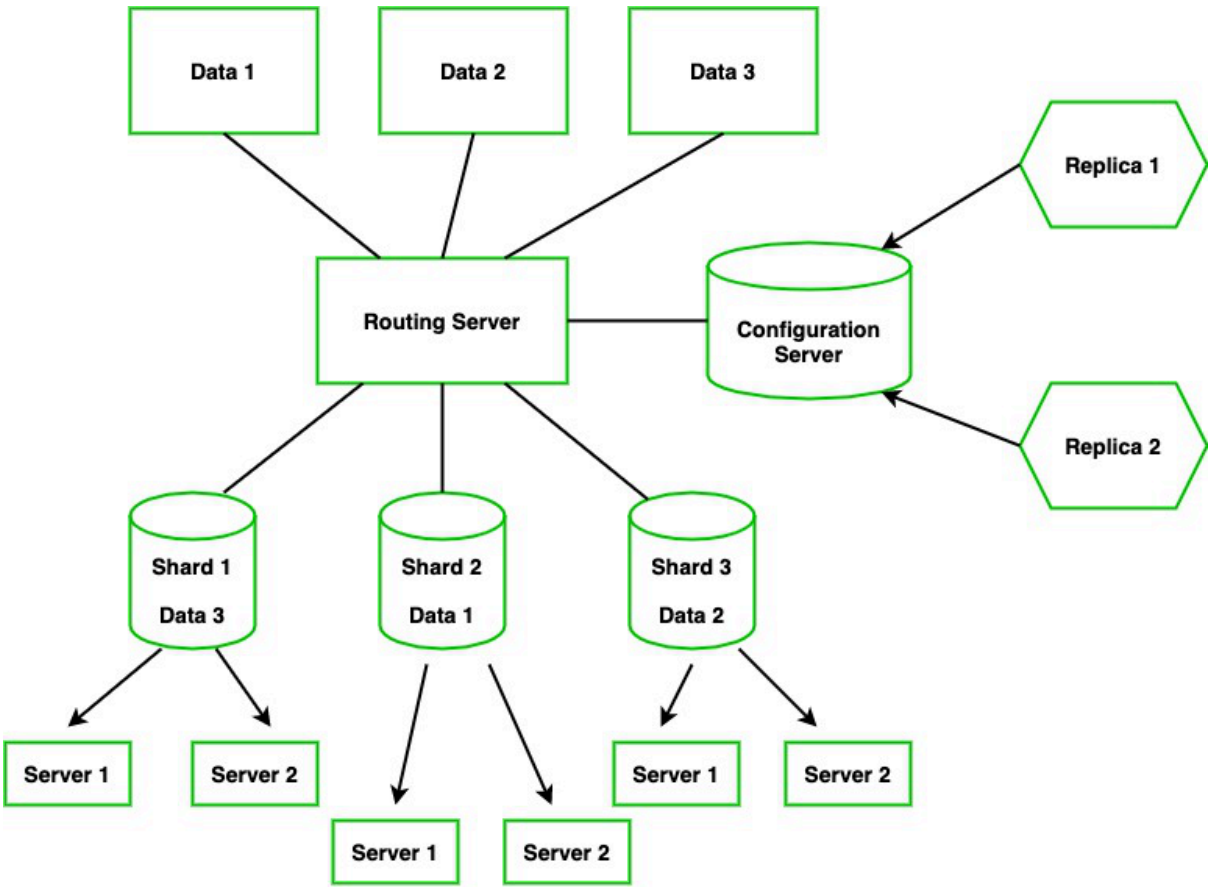
Need for Sharding

Database systems that have big data sets or high throughput requests can not be handled by a single server.

For example, High query flows can drain the CPU limit of the server and large data set stress the I/O capacity of the disk drive.

How does Sharding work?

Sharding determines the problem with horizontal scaling. It breaks the system dataset and store it over multiple servers, adding new servers to increase the volume as needed.



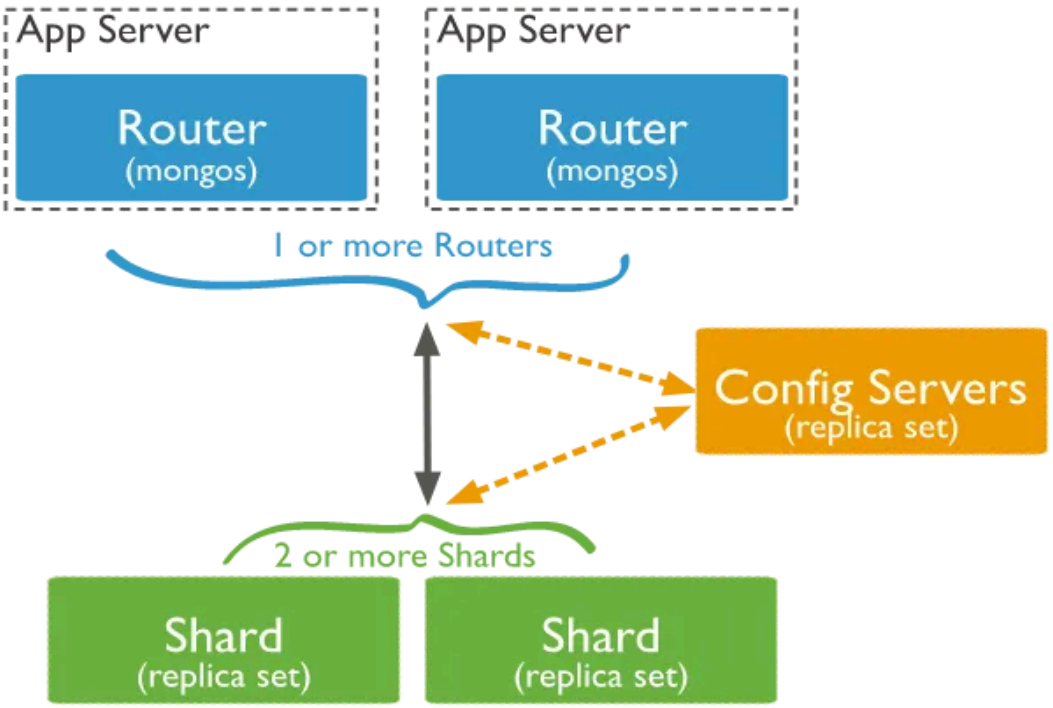
Advantages of Sharding

Sharding adds more server to a data field automatically adjust data loads across various servers

The number of operations each shard manage got reduced

It also increases the write capacity by splitting the write load over multiple instances.

Together using all three of them sharding can be enabled in MongoDB. The architecture diagram is as follows:



Shard Keys: A shard key in MongoDB is a designated field used to partition data across shards in a sharded cluster. It determines how data is distributed among shards, impacting query performance and scalability.

Chunks: A chunk is a contiguous range of data stored within a shard, dynamically split or merged based on the defined shard key, allowing for efficient distribution and management of data across the sharded cluster.