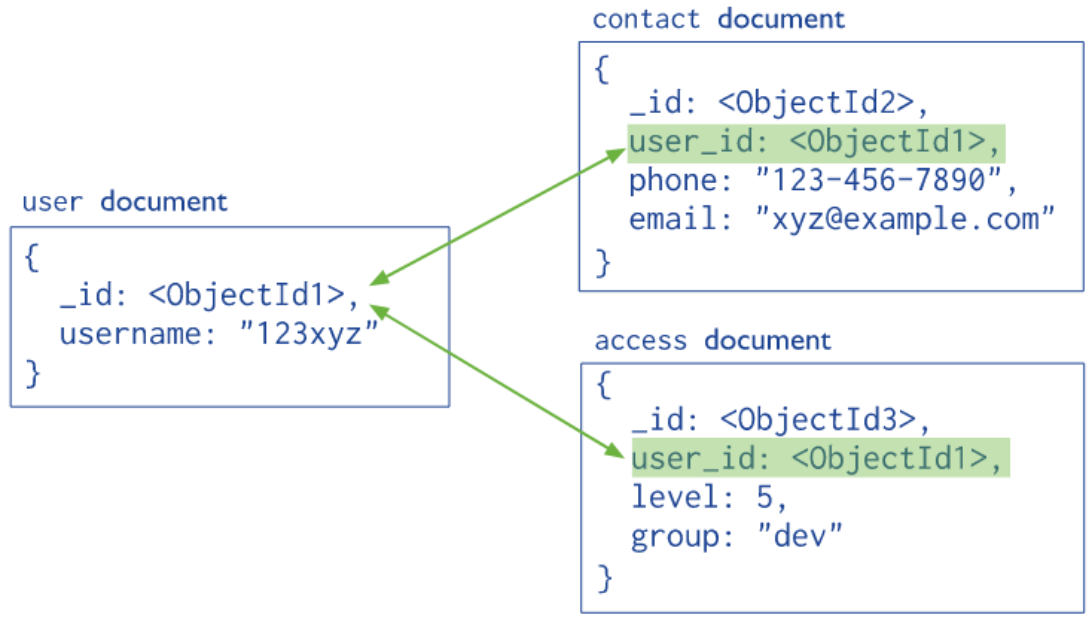
DESIGNING DOCUMENT DATABASES

DOCUMENT STRUCTURE

- Key decision to make when designing document dbs revolves around docuemnt structure and how to represent relationships between data.
- Simplest way: Embedded data
  - Represent relationships by storing related data in a single document structure
    - Object (embedded document)
    - Array
  - Denormalized
  - Allows retrieval and manipulation in a single db operation

REFERENCES

- Store relationships between data through links or references from one document to another
- Applications can then resolve these references to access related data
- Normalized



MONGO DB: HOW TO DESIGN A DATABASE MODEL

- Modeling relations
  - One to few: Embed the information in the container
    - Example: 1 person has a few addresses. Inside the person's object add an array with every address
  - One to many(thousands): Child referencing
    - Example: A product has multiple replacement parts. Each product has a reference to the replacement part ID.
  - One to squillions: Parent referencing
    - Example: An application has multiple log entries. Each entry has a reference to the application.

ONE-TO-FEW

```
{
    "_id": "ObjectId('AAA')",
    "name": "Joe Karlsson",
    "company": "MongoDB",
    "twitter": "@JoeKarlsson1",
    "twitch": "joe_karlsson",
    "tiktok": "joekarlsson",
    "website": "joekarlsson.com",
    "addresses": [
        { "street": "123 Sesame St", "city": "Anytown", "cc": "USA" },
        { "street": "123 Avenue Q",  "city": "New York", "cc": "USA" }
    ]
}
```

ONE-TO-MANY

```
{
    "name": "left-handed smoke shifter",
    "manufacturer": "Acme Corp",
    "catalog_number": "1234",
    "parts": ["ObjectID('AAAA')",
"ObjectID('BBBB')", "ObjectID('CCCC')"]
}
```

```
{
    "_id" : "ObjectID('AAAA')",
    "partno" : "123-aff-456",
    "name" : "#4 grommet",
    "qty": "94",
    "cost": "0.94",
    "price":" 3.99"
}
```

ONE-TO-SQUILLIONS

```
{
    "_id": ObjectID("AAAB"),
    "name": "goofy.example.com",
    "ipaddr": "127.66.66.66"
}
```

```
{
    "time": ISODate("2014-03-
28T09:42:41.382Z"),
    "message": "cpu is on fire!",
    "host": ObjectID("AAAB")
}
```

TWO-WAY REFERENCING

- Two-way referencing
  - Sometimes, you need both references: from the "one" side to the "many" side and references from the "many" side to the "one" side
    - Example: A product has multiple replacement parts. Each part has a reference to the product ID, and also each product has an array with the IDs of its replacement parts
    - **To consider:** Every update or delete operation must be performed twice

MANY-TO-MANY

```
{
    "_id": ObjectID("AAF1"),
    "name": "Kate Monster",
    "tasks": [ObjectID("ADF9"),
ObjectID("AE02"), ObjectID("AE73")]
}
```

```
{
    "_id": ObjectID("ADF9"),
    "description": "Write blog post about
MongoDB schema design",
    "due_date": ISODate("2014-04-01"),
    "owners": [ObjectID("AAF1"),
ObjectID("BB3G")]
}
```

DENORMALIZATION

- Process to eliminate the need to perform join queries for certain cases, at the price of some additional complexity when performing updates

TIP: Use denormalization when there's a high ratio of reads to updates

- Many-to-One denormalization
  - Used when there's a need for constant access to certain properties of the components
  - Example: A product has multiple replacement parts and you need to create lists of products with the names of the replacement parts.
    - Solution: Denormalize the replacement parts' names and put the information on every element into the parts' array
- One-to-Many
  - Used when there's a need for constant access to certain properties of the parent element
  - To ponder: Since you put additional information on every component, updating the parent originates multiple additional updates

GENERAL GUIDELINES

1. Favor embedding unless there is a compelling reason not to
2. If you need to access an object on its own, do not to embed it
3. If there are more than a couple of hundred documents on the "many" side, don't embed them
4. Consider the write/read ratio when denormalizing. A field that will mostly be read and only seldom updated is a good candidate for denormalization
5. How you model your data depends – entirely – on your particular application's data access patterns. You want to structure your data to match the ways that your application queries and updates it.

REFERENCES

- MongoDB Schema Design Best Practices:
  - https://www.mongodb.com/developer/article/mongodb-schema-design-best-practices/
- MongoDB Data Model Design
  - https://www.mongodb.com/docs/manual/core/data-model-design/#std-label-data-modeling-embedding

INDEXING DOCUMENT DATABASES

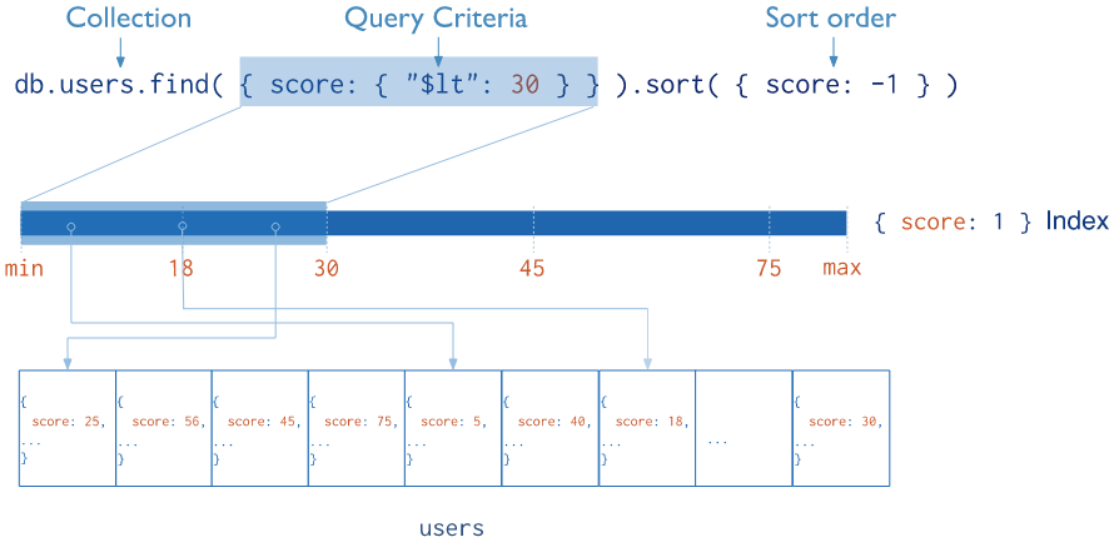
INDEXES

- Definition
  - An index stores the value of a specific field or set of fields, ordered by the value of the field
  - The ordering of the index entries supports efficient equality matches and range-based query operations
  - MongoDB creates a unique index on the _id field during the creation of a collection
- Types
  - Single field
  - Compound index
  - Multikey index
  - Geospatial index
  - Text index
  - Hash index

Syntax

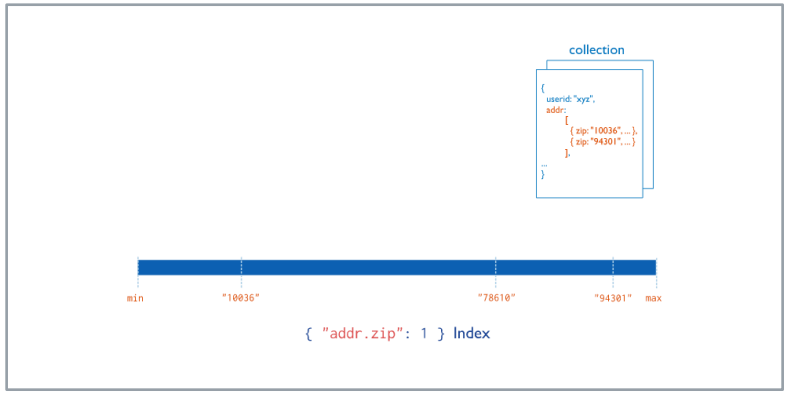
db.collection.createIndex(<key and index type specification>, <options>)

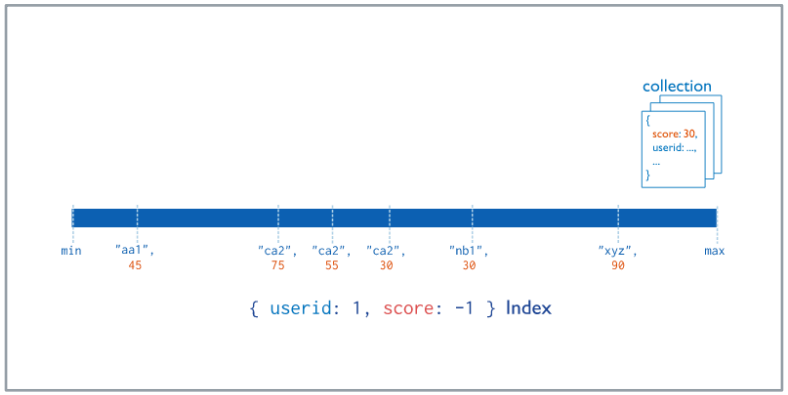
db.collection.createIndex({ name: -1 })

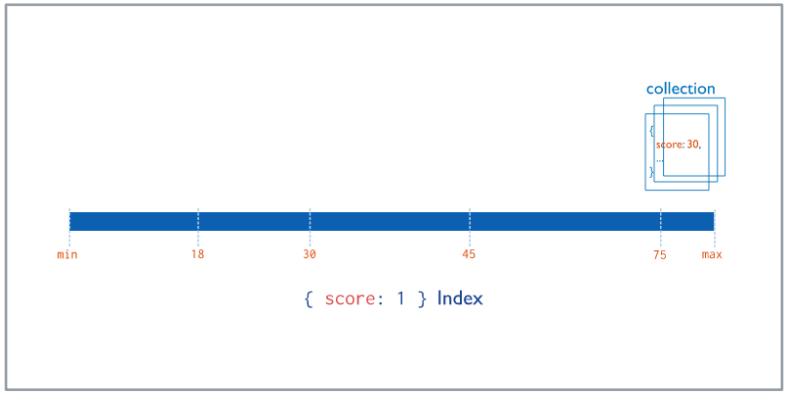


TYPES

- **Single field index**
  - Ascending/descending indexes on a single field of a document.
- **Compound index**
  - Ascending/descending indexes on multiple fields of a document.
- **Multikey index**
  - Index the content stored in arrays
  - It creates separate index entries for every element of the array
- **Text index**
  - It does not store language-specific stop words (e.g. “the”, “a”, “or”) and stem the words in a collection to only store root words
- **Geospatial index**
  - Used for efficient queries of geospatial coordinate data.
- **Hashed index**
  - Uses hash table instead of B-tree index structure. Only used for hash based sharding.
- **Cluster index**
  - Allows creation of clustered collections.







TRANSACTIONS IN DOCUMENT DATABASES

TRANSACTIONS

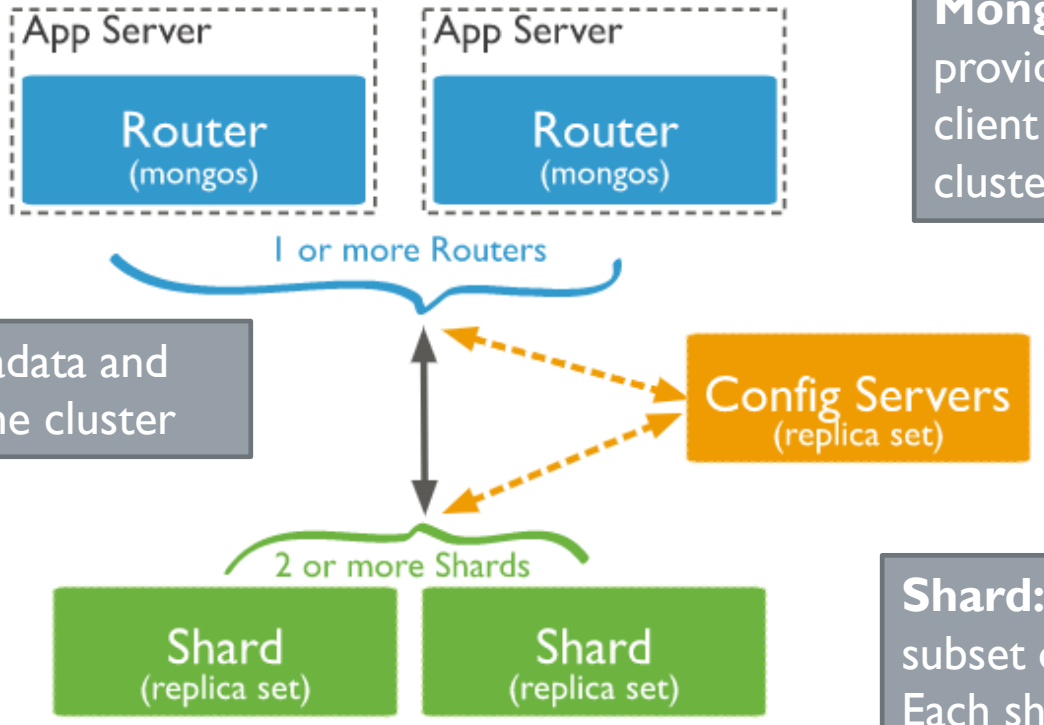
- An operation on a single document is atomic.
- Because you can use embedded documents and arrays to capture relationships between data in a single document structure instead of normalizing across multiple documents and collections, this single-document atomicity obviates the need for multi-document transactions for many practical use cases.
- For situations that require atomicity of reads and writes to multiple documents (in a single or multiple collections), MongoDB supports multi-document transactions.
- Distributed transactions can be used across multiple operations, collections, databases, documents.
  - In most cases, multi-document transaction incurs a greater performance cost over single document writes, and the availability of multi-document transactions should not be a replacement for effective schema design.

SHARDING

SHARDING

- **Definition**
  - Method for distributing data across multiple machines.
  - It's specially used to support deployments with very large data sets and high throughput operations.
- **Scenarios**
  - Database systems with large data sets or high throughput applications challenge the server's capacity
    - High query rates can exhaust the CPU capacity of the server
    - Working set sizes larger than the system's RAM stress the I/O capacity of disk drives
- Methods for addressing system growth
  - ***Vertical Scaling.*** Increase the capacity of a server (use a more powerful CPU, add more RAM, or increase the amount of storage space)
  - ***Horizontal Scaling.*** Involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required.

SHARDING ARCHITECTURE



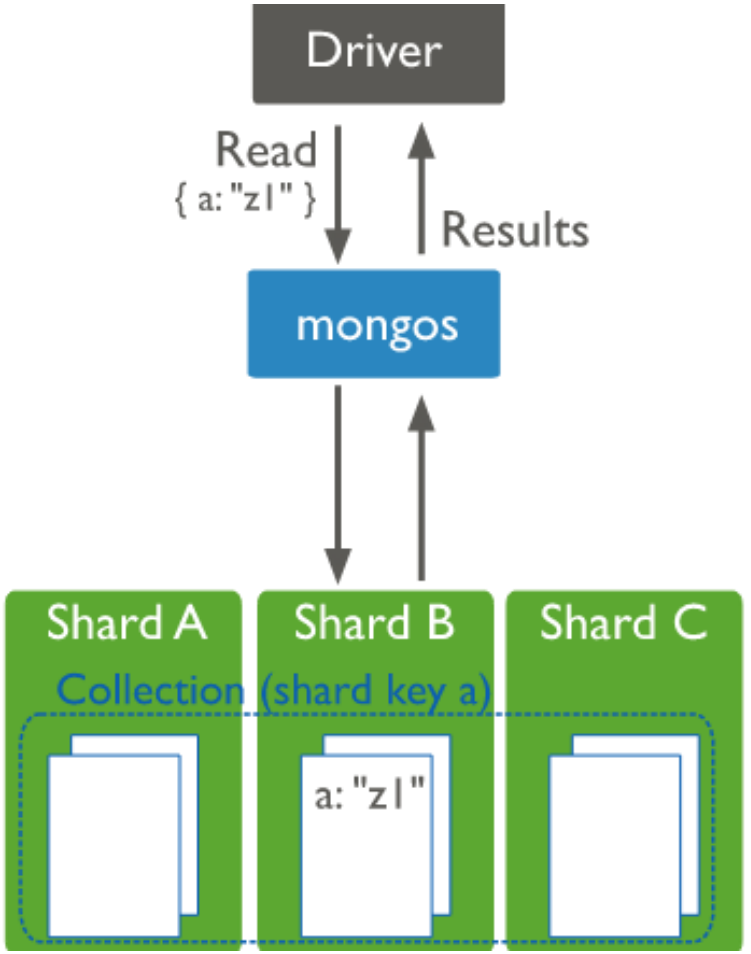
Mongos: Act as query routers, providing an interface between client applications and the sharded cluster.

Config server: Store metadata and configuration settings for the cluster

Shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.

MONGOS

- mongos instances route queries and write operations to shards in a sharded cluster.
- mongos provide the only interface to a sharded cluster from the perspective of applications.
  - Applications never connect or communicate directly with the shards.
- A mongos instance routes a query to a cluster by:
  - Determining the list of shards that must receive the query.
  - Establishing a cursor on all targeted shards.
  - The mongos then merges the data from each of the targeted shards and returns the result document.
    - Certain query modifiers, such as sorting, are performed on each shard before mongos retrieves the results.
- mongos performs a broadcast operation for queries that do not include the shard key, routing queries to all shards in the cluster.

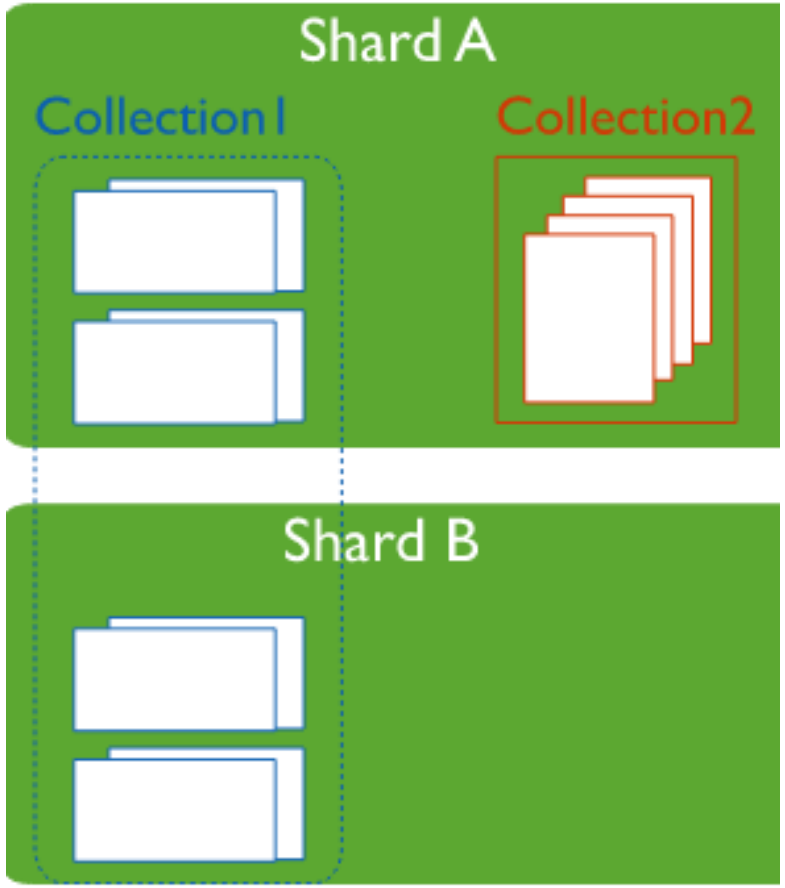


CONFIG SERVERS

- Config servers store the metadata for a sharded cluster.
- The metadata reflects state and organization for all data and components within the sharded cluster.
- The metadata includes the list of chunks on every shard and the ranges that define the chunks.
- The config servers also store Authentication configuration information such as Role-Based Access Control or internal authentication settings for the cluster.
- Each sharded cluster must have its own config servers.

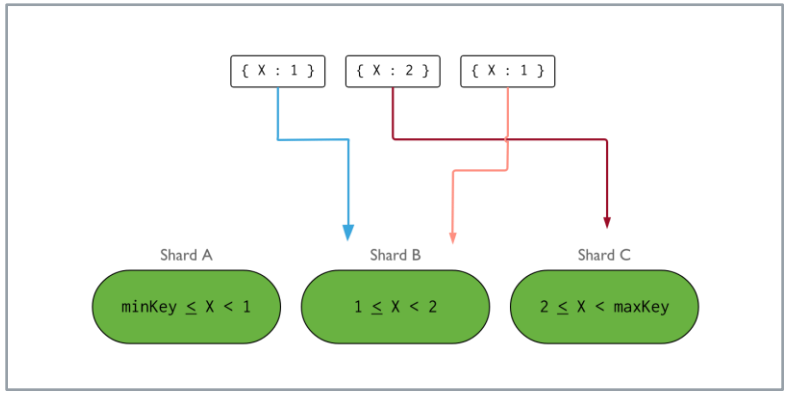
SHARDS

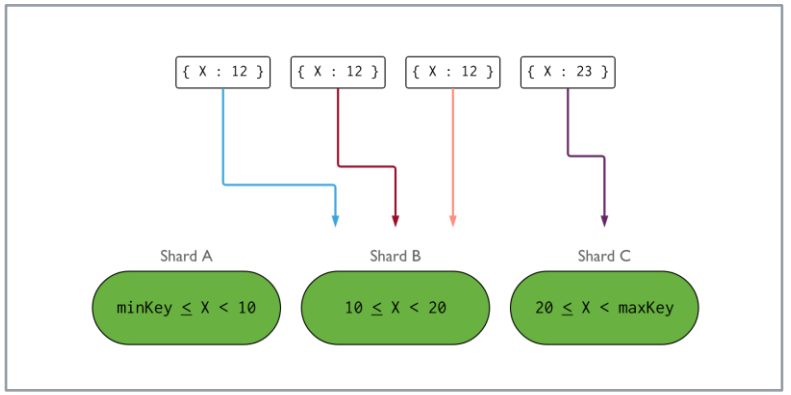
- A shard contains a subset of sharded data for a sharded cluster. Together, the cluster's shards hold the entire data set for the cluster.
  - Shards must be deployed as a replica set to provide redundancy and high availability.
- Primary Shard
  - Each database in a sharded cluster has a primary shard that holds all the un-sharded collections for that database. Each database has its own primary shard.

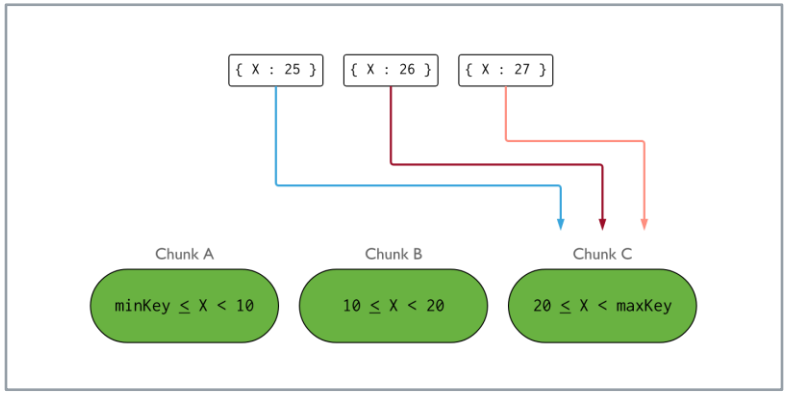


SHARD KEYS

- MongoDB uses the shard key to distribute the collection's documents across shards.
- The shard key consists of a field or multiple fields in the documents.
  - Documents in sharded collections can be missing the shard key fields.
  - Missing shard key fields are treated as having null values
- The choice of shard key affects the creation and distribution of chunks across the available shards. The distribution of data affects the efficiency and performance of operations within the sharded cluster.
- The ideal shard key allows MongoDB to distribute documents evenly throughout the cluster while also facilitating common query patterns.
- When you choose your shard key, consider:
  - the cardinality of the shard key
  - the frequency with which shard key values occur
  - whether a potential shard key grows monotonically
  - Sharding Query Patterns
  - Shard Key Limitations
    - Version >= 4.4 no shard key limitations
    - Version < 4.4 512 max shard key







REFERENCES

- https://www.mongodb.com/docs/manual/indexes/
- https://www.mongodb.com/docs/v6.0/core/transactions/
- https://www.mongodb.com/docs/manual/sharding/
- https://www.mongodb.com/docs/manual/core/sharding-choose-a-shard-key/#std-label-sharding-shard-key-selection