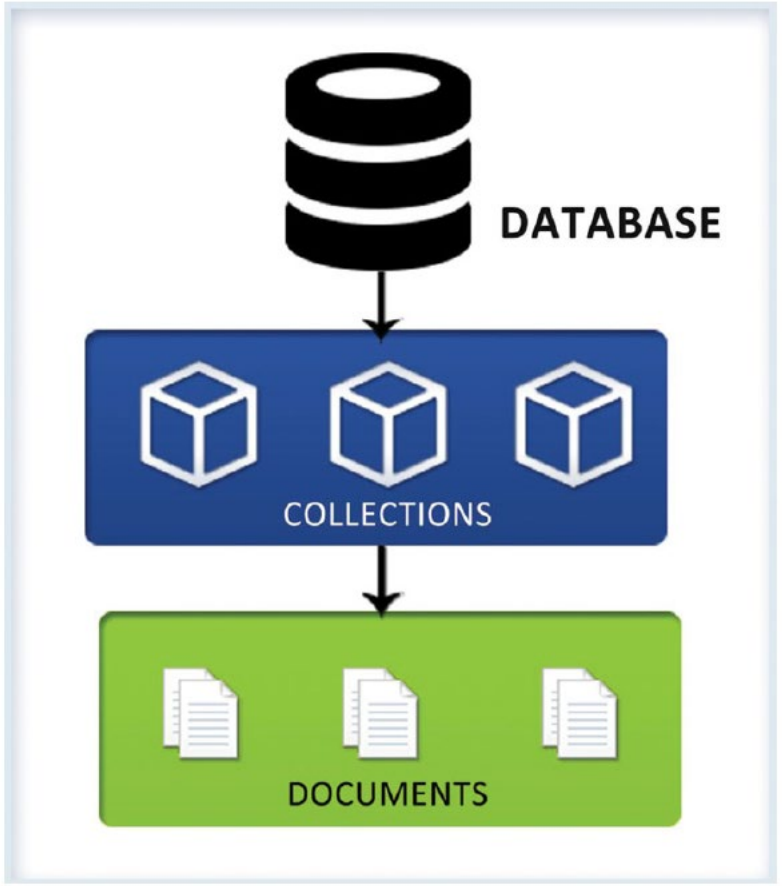
# Unit 5 - MongoDB

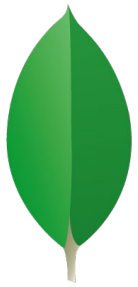
# MongoDB Introduction

- 2007, Dwight Merriman, Eliot Horowitz decided to build a database that would not comply with the RDBMS model
- Design Philosophy
  - Speed, Scalability, and Agility
  - Non-Relational Approach
  - JSON-Based Document Store
  - Performance vs. Features
  - Running the Database Anywhere

# MongoDB Data Model

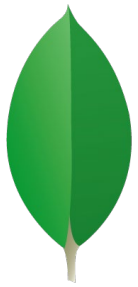


Database

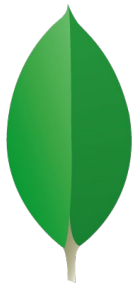


**Database**

- Made up of Multiple **Collections**.
- Created **on-the-fly** when referenced for the first time.

Collection  **Table**

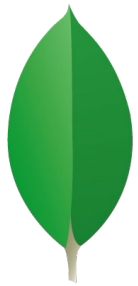
- Schema-less, and contains **Documents.**
- **Indexable** by one/more keys.
- Created **on-the-fly** when referenced for the first time.
- **Capped Collections**: Fixed size, older records get dropped after reaching the limit.

Document  **Record/Row**

- Stored in a **Collection**.
- Can have **_id** key – works like Primary keys in MySQL.
- Supported Relationships – **Embedded (or) References**.
- Document storage in **BSON** (Binary form of JSON).

# Understanding the Document Model.

```
var p = {
'_id': '3432',
'author': DBRef('User', 2),
'title': 'Introduction to MongoDB',
'body': 'MongoDB is an open sources.. ',
'timestamp': Date('01-04-12'),
'tags': ['MongoDB', 'NoSQL'],
'comments': [{'author': DBRef('User', 4),
                        'date': Date('02-04-12'),
                        'text': 'Did you see.. ',
                        'upvotes': 7, ... ]
}
> db.posts.save(p) //p is a document;
```



# MongoDB-Data Model-Embedded

- Embedded/De-normalized
- can have (embed) all the related data in a single document

```
{
        _id: ,
        Emp_ID: "10025AE336"
        Personal_details:{
                First_Name: "Radhika",
                Last_Name: "Sharma",
                Date_Of_Birth: "1995-09-26"
        },
        Contact: {
                e-mail: "radhika_sharma.123@gmail.com",
                phone: "9848022338"
        },
        Address: {
                city: "Hyderabad",
                Area: "Madapur",
                State: "Telangana"
        }
}
```

# MongoDB-Data Model-Normalized

- refer the sub documents in the original document, using references

**Employee:**

```
{
        _id: <ObjectId101>,
        Emp_ID: "10025AE336"
}
```

**Personal_details:**

```
{
        _id: <ObjectId102>,
        empDocID: " ObjectId101",
        First_Name: "Radhika",
        Last_Name: "Sharma",
        Date_Of_Birth: "1995-09-26"
}
```

**Contact:**

```
{
        _id: <ObjectId103>,
        empDocID: " ObjectId101",
        e-mail: "radhika_sharma.123@gmail.com",
        phone: "9848022338"
}
```

**Address:**

```
{
        _id: <ObjectId104>,
        empDocID: " ObjectId101",
        city: "Hyderabad",
        Area: "Madapur",
        State: "Telangana"
}
```

# Installation

- MongoDB Community Server Edition for Windows10 64 bit
- FIRST RUN COMMANDS
- ----------
- #Step 1 - Create a data folder
- C:\>md data
- C:\>md data\db

- #Step 2 - Specify this data path to the mongodb.exe
- C:\Users\XYZ> cd C:\Program Files\MongoDB\Server\4.2\bin
- C:\Program Files\MongoDB\Server\4.2\bin>mongod.exe --dbpath "C:\data"
- #This should show waiting for connections

- #Step 3 - Run mongodb.exe from another command prompt
- C:\Program Files\MongoDB\Server\4.2\bin>mongo.exe

- FROM NEXT RUN ONWARDS JUST EXECUTE
- ----------
- #Two different command prompt
- C:\Program Files\MongoDB\Server\4.2\bin>mongod.exe --dbpath "C:\data"
- C:\Program Files\MongoDB\Server\4.2\bin>mongo.exe

# Commands – Create database

- Create database:
- Syntax:

  use DATABASE_NAME
- Example:

  use sample
- Output:

```
> use sample
switched to db sample
```

# Commands – Show databases

Currently selected database

>db

List of databases – Empty dbs are not listed

>show dbs

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

```
> db
sample
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

# Commands – Create collections

Syntax:

db.createCollection(name, options)

In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

Example:

```
> db.createCollection("books")
{ "ok" : 1 }
>
```

# Commands - Collections

- Check Created collections:

show collections

```
> show collections
books
```

```
> db.createCollection("players", { capped : true, size : 6142800, max : 10000 } )
{ "ok" : 1 }
```

```
> db.movies.insert({"name" : "chamber of secrets"})
WriteResult({ "nInserted" : 1 })
```

```
> show collections
books
movies
players
```

# Some of the data types supported

- **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** − This type is used to store a boolean (true/ false) value.
- **Double** − This type is used to store floating point values.
- **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** − This type is used to store arrays or list or multiple values into one key.
- **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** − This datatype is used for embedded documents.
- **Null** − This type is used to store a Null value.
- **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date** − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** − This datatype is used to store the document’s ID.
- **Binary data** − This datatype is used to store binary data.
- **Code** − This datatype is used to store JavaScript code into the document.

# Commands – Insert document

- Syntax
- db.COLLECTION_NAME.insert(document)
- Single and multiple inserts possible – insertone vs insertmany
- Save is the other alternate for insert

# Commands – Query document

- Syntax
- >db.COLLECTION_NAME.find()
- Formatted Display
- >db.COLLECTION_NAME.find().pretty()
- Return any one document
- >db.COLLECTIONNAME.findOne()
- AND OR where clause is possible

# Commands – Update document

- Syntax

>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)

- By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.
- >db.books.update({'title':'MongoDB Overview'}, {$set:{'title':'New MongoDB Tutorial'}},{multi:true})

# Commands – Delete document

- Syntax

>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

- If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. **This is equivalent of SQL's truncate command.**
- > db.books.remove({}) WriteResult({ "nRemoved" : 2 }) > db.mycol.find() >

# Mongo DB Data Model Contd

- MongoDB is a document-based database. It uses Binary JSON for storing its data

```
{
"_id" : 1,
"name" : { "first" : "John", "last" : "Doe" },
"publications" : [
          {
            "title" : "First Book",
            "year" : 1989,
            "publisher" : "publisher1"
          },
          { "title" : "Second Book",
            "year" : 1999,
            "publisher" : "publisher2"
          }
]
}
```

# Mongo DB Data Model Contd

- Binary JSON - MongoDB stores the JSON document in a binary-encoded format. This is termed as BSON. The BSON data model is an **extended form of the JSON** data model
- BSON document is **fast**, highly traversable, and lightweight
- Supports **embedding of arrays** and objects within other arrays,
- also enables MongoDB to **reach inside the objects** to build indexes and match objects against queried expressions, both on top-level and nested BSON keys

# Mongo DB Data Model Contd _id field

- Documents are made up of **key-value pairs**
- Although a document can be compared to a row in RDBMS, unlike a row, documents have **flexible schema**
- A key, which is nothing but a label, can be roughly compared to the column name in RDBMS
- A key is used for querying data from the documents. Hence, like a RDBMS primary key (used to uniquely identify each row), you need to have a key that uniquely identifies each document within a collection. This is referred to as _id in MongoDB.

# Mongo DB Data Model Contd – Capped Collection

- MongoDB has a concept of capping the collection. This means it stores the documents in the collection in the inserted order.
- As the collection reaches its limit, the documents will be removed from the collection in FIFO (first in, first out) order.
- This means that the least recently inserted documents will be removed first. This is good for use cases where the order of insertion is required to be maintained automatically, and deletion of records after a fixed size is required. One such use cases is log files that get automatically truncated after a certain size.

# Mongo DB Data Model Contd – Polymorphic Schemas

- A polymorphic schema is a schema where a collection has documents of different types or schemas.
- A good example of this schema is a collection named Users. Some user documents might have an extra fax number or email address, while others might have only phone numbers, yet all these documents coexist within the same Users collection.
- This schema is generally referred to as a polymorphic schema

# MongoDB Architecture

- Core Processes
  - **mongod** , which is the core database process
  - **mongos** , which is the controller and query router for sharded clusters
  - **mongo** , which is the interactive MongoDB shell
- mongod
- This daemon handles all the data requests, manages the data format, and performs operations for background management.
- When a mongod is run without any arguments, it connects to the default data directory, which is C:\data\db or /data/db , and default port 27017, where it listens for socket connections.
- It’s important to ensure that the data directory exists and you have write permissions to the directory before the mongod process is started.

# MongoDB Architecture

- mongo
  - mongo provides an interactive JavaScript interface for the developer to test queries and operations directly on the database and for the system administrators to manage the database.
  - This is all done via the command line. When the mongo shell is started, it will connect to the default database called test.
  - This database connection value is assigned to global variable db .
  - you need to change the database from test to your database post the first connection is made. You can do this by using <databasename>.

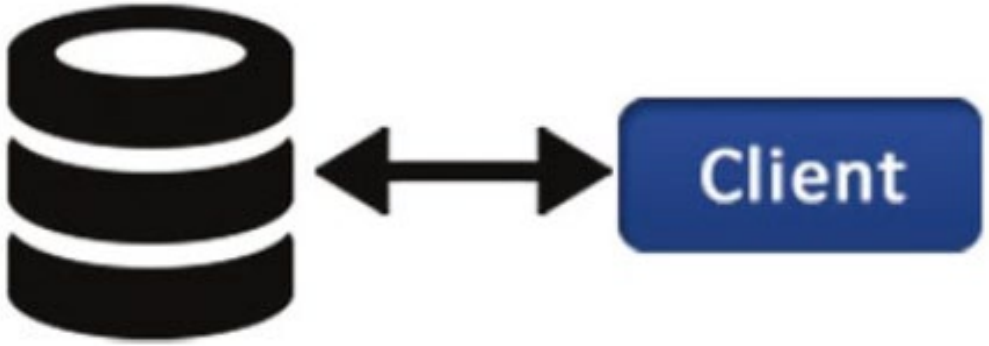
# MongoDB Architecture

- Mongos
  - mongos is used in MongoDB sharding.
  - It acts as a routing service that processes queries from the application layer and determines where in the sharded cluster the requested data is located.

# MongoDB Tools

- **mongodump** : This utility is used as part of an effective ***backup*** strategy. It creates a binary export of the database contents.
- **mongorestore** : The binary database dump created by the mongodump utility is imported to a ***new or an existing database*** using the mongorestore utility.
- **bsondump** : This utility converts the BSON files into human-readable formats such as JSON and CSV. For example, this utility can be used to read the output file generated by mongodump.
- **mongoimport , mongoexport : mongoimport** provides a method for taking data in JSON , CSV, or TSV formats and importing it into a mongod instance. Mongoexport provides a method to export data from a mongod instance into JSON, CSV, or TSV formats.
- **mongostat , mongotop , mongosniff** : These utilities provide diagnostic information related to the current operation of a mongod instance.

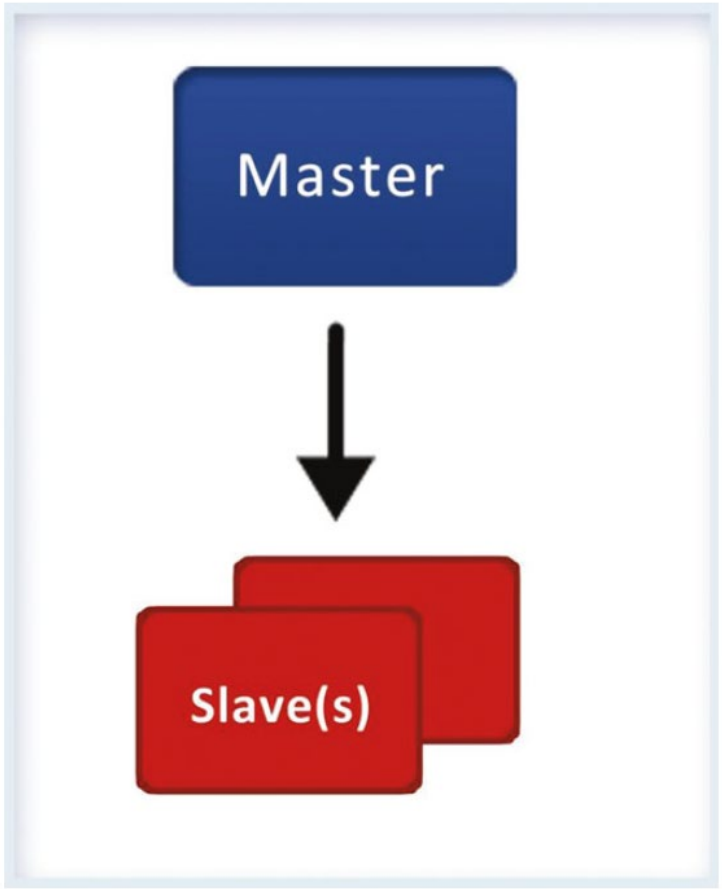
# Standalone Deployment



# Replication

- Master Slave
- Replication Set

# Replication – Master/Slave



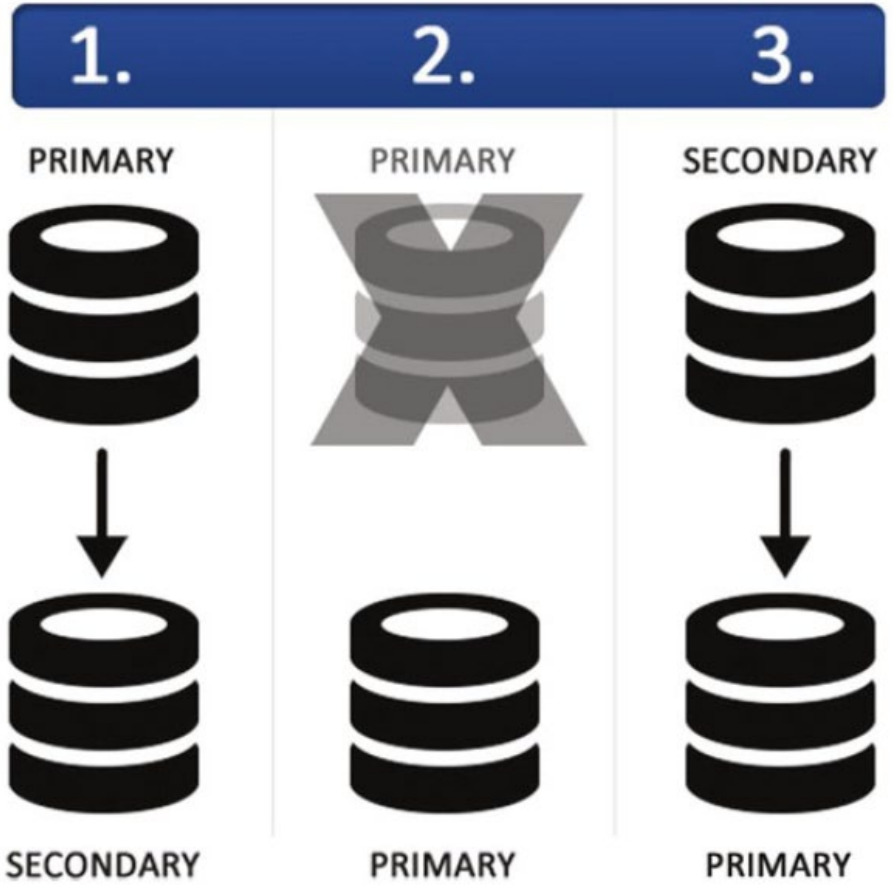
# Replication – Master/Slave

- The master node maintains a capped collection (oplog) that stores an ordered history of logical writes to the database.
- The slaves replicate the data using this oplog collection. Since the oplog is a capped collection, if the slave's state is far behind the master's state, the slave may become out of sync.
- In that scenario, the replication will stop and manual intervention will be needed to re-establish the replication.
- There are two main reasons behind a slave becoming out of sync:
  - The slave shuts down or stops and restarts later. During this time, the oplog may have deleted the log of operations required to be applied on the slave.
  - The slave is slow in executing the updates that are available from the master.

# Replication – Replica Set

- Replica sets are basically a type of master-slave replication but they provide automatic failover.
- A replica set has one master, which is termed as primary, and multiple slaves, which are termed as secondary in the replica set context;
- However, unlike master-slave replication, there’s no one node that is fixed to be primary in the replica set.
- If a master goes down in replica set, automatically one of the slave nodes is promoted to the master.
- The clients start connecting to the new master, and both data and application will remain available. In a replica set, this failover happens in an automated fashion.

# Replication – Replica Set



# Replication – Replica Set

1. The primary goes down, and the secondary is promoted as primary.
2. The original primary comes up, it acts as slave, and becomes the secondary node.

- The points to be noted are
  - A replica set is a mongod's cluster, which replicates among one another and ensures automatic failover.
  - In the replica set, one mongod will be the primary member and the others will be secondary members.
  - The primary member is elected by the members of the replica set. All writes are directed to the primary member whereas the secondary members replicate from the primary asynchronously using oplog.
  - The secondary's data sets reflect the primary data sets, enabling them to be promoted to primary in case of unavailability of the current primary.

# Replication – Replica Set

- Primary member : A replica set can have only one primary, which is elected by the voting nodes in the replica set. Any node with associated priority as 1 can be elected as a primary. The client redirects all the write operations to the primary member, which is then later replicated to the secondary members.
- Secondary member: A normal secondary member holds the copy of the data. The secondary member can vote and also can be a candidate for being promoted to primary in case of failover of the current primary.

# Replication – Replica Set

Types of Secondary:

- Priority 0
- Hidden
- Delayed
- Arbiters
- Non Voting

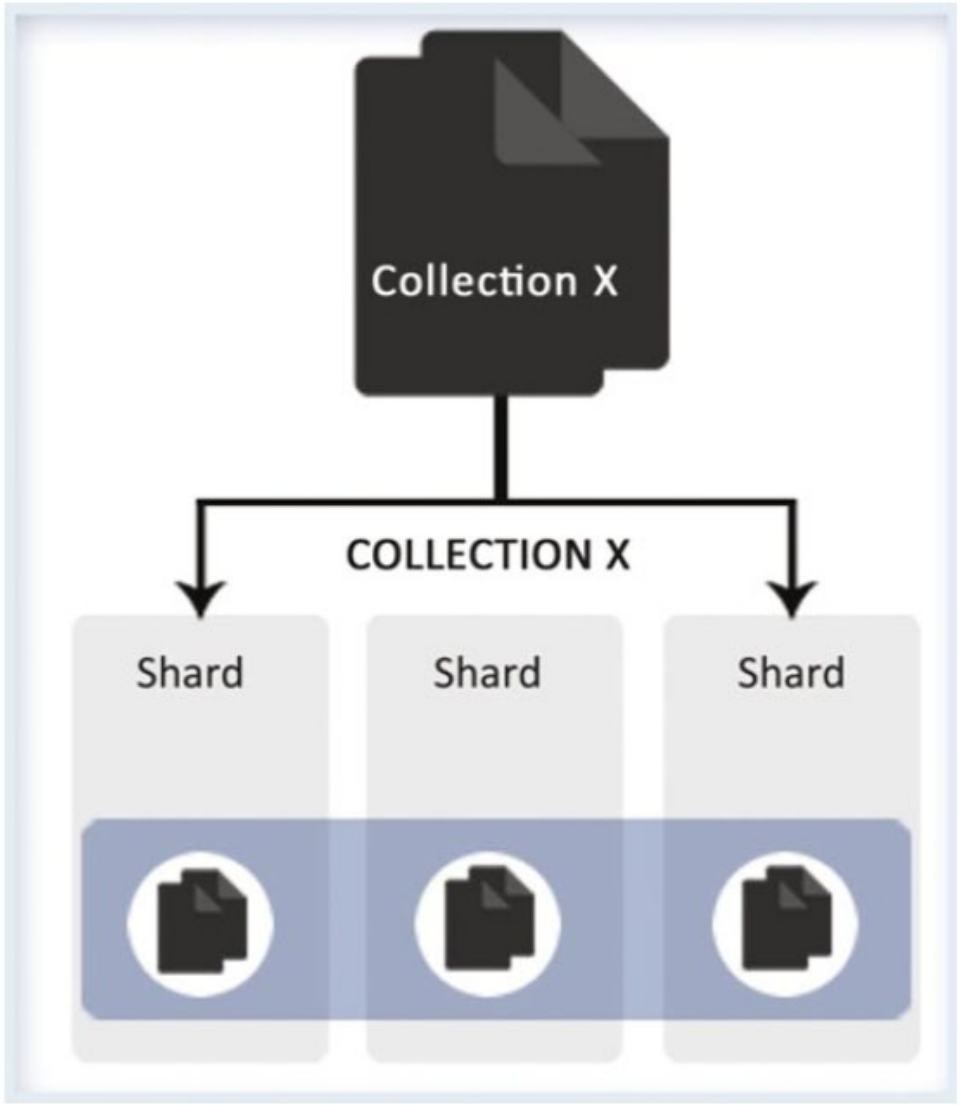
# Replication – Replica Set Elections

- If there are X servers with each server having 1 vote, then a server can become primary only when it has at least [(X/2) + 1] votes.
- If a server gets the required number of votes or more, then it will become primary.
- The primary that went down still remains part of the set; when it is up, it will act as a secondary server until the time it gets a majority of votes again.

# Sharding in MongoDB

- ideally the working set should fit in memory. The working set consists of the most requently accessed data and indexes
- the scaling is handled by scaling out the data horizontally, is also called sharding
- Sharding addresses the challenges of scaling to support large data sets and high throughput by horizontally dividing the datasets across servers where each server is responsible for handling its part of data and no one server is burdened. These servers are also called **shards**
- Every shard is an independent database. All the shards collectively make up a single logical database
- Sharding reduces the operations count handled by each shard. For example, when data is inserted, only the shards responsible for storing those records need to be accessed.
- The processes that need to be handled by each shard reduce as the cluster grows because the subset of data that the shard holds reduces. This leads to an increase in the throughput and capacity horizontally.
- Let’s assume you have a database that is **1TB** in size. If the number of shards is 4, you will have approximately 265GB of data handled by each shard, whereas if the number of shards is increased to 40, only 25GB of data will be held on each shard.

# Sharding in MongoDB



# Sharding in MongoDB

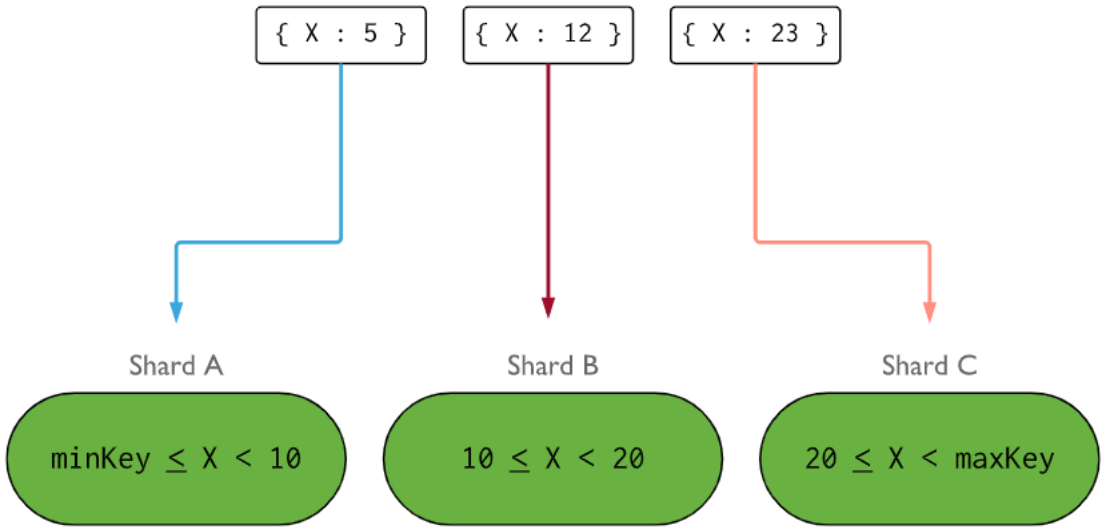
- Sharding increases deployment complexity
- Use sharding in the following instances:
  - The size of the dataset is huge and it has started challenging the capacity of a single system.
  - Since memory is used by MongoDB for quickly fetching data, it becomes important to scale out when the active work set limits are set to reach.
  - If the application is write-intensive, sharding can be used to spread the writes across multiple servers.
- Sharding is enabled in MongoDB via sharded clusters. The following are the components of a sharded cluster:
  - Shards
  - mongos
  - Config servers
- The shard is the component where the actual data is stored. For the sharded cluster, it holds a subset of data and can either be a mongod or a replica set. All shard's data combined together forms the complete dataset for the sharded cluster.
- Sharding is enabled per collection basis, so there might be collections that are not sharded. In every sharded cluster there's a primary shard where all the unsharded collections are placed in addition to the sharded collection data.

# Sharding in MongoDB

- Config servers hold the sharded cluster’s metadata. This metadata depicts the sharded system state and organization.
- The config server stores data for a single sharded cluster. The config servers should be available for the proper functioning of the cluster.
- One config server can lead to a cluster’s single point of failure. For production deployment it’s recommended to have at least three config servers, so that the cluster keeps functioning even if one config server is not accessible.
- A config server stores the data in the config database, which enables routing of the client requests to the respective data. This database should not be updated.
- MongoDB writes data to the config server only when the data distribution has changed for balancing the cluster.The **mongos** act as the routers. They are responsible for routing the read and write request from the application to the shards

# Types of Sharding – Range based sharding

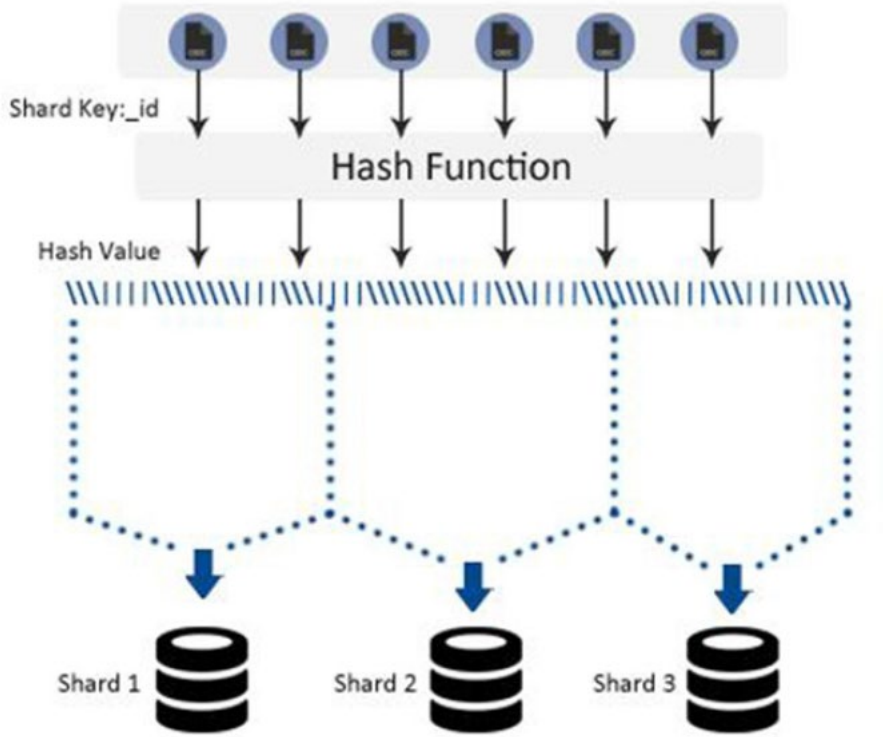
- Any indexed single/compound field that exists within all documents of the collection can be a **shard key**.
- MongoDB divides the documents based on the value of the field into chunks and distributes them across the shards
- In range-based partitioning , the shard key values are divided into ranges. The following image illustrates a sharded cluster using the field X as the shard key. If the values for X have a large range, low frequency, and change at a non-monotonic rate,



# Types of Sharding – Hash based sharding

- In hash-based partitioning , the data is distributed on the basis of the hash value of the shard field. If selected, this will lead to a more random distribution compared to range-based partitioning.
- It’s unlikely that the documents with close shard key will be part of the same chunk. For example, for ranges based on the hash of the id field, there will be a straight line of hash values, which will again be partitioned on basis of the number of shards. On the basis of the hash values, the documents will lie in either of the shards.
- Hashed keys are ideal for shard keys with fields that change monotonically like ObjectId values or timestamps.

A good example of this is the default _id field,
assuming it only contains ObjectId values.



# Chunks

- The data is moved between the shards in form of chunks. The shard key range is further partitioned into subranges, which are also termed as **chunks**
- For a sharded cluster, 64MB is the default chunk size. In most situations, this is an apt size for chunk slitting and migration.
- Let's discuss the execution of sharding and chunks with an example. Say you have a blog posts collection which is sharded on the field date . This implies that the collection will be split up on the basis of the date field values. Let's assume further that you have three shards. In this scenario the data might be distributed across shards as follows:
- Shard #1: Beginning of time up to July 2009
- Shard #2: August 2009 to December 2009
- Shard #3: January 2010 to through the end of time
- In order to retrieve documents from January 1, 2010 until today, the query is sent to mongos. In this scenario,
  1. The client queries mongos.
  2. The mongos know which shards have the data, so mongos sends the queries to Shard #3.
  3. Shard #3 executes the query and returns the results to mongos.
  4. Mongos combines the data received from various shards, which in this case is Shard #3 only, and returns the final result back to the client.
- Let's consider another scenario where you insert a new document. The new document has today's date.
- The sequences of events are as follows:
  1. The document is sent to the mongos.
  2. Mongos checks the date and on basis of that, sends the document to Shard #3.
  3. Shard #3 inserts the document.

# AWS DynamoDB

- Serverless
- Key value + Document model
- Availability, durability, and fault tolerance are built-in
- ACID transactions
- Active-active replication with global tables
- Amazon DynamoDB Streams as part of an event-driven architecture
- Secondary indexes
- Fine grained access control
- Encryption at rest
- Point-in-time recovery
- On-demand backup and restore
- Read/write capacity modes
- Standard Infrequent Access (Standard-IA) table class