SESSION 8
MONGODB REPLICATION AND SHARDING

Learning Objectives

In this session, students will learn to:

- Explain replication
- Describe the ways to implement replication
- Explain sharding
- Describe the ways to implement sharding

When dealing with large data sets ensuring high availability of the database is the most important requirement for any database technology. To ensure that the data requested by multiple global clients is available without downtime, multiple copies of data must be maintained. Replication refers to maintaining multiple copies of a database and allowing the copies to serve the clients in case the original database is unavailable. In addition, a fast-growing database can also be handled by distributing the data and maintaining it across multiple servers. This is referred to as sharding.

Replication helps in maintaining multiple copies of data on several servers to ensure high availability of data. Sharding helps in distributing data across multiple servers to manage high traffic and facilitate easy retrieval of data. This session covers the concept and implementation of replication and sharding in

MongoDB.

8.1 Replication

Replication is creating copies of existing data on different servers. It helps clients to access data without any interruptions, especially during technical glitches. Although data becomes redundant, replication facilitates high availability of data by ensuring that one copy of the data is available for access when the primary data source fails.

8.1.1 Replication Architecture

Let us now understand how MongoDB implements replication. The main data server in which the read and write operations are carried out on the database is known as the primary server. The servers which store copy of the database are called secondary servers. The complete set of a primary along with its secondary servers is called a replica set. A replica set must have a minimum of one primary and two secondary servers.

Figure 8.1 depicts the Primary-Secondary-Secondary (P-S-S) configuration of a replica set in MongoDB where a primary server has two secondary members. Each of the three servers have a separate `mongod` instance. Any write or update operation on the primary server is recorded in the operations log called the oplog. The oplog from the primary server is replicated by the secondary servers and then, the secondary servers update their respective databases from the oplog.

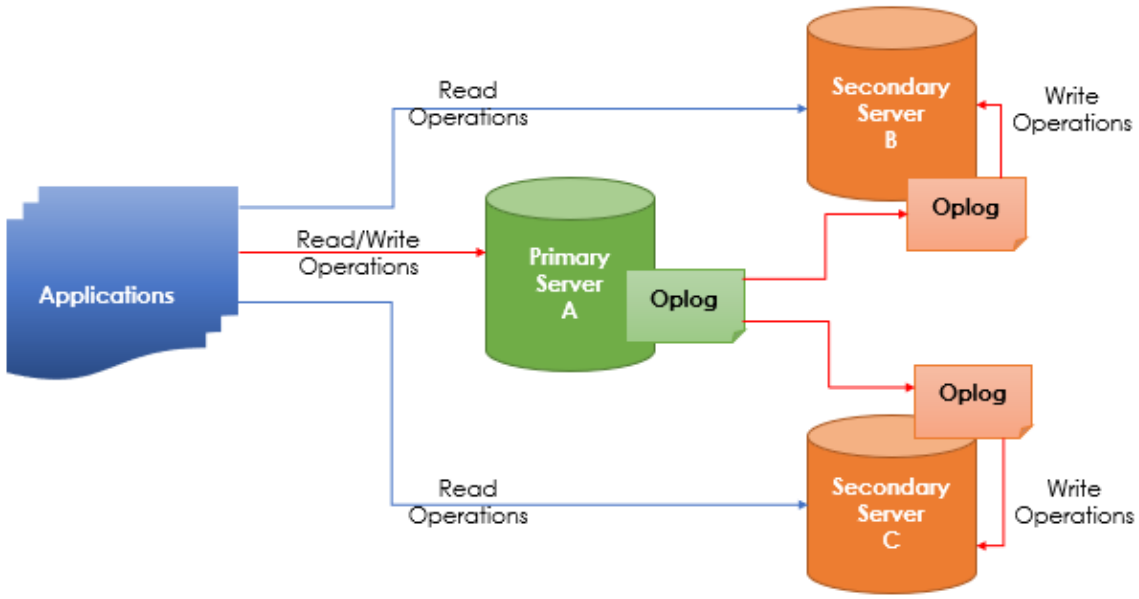
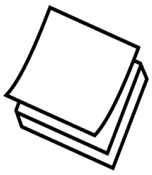


Figure 8.1: Replication Set Operations

In case the primary server fails, an election is held between the secondary servers.
Replica sets can trigger an election if:

- A new server is added.
- A new replica set is added.
- Secondary servers lose connectivity with primary server beyond the configured `timeout`(10 seconds by default).
- Replica set maintenance is scheduled.

During the election process, secondary servers can serve the read requests if configured. However, the write operation resumes only after the election process is completed. As a result of the election, one of the secondary servers becomes the primary server and will then be capable of handling both read and write operations on the database. Once the original primary server resumes its operation, the recently elected primary server becomes a secondary server once again.



A replica set can include a server that can only vote in election. Such a server is called arbiter and it does not contain copy of the databases. A configuration that has a primary server, a secondary server, and an arbiter is set to follow the Primary-Secondary-Arbiter (P-S-A) architecture.

8.1.2 Replication Methods

Table 8.1 lists the important methods employed to implement the replication process. Here, `rs` denotes replica set.

Name	Description
`rs.add`	Adds a new member to an existing replica set
`rs.addArb`	Adds an arbiter to the replica set
`rs.conf`	Returns a replica set configuration document
`rs.freeze`	Prevents the member from seeking the role of primary server in election
`rs.initiate`	Initiates a new replica set
`rs.printReplicationInfo`	Prints a formatted report of the replica set status as per the primary server
`rs.printSecondaryReplicationInfo`	Prints a formatted report of the replica set status as per the secondary server

Name	Description
`rs.reconfig`	Applies a new configuration object to an existing replica set
`rs.reconfigForPSASet`	Performs reconfiguration changes on a PSA replica set or on a replica set that is changing to a P-S-A architecture
`rs.remove`	Removes a member from a replica set
`rs.status`	Returns the state of the replica set as a document
`rs.stepDown`	Makes the current primary server to become a secondary server which forces an election
`rs.syncFrom`	Overrides the default sync target selection and sets the member of the specified target for this server to sync from

Table 8.1: Replication Methods

8.1.3 Implementing Replication Using the P-S-S Architecture

Now, let us see how to implement the P-S-S architecture in MongoDB with the help of an example. Consider that:

- The local machine will host all three servers: one primary and two secondaries.
- The `mongod` instance of the primary server will run through port `27017` and will await the requests from the application client.
- Two secondary servers will run through ports `27019` and `27020`, respectively.
- Thus, there will be three instances of `mongod` running on the local machine and `mongosh` will be used to communicate with these instances.
- Database, collection, and documents will be created in the primary server and can be read from all the three servers.
- The user must use port `27017` to run queries on the primary server.
- The user must use ports `27019` and `27020` to run queries on the secondary servers.

To implement replication:

1. Create three folders by name `rs1`, `rs2`, and `rs3` in the `c:\data` folder to store the three replica sets as shown in Figure 8.2.

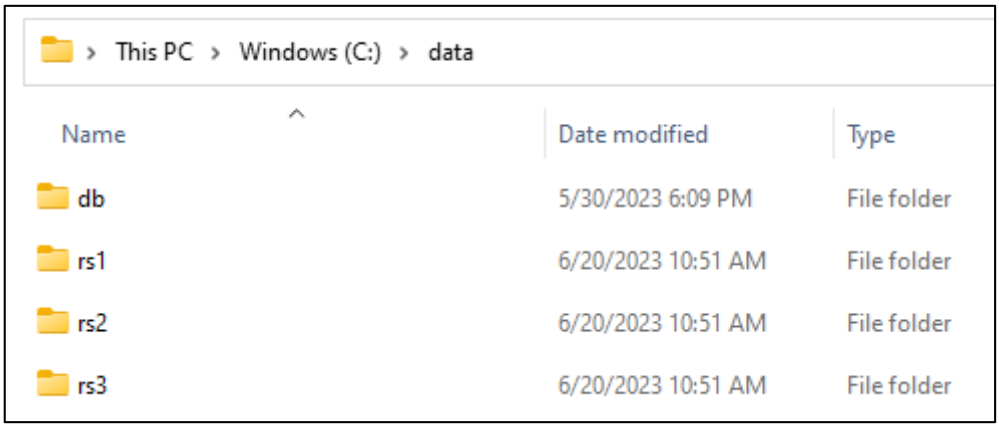


Figure 8.2: Creation of Replica Set Folders `rs1`, `rs2`, and `rs3`

2. Run `services.msc`.
3. To stop the running instance of MongoDB server. In the **Services** window, right-click MongoDB Server and then click **Stop**, as shown in Figure 8.3.

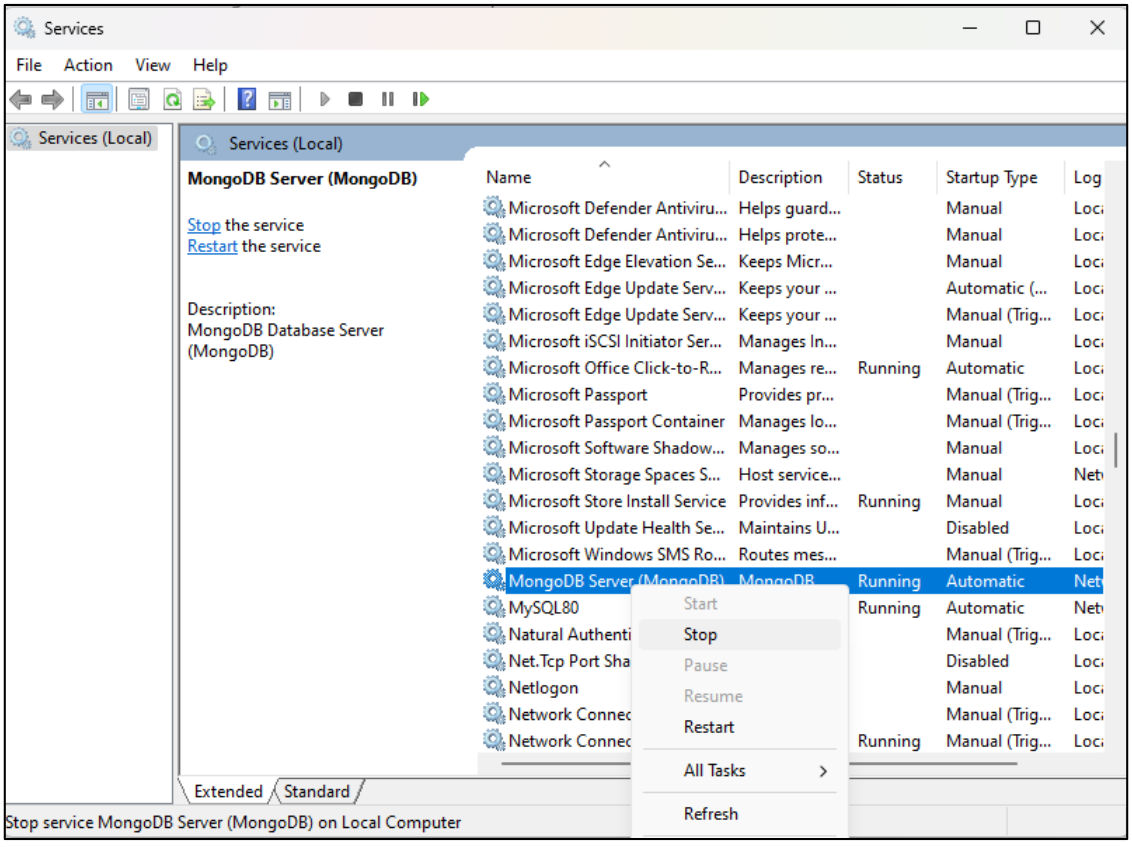


Figure 8.3: Stopping the MongoDB Instance

4. Open the command prompt.
5. Start the `mongod` instance connecting to the port 27017 using the command:

```
mongod --replSet replicaset --logpath
\data\rs1\1.log --dbpath \data\rs1 --port 27017
```

Figure 8.4 shows the output of the command.

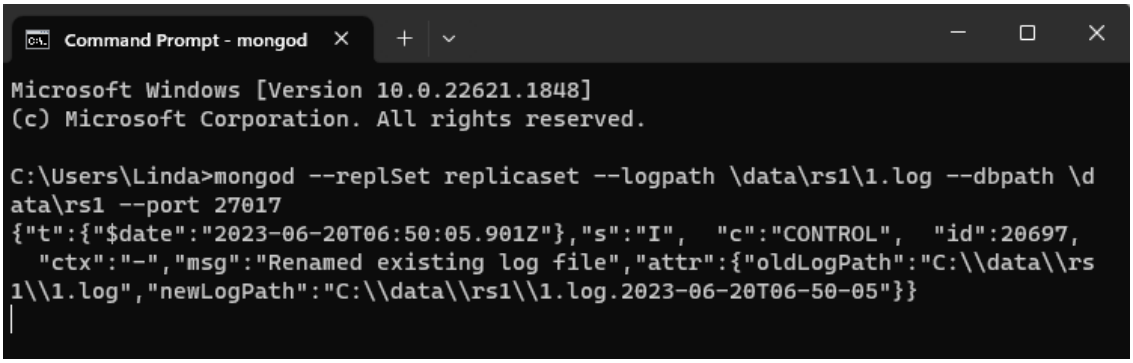
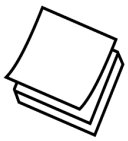


Figure 8.4: Primary Server: `mongoDb` Server Instance on Port 27017



When a user runs the command given in Figure 8.4, the output may be a simple cursor blinking if the command is run for the first time. Figure 8.4 shows the output when the same command is run for the second time.

6. In another command prompt, connect to the `mongod` instance using the `mongosh` command:

```
mongosh --port 27017
```

Figure 8.5 shows the output of the command.

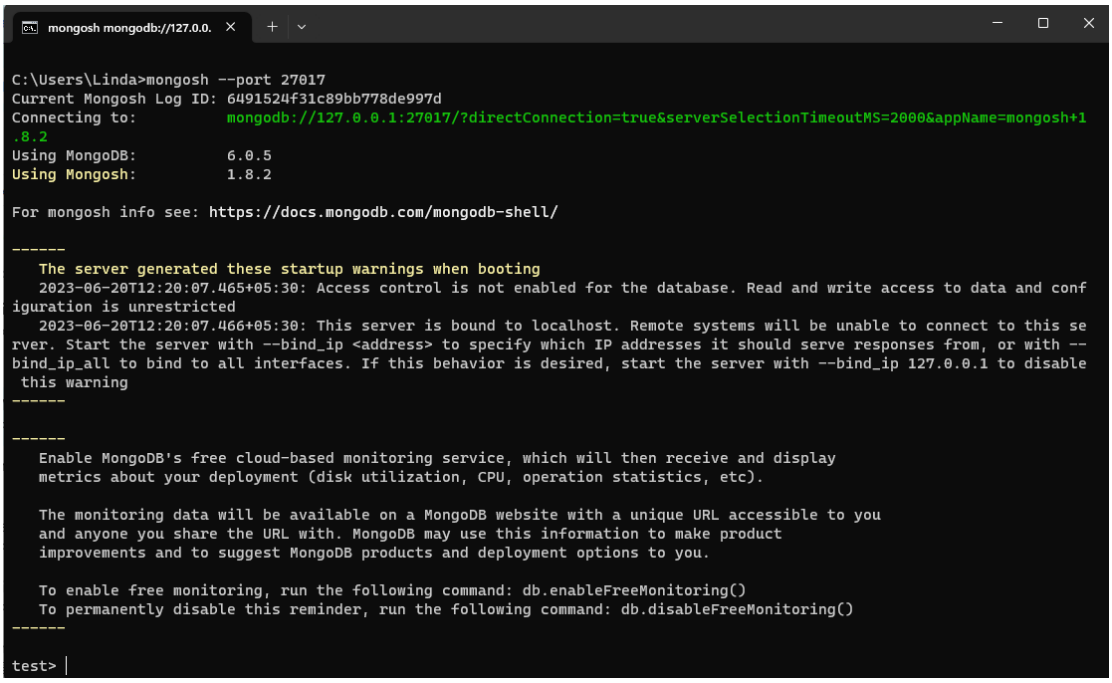


Figure 8.5: `mongoshell` Command to Connect to Port 27017

7. Configure the server to listen to port 27017 as primary server using the command as:

```
rsconfig={_id:"replicaset",members:[{_id:0,host:
"localhost:27017"}]}
```

Figure 8.6 shows the output of the command.

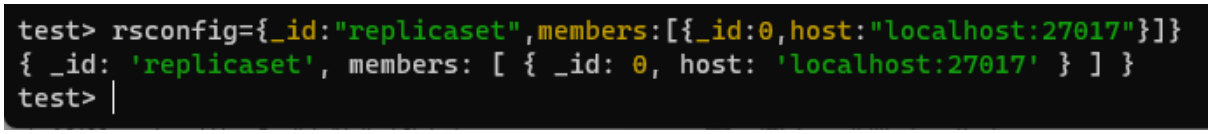


Figure 8.6: Configuring Port 27017 as Primary Server

8. Include the primary server as part of the replica set using the command:

```
rs.initiate(rsconfig)
```

Figure 8.7 shows the output of the command.

```
test> rs.initiate(rsconfig)
{ ok: 1 }
replicaset [direct: other] test>
```

Figure 8.7 Making the Primary Server as a Part of Replica Set

9. Confirm whether port 27017 has been configured as primary server using the command:

```
rs.status()
```

Figure 8.8 shows that port 27017 is assigned to the primary server.

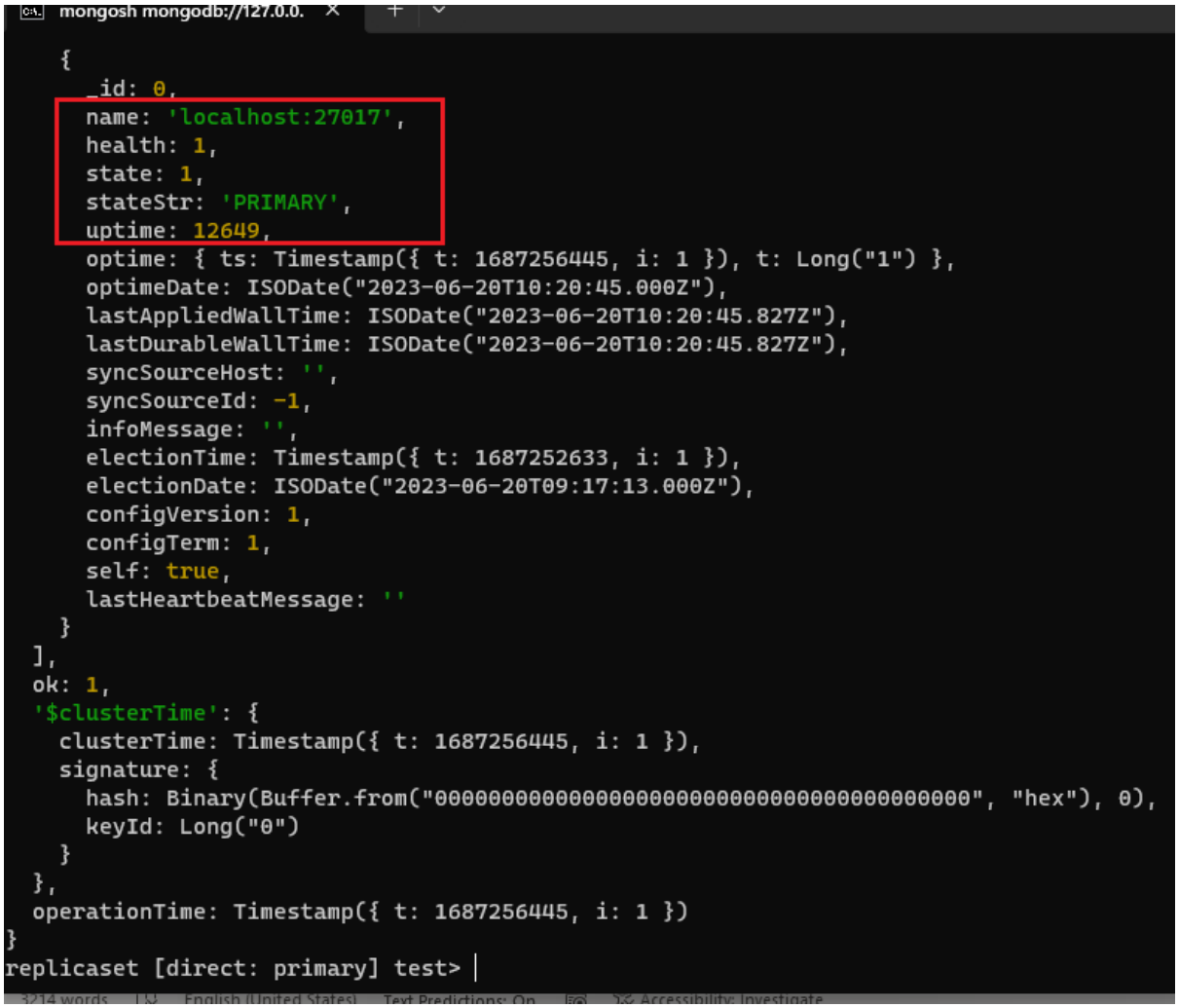


Figure 8.8: Confirming the Assignment of Port 27017 to Primary Server

Now that the primary server is configured, the next step is to configure the secondary servers.

10. Open a new command prompt.

11. Start the `mongod` instance connecting to the port 27019 using the command:

```
mongod --replSet replicaset --logpath
\data\rs2\2.log --dbpath \data\rs2 --port 27019
```

Figure 8.9 shows the output of the command.

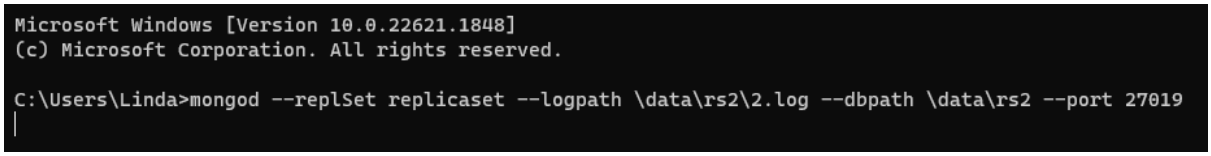


Figure 8.9: Secondary Server 1: mongoDb Server Instance on Port 27019

12. Open a new command prompt.

13. Start the `mongod` instance connecting to the port 27020 using the command:

```
mongod --replSet replicaset --logpath
\data\rs3\3.log --dbpath \data\rs3 --port 27020
```

Figure 8.10 shows the output of the command.

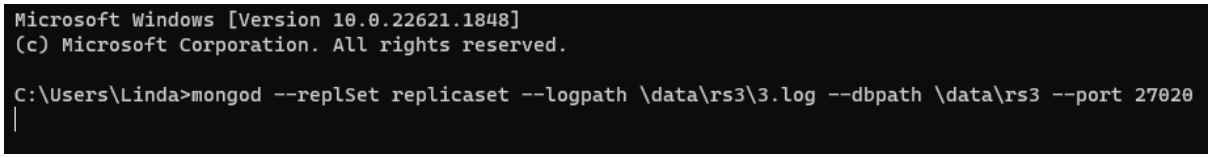


Figure 8.10: Secondary Server 2: mongoDb Server Instance on Port 27020

14. At the mongoshell prompt connected to the primary server, add the port 27019 to the primary server's replica set using the command:

```
rs.add("localhost:27019")
```

Figure 8.11 shows the output of the command.

```
replicaset [direct: primary] test> rs.add("localhost:27019")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687257996, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687257996, i: 1 })
}
replicaset [direct: primary] test>
```

Figure 8.11: Adding Port 27019 to the Primary Server Replica Set

15. Similarly, add the port 27020 to the primary server's replica set using the command:

```
rs.add("localhost:27020")
```

Figure 8.12 shows the output of the command.

```
replicaset [direct: primary] test> rs.add("localhost:27020")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687258159, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687258159, i: 1 })
}
replicaset [direct: primary] test>
```

Figure 8.12: Adding Port 27020 to the Primary Server Replica Set

16. Confirm the configurations of the three ports using the command:

```
rs.status()
```

Figures 8.13, 8.14, and 8.15 show the status of the replica set and the three ports.

```
replicaset [direct: primary] test> rs.status()
{
  set: 'replicaset',
  date: ISODate("2023-06-20T10:53:06.392Z"),
  myState: 1,
  term: Long("1"),
  syncSourceHost: '',
  syncSourceId: -1,
  heartbeatIntervalMillis: Long("2000"),
  majorityVoteCount: 2,
  writeMajorityCount: 2,
  votingMembersCount: 3,
  writableVotingMembersCount: 3,
  optimes: {
    lastCommittedOpTime: { ts: Timestamp({ t: 1687258377, i: 1 }), t: Long("1") },
    lastCommittedWallTime: ISODate("2023-06-20T10:52:57.145Z"),
    readConcernMajorityOpTime: { ts: Timestamp({ t: 1687258377, i: 1 }), t: Long("1") },
    appliedOpTime: { ts: Timestamp({ t: 1687258377, i: 1 }), t: Long("1") },
    durableOpTime: { ts: Timestamp({ t: 1687258377, i: 1 }), t: Long("1") },
    lastAppliedWallTime: ISODate("2023-06-20T10:52:57.145Z"),
    lastDurableWallTime: ISODate("2023-06-20T10:52:57.145Z")
  },
  lastStableRecoveryTimestamp: Timestamp({ t: 1687258327, i: 1 }),
  electionCandidateMetrics: {
    lastElectionReason: 'electionTimeout',
    lastElectionDate: ISODate("2023-06-20T09:17:13.020Z"),
    electionTerm: Long("1"),
    lastCommittedOpTimeAtElection: { ts: Timestamp({ t: 1687252632, i: 1 }), t: Long("-1") },
    lastSeenOpTimeAtElection: { ts: Timestamp({ t: 1687252632, i: 1 }), t: Long("-1") },
    numVotesNeeded: 1,
    priorityAtElection: 1,
    electionTimeoutMillis: Long("10000"),
    newTermStartDate: ISODate("2023-06-20T09:17:13.054Z"),
```

Figure 8.13: Status of Replica set

```
members: [
  {
    _id: 0,
    name: 'localhost:27017',
    health: 1,
    state: 1,
    stateStr: 'PRIMARY',
    uptime: 14581,
    optime: { ts: Timestamp({ t: 1687258377, i: 1 }), t: Long("1") },
    optimeDate: ISODate("2023-06-20T10:52:57.000Z"),
    lastAppliedWallTime: ISODate("2023-06-20T10:52:57.145Z"),
    lastDurableWallTime: ISODate("2023-06-20T10:52:57.145Z"),
    syncSourceHost: '',
    syncSourceId: -1,
    infoMessage: '',
    electionTime: Timestamp({ t: 1687252633, i: 1 }),
    electionDate: ISODate("2023-06-20T09:17:13.000Z"),
    configVersion: 5,
    configTerm: 1,
    self: true,
    lastHeartbeatMessage: ''
  },
```

Figure 8.14: Status of Port 27017

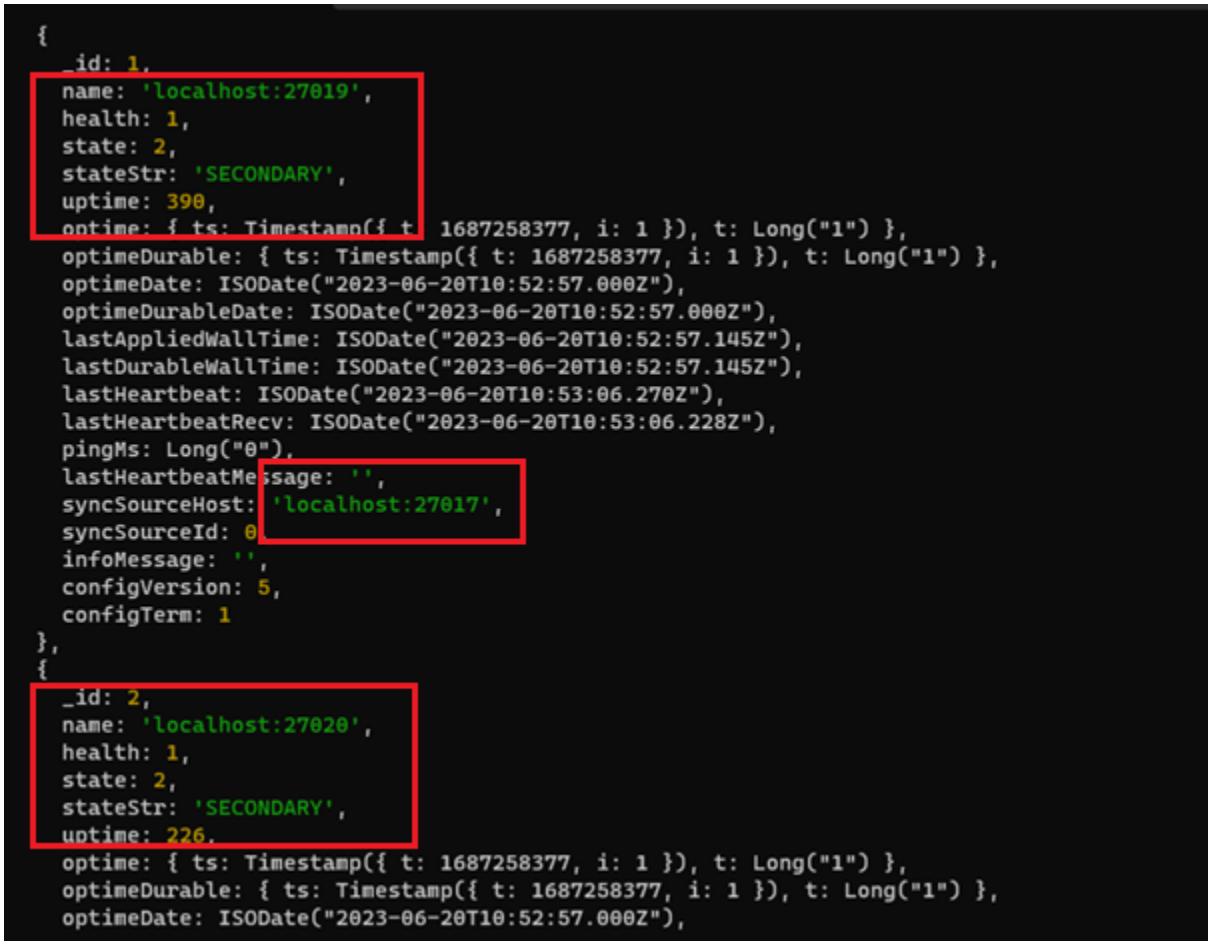


Figure 8.15: Status of Ports 27019 and 27020

Note that in Figure 8.15 the server instance on port 27017 is mentioned as the source host for the secondary server.

17. Check if the primary server is primary using the command:

```
rs.isMaster()
```

Figure 8.16 shows the output of the command. Note the phrase 'isMaster:true'.

```
replicaset [direct: primary] test> rs.isMaster()
{
  topologyVersion: {
    processId: ObjectId("64914c1f3d274b57f20991a7"),
    counter: Long("10")
  },
  hosts: [ 'localhost:27017', 'localhost:27019', 'localhost:27020' ],
  setName: 'replicaset',
  setVersion: 5,
  ismaster: true,
  secondary: false,
  primary: 'localhost:27017',
  me: 'localhost:27017',
  electionId: ObjectId("7fffffff0000000000000001"),
  lastWrite: {
    opTime: { ts: Timestamp({ t: 1687270855, i: 1 }), t: Long("1") },
    lastWriteDate: ISODate("2023-06-20T14:20:55.000Z"),
    majorityOpTime: { ts: Timestamp({ t: 1687270855, i: 1 }), t: Long("1") },
    majorityWriteDate: ISODate("2023-06-20T14:20:55.000Z")
  },
```

Figure 8.16: Confirming the Primary Node

18. Open another command prompt.

19. Use `mongosh` to communicate with the secondary server in port 27019.

```
mongosh --port 27019
```

Figure 8.17 shows the output of the command.

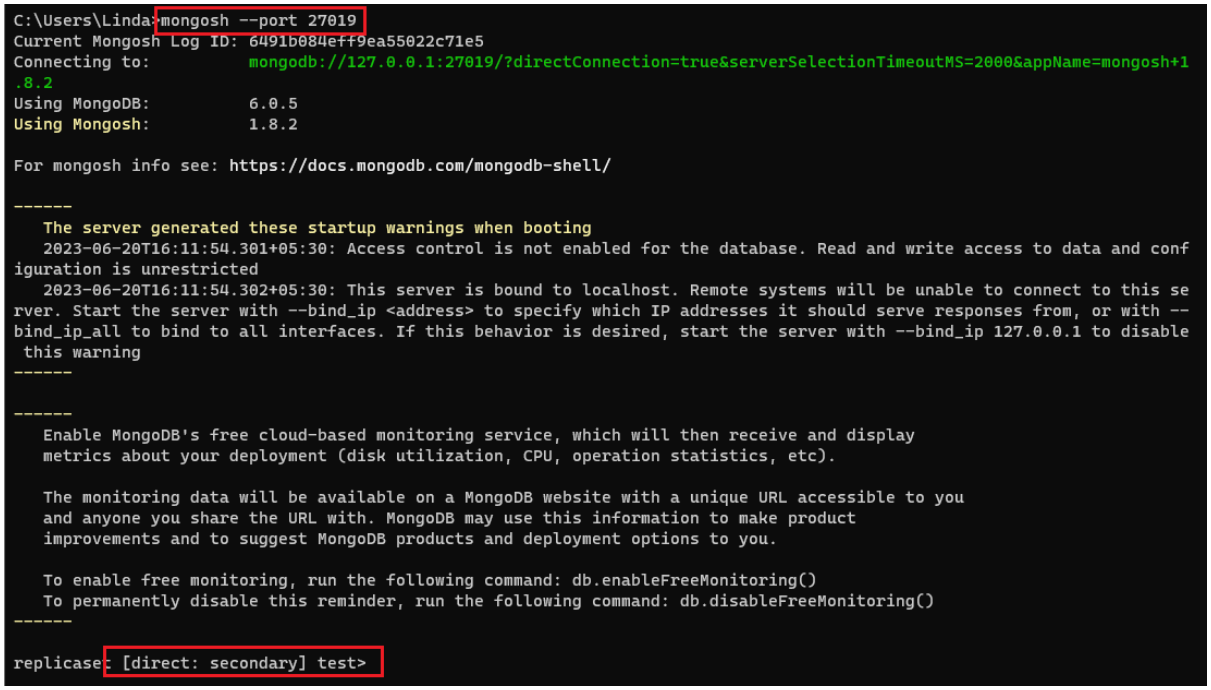


Figure 8.17: Secondary Server at Port 27019

20. Confirm that the secondary server is not primary using the command:

```
rs.isMaster()
```

Figure 8.18 shows the output of the command. Note the phrase '`isMaster:false`'.

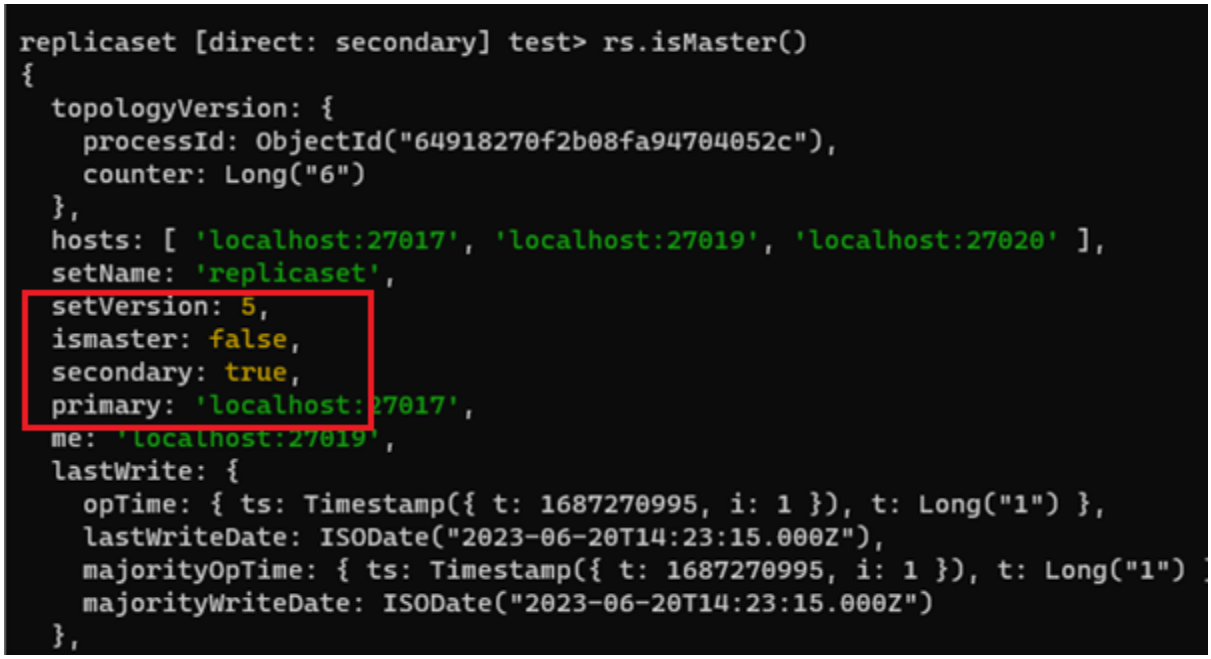


Figure 8.18: Confirming the Secondary Node

21. Open another command prompt.

22. Use `mongosh` to communicate with the secondary server in port 27020.

```
mongosh --port 27020
```

Figure 8.19 shows the output of the command.

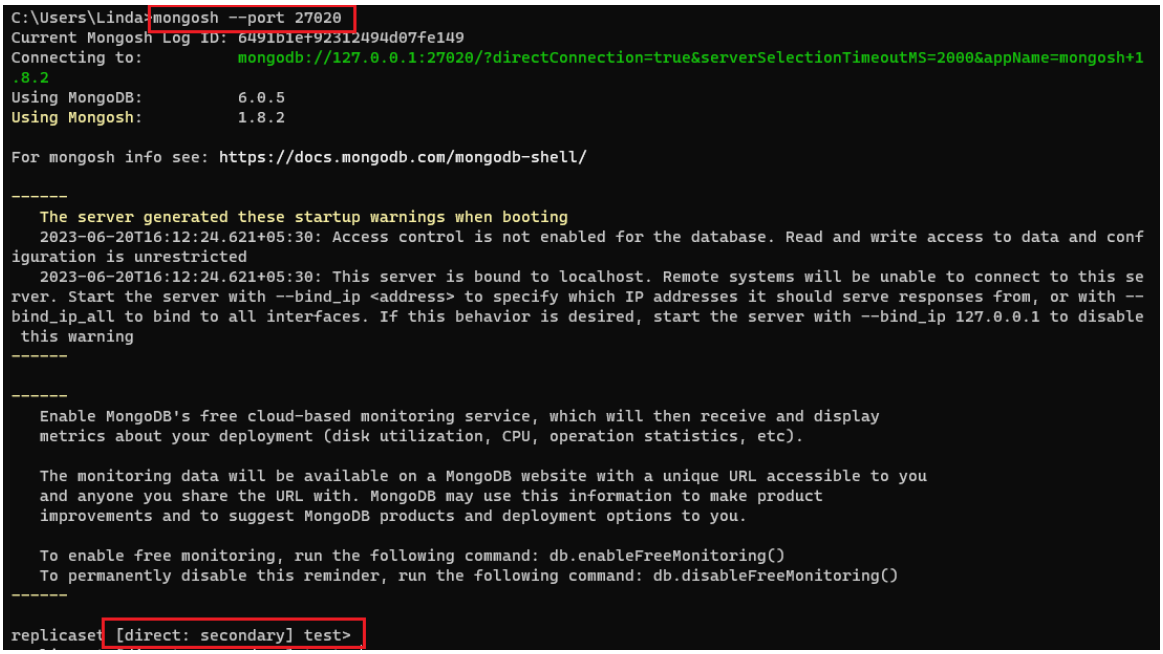


Figure 8.19: Secondary Server at Port 27020

Now that the replica set is set up, let us create a database collection and documents on the primary server and read from the secondary server.

Figure 8.20 shows the creation of database, collection, and document on the primary server.

```
replicaset [direct: primary] test> use sample_emp
switched to db sample_emp
replicaset [direct: primary] sample_emp> show collections

replicaset [direct: primary] sample_emp> db.empdetails.insertOne({Name:"David Jackson"})

{
  acknowledged: true,
  insertedId: ObjectId("6492c431ca833b9b775bc3f3")
}
replicaset [direct: primary] sample_emp> show collections
empdetails
replicaset [direct: primary] sample_emp> db.empdetails.find()
[
  { _id: ObjectId("6492c431ca833b9b775bc3f3"), Name: 'David Jackson' }
]
replicaset [direct: primary] sample_emp>
```

Figure 8.20: Creation of a Collection with a Document in Primary Server

Even though the changes are replicated to the secondary server, the user will not be able to read the details inserted in primary from the secondary. The user can verify this using the series of commands:

```
show dbs
use sample_emp
show collections
db.empdetails.find()
```

Figure 8.21 shows the output of the commands.

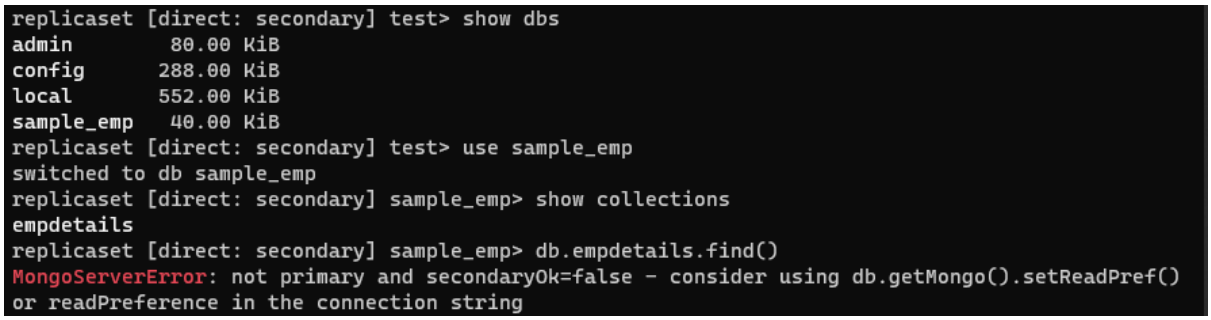


Figure 8.21: Reading a Document from Secondary Server

Note that the `secondayOk` must be set to true to allow the reading of collection contents from the secondary server. To do this, run the command:

```
rs.secondaryOk()
```

Figure 8.22 shows the output of this command.

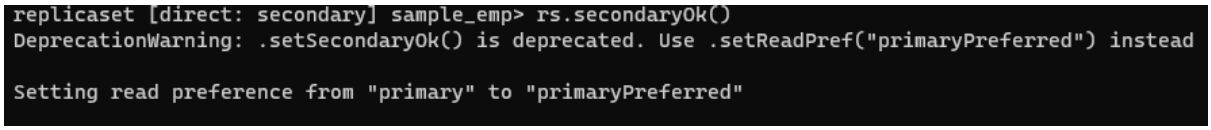


Figure 8.22: Allowing the Secondary Server to Serve Collection Queries

Figure 8.23 shows that the document inserted in the primary server can be read now from the secondary server.

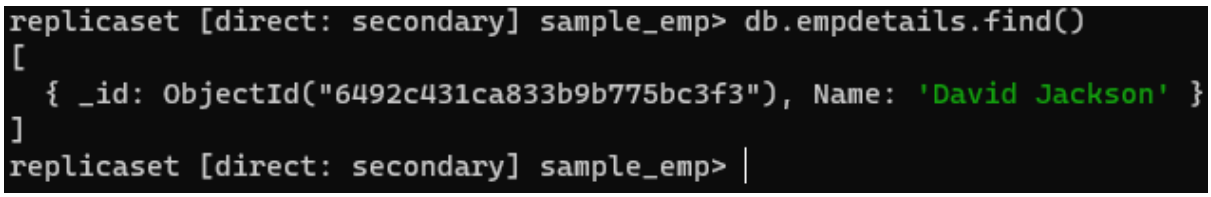


Figure 8.23: Reading Documents from the Secondary Server

Secondary servers can become primary servers as a result of an election. To force an election, the user can shut down the primary server using the command:

```
db.shutdownserver()
```

Figure 8.24 shows the result of shutting down the primary server.

```
replicaset [direct: primary] sample_emp> db.shutdownServer()
MongoNetworkError: connection 4 to 127.0.0.1:27017 closed
sample_emp>
```

Figure 8.24: Shutting Down the Primary Server

Now, one of the secondary servers automatically becomes the primary. To verify this, let us insert another document from the secondary server running on port 27019 using the command:

```
db.empdetails.insertOne({Name:"Elana Thomas"})
```

Figure 8.25 shows the output of this command and the result of running the `find` command on the `empdetails` collection.

```
replicaset [direct: secondary] sample_emp> db.empdetails.insertOne({Name:"Elana Thomas"})
{
  acknowledged: true,
  insertedId: ObjectId("6492d0b4e873a7aadecc6e22")
}
replicaset [direct: primary] sample_emp> db.empdetails.find()
[
  { _id: ObjectId("6492c431ca833b9b775bc3f3"), Name: 'David Jackson' },
  { _id: ObjectId("6492d0b4e873a7aadecc6e22"), Name: 'Elana Thomas' }
]
replicaset [direct: primary] sample_emp>
```

Figure 8.25: Secondary Server Elected as a Primary Server

Note that the server instance that was secondary has changed to primary and the document has been successfully inserted. Thus, the secondary server has automatically upgraded to be a primary server and has secured the write permissions as the original primary server is unavailable.

To return the elected primary server to its original state as secondary server, the user can use the `stepDown` command. However, the primary server that was shut down must be started using the command in a new command prompt:

```
mongod --replSet replicaset --logpath
\data\rs1\1.log --dbpath \data\rs1 --port 27017
```

In another new command prompt, the user can start the instance using the command:

```
mongosh --port 27017
```

Now, in the server instance of port 27019, the user can run the `stepDown` command as:

```
rs.stepDown()
```

Figure 8.26 shows the output of this command.

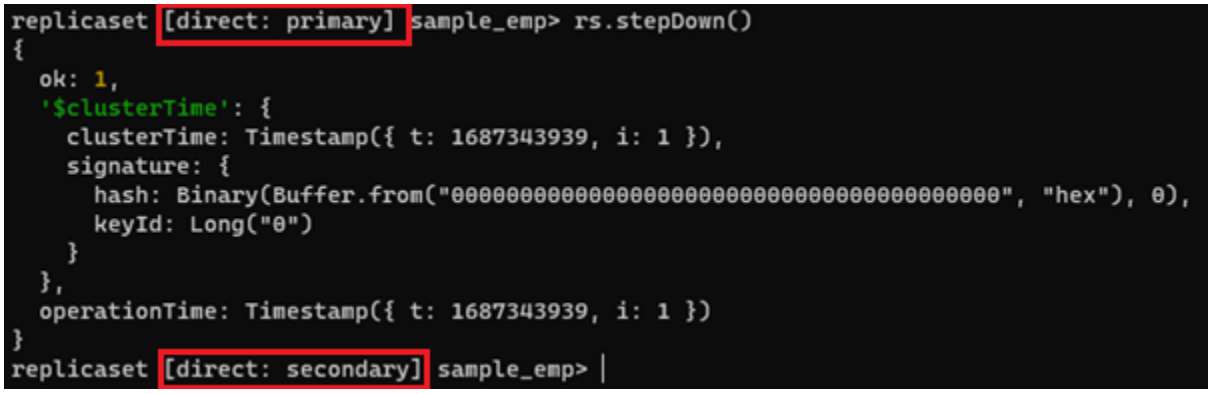


Figure 8.26: Secondary Server Steps Down from Primary Server Status

8.2 Sharding

Replication addresses the high availability of data servers by maintaining copies of the databases and ensuring that a secondary server takes over as a primary server when required. However, fast-growing large data sets may soon become challenging to handle for the primary server's CPU. The CPU, RAM, and Input/Output devices may be exhausted because of this. There are two ways of distributing data:

1. **Vertical scaling** involves increasing the RAM capacity, using a high-capacity processor, and increasing the storage space. Limitations in technology may prevent this set up from working efficiently for the given workload. Cloud-based providers have a vertical ceiling which prevents efficient processing of queries.

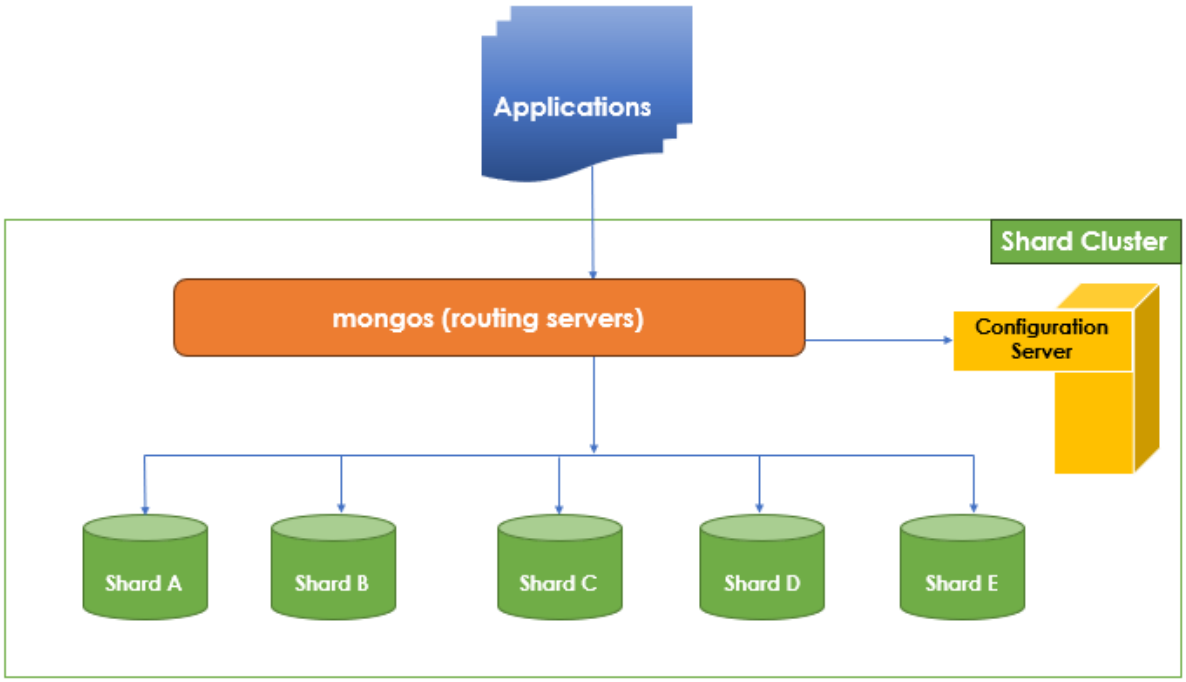
2. **Horizontal scaling** involves dividing the data across multiple servers. Each server is a high-end machine holding a subset of the large data set. Hence, the workload is divided and handled efficiently. In addition, the overall cost

is lower than investing on a single server. Sharding is the horizontal distribution of data across multiple servers.

When the load on a server increases, MongoDB supports distribution of data across different servers and thus supports horizontal scaling through sharding. In MongoDB, sharding is done at the collection level by distributing collections of data across multiple servers.

8.2.1 Shard Clusters

A shard cluster in MongoDB contains three components:



- **Shard Servers:** Data is distributed across the shard servers. Each shard server has a subset of sharded data and must be deployed as a replica set.
- **mongos:** The MongoDB instances `mongos` act as query routers. They provide an interface between the client application and shard cluster.
- **Configuration Servers:** Configuration information and metadata are stored in config servers. Each configuration server must also be deployed as a replica set.

8.2.2 Sharding Strategies

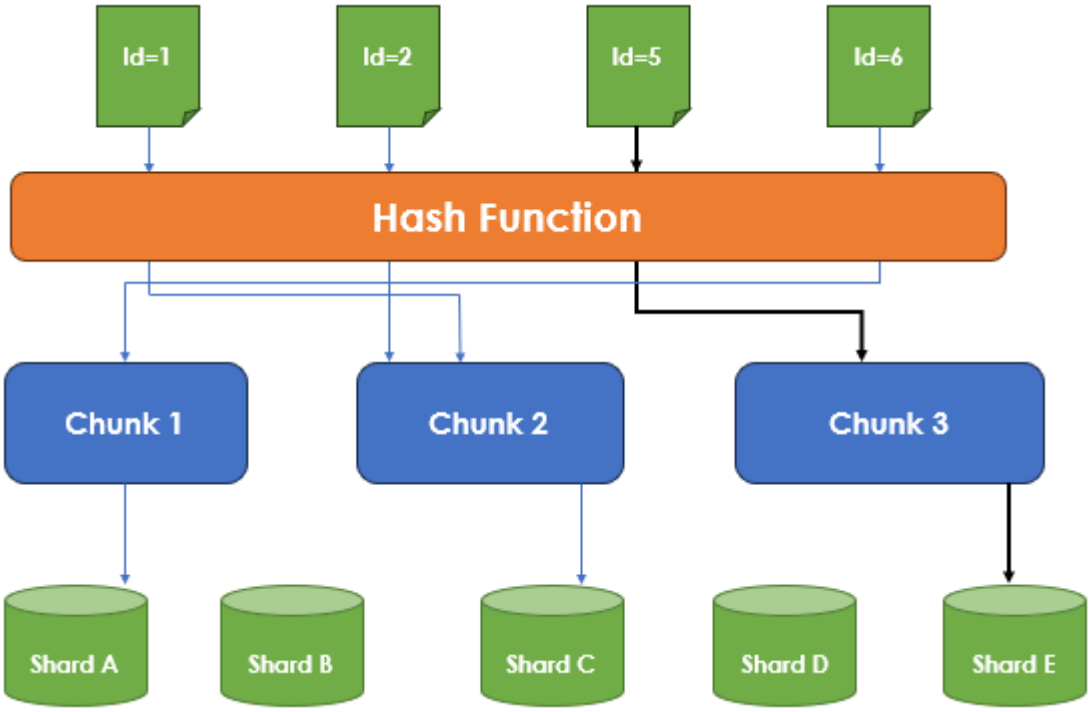
To distribute data across different shards, MongoDB uses shard keys. These shard keys are either single or compound index keys from the document which determine the distribution of data amongst the shard cluster. MongoDB divides the data in shards evenly into chunks. Each chunk has a span of index keys that are assigned to a range of shard key values.

The mapping of index keys to shard keys can be based on any of the two types of sharding strategies supported by MongoDB:

- **Hash-based Sharding**

  Hashing refers to the use of a mathematical function that uses the shard key of the document to compute hash values. Each chunk is then assigned a range of shard key fields based on the hash values. A chunk has a default size of 64 MB. When a user queries for data, MongoDB automatically computes the hash value and locates the chunk that contains the required data.

  Consider that the user has `Id` as the shard key for a collection.

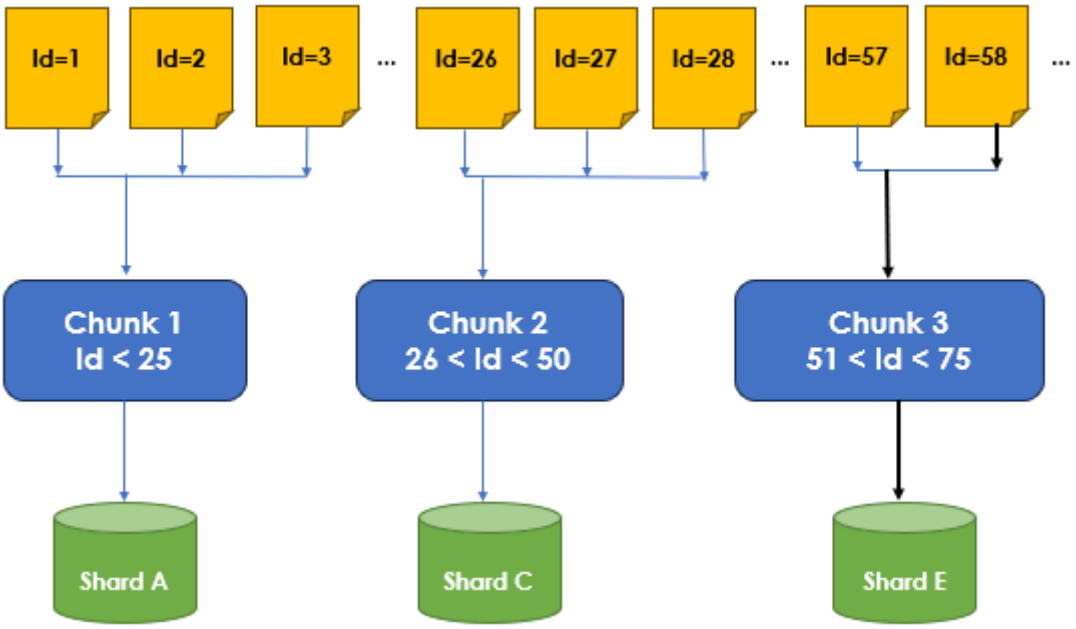


  If the user wants to locate a document with `Id` as `5`, then MongoDB uses the hash function to locate the chunk in which the document is stored. Here it is `Chunk 3` which is in `Shard E`.

  Note that documents with nearer shard key values may not be in the same shard or even in the adjacent shards. For example, document with `Id` as `5` is in `Shard E` while the document with `Id` as `6` is in `Shard A`.

- **Range-based Sharding**

  In range-based sharding, indexes are arranged in order and continuous ranges are formed. Each index value falls under a range and the range is assigned to shards.



Documents with nearer shard values are likely to be in the same shard.

8.2.3 Sharding Methods

Table 8.2 lists the important methods employed to implement the sharding process. Here, `sh` denotes sharding.

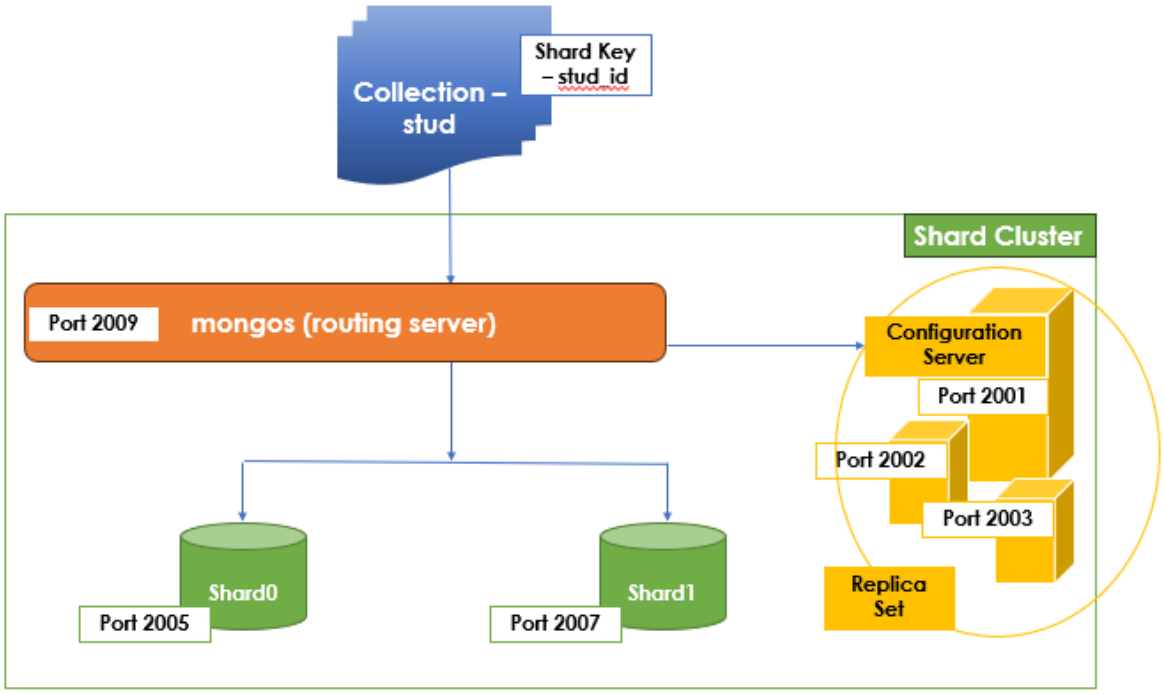
Name	Description
`sh.addShard()`	Adds a shard to the cluster
`sh.status()`	Reports on the status of a sharded cluster similar to `db.printShardingStatus()`
`sh.enableSharding()`	Creates a database
`sh.shardCollection()`	Enables sharding for a collection
`sh.moveChunk()`	Migrates a chunk to a cluster

Table 8.2: Sharding Methods

8.2.4 Implementing Sharding

Let us now see how to implement sharding with the help of an example. Consider that the user wants to store student details in the `stud` collection of the `studentdb` database. The data will be distributed across two shard servers `Shard0` and `Shard1` which will be configured to listen from the ports `2005` and `2007`, respectively. A routing server enabled by `mongos` will be configured to listen from the port `2009`. The configuration server replica set, primary `cs1`, and replicas, `cs2`, and `cs3`, will be configured to listen from ports `2001`, `2002`, and `2003`, respectively.

Hundred documents which consist of fields `stud_id` and `class` will be routed through the query router from mongo shell. The query will distribute the documents across the two shard servers.



Building the Configuration Server Replica Set

1. Create three directories namely `cs1`, `cs2`, and `cs3` to hold the data for configuration replica sets in the, `C:\data` folder.

2. Open a new command prompt and create a `mongod` instance to act as a primary configuration server using the command:

```
mongod --configsvr --replSet rshard --logpath
\data\cs1\1.log --dbpath \data\cs1 --port 2001
```

   Note that the replica set name is `rshard`.

3. Open a new command prompt and start communication with the configuration server, `cs1` through port `2001` using the command:

```
mongosh --port 2001
```

4. Configure cs1 as the primary shard configuration server which will be part of a replica set using the commands:

```
shardconfig={_id:"rshard",members:[{_id:0,host:
"localhost:2001"}]}

rs.initiate(shardconfig)
```

Figure 8.27 shows the output of these commands.

```
test> shardconfig={_id:"rshard",members:[{_id:0,host:"localhost:2001"}]}
{ _id: 'rshard', members: [ { _id: 0, host: 'localhost:2001' } ] }
test> rs.initiate(shardconfig)
{ ok: 1, lastCommittedOpTime: Timestamp({ t: 1687406954, i: 1 }) }
rshard [direct: other] test>
```

Figure 8.27: Configuring the Primary Configuration Server

5. Configure the secondary configuration servers in the replica set using the commands in two new command windows:

```
mongod --configsvr --replSet rshard --logpath
\data\cs2\2.log --dbpath \data\cs2 --port 2002

mongod --configsvr --replSet rshard --logpath
\data\cs3\3.log --dbpath \data\cs3 --port 2003
```

6. From the primary configuration server instance, add the secondary servers to the replica set using the commands:

```
rs.add("localhost:2002")
rs.add("localhost:2003")
```

7. Check the status of the configuration servers using the command:

```
rs.status()
```

Figure 8.28 shows the status of the configuration servers.

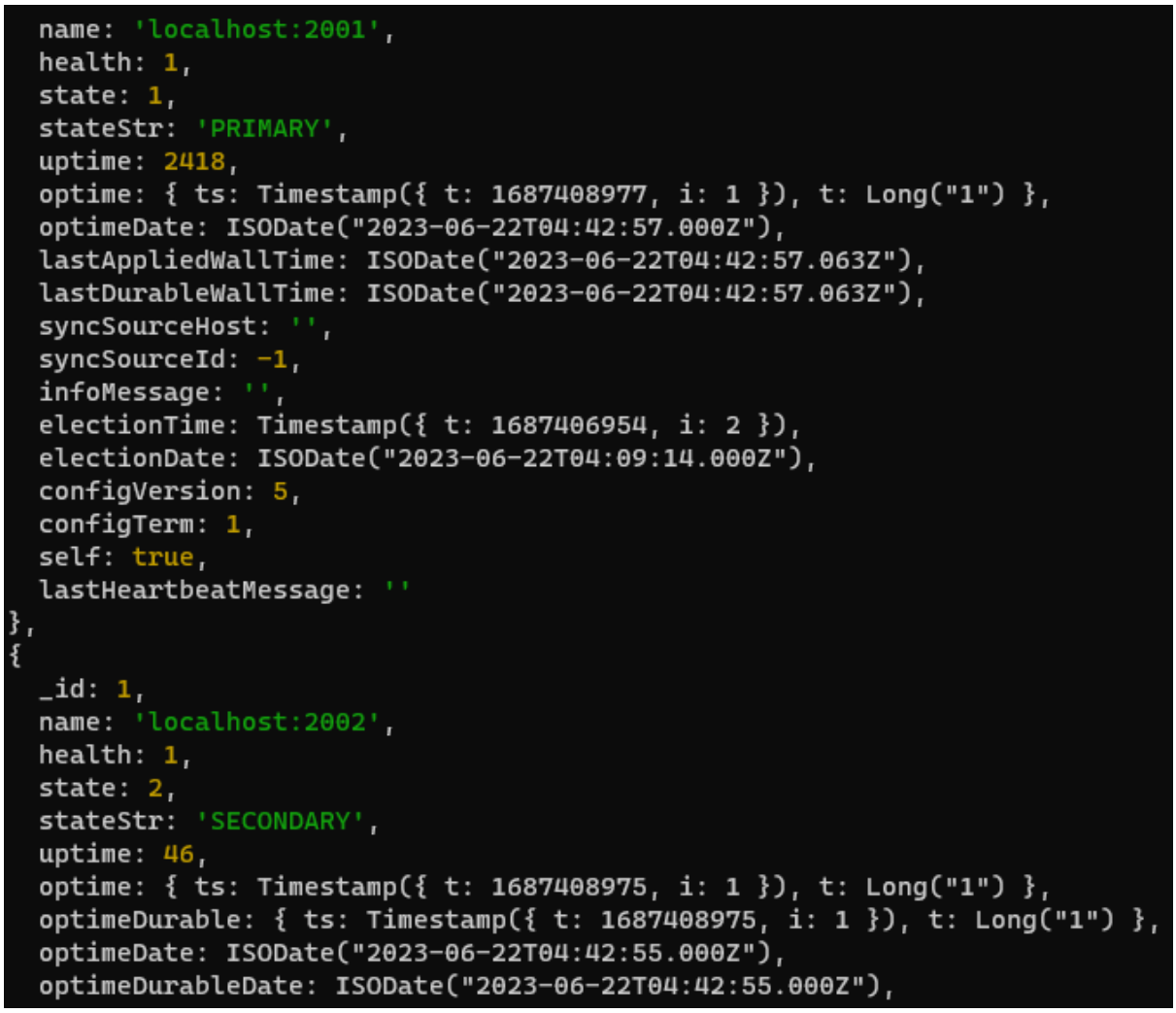


Figure 8.28: Status of the Configuration Server Replica Sets

Creating the Shard Servers Shard0 and Shard1

8. Create two new folders `Shard0` and `Shard1` in the path, `C:\data` to hold data for sharding.

9. Create two log folders one each inside the folders `Shard0` and `Shard1`.

10. Create a shard server `Shard0` as replica set to listen to port 2005. To do this, open a new command prompt and execute the command:

```
mongod --shardsvr --replSet shardrep --logpath
\data\Shard0\log\Shard0.log --dbpath
\data\Shard0 --port 2005
```

Note that the replica set name for `Shard0` is `shardrep`.

11. Open a new command prompt and start communication with the shard server, `Shard0` through port `2005` using the command:

```
mongosh --port 2005
```

12. Configure `Shard0` as the primary shard server which will be part of a replica set using the commands:

```
sconfig={_id:"shardrep",members:[{_id:0,host:
"localhost:2005"}]}

rs.initiate(sconfig)
```

Figure 8.29 shows the output of the commands.

```
test> sconfig={_id:"shardrep",members:[{_id:0,host:"localhost:2005"}]}
{ _id: 'shardrep', members: [ { _id: 0, host: 'localhost:2005' } ] }
test> rs.initiate(sconfig)
{ ok: 1 }
shardrep [direct: other] test>
```

Figure 8.29: `Shard0` Server Configured to Listen from Port `2005`

13. Similarly, create a shard server `Shard1` as replica set to listen to port 2007 using the commands.

```
mongod --shardsvr --replSet shard0rep --
logpath \data\Shard1\log\Shard1.log --dbpath
\data\Shard1 --port 2007

mongosh --port 2007

s2config={_id:"shard0rep",members:[{_id:0,hos
t:"localhost:2007"}]}

rs.initiate(s2config)
```

14. Open a new command prompt and start the routing server to listen to port 2009 using the primary config server using the command:

```
mongos --port 2009 --configdb rshard/localhost:2001
```

Figure 8.30 shows the output of this command.

Figure 8.30: Configuring the Routing Server

15. Open a new command prompt and establish communication with the routing server using the command:

```
mongosh --port 2009
```

16. Add the shard servers to the routing server using the commands:

```
sh.addShard("shardrep/localhost:2005")

sh.addShard("shard0rep/localhost:2007")
```

Figure 8.31 shows the output of this commands.

```
[direct: mongos] test> sh.addShard("shardrep/localhost:2005")
{
  shardAdded: 'shardrep',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687444154, i: 6 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687444154, i: 6 })
}
[direct: mongos] test> sh.addShard("shardrep/localhost:2007")
{
  shardAdded: 'shardrep',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687444619, i: 1 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687444619, i: 1 })
}
[direct: mongos] test>
```

Figure 8.31: Adding the Shards to the Router

17. Enable sharding for the `studentdb` database using the command:

```
sh.enableSharding("studentdb")
```

Figure 8.32 shows the output of the command.

```
[direct: mongos] test> sh.enableSharding("studentdb")
{
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687452540, i: 2 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687452540, i: 2 })
}
[direct: mongos] test>
```

Figure 8.32: Enable Sharding for the `studentdb` Database

18. Enable sharding for the collection `stud` in the database `studentdb` using the command:

```
sh.shardCollection("studentdb.stud",{"stud_id":"hashed"})
```

Figure 8.33 shows the output of the command.

```
[direct: mongos] test> sh.shardCollection("studentdb.stud",{"stud_id":"hashed"})
{
  collectionsharded: 'studentdb.stud',
  ok: 1,
  '$clusterTime': {
    clusterTime: Timestamp({ t: 1687452150, i: 34 }),
    signature: {
      hash: Binary(Buffer.from("0000000000000000000000000000000000000000", "hex"), 0),
      keyId: Long("0")
    }
  },
  operationTime: Timestamp({ t: 1687452150, i: 30 })
}
[direct: mongos] test>
```

Figure 8.33: Sharding the `stud` Collection with `stud_id` as Hashed Index

19. Insert records into the collection using the command:

```
for (i = 1; i <= 100; i++) { db.stud.insertMany([{
stud_id: i, class: "Grade-5" }]);}
```

Figure 8.34 shows the output of the command.

```
[direct: mongos] test> for (i = 1; i <= 100; i++) { db.stud.insertMany([{ stud_id: i, class: "Grade-5" }]);}
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64947c3c2572d3e93d716f17") }
}
[direct: mongos] test>
```

Figure 8.34: Inserting Records into the Collection

20. Check if the documents are sharded using the command:

```
db.stud.getShardDistribution()
```

Figure 8.35 shows the output of the command.

```
[direct: mongos] studentdb> db.stud.getShardDistribution()
Shard shardrep at shardrep/localhost:2005
{
  data: '2KiB',
  docs: 56,
  chunks: 2,
  'estimated data per chunk': '1KiB',
  'estimated docs per chunk': 28
}
---
Shard shard0rep at shard0rep/localhost:2007
{
  data: '2KiB',
  docs: 44,
  chunks: 2,
  'estimated data per chunk': '1KiB',
  'estimated docs per chunk': 22
}
---
Totals
{
  data: '5KiB',
  docs: 100,
  chunks: 4,
  'Shard shardrep': [ '56 % data', '56 % docs in cluster', '54B avg obj size on shard' ],
  'Shard shard0rep': [ '44 % data', '44 % docs in cluster', '54B avg obj size on shard' ]
}
[direct: mongos] studentdb>
```

Figure 8.35: Sharded Data

Note that a total of 100 documents were inserted using a loop from the router server listening to port 2009. The Shard0 server listening to port 2005 has two chunks and a total of 56 documents and each chunk has 28 documents. The Shard1 server listening to port 2007 has 44 documents and two chunks with each chunk containing 22 documents.

8.3 Summary

- Replication is making the data available in more than one place which enables easy querying and high data availability.
- PSS and PSA are some of the ways to implement replication.
- If the primary server fails, the secondaries hold an election, and the elected member takes the place of the primary server.
- Sharding is distributing data between shard servers and making them available for the queries from the application.
- The query router helps in routing the data amongst the shards.
- Maintaining the shard servers as replica sets helps in ensuring high availability of data.

Test Your Knowledge

1. Which of the following statements are true about the three-member (P-S-S) replica set in MongoDB?

    a. Only the primary server has all write operations.
    b. Data from the primary servers are replicated to the secondary servers.
    c. Any write or update operations on the primary server are recorded in the secondary server's log file.
    d. If the primary server is down, then randomly any one of the secondary servers becomes the primary server.

2. Which of the following events will trigger an election among the replica sets in MongoDB?

    a. When a secondary server tries to read the oplog file of the primary server
    b. When a new server is added to the replica set
    c. When the primary server fails
    d. When the secondary servers lose connectivity with the primary server beyond the configured timeout

3. Consider that you have started the `mongod` instance for the primary server with the replica set name as `primereplica` which is listening to port number 27018. Which of the following option is used to configure this primary server?

    a. `rsconfig={_id:"primereplica",primary:[{_id:0,port:"27018"}]}`
    b. `rsconfig={_id:"primereplica",members:[{_id:0,port:"27018"}]}`
    c. `rsconfig={_id:"primereplica",members:[{_id:0,host:"port:27018"}]}`
    d. `rsconfig={_id:"primereplica",members:[{_id:0,host:"localhost:27018"}]}`

4. Which of the following statement is not true about Sharding in MongoDB?

    a. It is a method of distributing a single dataset across multiple servers.
    b. It implements horizontal scaling of data.
    c. It implements vertical scaling of data.
    d. `mongos` provides an interface between the client application and the shard cluster.

5. Which of the following method will print the data distribution statistics for a sharded collection?

   a. `db.collection.getStatisticsDistribution()`
   b. `db.collection.getShardDistribution()`
   c. `db.collection.shardDistribution()`
   d. `db.collection.statisticsDistribution()`

Answers to Test Your Knowledge

1	a, b
2	b, c, d
3	d
4	c
5	b

Try it Yourself

1. Implement a three-member replica set with the P-S-S architecture by performing the following tasks:
    a. Start the `mongod` instance for the primary server with port number `27010` and replica set name as `studreplica`.
    b. Start the `mongod` instance for the two secondary servers with port numbers `27011`, and `27012`. Name the replica set as `studreplica`.
    c. Configure the primary server and add the secondary servers to the replica set.
    d. In the primary server, create a database with the name `Student_detail` that contains a collection named `Stud_mark`.
    e. Insert the following four documents into the `Stud_mark` collection.

```
[
    { name: "Adam",
      gender:"M",
      subjects:["Java","C","Python"],
      marks:[89,78,90],
      average:85.6
    },
    { name: "Franklin",
      gender:"M",
      subjects:["C","VB","Python"],
      marks:[78,85,89],
      average:84
  },
    { name: "Michael",
      gender:"M",
      subjects:["Java", "PHP"],
      marks:[88,89],
      average:88.5
    },
    { name: "Amelia",
      gender:"F",
      subjects:["Ruby","C++"],
      marks:[86,87],
      average: 86.5
    }
]
```

f. Check whether the `Student_detail` database is replicated in the secondary servers which are listening to ports `27011` and `27012`.
g. Force the primary server to shut down and check whether one of the secondary servers becomes the primary server.
h. Restore the role of the primary server to the server to listen to port `27010`.

2. Create a sharded cluster with the following components:
   a. Shard: Create two shard servers `shard1` and `shard2` where each shard server should be deployed as a replica set. Name the replica set name of the shard server as `empshardreplica`. The `shard1` and `shard2` servers will listen to ports `27013` and `27014`, respectively.
   b. Config servers: Create a configuration server with P-S-S architecture and name the replica set as `empconreplica`. The primary node of the config server group listens to port `27006` and the two secondary servers listen to ports `27007` and `27009`.
   c. `mongos`: Create a query routing server to listen to port `27005`.

3. Implement the sharding concept by performing the following tasks:
   a. Start the `mongos` routing server which listens to port `27010` using the configuration server `27006`.
   b. Add the shard servers `shard1` and `shar2` to the routing server.
   c. Enable sharding for the `emp` database and `emp_detail` collection. The `emp_detail` has two fields, `emp_id` and `designation` where `emp_id` is used as the hashed key.
   d. Insert 200 documents into the `emp_detail` collection using for loop. The `emp_id` value ranges from `1` to `200` and assign the `designation` as `software_engineer` for all the documents.
   e. Check for the shard distribution of data in the shard servers `shard1` and `shard2`.