4 Functionalities

Objectives

After completing this lesson, you should be able to do the following:

- Limiting the Columns retrieved
- Limit the rows that are retrieved
- Sort the rows that are retrieved by a query
- Indexing
- Replication and Sharding

MongoDB Projections

Projections

- In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document.
- If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

The find() Method

- MongoDB's **find()** method accepts second optional parameter that is list of fields that you want to retrieve.
- In MongoDB, when you execute **find()** method, then it displays all fields of a document.
- To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

> **NOTE**
>
> Unless the `_id` field is explicitly excluded in the projection document `_id: 0`, the `_id` field is returned.

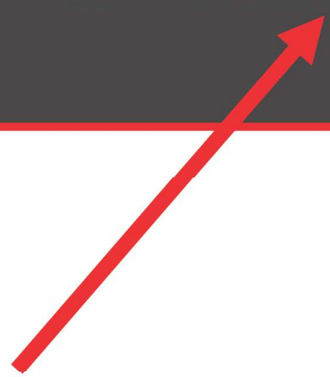
- The projection parameter determines which fields are returned in the matching documents. The projection parameter takes a document of the following form:
- Syntax

```
{ <field1>: <value>, <field2>: <value> ... }
```

- Example

```
db.bios.find( { }, { name: 1, contribs: 1 } )
```

```
> use EmployeeDB
switched to db EmployeeDB
> db.Employee.find({},{"Employeeid":1}).forEach(printjson);
{ "_id" : ObjectId("613267ff0bc1fda7d576ed0c"), "Employeeid" : 1 }
{ "_id" : ObjectId("6136c95f6bf57aa14000d299"), "Employeeid" : 1 }
{ "_id" : ObjectId("6136c95f6bf57aa14000d29a"), "Employeeid" : 2 }
{ "_id" : ObjectId("6136c95f6bf57aa14000d29b"), "Employeeid" : 3 }
{ "_id" : 1001 }
>
```



Fetched only EmployeeId Attribute

MongoDB Limit() and sort()

MongoDB order with Sort() & Limit() Query

What is Query Modifications?

- Mongo DB provides query modifiers such as the ‘limit’ and ‘Orders’ clause to provide more flexibility when executing queries.

MongoDB Limit Query Results

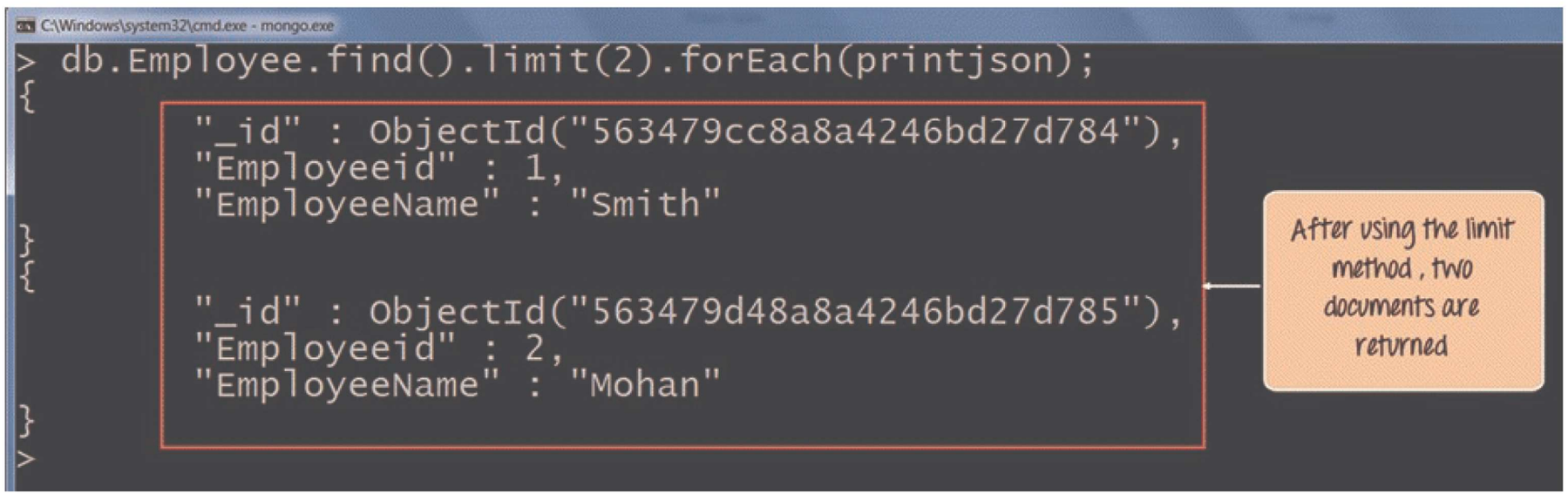
- This modifier is used to limit the number of documents which are returned in the result set for a query.

Syntax

```
db.COLLECTION_NAME.find().limit(NUMBER)
```

- It takes the find function which returns all of the documents in the collection but then uses the limit clause to limit the number of documents being returned to just 2.

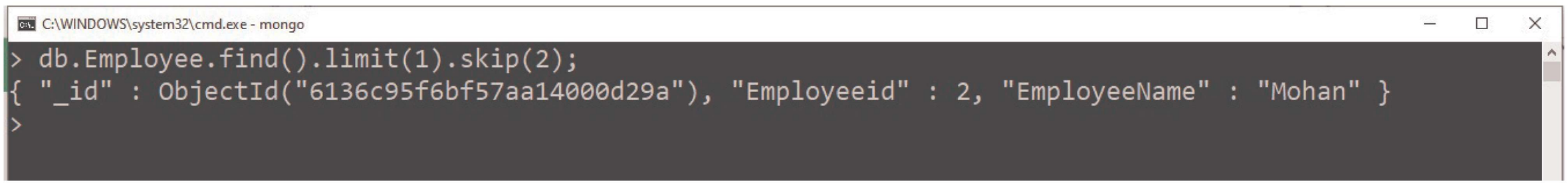
```
db.Employee.find().limit(2).forEach(printjson);
```



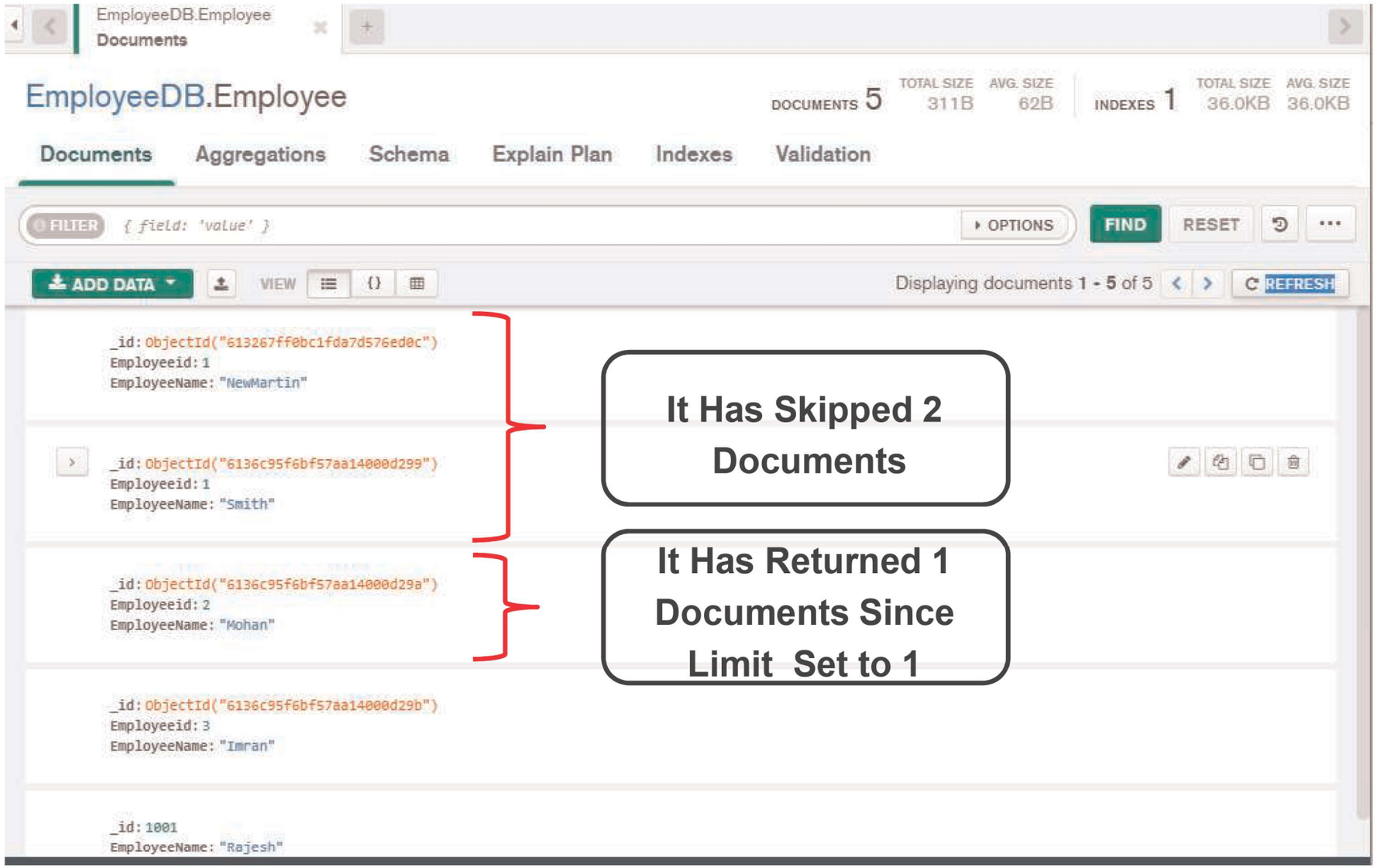
MongoDB skip() method

- In MongoDB, skip() method is used to skip the document. It is used with find() and limit() methods.
- **Syntax**

```
db.COLLECTION_NAME.find().limit(NUMBER).skip(NUMBER)
```



When Compared to



The sort() Method

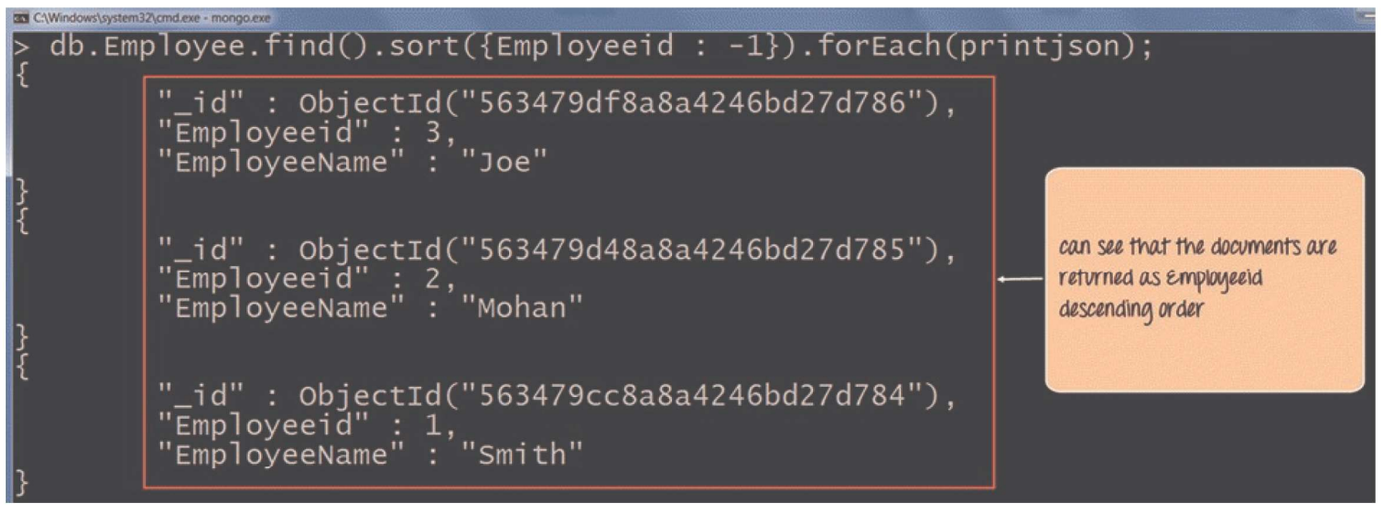
- To sort documents in MongoDB, you need to use **sort()** method.
- The method accepts a document containing a list of fields along with their sorting order.
- To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

- **Syntax**

```
db.COLLECTION_NAME.find().sort({KEY:1})
```

- It takes the sort function which returns all of the documents in the collection but then uses the modifier to change the order in which the records are returned.
- Here the -1 indicates that we want to return the documents based on the descending order of Employee id.

```
db.Employee.find().sort({Employeeid:-1}).forEach(printjson)
```



MongoDB Indexing

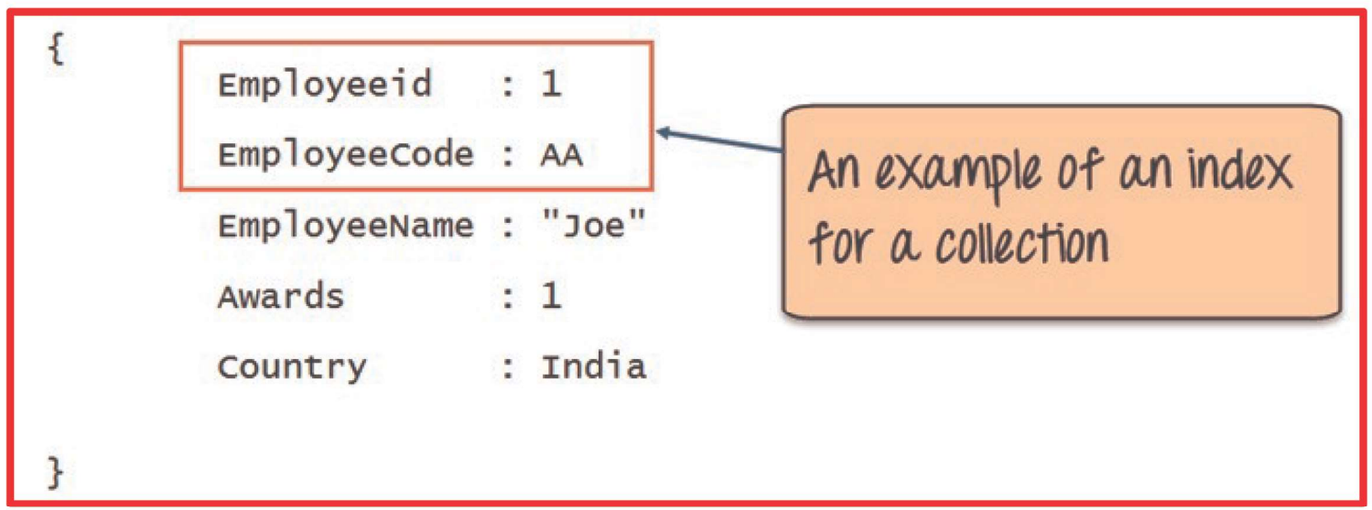
Introduction

- Indexes are very important in any database, and with MongoDB it's no different. With the use of Indexes, performing queries in MongoDB becomes more efficient.
- If you had a collection with thousands of documents with no indexes, and then you query to find certain documents, then in such case MongoDB would need to scan the entire collection to find the documents. But if you had indexes, MongoDB would use these indexes to limit the number of documents that had to be searched in the collection.
- Indexes are special data sets which store a partial part of the collection's data. Since the data is partial, it becomes easier to read this data. This partial set stores the value of a specific field or a set of fields ordered by the value of the field.

Understanding Impact of Indexes

- Now even though from the introduction we have seen that indexes are good for queries, but having too many indexes can slow down other operations such as the Insert, Delete and Update operation.
- If there are frequent insert, delete and update operations carried out on documents, then the indexes would need to change that often, which would just be an overhead for the collection.
- An index can either be based on just one field in the collection, or it can be based on multiple fields in the collection.

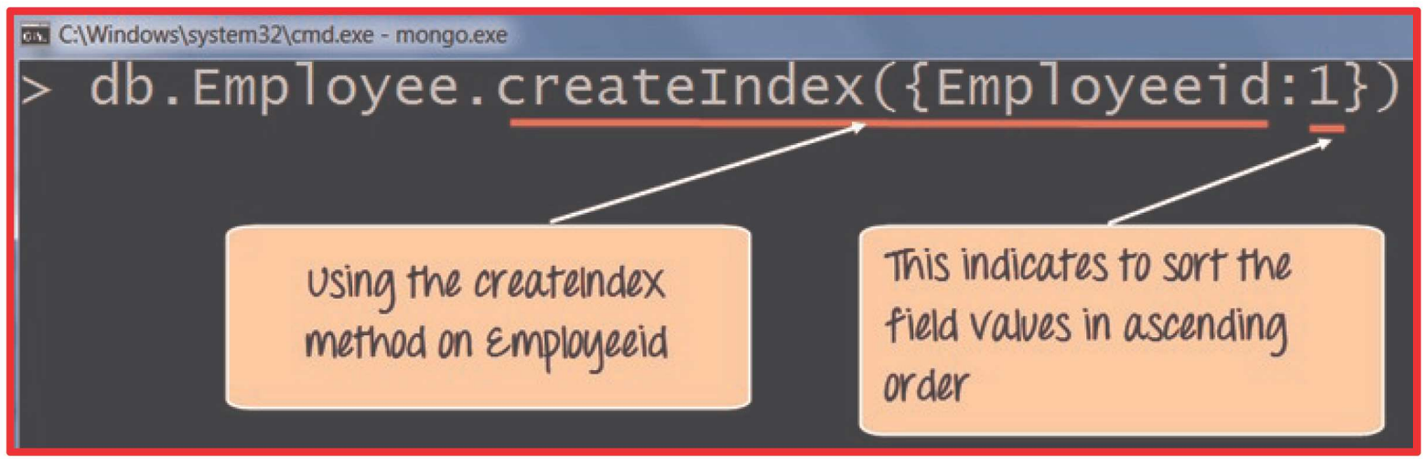
- In the sample, the Employeeid “1” and EmployeeCode “AA” are used to index the documents in the collection. So when a query search is made, these indexes will be used to quickly and efficiently find the required documents in the collection.
- So even if the search query is based on the EmployeeCode “AA”, that document would be returned.



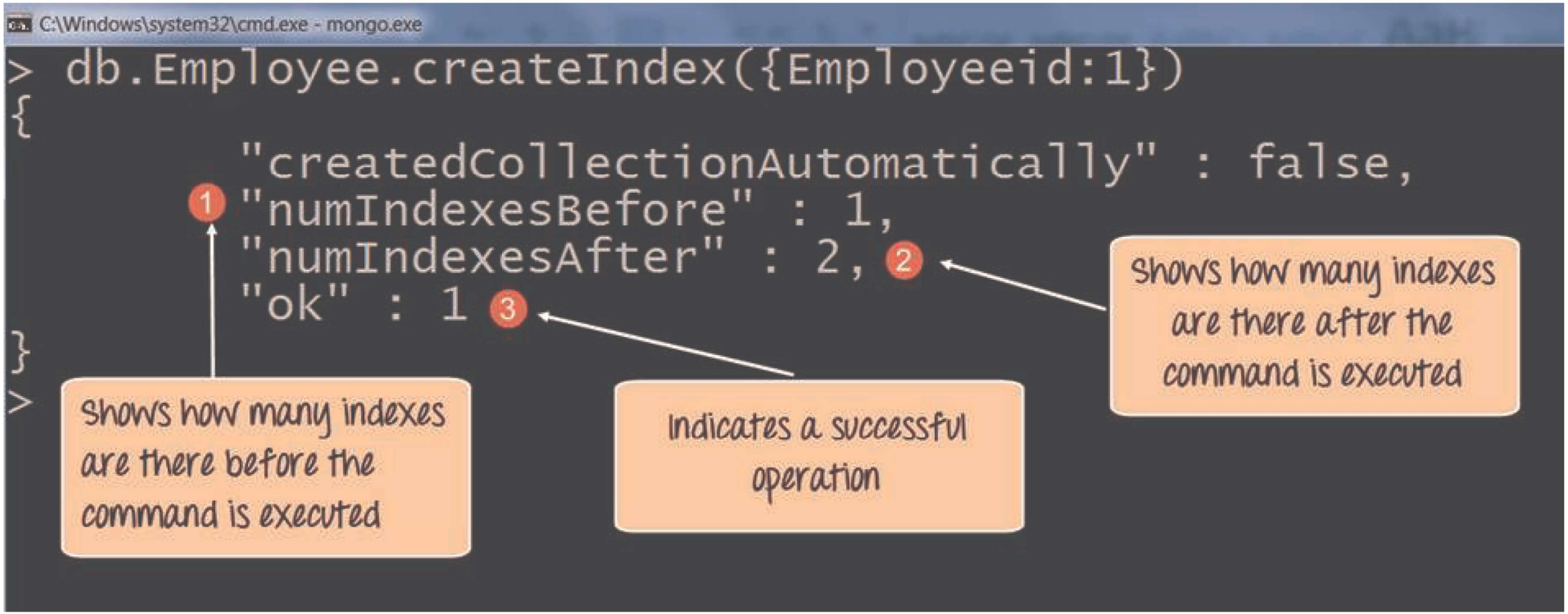
How to Create Indexes: createIndex()

- Creating an Index in MongoDB is done by using the "**createIndex**" method.
- The following example shows how add index to collection. Let's assume that we have our same Employee collection which has the Field names of "Employeeid" and "EmployeeName".

```
db.Employee.createIndex({Employeeid:1})
```



- The **createIndex** method is used to create an index based on the “Employeeid” of the document.
- The ‘1’ parameter indicates that when the index is created with the “Employeeid” Field values, they should be sorted in ascending order. Please note that this is different from the _id field (The id field is used to uniquely identify each document in the collection) which is created automatically in the collection by MongoDB. The documents will now be sorted as per the Employeeid and not the _id field.


C:\Windows\system32\cmd.exe - mongo.exe
> db.Employee.createIndex({Employeeid:1})
{
"createdCollectionAutomatically" : false,
1 "numIndexesBefore" : 1,
"numIndexesAfter" : 2, 2
"ok" : 1 3
}
Shows how many indexes are there before the command is executed
Indicates a successful operation
Shows how many indexes are there after the command is executed

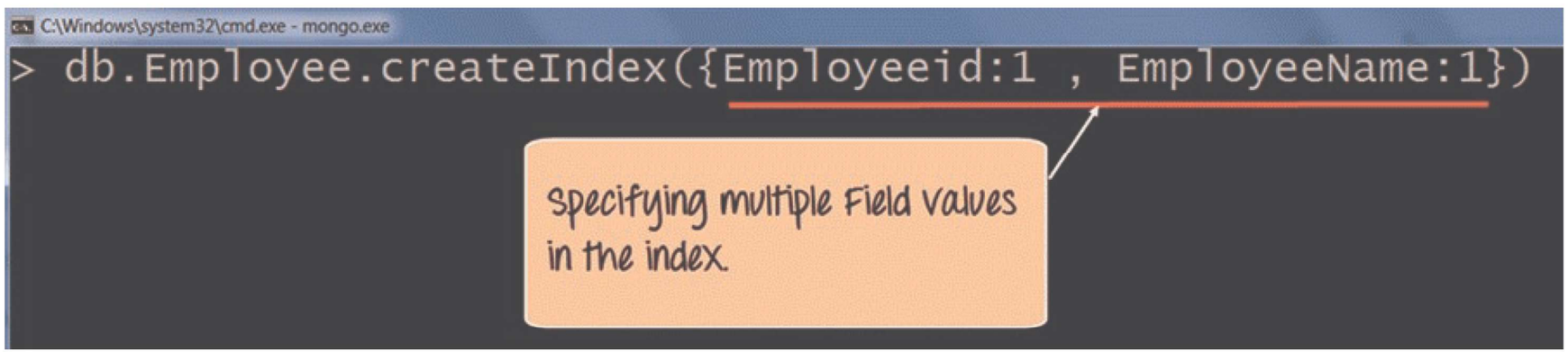
Creating an index based on one field value

1. The numIndexesBefore: 1 indicates the number of Field values (The actual fields in the collection) which were there in the indexes before the command was run. Remember that each collection has the _id field which also counts as a Field value to the index. Since the _id index field is part of the collection when it is initially created, the value of numIndexesBefore is 1.
2. The numIndexesAfter: 2 indicates the number of Field values which were there in the indexes after the command was run.
3. Here the “ok: 1” output specifies that the operation was successful, and the new index is added to the collection.

Creating an index based on multiple field values

- The createIndex method now takes into account multiple Field values which will now cause the index to be created based on the “Employeeid” and “EmployeeName”.
- The Employeeid:1 and EmployeeName:1 indicates that the index should be created on these 2 field values with the :1 indicating that it should be in ascending order.

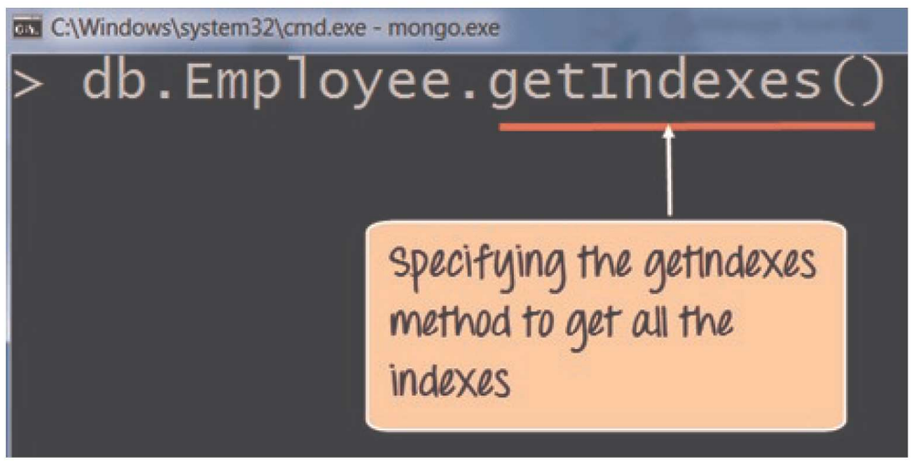
```
db.Employee.createIndex({Employeeid:1, EmployeeName:1])
```



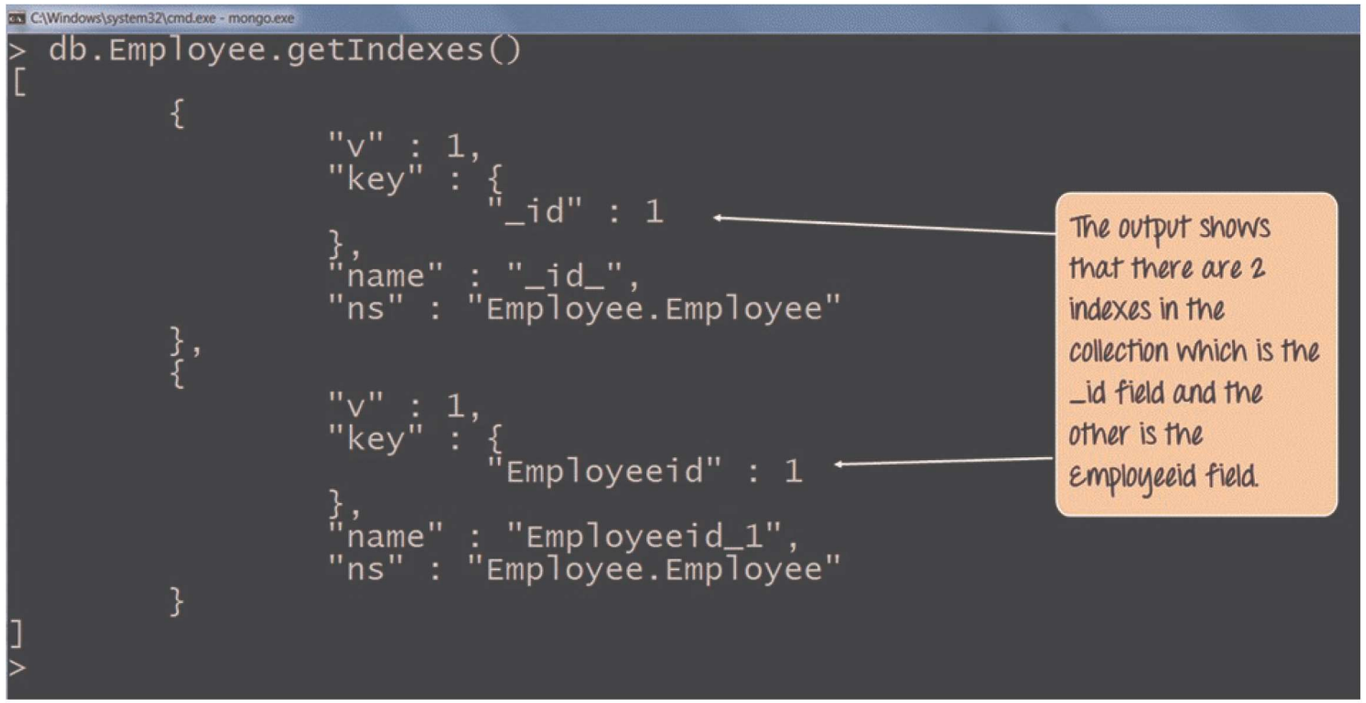
How to Find Indexes: getindexes()

- Finding an Index in MongoDB is done by using the **“getIndexes”** method.

```
db.Employee.getIndexes()
```



- The getIndexes method is used to find all of the indexes in a collection.

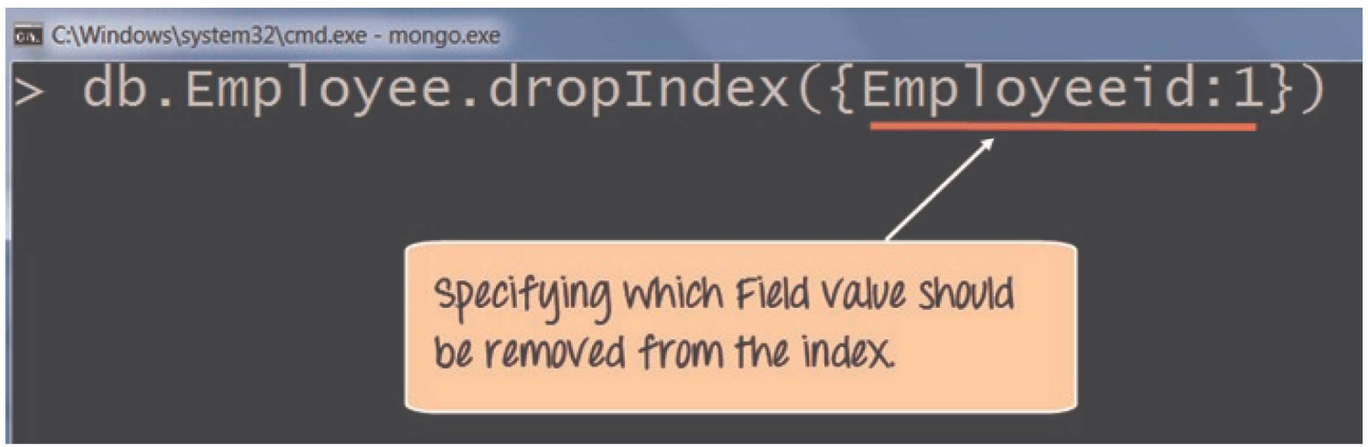


- The output returns a document which just shows that there are 2 indexes in the collection which is the _id field, and the other is the Employee id field.
- The :1 indicates that the field values in the index are created in ascending order.

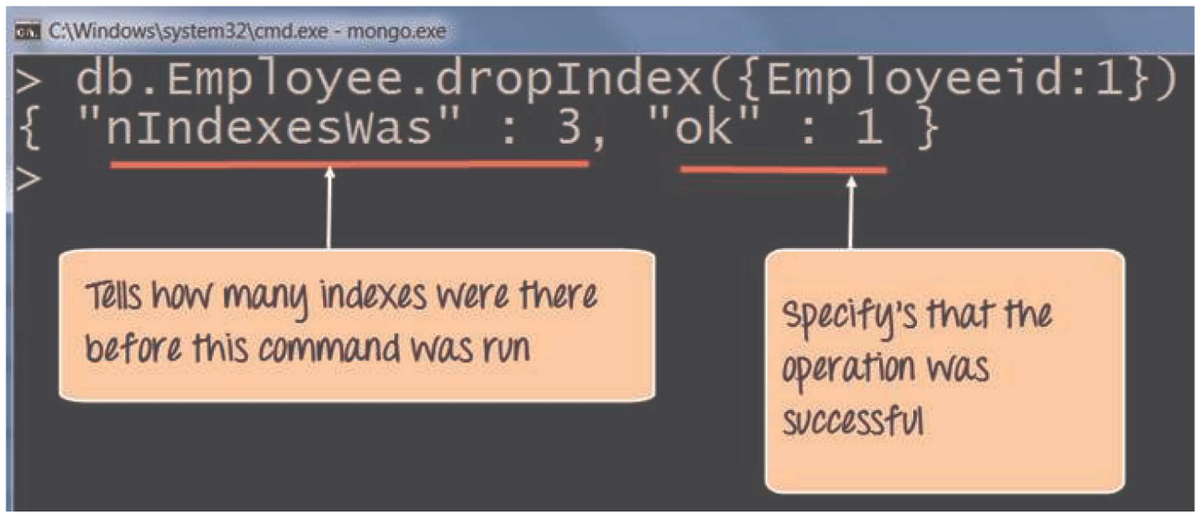
How to Drop Indexes: dropindex()

- Removing an Index in MongoDB is done by using the dropIndex method.

```
db.Employee.dropIndex({Employeeid:1})
```



- The dropIndex method takes the required Field values which needs to be removed from the Index.

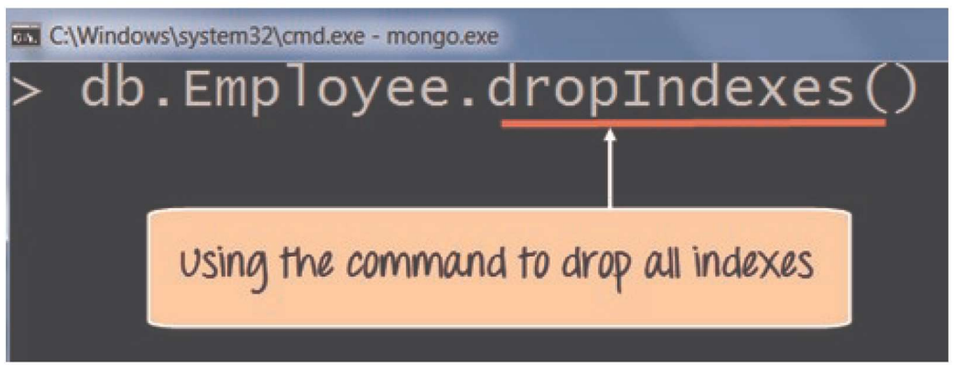


1. The nIndexesWas: 3 indicates the number of Field values which were there in the indexes before the command was run. Remember that each collection has the _id field which also counts as a Field value to the index.
2. The ok: 1 output specifies that the operation was successful, and the “Employeeid” field is removed from the index.

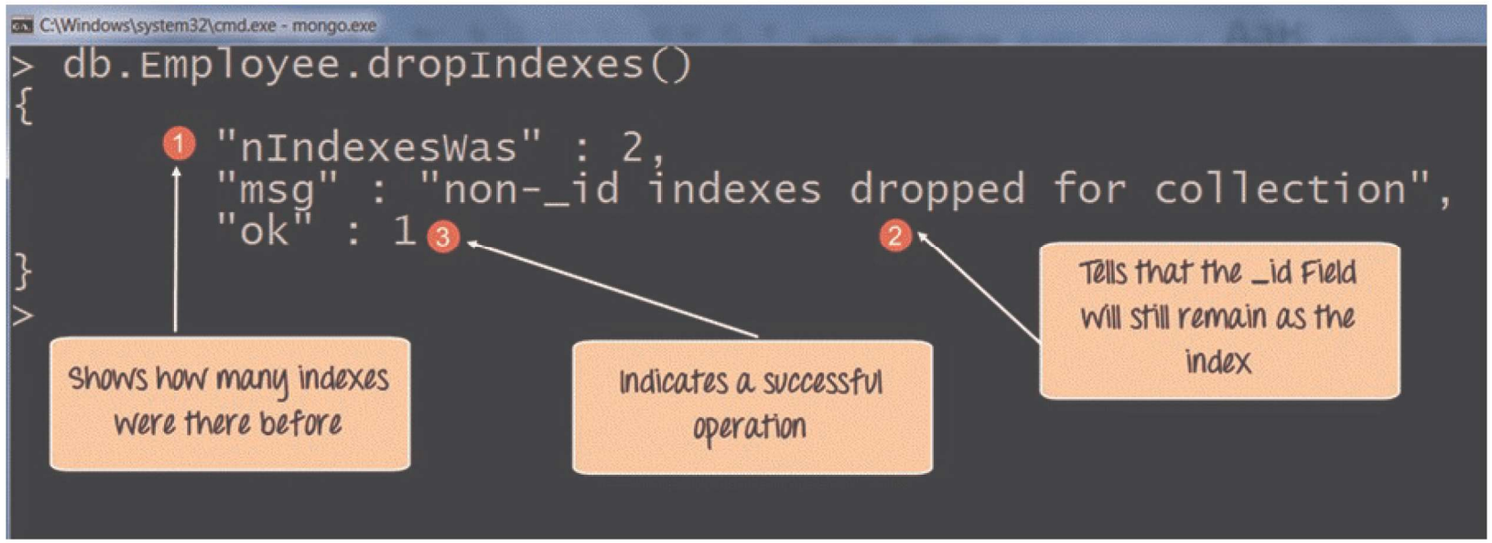
Remove All the Indexes

- To remove all of the indexes at once in the collection, one can use the dropIndexes command.

```
db.Employee.dropIndexes()
```



- The dropIndexes method will drop all of the indexes except for the _id index.



1. The nIndexesWas: 2 indicates the number of Field values which were there in the indexes before the command was run.
2. Remember again that each collection has the _id field which also counts as a Field value to the index, and that will not be removed by MongoDB and that is what this message indicates.
3. The ok: 1 output specifies that the operation was successful.

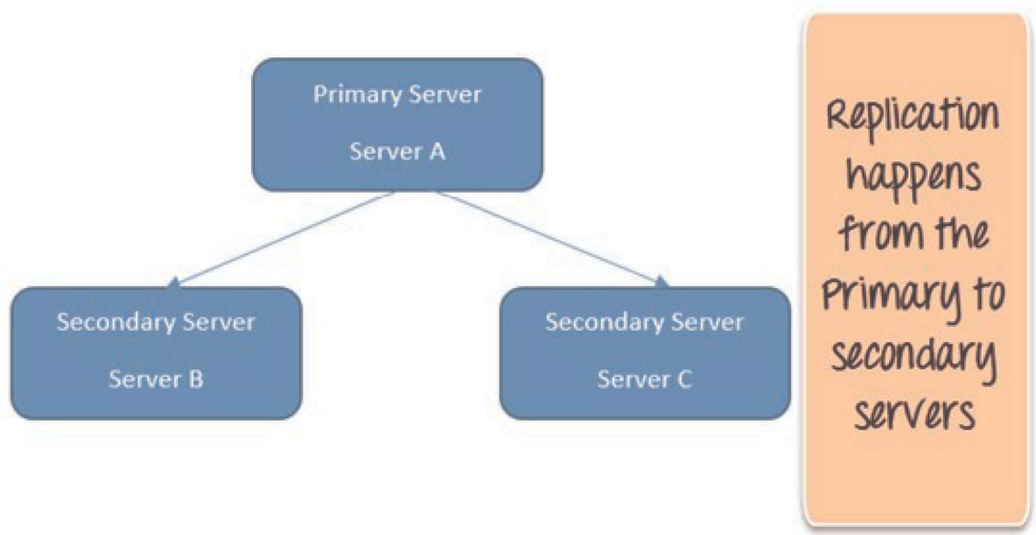
MongoDB Replication

What is MongoDB Replication?

- Replication is referred to the process of ensuring that the same data is available on more than one Mongo DB Server. This is sometimes required for the purpose of increasing data availability.
- Because if your main MongoDB Server goes down for any reason, there will be no access to the data. But if you had the data replicated to another server at regular intervals, you will be able to access the data from another server even if the primary server fails.
- Another purpose of replication is the possibility of load balancing. If there are many users connecting to the system, instead of having everyone connect to one system, users can be connected to multiple servers so that there is an equal distribution of the load.
- In MongoDB, multiple MongoDB Servers are grouped in sets called Replica sets. The Replica set will have a primary server which will accept all the write operation from clients. All other instances added to the set after this will be called the secondary instances which can be used primarily for all read operations.

How to Create Replica Set in MongoDB Adding the First Member using rs.initiate()

- To enable replication, we first need to create a replica set of MongoDB instances.
- Let's assume that for our example, we have 3 servers called ServerA, ServerB, and ServerC. In this configuration, ServerA will be our Primary server and ServerB and ServerC will be our secondary servers.



Why Replication?

1. To keep your data safe
2. High (24*7) availability of data
3. Disaster recovery
4. No downtime for maintenance (like backups, index rebuilds, compaction)
5. Read scaling (extra copies to read from)
6. Replica set is transparent to the application

How Replication Works in MongoDB

- MongoDB achieves replication by the use of replica set. A replica set is a group of **mongod** instances that host the same data set.
- In a replica, one node is primary node that receives all write operations. All other instances, such as secondary's, apply operations from the primary so that they have the same data set.
- Replica set can have only one primary node.

- Replica set is a group of two or more nodes (generally minimum 3 nodes are required).
- In a replica set, one node is primary node and remaining nodes are secondary.
- All data replicates from primary to secondary node.
- At the time of automatic failover or maintenance, election establishes for primary and a new primary node is elected.
- After the recovery of failed node, it again join the replica set and works as a secondary node.

Replica Set Features

1. A cluster of N nodes
2. Any one node can be primary
3. All write operations go to primary
4. Automatic failover
5. Automatic recovery
6. Consensus election of primary

Step 1 :

- Steps which need to be followed for creating the MongoDB replica set along with the addition of the first member to the set.

1. Ensure all mongod.exe instances added
   - Ensure that all mongod.exe instances which will be added to the replica set are installed on different servers. This is to ensure that even if one server goes down, the others will be available and hence other instances of MongoDB will be available.

Step 2 :

- All mongo.exe instances connect to each other
- Ensure that all mongo.exe instances can connect to each other.

```
mongo -host ServerB -port 27017
```

```
mongo -host ServerC -port 27017
```

Step 3:

- Start the first mongod.exe instance
- Start the first mongod.exe instance with the replSet option. This option provides a grouping for all servers which will be part of this replica set.

```
mongo -replSet "Replica1"
```

- Where “Replica1” is the name of your replica set. You can choose any meaningful name for your replica set name.

Step 4 and Step 5

Step 4 :

- First server is added to the replica set
- Now that the first server is added to the replica set, the next step is to initiate the replica set by issuing the following command **rs.initiate ().**

Step 5 :

- Verify the replica set
- Verify the replica set by issuing the command rs.conf() to ensure the replica set up properly.

Replica Set: Adding a Secondary using rs.add()

- The secondary servers can be added to the replica set by just using the rs.add command. This command takes in the name of the secondary servers and adds the servers to the replication set.
  1. Suppose if you have ServerA, ServerB, and ServerC, which are required to be part of your replica set and ServerA, is defined as the primary server in the replica set.
- To add ServerB and ServerC to the replica set issue the commands

```
rs.add("ServerB")
rs.add("ServerC")
```

Replica Set: Reconfiguring or Removing using rs.remove()

- To remove a server from the configuration set, we need to use the “rs.remove” command
  1. First perform a shutdown of the instance which you want to remove. One can do this by issuing the db.shutdownserver command from the mongo shell.
  2. Connect to the primary server
  3. Use the rs.remove command to remove the required server from the replica set. So suppose if you have a replica set with ServerA, ServerB, and ServerC, and you want to remove ServerC from the replica set, issue the command

```
rs.remove("ServerC")
```

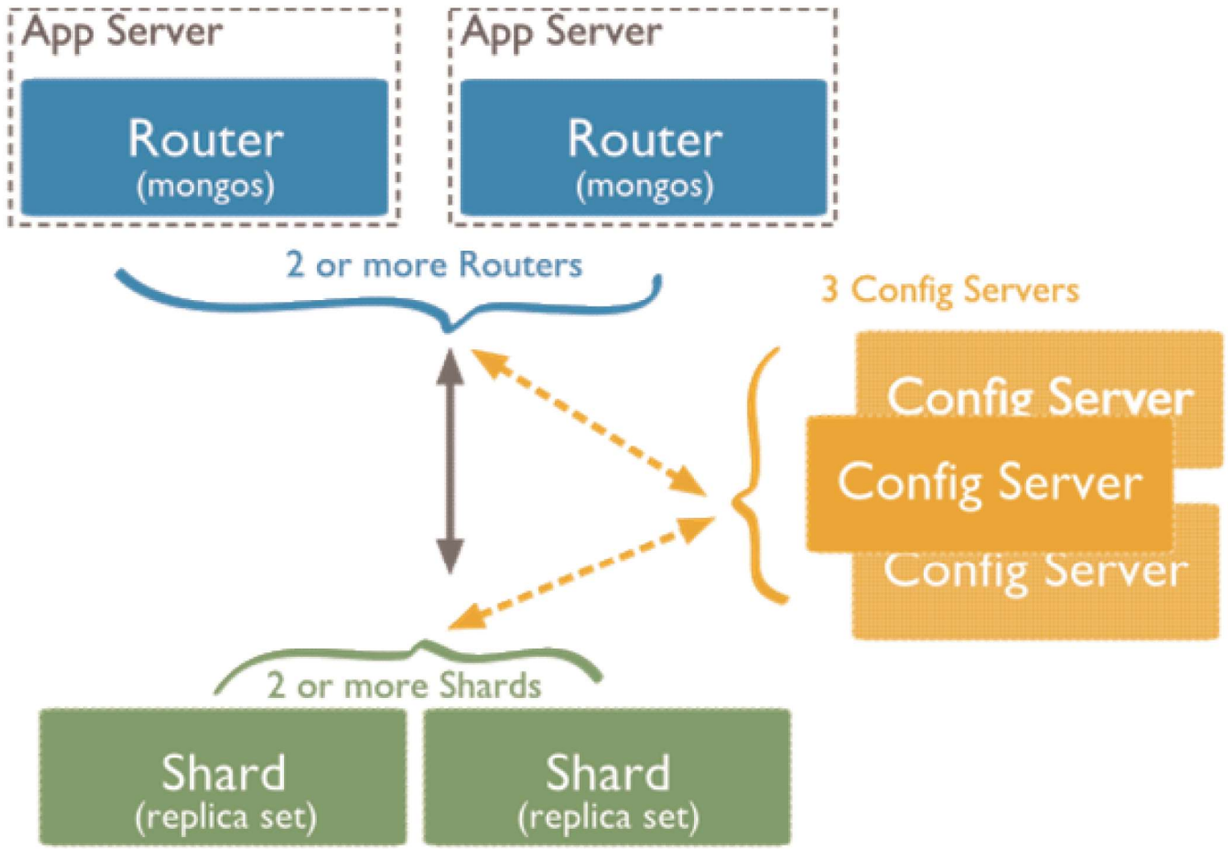
MongoDB Sharding

- Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth.
- Sharding is a concept in MongoDB, which splits large data sets into small data sets across multiple MongoDB instances.
- As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput.
- Sharding solves the problem with horizontal scaling. With sharding, you add more machines to support data growth and the demands of read and write operations.

Why Sharding?

- In replication, all writes go to master node
- Latency sensitive queries still go to master
- Single replica set has limitation of 12 nodes
- Memory can't be large enough when active dataset is big
- Local disk is not big enough
- Vertical scaling is too expensive

Sharding in MongoDB



- **A Shard** – This is the basic thing, and this is nothing but a MongoDB instance which holds the subset of the data. In production environments, all shards need to be part of replica sets.
- **Config server** – This is a mongodb instance which holds metadata about the cluster, basically information about the various mongodb instances which will hold the shard data.
- **A Router** – This is a mongodb instance which basically is responsible to re-directing the commands send by the client to the right servers.

Steps in Sharding Cluster

Step 1) Create a separate database for the config server.

```
mkdir /data/configdb
```

Step 2) Start the mongodb instance in configuration mode. Suppose if we have a server named Server D which would be our configuration server, we would need to run the below command to configure the server as a configuration server.

```
mongod -configdb ServerD: 27019
```

Step 3) Start the mongos instance by specifying the configuration server

```
mongos -configdb ServerD: 27019
```

Step 4) From the mongo shell connect to the mongo's instance

```
mongo -host ServerD -port 27017
```

Step 5) If you have Server A and Server B which needs to be added to the cluster

```
mongo -host ServerD -port 27017
```

Step 6) Enable sharding for the database. So if we need to shard the Employeedb database

```
sh.enableSharding(Employeedb)
```

Step 7) Enable sharding for the collection. So if we need to shard the Employee collection

```
Sh.shardCollection("db.Employee" , { "Employeeid" : 1 , "EmployeeName" : 1})
```

Summary

In this lesson, you should have learned how to:

- Limiting the Columns retrieved
- Limit the rows that are retrieved
- Sort the rows that are retrieved by a query
- Indexing
- Replication and Sharding