MongoDB

MongoDB

- MongoDB is an open-source document database and leading NoSQL database.
- MongoDB is written in C++.
- This will give you great understanding on MongoDB concepts needed to create and deploy a highly scalable and performance-oriented database.

MongoDB - Contents

MongoDB – Contents Contd.

MongoDB – Contents Contd.

MongoDB – Contents Contd.

MongoDB – Contents Contd.

MongoDB – Contents Contd.

MongoDB - Overview

- MongoDB is a cross-platform
- Document oriented database
- Provides high performance
- Provide high availability,
- Easily scalable
- MongoDB works on concept of collection and document.

MongoDB – Overview Contd.

- Database
  - Database is a physical container for collections.
  - Each database gets its own set of files on the file system.
  - A single MongoDB server typically has multiple databases.
- Collection
  - Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table.
  - A collection exists within a single database.
  - Collections do not enforce a schema.
  - Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

MongoDB – Overview Contd.

- Document
  - A document is a set of key-value pairs.
  - Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure,
  - Common fields in a collection's documents may hold different types of data.

MongoDB – Overview Contd.

- The following table shows the relationship of RDBMS terminology with MongoDB.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

MongoDB – Overview Contd.

Database Server and Client	
Mysqld/Oracle	mongod
mysql/sqlplus	mongo

Sample Document
Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

MongoDB – Overview Contd.

```
{
 title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'tutorials point',
  url: 'http://www.tutorialspoint.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user:'user1',
      message: 'My first comment',
      dateCreated: new Date (2011,1,20,2,15),
      like: 0
    },
    {
      user:'user2',
      message: 'My second comments',
      dateCreated: new Date (2011,1,25,7,45),
      like: 5
```

MongoDB – Overview Contd.

- **_id** is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

MongoDB - Advantages

- Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.
- Advantages of MongoDB over RDBMS
  - **Schema less** − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
  - Structure of a single object is clear.
  - No complex joins.
  - Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
  - Tuning.
  - **Ease of scale-out** − MongoDB is easy to scale.
  - Conversion/mapping of application objects to database objects not needed.
  - Uses internal memory for storing the (windowed) working set, enabling faster access of data

Why Use MongoDB?

- **Document Oriented Storage** − Data is stored in the form of JSON style documents.
- Index on any attribute
- Replication and high availability
- Auto-sharding
- Rich queries
- Fast in-place updates
- Professional support by MongoDB

Where to Use MongoDB?

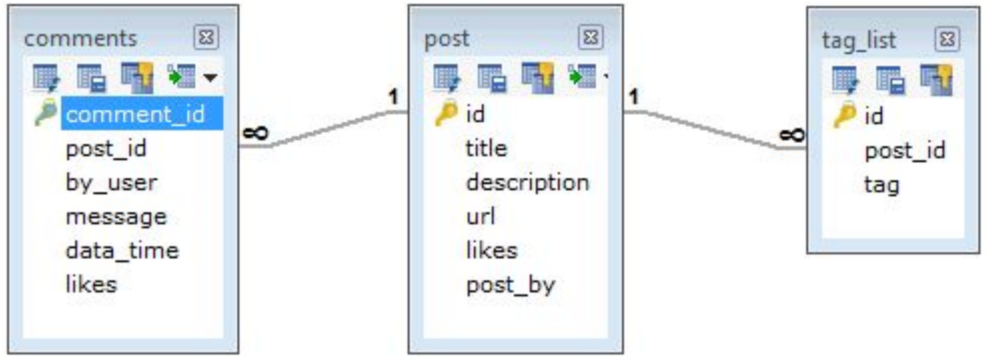
- Big Data
- Content Management and Delivery
- Mobile and Social Infrastructure
- User Data Management
- Data Hub

MongoDB - Data Modelling Contd.

- Example
  - Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.
  - Every post has the unique title, description and url.
  - Every post can have one or more tags.
  - Every post has the name of its publisher and total number of likes.
  - Every post has comments given by users along with their name, message, date-time and likes.
  - On each post, there can be zero or more comment

MongoDB - Data Modelling Contd.

- In RDBMS schema, design for above requirements will have minimum three tables.



MongoDB - Data Modelling Contd.

```
{
  _id: POST_ID
  title: TITLE_OF_POST,
  description: POST_DESCRIPTION,
  by: POST_BY,
  url: URL_OF_POST,
  tags: [TAG1, TAG2, TAG3],
  likes: TOTAL_LIKES,
  comments: [
    {
      user:'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    },
    {
      user:'COMMENT_BY',
      message: TEXT,
      dateCreated: DATE_TIME,
      like: LIKES
    }
  ]
}
```

Create Database

- In this, we will see how to create a database in MongoDB.

- The use Command
- MongoDB use DATABASE_NAME is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.
- Syntax
- Basic syntax of use DATABASE statement is as follows −

```
use DATABASE_NAME
```

Create Database Contd.

- Example
- If you want to create a database with name <mydb>, then use DATABASE statement would be as follows −

- >use mydb
- switched to db mydb
- To check your currently selected database, use the command db

- >db
- mydb
- If you want to check your databases list, use the command show dbs.

- >show dbs
- local   0.78125GB
- test   0.23012GB

Create Database Contd.

- Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

- >db.movie.insert({"name":"tutorials point"})
- >show dbs
- local   0.78125GB
- mydb   0.23012GB
- test   0.23012GB
- In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.

Drop Database

- In this , we will see how to drop a database using MongoDB command.
  - The dropDatabase() Method
  - MongoDB db.dropDatabase() command is used to drop a existing database.
- Syntax
- Basic syntax of dropDatabase() command is as follows −

```
db.dropDatabase()
```

- This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

Drop Database Contd.

- Example
- First, check the list of available databases by using the command, show dbs.
- >show dbs
- local   0.78125GB
- mydb   0.23012GB
- test   0.23012GB
- >
- If you want to delete new database <mydb>, then dropDatabase() command would be as follows −

Drop Database Contd.

- >use mydb
- switched to db mydb
- >db.dropDatabase()
- >{ "dropped" : "mydb", "ok" : 1 }
- >
- Now check list of databases.

- >show dbs
- local   0.78125GB
- test   0.23012GB
- >

Create Collection

- In this, we will see how to create a collection using MongoDB.
- The createCollection() Method

  MongoDB db.createCollection(name, options)
- is used to create collection.
- Syntax
- Basic syntax of createCollection() command is as follows −

  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.

Create Collection

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Create Collection Contd.

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexID	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false. – not applicable now
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

Create Collection Contd.

- Examples
- Basic syntax of createCollection() method without options is as follows –

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
>
```

- You can check the created collection by using the command show collections.

Create Collection Contd.

```
>show collections
mycollection
system.indexes
```

- The following example shows the syntax of createCollection() method with few important options −

- >db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 })
- { "ok" : 1 }
- >

Create Collection Contd.

- In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

```
>db.tutorialspoint.insert({"name" : "tutorialspoint"})
>show collections
mycol
mycollection
system.indexes
tutorialspoint
>
```

Mongodb drop collection

- In this, we will see how to drop a collection using MongoDB.
- The drop() Method
- MongoDB's db.collection.drop() is used to drop a collection from the database.
- Syntax
- Basic syntax of drop() command is as follows −

```
db.COLLECTION_NAME.drop()
```

Mongodb drop collection Contd.

- Example
- First, check the available collections into your database mydb.

```
>use mydb
switched to db mydb
>show collections
mycol
mycollection
system.indexes
tutorialspoint
>
```

Mongodb drop collection Contd.

- Now drop the collection with the name mycollection.

```
>db.mycollection.drop()
true
>
```

- Again check the list of collections into database.

```
>show collections
mycol
system.indexes
tutorialspoint
>
```

- drop() method will return true, if the selected collection is dropped successfully, otherwise it will return false.

MongoDB - Datatypes

- MongoDB supports many datatypes. Some of them are −
  - **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** −timestamp. This can be handy for recording when a document has been modified or added.

MongoDB – Datatypes Contd.

- **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.
- **Regular expression** − This datatype is used to store regular expression.

MongoDB - Insert Document

- In this, we will learn how to insert document in MongoDB collection.
- The insert() Method
- To insert data into MongoDB collection, you need to use MongoDB's insert() or save() method.

- Syntax
- The basic syntax of insert() command is as follows −

- `>db.COLLECTION_NAME.insert(document)`

MongoDB - Insert Document Contd.

- In this, we will learn how to insert document in MongoDB collection.

- The insert() Method
- To insert data into MongoDB collection, you need to use MongoDB's insert() or save() method.

- Syntax
- The basic syntax of insert() command is as follows −

- >db.COLLECTION_NAME.insert(document)

MongoDB - Insert Document Contd.

- Example
- >db.mycol.insert({
-   _id: ObjectId(7df78ad8902c),
-   title: 'MongoDB Overview',
-   description: 'MongoDB is no sql database',
-   by: 'tutorials point',
-   url: 'http://www.tutorialspoint.com',
-   tags: ['mongodb', 'database', 'NoSQL'],
-   likes: 100
- })
- Here mycol is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

- In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.

MongoDB - Insert Document Contd.

- _id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows −

- _id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id,
- 3 bytes incrementer)
- To insert multiple documents in a single query, you can pass an array of documents in insert() command.

MongoDB - Insert Document Contd.

- _Example
- >db.post.insert([
-    {
-       title: 'MongoDB Overview',
-       description: 'MongoDB is no sql database',
-       by: 'tutorials point',
-       url: 'http://www.tutorialspoint.com',
-       tags: ['mongodb', 'database', 'NoSQL'],
-       likes: 100
-    },
-
-    {

MongoDB - Insert Document Contd.

- title: 'NoSQL Database',
-        description: 'NoSQL database doesn't have tables',
-        by: 'tutorials point',
-        url: 'http://www.tutorialspoint.com',
-        tags: ['mongodb', 'database', 'NoSQL'],
-        likes: 20,
-        comments: [
-           {
-              user:'user1',
-              message: 'My first comment',
-              dateCreated: new Date(2013,11,10,2,35),
-              like: 0
-           }
-        ]
-     }
-])

MongoDB - Insert Document Contd.

- To insert the document you can use db.post.save(document) also. If you don't specify _id in the document then save() method will work same as insert() method. If you specify _id then it will replace whole data of document containing _id as specified in save() method.

MongoDB - Capped Collections

- **Capped collections** are fixed-size circular collections that follow the insertion order to support high performance for create, read, and delete operations. By circular, it means that when the fixed size allocated to the collection is exhausted, it will start deleting the oldest document in the collection without providing any explicit commands.
- Capped collections restrict updates to the documents if the update results in increased document size. Since capped collections store documents in the order of the disk storage, it ensures that the document size does not increase the size allocated on the disk. Capped collections are best for storing log information, cache data, or any other high volume data.

MongoDB - Capped Collections Contd.

- **>db.createCollection("cappedLogCollection",{capped:true,size:10000})**
- **In addition to collection size, we can also limit the number of documents in the collection using the max parameter −**
- >db.createCollection("cappedLogCollection",{capped:true,size:10000, max:1000})
- If you want to check whether a collection is capped or not, use the following isCapped command −

MongoDB - Capped Collections Contd.

- **>db.createCollection("cappedLogCollection",{capped:true,size:10000})**
- **In addition to collection size, we can also limit the number of documents in the collection using the max parameter −**
- >db.createCollection("cappedLogCollection",{capped:true,size:10000, max:1000})
- If you want to check whether a collection is capped or not, use the following isCapped command −

MongoDB - Capped Collections Contd.

- **>db.cappedLogCollection.isCapped()**
- **If there is an existing collection which you are planning to convert to capped, you can do it with the following code –**

- **>db.runCommand({"convertToCapped":"posts",size:10000})**
- **This code would convert our existing collection posts to a capped collection.**

MongoDB - Capped Collections Contd.

- **Querying Capped Collection**
- **By default, a find query on a capped collection will display results in insertion order. But if you want the documents to be retrieved in reverse order, use the sort command as shown in the following code –**
- **>db.cappedLogCollection.find().sort({$natural:-1})**
- **There are few other important points regarding capped collections worth knowing**

MongoDB - Capped Collections Contd.

- **We cannot delete documents from a capped collection.**
- **There are no default indexes present in a capped collection, not even on _id field.**
- **While inserting a new document, MongoDB does not have to actually look for a place to accommodate new document on the disk. It can blindly insert the new document at the tail of the collection. This makes insert operations in capped collections very fast.**

MongoDB - Capped Collections Contd.

- **Similarly, while reading documents MongoDB returns the documents in the same order as present on disk. This makes the read operation very fast.**

MongoDB - Query Document

- In this , we will learn how to query document from MongoDB collection.

- The find() Method
- To query data from MongoDB collection, you need to use MongoDB's find() method.

- Syntax
- The basic syntax of find() method is as follows −

- >db.COLLECTION_NAME.find()
- find() method will display all the documents in a non-structured way.

MongoDB - Query Document Contd.

- The pretty() Method
- To display the results in a formatted way, you can use pretty() method.
- Syntax
- >db.mycol.find().pretty()
- Example

```
>db.mycol.find().pretty()
{
  "_id": ObjectId(7df78ad8902c),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database",
  "by": "tutorials point",
  "url": "http://www.tutorialspoint.com",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": "100"
}
>
```

- Apart from find() method, there is findOne() method, that returns only one document.

MongoDB - Query Document Contd.

- RDBMS Where Clause Equivalents in MongoDB
- To query the document on the basis of some condition, you can use following operations

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$lt:<value>}}	db.mycol.find({"likes":{$lt:50}}).pretty()	where likes < 50
Less Than Equals	{<key>:{$lte:<value>}}	db.mycol.find({"likes":{$lte:50}}).pretty()	where likes <= 50
Greater Than	{<key>:{$gt:<value>}}	db.mycol.find({"likes":{$gt:50}}).pretty()	where likes > 50
Greater Than Equals	{<key>:{$gte:<value>}}	db.mycol.find({"likes":{$gte:50}}).pretty()	where likes >= 50
Not Equals	{<key>:{$ne:<value>}}	db.mycol.find({"likes":{$ne:50}}).pretty()	where likes != 50

AND in MongoDB.

MongoDB - Query Document Contd.

- Syntax
- In the find() method, if you pass multiple keys by separating them by ',' then MongoDB treats it as AND condition. Following is the basic syntax of AND −

```
>db.mycol.find(
  {
    $and: [
      {key1: value1}, {key2:value2}
    ]
  }
).pretty()
```

MongoDB - Query Document Contd.

- Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

```
>db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty() {
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials point",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

MongoDB - Query Document Contd.

- Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview‘ and likes =100. all conditions are separated by ,.

```
>db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB
Overview"},{“likes”:100}]}).pretty() {
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials point",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

MongoDB - Query Document Contd.

- Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

```
>db.mycol.find({$and:[{"by":"tutorials point"},{"title": "MongoDB
Overview"}]}).pretty() {
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials point",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
```

For the above given example, equivalent where clause will be ' where by = 'tutorials point' AND title = 'MongoDB Overview' '. You can pass any number of key, value pairs in find clause.

MongoDB - Query Document Contd.

OR in MongoDB

Syntax

To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR −

```
>db.mycol.find(
  {
    $or: [
      {key1: value1}, {key2:value2}
    ]
  }
).pretty()
```

MongoDB - Query Document Contd.

- Example
- Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

```
>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}).pretty()
{
  "_id": ObjectId(7df78ad8902c),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database",
  "by": "tutorials point",
  "url": "http://www.tutorialspoint.com",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": "100"
}
>
```

MongoDB - Query Document Contd.

- Example
- Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview‘ or likes are 100.

```
>db.mycol.find({$or:[{"by":"tutorials point"},{"title": "MongoDB
Overview"},{“likes”:100}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials point",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
>
```

MongoDB - Query Document Contd.

- Using AND and OR Together
- Example
- The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is 'where likes>10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')'

MongoDB - Query Document Contd.

```
>db.mycol.find({"likes": {$gt:10}, $or: [{"by": "tutorials point"},
   {"title": "MongoDB Overview"}]}).pretty()
{
   "_id": ObjectId(7df78ad8902c),
   "title": "MongoDB Overview",
   "description": "MongoDB is no sql database",
   "by": "tutorials point",
   "url": "http://www.tutorialspoint.com",
   "tags": ["mongodb", "database", "NoSQL"],
   "likes": "100"
}
>
```

MongoDB - Update Document

- MongoDB's update() and save() methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.
- MongoDB Update() Method
- The update() method updates the values in the existing document.

- Syntax
- The basic syntax of update() method is as follows −

```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)
```

MongoDB - Update Document

- Example
- Consider the mycol collection has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

- Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

```
>db.mycol.update({'title':'MongoDB Overview'},{$set:{'title':'New MongoDB
Tutorial'}})
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"New MongoDB Tutorial"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
>
```

MongoDB - Update Document

- By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

- >db.mycol.update({'title':'MongoDB Overview'},
- {$set:{'title':'New MongoDB Tutorial'}},{multi:true})

MongoDB - Update Document

- MongoDB Save() Method
- The save() method replaces the existing document with the new document passed in the save() method.
- Syntax
- The basic syntax of MongoDB save() method is shown below −

- >db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})

MongoDB - Update Document

- Example
- Following example will replace the document with the _id '5983548781331adf45ec7'.

```
>db.mycol.save(
  {
    "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point New Topic",
      "by":"Tutorials Point"
  }
)
>db.mycol.find()
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"Tutorials Point New Topic",
  "by":"Tutorials Point"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
>
```

MongoDB - Delete Document

- In this chapter, we will learn how to delete a document using MongoDB.
- The remove() Method
- MongoDB's remove() method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.
- deletion criteria − (Optional) deletion criteria according to documents will be removed.
- justOne − (Optional) if set to true or 1, then remove only one document.
- Syntax
- Basic syntax of remove() method is as follows −
- >db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

MongoDB - Delete Document Contd.

- Example
- Consider the mycol collection has the following data.

- { "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
- { "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
- { "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
- Following example will remove all the documents whose title is 'MongoDB Overview'.

- >db.mycol.remove({'title':'MongoDB Overview'})
- >db.mycol.find()
- { "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
- { "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
- >

MongoDB - Delete Document Contd.

- Remove Only One
  - If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.
  - >db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)
- Remove All Documents
  - 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.mycol.remove()
  - >db.mycol.find()
  - >

MongoDB - Projection

- 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, explained in MongoDB Query Document 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.
- Syntax
- The basic syntax of find() method with projection is as follows −

- `>db.COLLECTION_NAME.find({},{KEY:1})`

MongoDB – Projection Contd.

- Example
- Consider the collection mycol has the following data −

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display the title of the document while querying the document.

```
>db.mycol.find({},{"title":1,_id:0})
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
>
```

- Please note _id field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0.

MongoDB - Limit Records

- In this chapter, we will learn how to limit records using MongoDB.
- The Limit() Method
- To limit the records in MongoDB, you need to use limit() method. The method accepts one number type argument, which is the number of documents that you want to be displayed.
- Syntax
- The basic syntax of limit() method is as follows −
- >db.COLLECTION_NAME.find().limit(NUMBER)

MongoDB - Limit Records Contd.

- Example
- Consider the collection myycol has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

- Following example will display only two documents while querying the document.

```
>db.mycol.find({},{"title":1,_id:0}).limit(2)
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
>
```

- If you don't specify the number argument in limit() method then it will display all documents from the collection.

MongoDB - Limit Records Contd.

- MongoDB Skip() Method
- Apart from limit() method, there is one more method skip() which also accepts number type argument and is used to skip the number of documents.
- Syntax
- The basic syntax of skip() method is as follows −

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

MongoDB - Limit Records Contd.

- Example
- Following example will display only the second document.

- >db.mycol.find({},{"title":1,_id:0}).limit(1).skip(1)
- {"title":"NoSQL Overview"}
- >
- Please note, the default value in skip() method is 0.

MongoDB - Sort Records

- In this chapter, we will learn how to sort records in MongoDB.

- 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
- The basic syntax of sort() method is as follows −

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

MongoDB - Sort Records Contd.

- Example
- Consider the collection myycol has the following data.

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

- Following example will display the documents sorted by title in the descending order.

```
>db.mycol.find({},{"title":1,_id:0}).sort({"title":-1})
{"title":"Tutorials Point Overview"}
{"title":"NoSQL Overview"}
{"title":"MongoDB Overview"}
>
```

- Please note, if you don't specify the sorting preference, then sort() method will display the documents in ascending order.

MongoDB - Indexing

- Indexes support the efficient resolution of queries.
- Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement.
- This scan is highly inefficient and require MongoDB to process a large volume of data.
- Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form.
- The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

MongoDB – Indexing Contd.

- The createIndex() Method
  - To create an index you need to use ensureIndex() method of MongoDB.
- Syntax
- The basic syntax of ensureIndex() method is as follows().

```
>db.COLLECTION_NAME.createIndex({KEY:1})
```

- Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.
- Example

```
>db.mycol.createIndex({"title":1})
```

In createIndex() method you can pass multiple fields, to create index on multiple fields.

```
>db.mycol.createIndex({"title":1,"description":-1})
```

MongoDB – Indexing Contd.

- **createIndex()** method also accepts list of options (which are optional). Following is the list –

Parameter	Type	Description
background	Boolean	Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is **false**.
unique	Boolean	Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is **false**.

MongoDB – Indexing Contd.

name	string	The name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order.
dropDups	Boolean	Creates a unique index on a field that may have duplicates. MongoDB indexes only the first occurrence of a key and removes all documents from the collection that contain subsequent occurrences of that key. Specify true to create unique index. The default value is **false**.
sparse	Boolean	If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is **false**.

MongoDB – Indexing Contd.

expireAfterSeconds	integer	Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection.
v	index version	The index version number. The default index version depends on the version of MongoDB running when creating the index.
weights	document	The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score.

MongoDB – Indexing Contd.

default_language	string	For a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is **english**.
language_override	string	For a text index, specify the name of the field in the document that contains, the language to override the default language. The default value is language.

MongoDB - Aggregation

- Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

- In SQL count(*) and with group by is an equivalent of mongodb aggregation.
- The aggregate() Method
  - For the aggregation in MongoDB, you should use aggregate() method.
- Syntax
  - Basic syntax of aggregate() method is as follows −

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

MongoDB – Aggregation Contd.

- Example
- In the collection you have the following data −
- {
-    _id: ObjectId(7df78ad8902c)
-    title: 'MongoDB Overview',
-    description: 'MongoDB is no sql database',
-    by_user: 'tutorials point',
-    url: 'http://www.tutorialspoint.com',
-    tags: ['mongodb', 'database', 'NoSQL'],
-    likes: 100
- },
- {
-    _id: ObjectId(7df78ad8902d)
-    title: 'NoSQL Overview',
-    description: 'No sql database is very fast',

MongoDB – Aggregation Contd.

- by_user: 'tutorials point',
- url: 'http://www.tutorialspoint.com',
- tags: ['mongodb', 'database', 'NoSQL'],
- likes: 10
- },
- {
- _id: ObjectId(7df78ad8902e)
- title: 'Neo4j Overview',
- description: 'Neo4j is no sql database',
- by_user: 'Neo4j',
- url: 'http://www.neo4j.com',
- tags: ['neo4j', 'database', 'NoSQL'],
- likes: 750
- },

MongoDB – Aggregation Contd.

- Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method −
- > db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
- {
-    "result" : [
-       {
-          "_id" : "tutorials point",
-          "num_tutorial" : 2
-       },
-       {
-          "_id" : "Neo4j",
-          "num_tutorial" : 1
-       }
-    ],
-    "ok" : 1
- }
- >
- Sql equivalent query for the above use case will be select by_user, count(*) from mycol group by by_user.

MongoDB – Aggregation Contd.

- In the above example, we have grouped documents by field by_user and on each occurrence of by_user previous value of sum is incremented. Following is a list of available aggregation expressions.

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])

MongoDB – Aggregation Contd.

Expression	Description	Example
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])

MongoDB – Aggregation Contd.

Expression	Description	Example
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])

MongoDB – Aggregation Contd.

Expression	Description	Example
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

MongoDB – Aggregation Contd.

- Pipeline Concept
- In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on. MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

MongoDB – Aggregation Contd.

- Following are the possible stages in aggregation framework −
    - $project − Used to select some specific fields from a collection.
    - $match − This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
    - $group − This does the actual aggregation as discussed above.
    - $sort − Sorts the documents.
    - $skip − With this, it is possible to skip forward in the list of documents for a given amount of documents.
    - $limit − This limits the amount of documents to look at, by the given number starting from the current positions.
    - $unwind − This is used to unwind document that are using arrays. When using an array, the data is kind of pre-joined and this operation will be undone with this to have individual documents again. Thus with this stage we will increase the amount of documents for the next stage.

MongoDB - Replication

- Replication is the process of synchronizing data across multiple servers.
- Replication provides redundancy and increases data availability with multiple copies of data on different database servers.
- Replication protects a database from the loss of a single server.
- Replication also allows you to recover from hardware failure and service interruptions.
- With additional copies of the data, you can dedicate one to disaster recovery, reporting, or backup.

MongoDB – Replication Contd.

- Why Replication?
  - To keep your data safe
  - High (24*7) availability of data
  - Disaster recovery
  - No downtime for maintenance (like backups, index rebuilds, compaction)
  - Read scaling (extra copies to read from)
  - Replica set is transparent to the application

MongoDB – Replication Contd.

How Replication Works in MongoDB

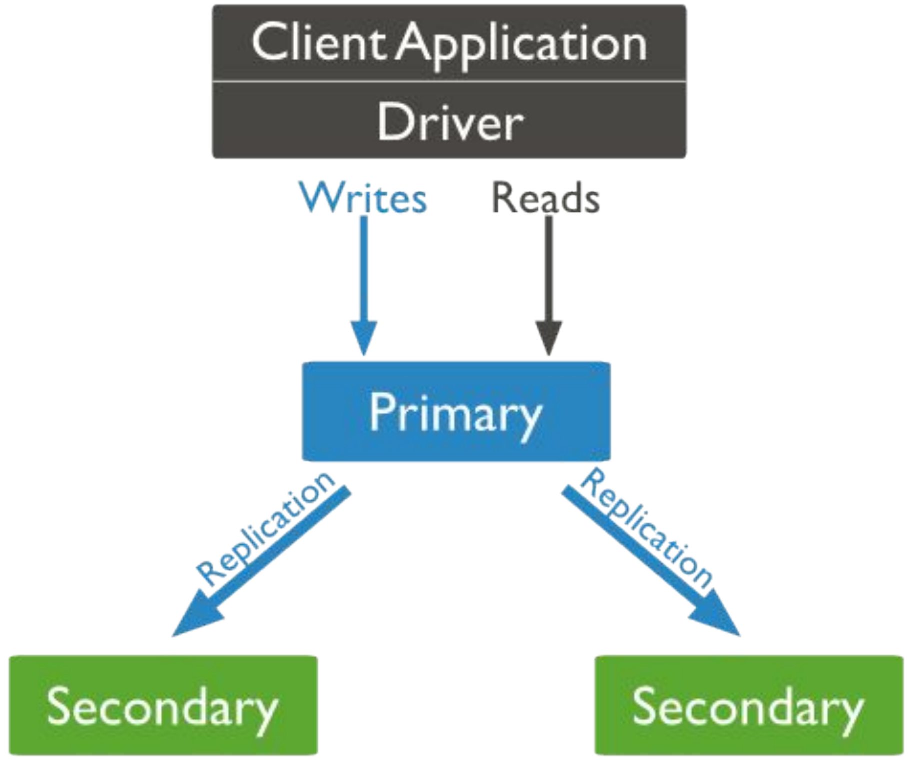
- MongoDB achieves replication by the use of replica set. A replica set is a group of mongod instances that host the same data set. In a replica, one node is primary node that receives all write operations. All other instances, such as secondaries, apply operations from the primary so that they have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

MongoDB – Replication Contd.

- 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.

MongoDB – Replication Contd.

- A typical diagram of MongoDB replication is shown in which client application always interact with the primary node and the primary node then replicates the data to the secondary nodes.



MongoDB – Replication Contd.

- Replica Set Features
- A cluster of N nodes
- Any one node can be primary
- All write operations go to primary
- Automatic failover
- Automatic recovery
- Consensus election of primary

MongoDB – Replication Contd.

- Set Up a Replica Set
- In this, we will convert standalone MongoDB instance to a replica set. To convert to replica set, following are the steps –

- Shutdown already running MongoDB server.

- mongod --port "PORT" --dbpath "YOUR_DB_DATA_PATH" --replSet "REPLICA_SET_INSTANCE_NAME"

MongoDB – Replication Contd.

- Example
- mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
- It will start a mongod instance with the name rs0, on port 27017.
- Now start the command prompt and connect to this mongod instance.
- In Mongo client, issue the command rs.initiate() to initiate a new replica set.
- To check the replica set configuration, issue the command rs.conf(). To check the status of replica set issue the command rs.status().

MongoDB – Replication Contd.

- Add Members to Replica Set
- To add members to replica set, start mongod instances on multiple machines. Now start a mongo client and issue a command rs.add().

- Syntax
- The basic syntax of rs.add() command is as follows −

- >rs.add(HOST_NAME:PORT)

MongoDB – Replication Contd.

- Example
- Suppose your mongod instance name is mongod1.net and it is running on port 27017. To add this instance to replica set, issue the command rs.add() in Mongo client.

```
>rs.add("mongod1.net:27017")
```

- You can add mongod instance to replica set only when you are connected to primary node. To check whether you are connected to primary or not, issue the command db.isMaster() in mongo client.

MongoDB - Sharding

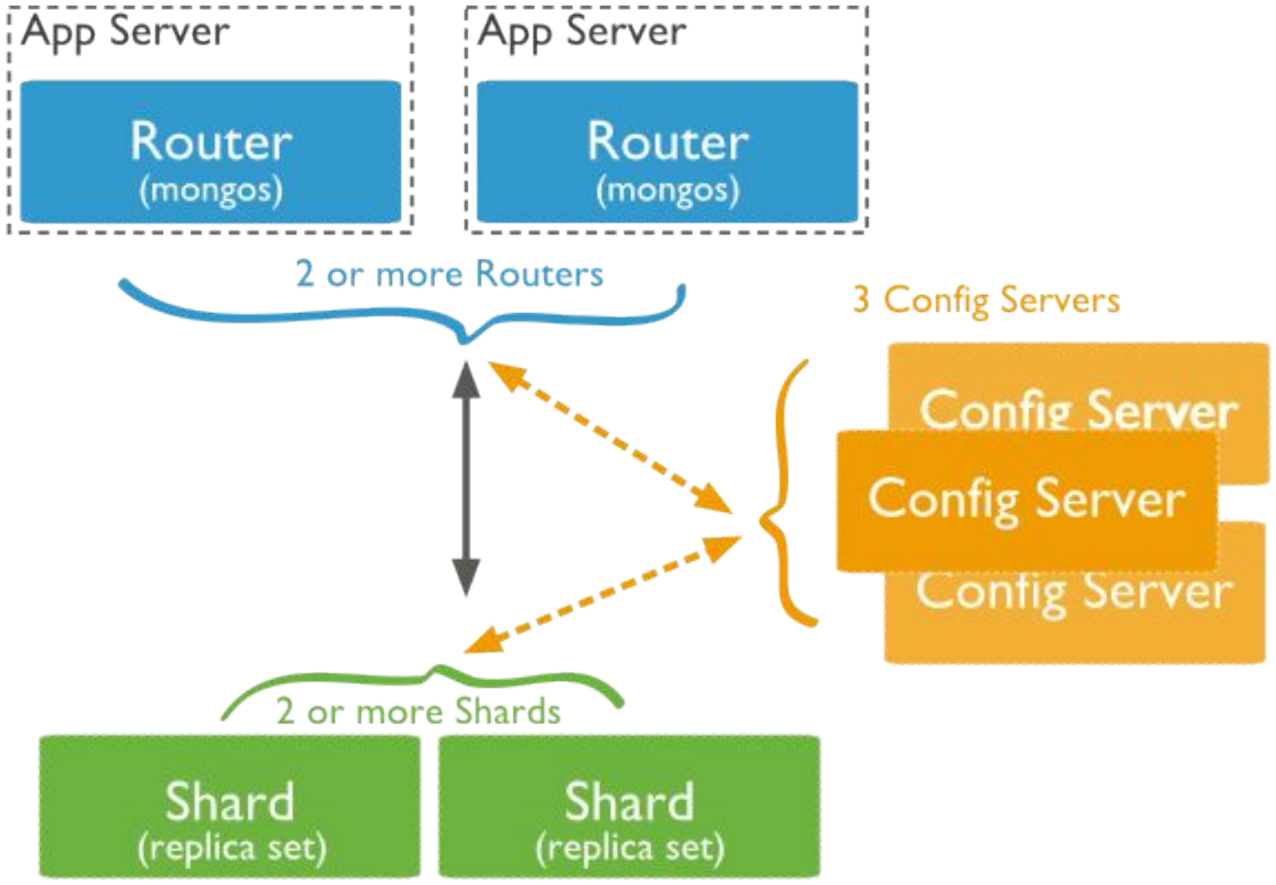
- Sharding is the process of storing data records across multiple machines and it is MongoDB's approach to meeting the demands of data growth.
- As the size of the data increases, a single machine may not be sufficient to store the data nor provide an acceptable read and write throughput.
- Sharding 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

The following diagram shows the sharding in MongoDB using sharded cluster.



Sharding in MongoDB

- In the following diagram, there are three main components −
  - **Shards** − Shards are used to store data. They provide high availability and data consistency. In production environment, each shard is a separate replica set.
  - **Config Servers** − Config servers store the cluster's metadata. This data contains a mapping of the cluster's data set to the shards. The query router uses this metadata to target operations to specific shards. In production environment, sharded clusters have exactly 3 config servers.
  - **Query Routers** − Query routers are basically mongo instances, interface with client applications and direct operations to the appropriate shard. The query router processes and targets the operations to shards and then returns results to the clients. A sharded cluster can contain more than one query router to divide the client request load. A client sends requests to one query router. Generally, a sharded cluster have many query routers.
-

MongoDB - Create Backup

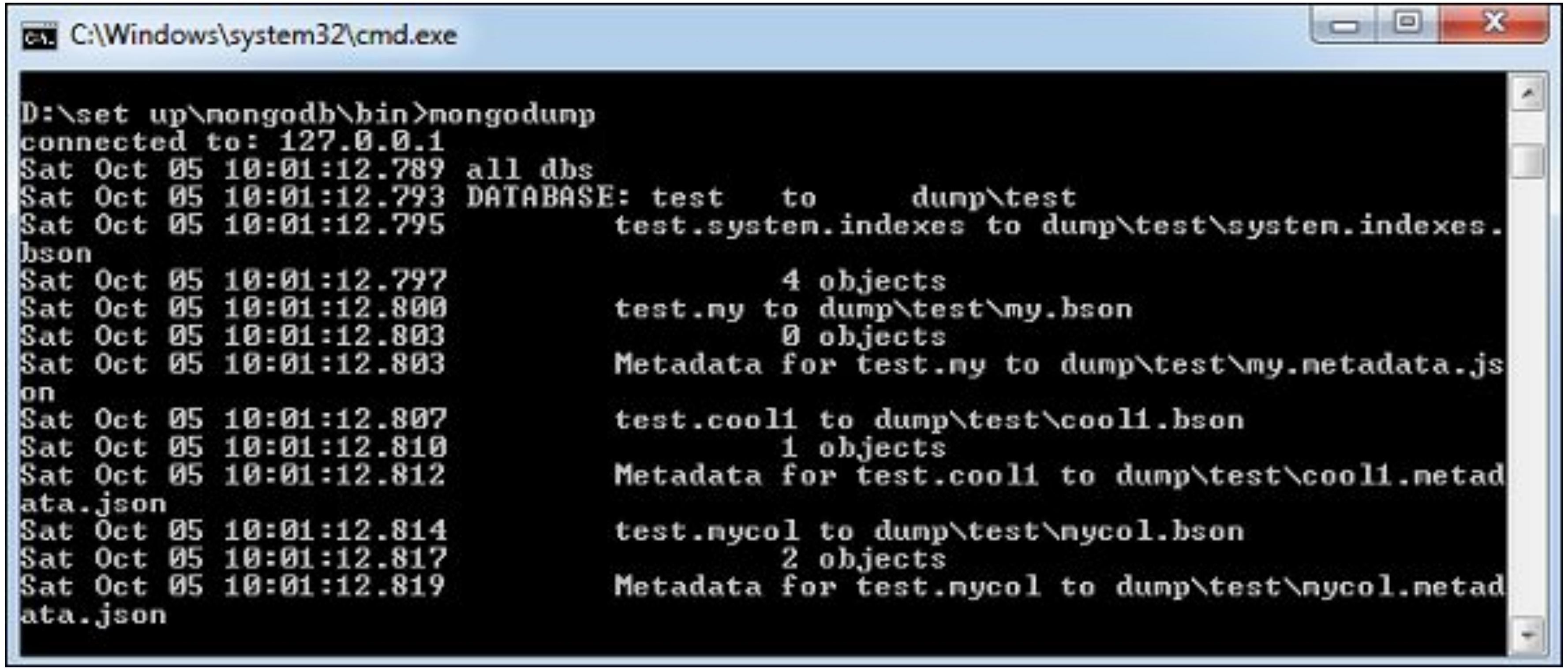
- Dump MongoDB Data
- To create backup of database in MongoDB, you should use **mongodump**command. This command will dump the entire data of your server into the dump directory. There are many options available by which you can limit the amount of data or create backup of your remote server.
- Syntax
- The basic syntax of **mongodump** command is as follows −

```
>mongodump
```

MongoDB - Create Backup Contd.

- Example
- Start your mongod server. Assuming that your mongod server is running on the localhost and port 27017, open a command prompt and go to the bin directory of your mongodb instance and type the command **mongodump**
- Consider the mycol collection has the following data.
- >mongodump

MongoDB - Create Backup Contd.



```
D:\set up\mongodb\bin>mongodump
connected to: 127.0.0.1
Sat Oct 05 10:01:12.789 all dbs
Sat Oct 05 10:01:12.793 DATABASE: test   to     dump\test
Sat Oct 05 10:01:12.795         test.system.indexes to dump\test\system.indexes.
bson
Sat Oct 05 10:01:12.797                  4 objects
Sat Oct 05 10:01:12.800         test.my to dump\test\my.bson
Sat Oct 05 10:01:12.803                  0 objects
Sat Oct 05 10:01:12.803         Metadata for test.my to dump\test\my.metadata.js
on
Sat Oct 05 10:01:12.807         test.cool1 to dump\test\cool1.bson
Sat Oct 05 10:01:12.810                  1 objects
Sat Oct 05 10:01:12.812         Metadata for test.cool1 to dump\test\cool1.metad
ata.json
Sat Oct 05 10:01:12.814         test.mycol to dump\test\mycol.bson
Sat Oct 05 10:01:12.817                  2 objects
Sat Oct 05 10:01:12.819         Metadata for test.mycol to dump\test\mycol.metad
ata.json
```

MongoDB - Create Backup Contd.

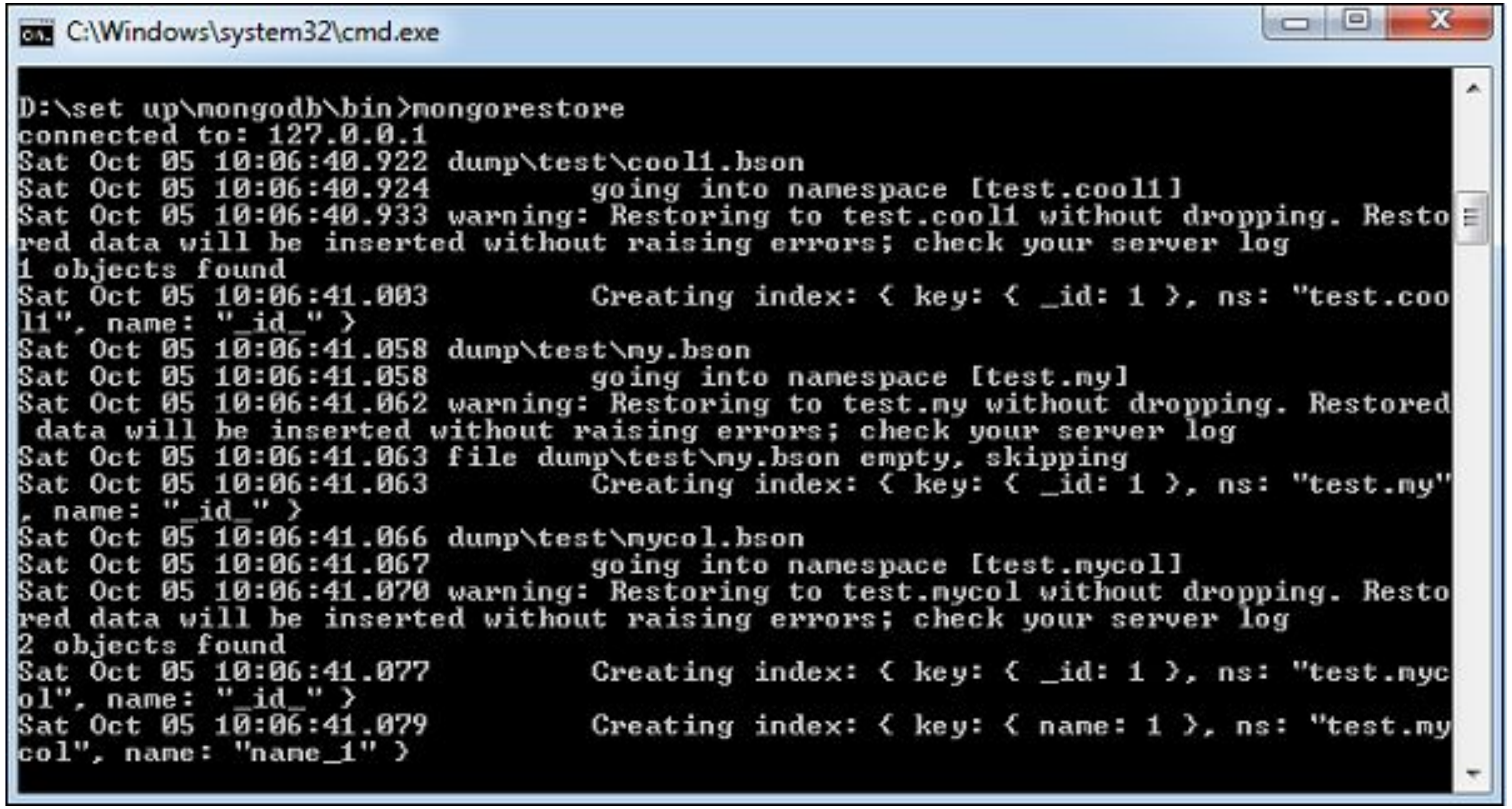
- Following is a list of available options that can be used with the **mongodump** command.
-

Syntax	Description	Example
mongodump --host HOST_NAME --port PORT_NUMBER	This commmand will backup all databases of specified mongod instance.	mongodump --host tutorialspoint.com --port 27017
mongodump --dbpath DB_PATH --out BACKUP_DIRECTORY	This command will backup only specified database at specified path.	mongodump --dbpath /data/db/ --out /data/backup/
mongodump --collection	This command will backup only specified collection of	mongodump --collection mycol --db

Restore data

- To restore backup data MongoDB's **mongorestore** command is used. This command restores all of the data from the backup directory.
- Syntax
- The basic syntax of **mongorestore** command is −
- >mongorestore
- Following is the output of the command −

Restore data Contd.



```
D:\set up\mongodb\bin>mongorestore
connected to: 127.0.0.1
Sat Oct 05 10:06:40.922 dump\test\cool1.bson
Sat Oct 05 10:06:40.924         going into namespace [test.cool1]
Sat Oct 05 10:06:40.933 warning: Restoring to test.cool1 without dropping. Resto
red data will be inserted without raising errors; check your server log
1 objects found
Sat Oct 05 10:06:41.003         Creating index: { key: { _id: 1 }, ns: "test.coo
l1", name: "_id_" }
Sat Oct 05 10:06:41.058 dump\test\my.bson
Sat Oct 05 10:06:41.058         going into namespace [test.my]
Sat Oct 05 10:06:41.062 warning: Restoring to test.my without dropping. Restored
 data will be inserted without raising errors; check your server log
Sat Oct 05 10:06:41.063 file dump\test\my.bson empty, skipping
Sat Oct 05 10:06:41.063         Creating index: { key: { _id: 1 }, ns: "test.my"
, name: "_id_" }
Sat Oct 05 10:06:41.066 dump\test\mycol.bson
Sat Oct 05 10:06:41.067         going into namespace [test.mycol]
Sat Oct 05 10:06:41.070 warning: Restoring to test.mycol without dropping. Resto
red data will be inserted without raising errors; check your server log
2 objects found
Sat Oct 05 10:06:41.077         Creating index: { key: { _id: 1 }, ns: "test.myc
ol", name: "_id_" }
Sat Oct 05 10:06:41.079         Creating index: { key: { name: 1 }, ns: "test.my
col", name: "name_1" }
```

MongoDB - Deployment

- When you are preparing a MongoDB deployment, you should try to understand how your application is going to hold up in production. It’s a good idea to develop a consistent, repeatable approach to managing your deployment environment so that you can minimize any surprises once you’re in production.
- The best approach incorporates prototyping your set up, conducting load testing, monitoring key metrics, and using that information to scale your set up. The key part of the approach is to proactively monitor your entire system - this will help you understand how your production system will hold up before deploying, and determine where you will need to add capacity. Having insight into potential spikes in your memory usage, for example, could help put out a write-lock fire before it starts.

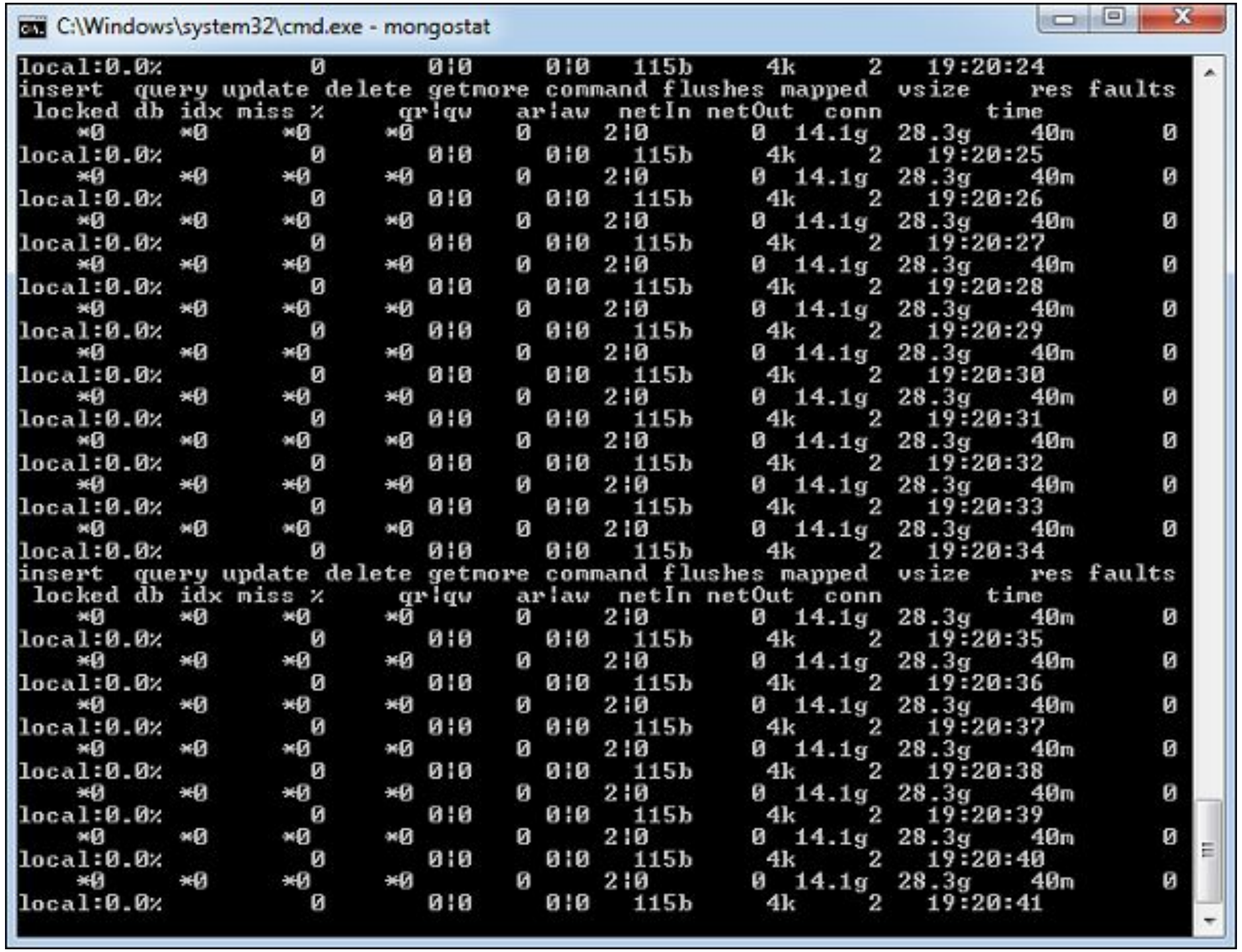
MongoDB – Deployment Contd.

- To monitor your deployment, MongoDB provides some of the following commands −
- mongostat
- This command checks the status of all running mongod instances and return counters of database operations. These counters include inserts, queries, updates, deletes, and cursors. Command also shows when you’re hitting page faults, and showcase your lock percentage. This means that you're running low on memory, hitting write capacity or have some performance issue.
- To run the command, start your mongod instance. In another command prompt, go to **bin** directory of your mongodb installation and type **mongostat**.
- D:\set up\mongodb\bin>mongostat

MongoDB – Deployment Contd.

- To monitor your deployment, MongoDB provides some of the following commands −
- mongostat
- This command checks the status of all running mongod instances and return counters of database operations. These counters include inserts, queries, updates, deletes, and cursors. Command also shows when you’re hitting page faults, and showcase your lock percentage. This means that you're running low on memory, hitting write capacity or have some performance issue.
- To run the command, start your mongod instance. In another command prompt, go to **bin** directory of your mongodb installation and type **mongostat**.
- D:\set up\mongodb\bin>mongostat

MongoDB – Deployment Contd.



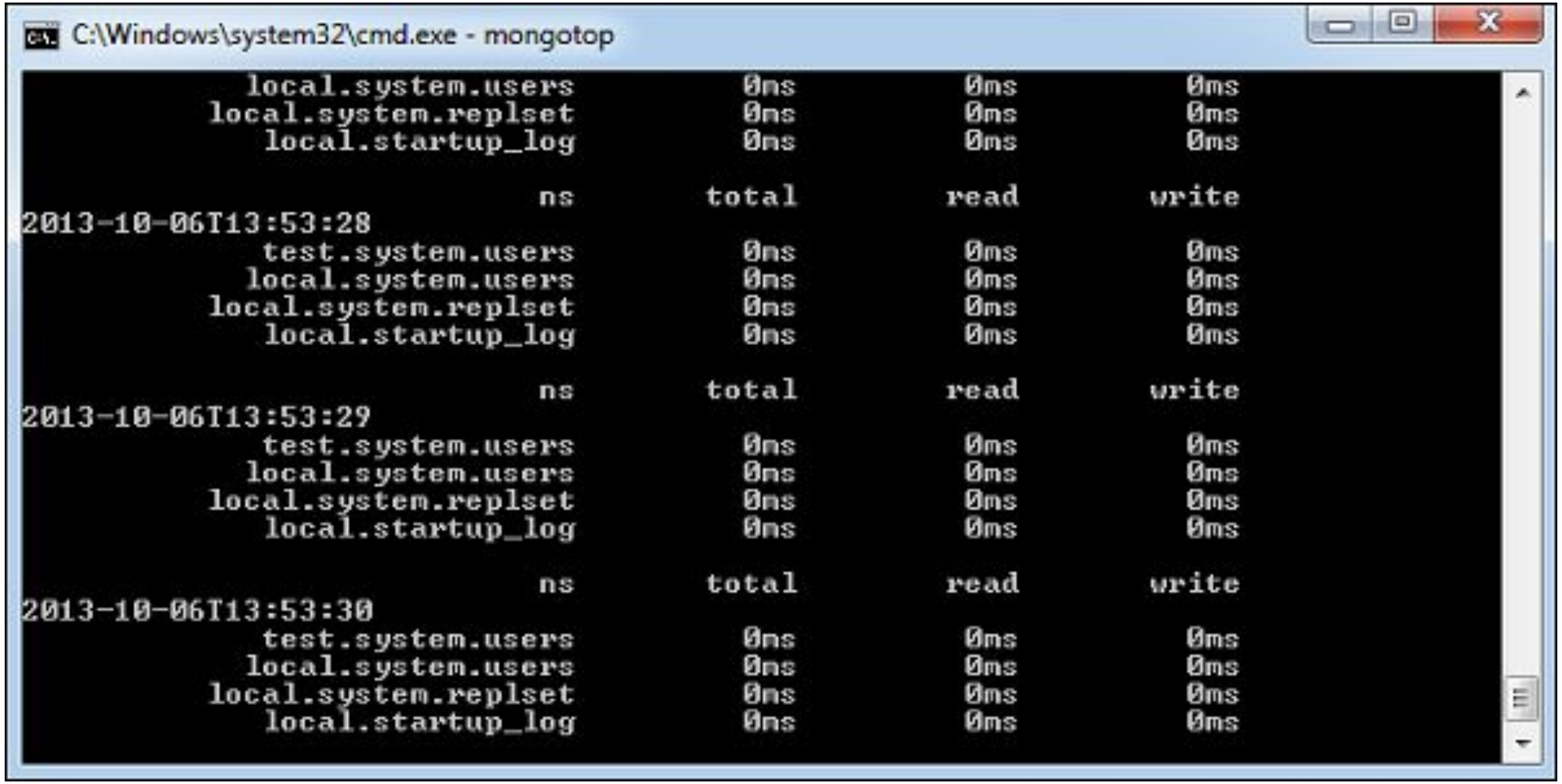
C:\Windows\system32\cmd.exe - mongostat

```
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:24
insert  query update delete getmore command flushes mapped  vsize    res faults
 locked db idx miss %      qr|qw   ar|aw  netIn netOut  conn       time
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:25
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:26
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:27
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:28
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:29
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:30
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:31
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:32
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:33
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:34
insert  query update delete getmore command flushes mapped  vsize    res faults
 locked db idx miss %      qr|qw   ar|aw  netIn netOut  conn       time
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:35
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:36
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:37
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:38
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:39
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:40
    *0     *0     *0     *0       0     2|0       0  14.1g  28.3g    40m      0
local:0.0%          0       0|0     0|0   115b     4k     2   19:20:41
```

MongoDB – Deployment Contd.

- MONGOTOP
- This command tracks and reports the read and write activity of MongoDB instance on a collection basis. By default, **mongotop** returns information in each second, which you can change it accordingly. You should check that this read and write activity matches your application intention, and you’re not firing too many writes to the database at a time, reading too frequently from a disk, or are exceeding your working set size.
- To run the command, start your mongod instance. In another command prompt, go to **bin** directory of your mongodb installation and type **mongotop**.

MongoDB – Deployment Contd.


C:\Windows\system32\cmd.exe - mongotop

```
            local.system.users           0ms           0ms           0ms
          local.system.replset           0ms           0ms           0ms
             local.startup_log           0ms           0ms           0ms

                            ns         total          read         write
2013-10-06T13:53:28
             test.system.users           0ms           0ms           0ms
            local.system.users           0ms           0ms           0ms
          local.system.replset           0ms           0ms           0ms
             local.startup_log           0ms           0ms           0ms

                            ns         total          read         write
2013-10-06T13:53:29
             test.system.users           0ms           0ms           0ms
            local.system.users           0ms           0ms           0ms
          local.system.replset           0ms           0ms           0ms
             local.startup_log           0ms           0ms           0ms

                            ns         total          read         write
2013-10-06T13:53:30
             test.system.users           0ms           0ms           0ms
            local.system.users           0ms           0ms           0ms
          local.system.replset           0ms           0ms           0ms
             local.startup_log           0ms           0ms           0ms
```

MongoDB - Relationships

- Relationships in MongoDB represent how various documents are logically related to each other. Relationships can be modeled via Embedded and Referenced approaches. Such relationships can be either 1:1, 1:N, N:1 or N:N.

- Let us consider the case of storing addresses for users. So, one user can have multiple addresses making this a 1:N relationship.

MongoDB – Relationships Contd.

- Following is the sample document structure of user document −
- {
-   "_id":ObjectId("52ffc33cd85242f436000001"),
-   "name": "Tom Hanks",
-   "contact": "987654321",
-   "dob": "01-01-1991"
- }
- Following is the sample document structure of address document −
- {
-   "_id":ObjectId("52ffc4a5d85242602e000000"),
-   "building": "22 A, Indiana Apt",
-   "pincode": 123456,
-   "city": "Los Angeles",
-   "state": "California"
- } .

MongoDB – Relationships Contd.

- Modeling Embedded Relationships
- In the embedded approach, we will embed the address document inside the user document.

```
•  {
•     "_id":ObjectId("52ffc33cd85242f436000001"),
•     "contact": "987654321",
•     "dob": "01-01-1991",
•     "name": "Tom Benzamin",
•     "address": [
•        {
•           "building": "22 A, Indiana Apt",
•           "pincode": 123456,
•           "city": "Los Angeles",
•           "state": "California"
•        },
•        {
•           "building": "170 A, Acropolis Apt",
•           "pincode": 456789,
•           "city": "Chicago",
•           "state": "Illinois"
•        }
•     ]
•  }
```

MongoDB – Relationships Contd.

- This approach maintains all the related data in a single document, which makes it easy to retrieve and maintain. The whole document can be retrieved in a single query such as −

- >db.users.findOne({"name":"Tom Benzamin"},{"address":1})
- Note that in the above query, db and users are the database and collection respectively.}

MongoDB – Relationships Contd.

- The drawback is that if the embedded document keeps on growing too much in size, it can impact the read/write performance.

- Modeling Referenced Relationships
- This is the approach of designing normalized relationship. In this approach, both the user and address documents will be maintained separately but the user document will contain a field that will reference the address document's id field.

MongoDB – Relationships Contd.

- {
- "_id":ObjectId("52ffc33cd85242f436000001"),
- "contact": "987654321",
- "dob": "01-01-1991",
- "name": "Tom Benzamin",
- "address_ids": [
-   ObjectId("52ffc4a5d85242602e000000"),
-   ObjectId("52ffc4a5d85242602e000001")
-]
- }

MongoDB – Relationships Contd.

- As shown above, the user document contains the array field address_ids which contains ObjectIds of corresponding addresses. Using these ObjectIds, we can query the address documents and get address details from there. With this approach, we will need two queries: first to fetch the address_ids fields from user document and second to fetch these addresses from address collection.

MongoDB – Relationships Contd.

- >var result = db.users.findOne({"name":"Tom Benzamin"},{"address_ids":1})
- >var addresses = db.address.find({"_id":{"$in":result["address_ids"]}})

MongoDB - Database References

- As seen in the last chapter of MongoDB relationships, to implement a normalized database structure in MongoDB, we use the concept of **Referenced Relationships** also referred to as **Manual References** in which we manually store the referenced document's id inside other document. However, in cases where a document contains references from different collections, we can use **MongoDB DBRefs**.

DBRefs vs Manual References

- As an example scenario, where we would use DBRefs instead of manual references, consider a database where we are storing different types of addresses (home, office, mailing, etc.) in different collections (address_home, address_office, address_mailing, etc). Now, when a **user** collection's document references an address, it also needs to specify which collection to look into based on the address type. In such scenarios where a document references documents from many collections, we should use DBRefs.

Using DBRefs

- There are three fields in DBRefs −
  - **$ref** − This field specifies the collection of the referenced document
  - **$id** − This field specifies the _id field of the referenced document
  - **$db** − This is an optional field and contains the name of the database in which the referenced document lies

Using DBRefs Contd.

- Consider a sample user document having DBRef field address as shown in the code snippet −

```
{
  "_id":ObjectId("53402597d852426020000002"),
  "address": {
  "$ref": "address_home",
  "$id": ObjectId("534009e4d852427820000002"),
  "$db": "tutorialspoint"},
  "contact": "987654321",
  "dob": "01-01-1991",
  "name": "Tom Benzamin"
}
```

Using DBRefs Contd.

- The address DBRef field here specifies that the referenced address document lies in address_home collection under tutorialspoint database and has an id of 534009e4d852427820000002.
- The following code dynamically looks in the collection specified by $ref parameter (address_home in our case) for a document with id as specified by $id parameter in DBRef.

Using DBRefs Contd.

```
>var user = db.users.findOne({"name":"Tom Benzamin"})
>var dbRef = user.address
>db[dbRef.$ref].findOne({"_id":(dbRef.$id)})
```

- The above code returns the following address document present in address_home collection −

```
{
  "_id" : ObjectId("534009e4d852427820000002"),
  "building" : "22 A, Indiana Apt",
  "pincode" : 123456,
  "city" : "Los Angeles",
  "state" : "California"
}
```

MongoDB - Covered Queries

- What is a Covered Query?
- As per the official MongoDB documentation, a covered query is a query in which −
  - All the fields in the query are part of an index.
  - All the fields returned in the query are in the same index.
- Since all the fields present in the query are part of an index, MongoDB matches the query conditions and returns the result using the same index without actually looking inside the documents. Since indexes are present in RAM, fetching data from indexes is much faster as compared to fetching data by scanning documents.

MongoDB - Covered Queries

- Using Covered Queries
- To test covered queries, consider the following document in the users collection −

- {
-    "_id": ObjectId("53402597d852426020000002"),
-    "contact": "987654321",
-    "dob": "01-01-1991",
-    "gender": "M",
-    "name": "Tom Benzamin",
-    "user_name": "tombenzamin"
- }

MongoDB - Covered Queries

- We will first create a compound index for the users collection on the fields gender and user_name using the following query −

- >db.users.createIndex({gender:1,user_name:1})
- Now, this index will cover the following query −

- >db.users.find({gender:"M"},{user_name:1,_id:0})
- That is to say that for the above query, MongoDB would not go looking into database documents. Instead it would fetch the required data from indexed data which is very fast.

MongoDB - Covered Queries

- Since our index does not include _id field, we have explicitly excluded it from result set of our query, as MongoDB by default returns _id field in every query. So the following query would not have been covered inside the index created above −

- >db.users.find({gender:"M"},{user_name:1})
- Lastly, remember that an index cannot cover a query if −
  - Any of the indexed fields is an array
  - Any of the indexed fields is a subdocument

MongoDB - Analyzing Queries

- Analyzing queries is a very important aspect of measuring how effective the database and indexing design is. We will learn about the frequently used $explain and $hint queries.

- Using $explain
- The $explain operator provides information on the query, indexes used in a query and other statistics. It is very useful when analyzing how well your indexes are optimized.

MongoDB - Analyzing Queries Contd.

- In the last chapter, we had already created an index for the users collection on fields gender and user_name using the following query –

- >db.users.ensureIndex({gender:1,user_name:1})
- We will now use $explain on the following query –
- >db.users.find({gender:"M"},{user_name:1,_id:0}).explain()
- The above explain() query returns the following analyzed result –

MongoDB - Analyzing Queries Contd.

```
• {
•   "cursor" : "BtreeCursor gender_1_user_name_1",
•   "isMultiKey" : false,
•   "n" : 1,
•   "nscannedObjects" : 0,
•   "nscanned" : 1,
•   "nscannedObjectsAllPlans" : 0,
•   "nscannedAllPlans" : 1,
•   "scanAndOrder" : false,
•   "indexOnly" : true,
•   "nYields" : 0,
•   "nChunkSkips" : 0,
•   "millis" : 0,
•   "indexBounds" : {
•     "gender" : [
```

MongoDB - Analyzing Queries Contd.

```
•            [
•               "M",
•               "M"
•            ]
•         ],
•         "user_name" : [
•            [
•               {
•                  "$minElement" : 1
•               },
•               {
•                  "$maxElement" : 1
•               }
•            ]
•         ]
•      }
•   }
```

MongoDB - Analyzing Queries Contd.

- We will now look at the fields in this result set −
- The true value of indexOnly indicates that this query has used indexing.
- The cursor field specifies the type of cursor used. BTreeCursor type indicates that an index was used and also gives the name of the index used. BasicCursor indicates that a full scan was made without using any indexes.
- In indicates the number of documents matching returned.
- InscannedObjects indicates the total number of documents scanned.
- Inscanned indicates the total number of documents or index entries scanned.

MongoDB - Analyzing Queries Contd.

- Using $hint
- The $hint operator forces the query optimizer to use the specified index to run a query. This is particularly useful when you want to test performance of a query with different indexes. For example, the following query specifies the index on fields gender and user_name to be used for this query −

- >db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender:1,user_name:1})
- To analyze the above query using $explain −

- >db.users.find({gender:"M"},{user_name:1,_id:0}).hint({gender:1,user_name:1}).e
  xplai

MongoDB - Atomic Operations

- MongoDB does not support **multi-document atomic transactions**. However, it does provide atomic operations on a single document. So if a document has hundred fields the update statement will either update all the fields or none, hence maintaining atomicity at the document-level.
- Model Data for Atomic Operations
- The recommended approach to maintain atomicity would be to keep all the related information, which is frequently updated together in a single document using **embedded documents**. This would make sure that all the updates for a single document are atomic.

MongoDB - Atomic Operations Contd.

- Consider the following products document −
- {
-   "_id":1,
-   "product_name": "Samsung S3",
-   "category": "mobiles",
-   "product_total": 5,
-   "product_available": 3,
-   "product_bought_by": [
-     {
-       "customer": "john",
-       "date": "7-Jan-2014"
-     },
-     {
-       "customer": "mark",
-       "date": "8-Jan-2014"
-     }
-   ]
- }

MongoDB - Atomic Operations Contd.

- In this document, we have embedded the information of the customer who buys the product in the product_bought_by field. Now, whenever a new customer buys the product, we will first check if the product is still available using product_available field. If available, we will reduce the value of product_available field as well as insert the new customer's embedded document in the product_bought_by field. We will use findAndModify command for this functionality because it searches and updates the document in the same go.

```
• >db.products.findAndModify({
•   query:{_id:2,product_available:{$gt:0}},
•   update:{
•      $inc:{product_available:-1},
•      $push:{product_bought_by:{customer:"rob",date:"9-Jan-2014"}}
•   }
• })
```

MongoDB - Atomic Operations Contd.

- Our approach of embedded document and using findAndModify query makes sure that the product purchase information is updated only if it the product is available. And the whole of this transaction being in the same query, is atomic.
- In contrast to this, consider the scenario where we may have kept the product availability and the information on who has bought the product, separately. In this case, we will first check if the product is available using the first query. Then in the second query we will update the purchase information. However, it is possible that between the executions of these two queries, some other user has purchased the product and it is no more available. Without knowing this, our second query will update the purchase information based on the result of our first query. This will make the database inconsistent because we have sold a product which is not available

MongoDB - Advanced Indexing

- Consider the following document of the users collection −

```
{
  "address": {
    "city": "Los Angeles",
    "state": "California",
    "pincode": "123"
  },
  "tags": [
    "music",
    "cricket",
    "blogs"
  ],
  "name": "Tom Benzamin"
}
```

- The above document contains an address sub-document and a tags array.

MongoDB - Advanced Indexing contd.

- Indexing Array Fields
- Suppose we want to search user documents based on the user’s tags. For this, we will create an index on tags array in the collection.

- Creating an index on array in turn creates separate index entries for each of its fields. So in our case when we create an index on tags array, separate indexes will be created for its values music, cricket and blogs

MongoDB - Advanced Indexing contd.

- To create an index on tags array, use the following code −
- >db.users.createIndex({"tags":1})
- After creating the index, we can search on the tags field of the collection like this −
- >db.users.find({tags:"cricket"})
- To verify that proper indexing is used, use the following explain command −
- >db.users.find({tags:"cricket"}).explain()
- The above command resulted in "cursor" : "BtreeCursor tags_1" which confirms that proper indexing is used.

MongoDB - Advanced Indexing contd.

- Indexing Sub-Document Fields
- Suppose that we want to search documents based on city, state and pincode fields. Since all these fields are part of address sub-document field, we will create an index on all the fields of the sub-document.
- For creating an index on all the three fields of the sub-document, use the following code −
- >db.users.createIndex({"address.city":1,"address.state":1,"address.pincode":1})
- Once the index is created, we can search for any of the sub-document fields utilizing this index as follows −
- >db.users.find({"address.city":"Los Angeles"})
- Remember that the query expression has to follow the order of the index specified. So the index created above would support the following queries −
- >db.users.find({"address.city":"Los Angeles","address.state":"California"})

MongoDB - Indexing Limitations

- In this , we will learn about Indexing Limitations and its other components.
- Extra Overhead
- Every index occupies some space as well as causes an overhead on each insert, update and delete. So if you rarely use your collection for read operations, it makes sense not to use indexes.
- RAM Usage
- Since indexes are stored in RAM, you should make sure that the total size of the index does not exceed the RAM limit. If the total size increases the RAM size, it will start deleting some indexes, causing performance loss.

MongoDB - Indexing Limitations Contd.

- Query Limitations
- Indexing can't be used in queries which use –

- Regular expressions or negation operators like $nin, $not, etc.
- Arithmetic operators like $mod, etc.
- $where clause
- Hence, it is always advisable to check the index usage for your queries.

- Index Key Limits
- Starting from version 2.6, MongoDB will not create an index if the value of existing index field exceeds the index key limit.

MongoDB - Indexing Limitations Contd.

- Inserting Documents Exceeding Index Key Limit
- MongoDB will not insert any document into an indexed collection if the indexed field value of this document exceeds the index key limit. Same is the case with mongorestore and mongoimport utilities.

- Maximum Ranges
- A collection cannot have more than 64 indexes.
- The length of the index name cannot be longer than 125 characters.
- A compound index can have maximum 31 fields indexed.

MongoDB – ObjectId

- We have been using MongoDB Object Id in all the previous chapters. In this chapter, we will understand the structure of ObjectId.
- An **ObjectId** is a 12-byte BSON type having the following structure:
    - ⍰ The first 4 bytes representing the seconds since the unix epoch
    - ⍰ The next 3 bytes are the machine identifier
    - ⍰ The next 2 bytes consists of process id
    - ⍰ The last 3 bytes are a random counter value

- MongoDB uses ObjectIds as the default value of **_id** field of each document, which is generated while the creation of any document. The complex combination of ObjectId makes all the _id fields unique.

MongoDB – ObjectId Contd.

- Creating New ObjectId
- To generate a new ObjectId use the following code:
- >newObjectId = ObjectId()
- The above statement returned the following uniquely generated id:
- ObjectId("5349b4ddd2781d08c09890f3")
- Instead of MongoDB generating the ObjectId, you can also provide a 12-byte id:
- >myObjectId = ObjectId("5349b4ddd2781d08c09890f4")

MongoDB – ObjectId Contd.

- Creating Timestamp of a Document
- Since the _id ObjectId by default stores the 4-byte timestamp, in most cases you do not need to store the creation time of any document. You can fetch the creation time of a document using getTimestamp method:
- >ObjectId("5349b4ddd2781d08c09890f4").getTimestamp()
- This will return the creation time of this document in ISO date format:
- ISODate("2014-04-12T21:49:17Z")

MongoDB – ObjectId Contd.

- **Converting ObjectId to String**
- In some cases, you may need the value of ObjectId in a string format. To convert the ObjectId in string, use the following code:
- >newObjectId.str
- The above code will return the string format of the Guid:
- 5349b4ddd2781d08c09890f3

MongoDB - Map Reduce

- As per the MongoDB documentation, Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results. MongoDB uses mapReduce command for map-reduce operations. MapReduce is generally used for processing large data sets.
- MapReduce Command
- Following is the syntax of the basic mapReduce command −
- >db.collection.mapReduce(
- function() {emit(key,value);}, //map function
- function(key,values) {return reduceFunction}, { //reduce function
- out: collection,
- query: document,
- sort: document,
- limit: number
- }
-)

MongoDB - Map Reduce Contd.

- The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.
- In the above syntax -
- □ map is a javascript function that maps a value with a key and emits a key-value pair
- □ reduce is a javascript function that reduces or groups all the documents having the same key
- □ out specifies the location of the map-reduce query result
- □ query specifies the optional selection criteria for selecting documents
- □ sort specifies the optional sort criteria
- □ limit specifies the optional maximum number of documents to be returned

MongoDB - Map Reduce Contd.

- Using MapReduce
- Consider the following document structure storing user posts. The document stores user_name of the user and the status of post.
- {
- "post_text": "tutorialspoint is an awesome website for tutorials",
- "user_name": "mark",
- "status":"active"
- }
- Now, we will use a mapReduce function on our posts collection to select all the active posts, group them on the basis of user_name and then count the number of posts by each user using the following code −
- >db.posts.mapReduce(
- function() { emit(this.user_id,1); },
- function(key, values) {return Array.sum(values)}, {
- query:{status:"active"},
- out:"post_total"
- }
-)

MongoDB - Map Reduce Contd.

- The above mapReduce query outputs the following result −
- {
- "result" : "post_total",
- "timeMillis" : 9,
- "counts" : {
- "input" : 4,
- "emit" : 4,
- "reduce" : 2,
- "output" : 2
- },
- "ok" : 1,
- }
- The result shows that a total of 4 documents matched the query (status:"active"), the map function emitted 4 documents with key-value pairs and finally the reduce function grouped mapped documents having the same keys into 2.

MongoDB - Map Reduce Contd.

- To see the result of this mapReduce query, use the find operator −
- >db.posts.mapReduce(
- function() { emit(this.user_id,1); },
- function(key, values) {return Array.sum(values)}, {
- query:{status:"active"},
- out:"post_total"
- }
-).find()
- The above query gives the following result which indicates that both users tom and mark have two posts in active states −
- { "_id" : "tom", "value" : 2 }
- { "_id" : "mark", "value" : 2 }
- In a similar manner, MapReduce queries can be used to construct large complex aggregation queries. The use of custom Javascript functions make use of MapReduce which is very flexible and powerful.

MongoDB - Text Search

- Starting from version 2.4, MongoDB started supporting text indexes to search inside string content. The Text Search uses stemming techniques to look for specified words in the string fields by dropping stemming stop words like a, an, the, etc. At present, MongoDB supports around 15 languages.
- Enabling Text Search
- Initially, Text Search was an experimental feature but starting from version 2.6, the configuration is enabled by default. But if you are using the previous version of MongoDB, you have to enable text search with the following code:
- `>db.adminCommand({setParameter:true,textSearchEnabled:true})`

MongoDB - Text Search Contd.

- Creating Text Index
- Consider the following document under posts collection containing the post text and its tags:
- {
- "post_text": "enjoy the mongodb articles on tutorialspoint",
- "tags": [
- "mongodb",
- "tutorialspoint"
-]
- }
- We will create a text index on post_text field so that we can search inside our posts' text:
- >db.posts.ensureIndex({post_text:"text"})

MongoDB - Text Search Contd.

- Using Text Index
- Now that we have created the text index on post_text field, we will search for all the posts having the word tutorialspoint in their text.
- >db.posts.find({$text:{$search:"tutorialspoint"}})
- The above command returned the following result documents having the word tutorialspoint in their post text:
- {
- "_id" : ObjectId("53493d14d852429c10000002"),
- "post_text" : "enjoy the mongodb articles on tutorialspoint",
- "tags" : ["mongodb", "tutorialspoint"]
- }
- {
- "_id" : ObjectId("53493d1fd852429c10000003"),
- "post_text" : "writing tutorials on mongodb",
- "tags" : ["mongodb", "tutorial"]
- }

MongoDB - Text Search Contd.

- If you are using old versions of MongoDB, you have to use the following command:
- >db.posts.runCommand("text",{search:" tutorialspoint "})
- Using Text Search highly improves the search efficiency as compared to normal search.
- Deleting Text Index
- To delete an existing text index, first find the name of index using the following query:
- >db.posts.getIndexes()
- After getting the name of your index from above query, run the following command. Here, post_text_text is the name of the index.
- >db.posts.dropIndex("post_text_text")

Mongodb – Regular Expressions

- Regular Expressions are frequently used in all languages to search for a pattern or word in any string. MongoDB also provides functionality of regular expression for string pattern matching using the $regex operator. MongoDB uses PCRE (Perl Compatible Regular Expression) as regular expression language.

- Unlike text search, we do not need to do any configuration or command to use regular expressions.

Mongodb – Regular Expressions Contd.

- Consider the following document structure under posts collection containing the post text and its tags –

```
{
   "post_text": "enjoy the mongodb articles on tutorialspoint",
   "tags": [
      "mongodb",
      "tutorialspoint"
   ]
}
```

Using regex Expression

- The following regex query searches for all the posts containing string tutorialspoint in it −

```
>db.posts.find({post_text:{$regex:"tutorialspoint"}})
```

- The same query can also be written as −

```
>db.posts.find({post_text:/tutorialspoint/})
```

- Using regex Expression with Case Insensitive
- To make the search case insensitive, we use the $options parameter with value $i. The following command will look for strings having the word tutorialspoint, irrespective of smaller or capital case −

```
>db.posts.find({post_text:{$regex:"tutorialspoint",$options:"$i"}})
```

- One of the results returned from this query is the following document which contains the word tutorialspoint in different cases −

```
{
  "_id" : ObjectId("53493d37d852429c10000004"),
  "post_text" : "hey! this is my post on TutorialsPoint",
  "tags" : [ "tutorialspoint" ]
}
```

Using regex for Array Elements

- We can also use the concept of regex on array field. This is particularly very important when we implement the functionality of tags. So, if you want to search for all the posts having tags beginning from the word tutorial (either tutorial or tutorials or tutorialpoint or tutorialphp), you can use the following code −

```
>db.posts.find({tags:{$regex:"tutorial"}})
```

Optimizing Regular Expression Queries

- If the document fields are indexed, the query will use make use of indexed values to match the regular expression. This makes the search very fast as compared to the regular expression scanning the whole collection.
- If the regular expression is a prefix expression, all the matches are meant to start with a certain string characters. For e.g., if the regex expression is ^tut, then the query has to search for only those strings that begin with tut.

Working with RockMongo

- RockMongo is a MongoDB administration tool using which you can manage your server, databases, collections, documents, indexes, and a lot more. It provides a very user-friendly way for reading, writing, and creating documents. It is similar to PHPMyAdmin tool for PHP and MySQL.

- Downloading RockMongo
- You can download the latest version of RockMongo from here: http://rockmongo.com/downloads

Installing RockMongo

- Once downloaded, you can unzip the package in your server root folder and rename the extracted folder to rockmongo. Open any web browser and access the index.php page from the folder rockmongo. Enter admin/admin as username/password respectively.

Working with RockMongo

- We will now be looking at some basic operations that you can perform with RockMongo.
- Creating New Database
  - To create a new database, click Databases tab. Click Create New Database. On the next screen, provide the name of the new database and click on Create. You will see a new database getting added in the left panel.

Working with RockMongo Contd.

- Creating New Collection
- To create a new collection inside a database, click on that database from the left panel. Click on the New Collection link on top. Provide the required name of the collection. Do not worry about the other fields of Is Capped, Size and Max. Click on Create. A new collection will be created and you will be able to see it in the left panel.

Working with RockMongo Contd.

- Creating New Collection
- To create a new collection inside a database, click on that database from the left panel. Click on the New Collection link on top. Provide the required name of the collection. Do not worry about the other fields of Is Capped, Size and Max. Click on Create. A new collection will be created and you will be able to see it in the left panel.

Working with RockMongo Contd.

- Creating New Document
  - To create a new document, click on the collection under which you want to add documents. When you click on a collection, you will be able to see all the documents within that collection listed there. To create a new document, click on the Insert link at the top. You can enter the document's data either in JSON or array format and click on Save.

Working with RockMongo Contd.

- Export/Import Data
  - To import/export data of any collection, click on that collection and then click on Export/Import link on the top panel. Follow the next instructions to export your data in a zip format and then import the same zip file to import back data.

MongoDB - GridFS

- GridFS is the MongoDB specification for storing and retrieving large files such as images, audio files, video files, etc. It is kind of a file system to store files but its data is stored within MongoDB collections. GridFS has the capability to store files even greater than its document size limit of 16MB.

MongoDB – GridFS Contd.

- GridFS divides a file into chunks and stores each chunk of data in a separate document, each of maximum size 255k.

- GridFS by default uses two collections fs.files and fs.chunks to store the file's metadata and the chunks. Each chunk is identified by its unique _id ObjectId field. The fs.files severs as a parent document. The files_id field in the fs.chunks document links the chunk to its parent.

MongoDB – GridFS Contd.

- Following is a sample document of fs.files collection −

```
{
   "filename": "test.txt",
   "chunkSize": NumberInt(261120),
   "uploadDate": ISODate("2014-04-13T11:32:33.557Z"),
   "md5": "7b762939321e146569b07f72c62cca4f",
   "length": NumberInt(646)
}
```

MongoDB – GridFS Contd.

- The document specifies the file name, chunk size, uploaded date, and length.

- Following is a sample document of fs.chunks document −

```
{
  "files_id": ObjectId("534a75d19f54bfec8a2fe44b"),
  "n": NumberInt(0),
  "data": "Mongo Binary Data"
}
```

MongoDB – GridFS Contd.

- The document specifies the file name, chunk size, uploaded date, and length.
- Following is a sample document of fs.chunks document −

```
{
  "files_id": ObjectId("534a75d19f54bfec8a2fe44b"),
  "n": NumberInt(0),
  "data": "Mongo Binary Data"
}
```

MongoDB – GridFS Contd.

- Adding Files to GridFS
- Now, we will store an mp3 file using GridFS using the put command. For this, we will use the mongofiles.exe utility present in the bin folder of the MongoDB installation folder.

- Open your command prompt, navigate to the mongofiles.exe in the bin folder of MongoDB installation folder and type the following code

```
>mongofiles.exe -d gridfs put song.mp3
```

MongoDB – GridFS Contd.

- Here, gridfs is the name of the database in which the file will be stored. If the database is not present, MongoDB will automatically create a new document on the fly. Song.mp3 is the name of the file uploaded. To see the file's document in database, you can use find query −

```
>db.fs.files.find()
```

- The above command returned the following document −

```
{
  _id: ObjectId('534a811bf8b4aa4d33fdf94d'),
  filename: "song.mp3",
  chunkSize: 261120,
  uploadDate: new Date(1397391643474), md5:
"e4f53379c909f7bed2e9d631e15c1c41",
  length: 10401959
}
```

MongoDB – GridFS Contd.

- We can also see all the chunks present in fs.chunks collection related to the stored file with the following code, using the document id returned in the previous query −

```
>db.fs.chunks.find({files_id:ObjectId('534a811bf8b4aa4d33fdf94d')})
```

- In my case, the query returned 40 documents meaning that the whole mp3 document was divided in 40 chunks of data.

MongoDB - Capped Collections

- Capped collections are fixed-size circular collections that follow the insertion order to support high performance for create, read, and delete operations. By circular, it means that when the fixed size allocated to the collection is exhausted, it will start deleting the oldest document in the collection without providing any explicit commands.

MongoDB - Capped Collections Contd.

- Capped collections restrict updates to the documents if the update results in increased document size. Since capped collections store documents in the order of the disk storage, it ensures that the document size does not increase the size allocated on the disk. Capped collections are best for storing log information, cache data, or any other high volume data.

MongoDB - Capped Collections Contd.

- Creating Capped Collection
- To create a capped collection, we use the normal createCollection command but with capped option as true and specifying the maximum size of collection in bytes.

```
>db.createCollection("cappedLogCollection",{capped:true,size:10000})
```

MongoDB - Capped Collections Contd.

- In addition to collection size, we can also limit the number of documents in the collection using the max parameter −

```
>db.createCollection("cappedLogCollection",{capped:true,size:10000,max:1000
})
```

- If you want to check whether a collection is capped or not, use the following isCapped command −

```
>db.cappedLogCollection.isCapped()
```

MongoDB - Capped Collections Contd.

- If there is an existing collection which you are planning to convert to capped, you can do it with the following code −

- >db.runCommand({"convertToCapped":"posts",size:10000})
- This code would convert our existing collection posts to a capped collection.

MongoDB - Capped Collections Contd.

- Querying Capped Collection
- By default, a find query on a capped collection will display results in insertion order. But if you want the documents to be retrieved in reverse order, use the sort command as shown in the following code –

```
>db.cappedLogCollection.find().sort({$natural:-1})
```

MongoDB - Capped Collections Contd.

- There are few other important points regarding capped collections worth knowing
  - We cannot delete documents from a capped collection.
  - There are no default indexes present in a capped collection, not even on _id field.
  - While inserting a new document, MongoDB does not have to actually look for a place to accommodate new document on the disk. It can blindly insert the new document at the tail of the collection. This makes insert operations in capped collections very fast.
  - Similarly, while reading documents MongoDB returns the documents in the same order as present on disk. This makes the read operation very fast.

MongoDB - Auto-Increment Sequence

- MongoDB does not have out-of-the-box auto-increment functionality, like SQL databases. By default, it uses the 12-byte ObjectId for the _id field as the primary key to uniquely identify the documents. However, there may be scenarios where we may want the _id field to have some auto-incremented value other than the ObjectId.
- Since this is not a default feature in MongoDB, we will programmatically achieve this functionality by using a counters collection as suggested by the MongoDB documentation.

MongoDB - Auto-Increment Sequence Contd.

- Using Counter Collection
- Consider the following products document. We want the _id field to be an auto-incremented integer sequence starting from 1,2,3,4 upto n.

```
{
  "_id":1,
  "product_name": "Apple iPhone",
  "category": "mobiles"
}
```

- For this, create a counters collection, which will keep track of the last sequence value for all the sequence fields.

```
>db.createCollection("counters")
```

MongoDB - Auto-Increment Sequence Contd.

- Using Counter Collection
- Consider the following products document. We want the _id field to be an auto-incremented integer sequence starting from 1,2,3,4 upto n.

```
{
  "_id":1,
  "product_name": "Apple iPhone",
  "category": "mobiles"
}
```

- For this, create a counters collection, which will keep track of the last sequence value for all the sequence fields.

```
>db.createCollection("counters")
```

MongoDB - Auto-Increment Sequence Contd.

- Now, we will insert the following document in the counters collection with productid as its key −

```
{
  "_id":"productid",
  "sequence_value": 0
}
```

- The field sequence_value keeps track of the last value of the sequence.
- Use the following code to insert this sequence document in the counters collection −

```
>db.counters.insert({_id:"productid",sequence_value:0})
```

MongoDB - Auto-Increment Sequence Contd.

- Using the Javascript Function
- We will now use the function getNextSequenceValue while creating a new document and assigning the returned sequence value as document's _id field.
- Insert two sample documents using the following code −

```
>db.products.insert({
  "_id":getNextSequenceValue("productid"),
  "product_name":"Apple iPhone",
  "category":"mobiles"
})
>db.products.insert({
  "_id":getNextSequenceValue("productid"),
  "product_name":"Samsung S3",
  "category":"mobiles"
})
```

- As you can see, we have used the getNextSequenceValue function to set value for the _id field.

MongoDB!COMPONENTS

- Welcome to MongoDB!COMPONENTS
  - bin/mongod - The database process.
  - bin/mongos - Sharding controller.
  - bin/mongo - The database shell (uses interactive javascript).
- UTILITIES
  - bin/mongodump   - MongoDB dump tool - for backups, snapshots, etc..
  - bin/mongorestore   - MongoDB restore a dump
  - bin/mongoexport   - Export a single collection to test (JSON, CSV)
  - bin/mongoimport   - Import from JSON or CSV
  - bin/mongofiles   - Utility for putting and getting files from MongoDB GridFS
  - bin/mongostat   - Show performance statisticsRUNNING
  - Bin/mongotop   - show actIvity details