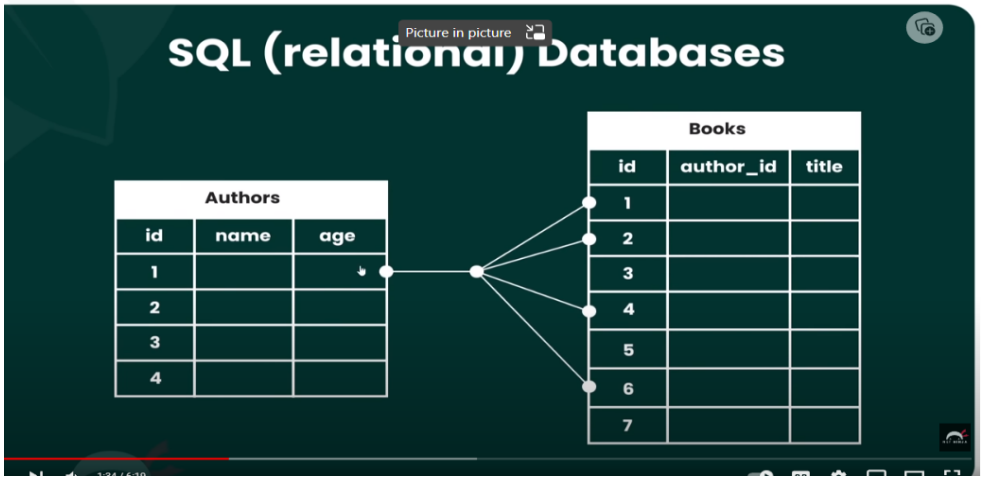
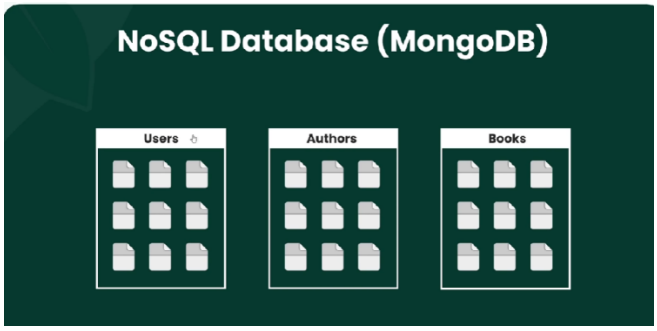
MongoDB Tutorial :

- **NoSQL database**
- **Unstructred Datbase**
- **Fix Schema (Structure)**

In SQL database data is stored in table in terms of rows and column.In SQL database data is related with each other.



In MongoDB ,Instead of rows and columns there are collections and documents.document inside collection is in form of key value pair.



- **It allow to store nested document inside other document.**

- **We can host mongodb database on cloud using its ATLAS service.**

Collection and Documents :

- Each collection contain Multiple documents.
- Document is individual entity in collection which contain data as key value pair and each documents have its unique id for identification.
- Document is similar to json object .
- Document which is present inside other document called nested document.

e.g

{

“title”:”Rich and poor” ,

“Author”: {

“name” :” xyx” ,

“email”:”xyz@gmail.com”

},

“tags”:[“vdo games”,”reviews”],

“upvotes”:20

Id:Objectid(123…)

}

NOSQL Database Types :

1)Key Value pair : Amazon Dynamodb,Oracle NoSQL

2)Document Based : MongoDB,CouchDB

3)Graph Based : Infogrid.

4)Column Based : Cassendra,HBase.

- **Advantages :**

  1)Schema less: MongoDB is a document database in which onecollection holds different documents. Number of fields, content and size of the document can differ from one document to another.

  2)Fast performance

  3)Big data Capability.

  4)Ease of scale-out: MongoDB is easy to scale.

- **Why Mongodb :**
  - Document based as JSON
  - Index on Any Attribute
  - Fast and Flexible .
- **Where To Use :** Big data ,User data management and Social Infrastructure

RDBMS	MongoDB
It is a relational database.	It is a non-relational and document-oriented database.
Not suitable for hierarchical data storage.	Suitable for hierarchical data storage.
It is vertically scalable i.e increasing RAM.	It is horizontally scalable i.e we can add more servers.
It has a predefined schema.	It has a dynamic schema.
It is quite vulnerable to SQL injection.	It is not affected by SQL injection.
It centers around ACID properties (Atomicity, Consistency, Isolation, and Durability).	It centers around the CAP theorem (Consistency, Availability, and Partition tolerance).
It is row-based.	It is document-based.
It is slower in comparison with MongoDB.	It is almost 100 times faster than RDBMS.
Supports complex joins.	No support for complex joins.
It is column-based.	It is field-based.
It does not provide JavaScript client for querying.	It provides a JavaScript client for querying.
It supports SQL query language only.	It supports JSON query language along with SQL.

Last Updated : 22 Feb, 2023

MongoDB	MySQL
MongoDB is an open-source database developed by MongoDB, Inc. MongoDB stores data in JSON-like documents that can vary in structure. It is a popular NoSQL database.	MySQL is a popular open-source relational database management system (RDBMS) that is developed, distributed and supported by Oracle Corporation.

How Data is Stored?

MongoDB	MySQL
In MongoDB, each individual records are stored as 'documents'.	In MySQL, each individual records are stored as 'rows' in a table.

HIERARCHICAL UPPER OF A RECORD

MongoDB	MySQL
Documents belonging to a particular class or group as stored in a 'collection'. Example: collection of users.	A 'table' is used to store rows (records) of similar type.

SQL or NoSQL

MongoDB	MySQL
MongoDB is what is called a NoSQL database. This means that pre-defined structure for the incoming data can be defined and adhered to but also, if required different documents in a collection can have different structures. It has a dynamic schema.	MySQL as the name suggests uses Structured Query Language (SQL) for database access. The schema can not be changed. The inputs following the given schema are only entered.

MongoDB Cursor :

In MongoDB, when the find() method is used to find the documents present in the given collection, then this method returned a pointer which will points to the documents of the collection, now this pointer is known as cursor.

```
Microsoft Windows [Version 10.0.22000.2057]
(c) Microsoft Corporation. All rights reserved.

C:\Users\omkar>monfgosh
'monfgosh' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\omkar>mongosh
Current Mongosh Log ID: 64ca51f1a9c25612e6e73231
Connecting to:          mongodb://127.0.0.1:27017/?directConnect
Using MongoDB:          6.0.8
Using Mongosh:          1.10.2

For mongosh info see: https://docs.mongodb.com/mongodb-shell/

------
   The server generated these startup warnings when booting
   2023-08-02T18:16:47.879+05:30: Access control is not enabled
------

Warning: Found ~/.mongorc.js, but not ~/.mongoshrc.js. ~/.mongor
  You may want to copy or rename ~/.mongorc.js to ~/.mongoshrc.j
test> show dbs
admin   40.00 KiB
config  72.00 KiB
db      36.00 KiB
local   72.00 KiB
mydb    48.00 KiB
test> use db
switched to db db
db> db.createCollection("Student")
{ ok: 1 }
db> show collections
Data
Student
```

CRUD Operations :

MongoDB Methods :

1)Insert data into Document :

```
db> db.Student.insert({"id":1,"name":"omkar","age":20})
DeprecationWarning: Collection.insert() is deprecated. Use insertOn
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64ca574da9c25612e6e73232") }
}
```

_2)Display all Data From Document :

```
db> db.Student.find()
[
  {
    _id: ObjectId("64ca574da9c25612e6e73232"),
    id: 1,
    name: 'omkar',
    age: 20
  }
]
```

-Display Requested Data From Document :

```
db> db.Student.find({"id":1})
[
  {
    _id: ObjectId("64ca574da9c25612e6e73232"),
    id: 1,
    name: 'omi',
    age: 20
  }
]
```

-Pretty() : used to format data

```
db> db.Student.find().pretty()
[
  {
    _id: ObjectId("64ca574da9c25612e6e73232"),
    id: 1,
    name: 'omi',
    age: 20
  }
]
```

3)Update Data From Document :

```
db> db.Student.update({"name":"omkar"},{$set:{"name":"omi"}})
DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany,
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

4) Delete data From Document :

```
  _id: ObjectId("64ca574da9c25612e6e73232"),
  id: 1,
  name: 'omi',
  age: 20
},
{ _id: ObjectId("64ca5a19a9c25612e6e73233"), id: 2, name: 'aniket' }

b> db.Student.remove({"id":1})
eprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMa
 acknowledged: true, deletedCount: 1 }
b> db.Student.find()

{ _id: ObjectId("64ca5a19a9c25612e6e73233"), id: 2, name: 'aniket' }
```

-To remove all data in document

```
db> db.Student.remove()
```

5)Sort data in Document :

We use sort method for it where we provide key on which basis we have to sort if we provide 1 it will sort in ascending order and if we provide -1 it will sort it in descending order.

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTi
db> use db
already on db db
db> db.createCollection("Numbers")
{ ok: 1 }
db> db.Numbers.insert({"id":3})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64cbb163e5bbe89fbd7cfd71") }
}
db> db.Numbers.insert({"id":2})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64cbb167e5bbe89fbd7cfd72") }
}
db> db.Numbers.insert({"id":1})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64cbb16ce5bbe89fbd7cfd73") }
}
db> db.Numbers.insert({"id":5})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64cbb171e5bbe89fbd7cfd74") }
}
db> db.Numbers.insert({"id":4})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64cbb174e5bbe89fbd7cfd75") }
}
db> db.Numbers.find()
[
  { _id: ObjectId("64cbb163e5bbe89fbd7cfd71"), id: 3 },
  { _id: ObjectId("64cbb167e5bbe89fbd7cfd72"), id: 2 },
  { _id: ObjectId("64cbb16ce5bbe89fbd7cfd73"), id: 1 },
  { _id: ObjectId("64cbb171e5bbe89fbd7cfd74"), id: 5 },
  { _id: ObjectId("64cbb174e5bbe89fbd7cfd75"), id: 4 }
]
db> db.Numbers.find().sort({"id":1})
[
  { _id: ObjectId("64cbb16ce5bbe89fbd7cfd73"), id: 1 },
  { _id: ObjectId("64cbb167e5bbe89fbd7cfd72"), id: 2 },
  { _id: ObjectId("64cbb163e5bbe89fbd7cfd71"), id: 3 },
  { _id: ObjectId("64cbb174e5bbe89fbd7cfd75"), id: 4 },
  { _id: ObjectId("64cbb171e5bbe89fbd7cfd74"), id: 5 }
]
db> db.Numbers.find().sort({"id":-1})
[
  { _id: ObjectId("64cbb171e5bbe89fbd7cfd74"), id: 5 },
  { _id: ObjectId("64cbb174e5bbe89fbd7cfd75"), id: 4 },
  { _id: ObjectId("64cbb163e5bbe89fbd7cfd71"), id: 3 },
  { _id: ObjectId("64cbb167e5bbe89fbd7cfd72"), id: 2 },
  { _id: ObjectId("64cbb16ce5bbe89fbd7cfd73"), id: 1 }
]
db>
```

☂ **Conditional Operators :** To filter data From Documents .

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

```
db> db.Numbers.find({"id":{$lt:3}}).pretty()
[
  { _id: ObjectId("64cbb167e5bbe89fbd7cfd72"), id: 2 },
  { _id: ObjectId("64cbb16ce5bbe89fbd7cfd73"), id: 1 }
]
db> db.Numbers.find({"id":{$gt:3}})
[
  { _id: ObjectId("64cbb171e5bbe89fbd7cfd74"), id: 5 },
  { _id: ObjectId("64cbb174e5bbe89fbd7cfd75"), id: 4 }
]
db> db.Numbers.find({"id":{$lte:3}}).pretty()
[
  { _id: ObjectId("64cbb163e5bbe89fbd7cfd71"), id: 3 },
  { _id: ObjectId("64cbb167e5bbe89fbd7cfd72"), id: 2 },
  { _id: ObjectId("64cbb16ce5bbe89fbd7cfd73"), id: 1 }
]
db> db.Numbers.find({"id":{$gte:3}})
[
  { _id: ObjectId("64cbb163e5bbe89fbd7cfd71"), id: 3 },
  { _id: ObjectId("64cbb171e5bbe89fbd7cfd74"), id: 5 },
  { _id: ObjectId("64cbb174e5bbe89fbd7cfd75"), id: 4 }
]
db> db.Numbers.find({"id":{$ne:3}})
[
  { _id: ObjectId("64cbb167e5bbe89fbd7cfd72"), id: 2 },
  { _id: ObjectId("64cbb16ce5bbe89fbd7cfd73"), id: 1 },
  { _id: ObjectId("64cbb171e5bbe89fbd7cfd74"), id: 5 },
  { _id: ObjectId("64cbb174e5bbe89fbd7cfd75"), id: 4 }
]
```

AND and OR in MongoDB:

In find() , method if we pass multiple keys separating by comma then it is treated as AND in MongoDB.

```
db> db.createCollection("omkar")
{ ok: 1 }
db> db.omkar.insert({"id":1,"name":"omkar","Qualif":"BE","batch":"2023"})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64cbb558e5bbe89fbd7cfd76") }
}
db> db.omkar.insert({"id":2,"name":"shubz","Qualif":"BE","batch":"2022"})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64cbb568e5bbe89fbd7cfd77") }
}
db> db.Om.find().pretty()

db> db.omkar.find().pretty()
[
  {
    _id: ObjectId("64cbb558e5bbe89fbd7cfd76"),
    id: 1,
    name: 'omkar',
    Qualif: 'BE',
    batch: '2023'
  },
  {
    _id: ObjectId("64cbb568e5bbe89fbd7cfd77"),
    id: 2,
    name: 'shubz',
    Qualif: 'BE',
    batch: '2022'
  }
]
```

Logical Operators :

AND : , treated as AND

```
db> db.omkar.find({"id":1,"batch":"2023"})
[
  {
    _id: ObjectId("64cbb558e5bbe89fbd7cfd76"),
    id: 1,
    name: 'omkar',
    Qualif: 'BE',
    batch: '2023'
  }
]
```

OR :

```
db> db.omkar.find({  $or : [  {"id":1,"batch":"2023"}]})
[
  {
    _id: ObjectId("64cbb558e5bbe89fbd7cfd76"),
    id: 1,
    name: 'omkar',
    Qualif: 'BE',
    batch: '2023'
  }
]
```

NOT :

```
omk> db.stu.find({"Name":{$not :{$eq : "Omkar"}}})
[
  {
    _id: ObjectId("64d480d1dd0ea0075c4a7282"),
    id: 2,
    Name: 'Yuvi',
    Skill: [ 'MariaDb', 'Infogrid' ]
  },
  {
    _id: ObjectId("64d482b8dd0ea0075c4a7283"),
    id: 3,
    Name: 'Yuv',
    Skill: [ 'MariaDb', 'Infogrid', 'MongoDB' ]
  },
  {
    _id: ObjectId("64d48478dd0ea0075c4a7285"),
    id: 2,
    Name: 'Yuvi',
    Skill: [ 'MariaDb', 'Infogrid' ]
  }
]
```

Adding Single & Multiple Documents in collection :

```
MongoshInvalidInputError: [COMMON-10001] 'om' is not a valid argument for "show".
mydb> db.om.find()
[
  {
    _id: ObjectId("64cd0420030c83edd64ae23c"),
    id: 1,
    name: 'omkar',
    age: 30
  }
]
mydb> db.om.insertOne({"id":2,"name":"anu"})
{
  acknowledged: true,
  insertedId: ObjectId("64cd0505aa9eefa489ed7e80")
}
mydb> db.om.insertMany({"id":3,"name":"yuvi"},{"id":4,"Temp":["cool","HOT"]})
MongoInvalidArgumentError: Argument "docs" must be an array of documents
mydb> db.om.insertMany([{"id":3,"name":"yuvi"},{"id":4,"Temp":["cool","HOT"]}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("64cd055aaa9eefa489ed7e81"),
    '1': ObjectId("64cd055aaa9eefa489ed7e82")
  }
}
```

Find() : It show all the documents in collection. And if we specify argument then it help to filter documents like db.om.find({“age”:{ $lt:10}})

findOne() : It provide one record .

```
mydb> db.om.findOne()
{
  _id: ObjectId("64cd0420030c83edd64ae23c"),
  id: 1,
  name: 'omkar',
  age: 30
}
mydb>
```

updateOne() and updateMany() :

```
switched to db mydb
mydb> db.om.find()
[
  {
    _id: ObjectId("64cd0420030c83edd64ae23c"),
    id: 1,
    name: 'omkar',
    age: 30
  },
  { _id: ObjectId("64cd0505aa9eefa489ed7e80"), id: 2, name: 'anu' },
  { _id: ObjectId("64cd055aaa9eefa489ed7e81"), id: 3, name: 'yuvi' },
  {
    _id: ObjectId("64cd055aaa9eefa489ed7e82"),
    id: 4,
    Temp: [ 'cool', 'HOT' ]
  }
]
mydb> db.om.updateOne({"id":1},{$set : { "name" : "omi"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
mydb> db.om.updateMany({},{$inc: {"id":1}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
mydb> db.om.find()
[
  {
    _id: ObjectId("64cd0420030c83edd64ae23c"),
    id: 2,
    name: 'omi',
    age: 30
  },
  { _id: ObjectId("64cd0505aa9eefa489ed7e80"), id: 3, name: 'anu' },
  { _id: ObjectId("64cd055aaa9eefa489ed7e81"), id: 4, name: 'yuvi' },
  {
    _id: ObjectId("64cd055aaa9eefa489ed7e82"),
    id: 5,
    Temp: [ 'cool', 'HOT' ]
  }
```

deleteOne() and deleteMany() :

```
]
mydb> db.om.deleteOne({"id":2})
{ acknowledged: true, deletedCount: 1 }
mydb> db.om.find()
[
  { _id: ObjectId("64cd0505aa9eefa489ed7e80"), id: 3, name: 'anu' },
  { _id: ObjectId("64cd055aaa9eefa489ed7e81"), id: 4, name: 'yuvi' },
  {
    _id: ObjectId("64cd055aaa9eefa489ed7e82"),
    id: 5,
    Temp: [ 'cool', 'HOT' ]
  }
]
mydb> db.om.deleteMany({"id":{$gt : 3}})
{ acknowledged: true, deletedCount: 2 }
mydb> db.om.find()
[ { _id: ObjectId("64cd0505aa9eefa489ed7e80"), id: 3, name: 'anu' } ]
mydb>
```

Indexing in MongoDB :

MongoDB uses indexing in order to make the query processing more efficient. If there is no indexing, then the MongoDB must scan every document in the collection and retrieve only those documents that match the query. Indexes are special data structures that stores some information related to the documents such that it becomes easy for MongoDB to find the right data file.

```
omk> db.dd.createIndex({Age:1})
Age_1
omk> db.dd.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { Age: 1 }, name: 'Age_1' }
]
omk> db.dd.dropIndex({KEY : 1})
MongoshInternalError: can't find index with key: { KEY: 1 }
omk> db.dd.dropIndex({Age : 1})
{ nIndexesWas: 2, ok: 1 }
omk> db.dd.getIndexes()
[ { v: 2, key: { _id: 1 }, name: '_id_' } ]
omk>
```

Types of Index :

1) **Single Field Index :** A single field index means index on a single field of a document. This index is helpful for fetching data in ascending as well as descending order.

```
omk> db.dd.createIndex({Age:1})
Age_1
omk> db.dd.getIndexes()
[
  { v: 2, key: { _id: 1 }, name: '_id_' },
  { v: 2, key: { Age: 1 }, name: 'Age_1' }
]
```

2) **Compound Index :** the compound index is an index where a single index structure holds multiple references.

```
omk> db.dd.find().pretty()
[
  {
    _id: ObjectId("64d31ce121ca320bbbffe0f4"),
    title: 'MongoDB and Python',
    description: 'MongoDB is no SQL database'
  },
  { _id: ObjectId("64d31dd021ca320bbbffe0f9"), Name: 'omkar', Age: 12 },
  { _id: ObjectId("64d31eb1766e40e60cc6b171"), Name: '4', Age: 3 },
  { _id: ObjectId("64d31eb2766e40e60cc6b172"), Name: '2', Age: 3 },
  { _id: ObjectId("64d31eb4766e40e60cc6b173"), Name: '3', Age: 2 },
  { _id: ObjectId("64d31eb4766e40e60cc6b174"), Name: '2', Age: 2 },
  { _id: ObjectId("64d31eb5766e40e60cc6b175"), Name: '2', Age: 2 }
]
omk> db.dd.createIndex({Name:1,Age:1})
Name_1_Age_1
omk>
```

3) **MultiKey Index :** MongoDB uses the multikey indexes to index the values stored in arrays. When we index a field that holds an array value then MongoDB automatically creates a separate index of each and every value present in that array. Using these multikey indexes we can easily find a document that contains an array by matching the items. In MongoDB, you don't need to explicitly specify the multikey index because MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value.

```
omk> db.stu.insert({"id":2,"Name":"Yuvi","Skill":["MariaDb","Infogrid"]})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64d480d1dd0ea0075c4a7282") }
}
omk> db.stu.find().pretty()
[
  {
    _id: ObjectId("64d480b3dd0ea0075c4a7281"),
    id: 1,
    Name: 'Omkar',
    Skill: [ 'MYSQL', 'MongoDB' ]
  },
  {
    _id: ObjectId("64d480d1dd0ea0075c4a7282"),
    id: 2,
    Name: 'Yuvi',
    Skill: [ 'MariaDb', 'Infogrid' ]
  }
]
omk> db.stu.createIndex({Skill:1})
Skill_1
```

4)Text Index : In MongoDB, we can create text indexes using db.collectionName.createIndex() method. So, to index a field that contains either string or an array of string elements, pass a document in the createIndex() method that contains the field and the string literal(i.e., "text"). Using this method you are

allowed to index multiple fields for the text index. Also, a compound index can contain text index key in combination with ascending and descending index key. And if you want to drop a text index, just use the index name.

```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
omk> db.stu.insert({"id":3,"Name":"Yuv","Skill":["MariaDb","Infogrid","MongoDB"]})
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64d482b8dd0ea0075c4a7283") }
}
omk> db.stu.find().pretty()
[
  {
    _id: ObjectId("64d480b3dd0ea0075c4a7281"),
    id: 1,
    Name: 'Omkar',
    Skill: [ 'MYSQL', 'MongoDB' ]
  },
  {
    _id: ObjectId("64d480d1dd0ea0075c4a7282"),
    id: 2,
    Name: 'Yuvi',
    Skill: [ 'MariaDb', 'Infogrid' ]
  },
  {
    _id: ObjectId("64d482b8dd0ea0075c4a7283"),
    id: 3,
    Name: 'Yuv',
    Skill: [ 'MariaDb', 'Infogrid', 'MongoDB' ]
  }
]
omk> db.stu.createIndex({Name:"text"})
Name_text
omk> db.stu.find({$text : {$search :"Yuvi"}})
[
  {
    _id: ObjectId("64d480d1dd0ea0075c4a7282"),
    id: 2,
    Name: 'Yuvi',
    Skill: [ 'MariaDb', 'Infogrid' ]
omk>
```

```
> db.studentsposts.find({$text:{$search: "mongodb"}}).pretty()
{
        "_id" : ObjectId("60193d0d0cf217478ba935ac"),
        "title" : "fun with mongodb",
        "tags" : [
                "mongodb",
                "geeksforgeeks"
        ]
}
{
        "_id" : ObjectId("60193d280cf217478ba935ad"),
        "title" : "fun in learning  mongodb",
        "tags" : [
                "mongodb learning",
                "geeksforgeeks"
        ]
}
>
```

Mongo DB Advance :

1)Import json file to mongoDB Collection : mongoimport --jsonArray --db omk --collection student --file H:\students.json

2)Export data in from MongoDbB Collection to json file:

mongoexport –db omk –collection students –out D:\om.json