**No-SQL** : No-SQL stands for Not Only SQL.No-SQL is a not relational based database.No-SQL databases does not follow the rules of RDMS and No-SQL databases does not use SQL to query the data.

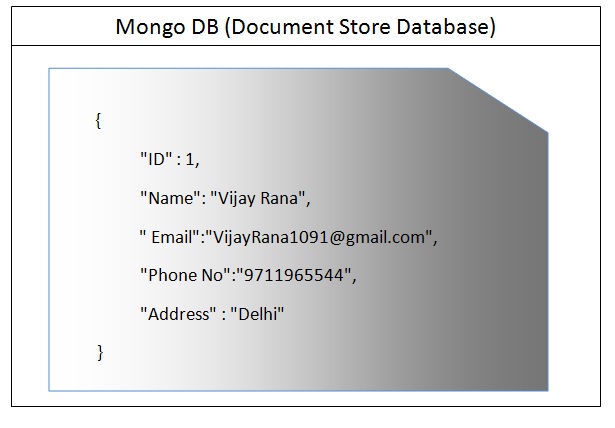
No-SQL databases can be divided in to four categories

1. Document Store  2. Graph Store 3. Column Value Store 5. Key-Value Store

MongoDB comes under Document Store database

**Document Store Databases :**In Document Store Database, Data stored in form of documents. it extends the Key value Store Database Concept.

Mongo DB and Couch DB are two main Document Store Databases .Mongo DB stores data in form of Documents. Here is an example of sample document.



Set up Mongo DB on Machine

Download MongoDB from below MongoDB official Site (Download MSI file as per your Operating System)

<https://www.mongodb.org/downloads#production>

Install MSI file (In case of Window Operation System,Default Location is (C:\Program Files\MongoDB)

Copy this folder and Place this in your C Drive or any other drive and Create a Folder in C drive named data and a Folder named db inside data folder.

So In C Drive we have two folders now

**1. MongoDB** (Copied from Installation Location)

**2. data/db** (Create a folder named data and a folder named db inside data Folder)

Start Mongo DB

Go to the bin folder inside MongoDB folder which we placed in c Drive

In my case bin Folder is inside MongoDB folder(It can be inside server folder as well as below)

C:\Program Files\MongoDB\Server\3.0\bin

and double click on the Mongod.exe (It will open a command Prompt and it will start the MongoDB server)

Last line should say "Waiting for connections..." as below it means our Mongo DB server has been started successfully

https://www.codeproject.com/KB/NoSQL/1087008/MongoD.jpg

Download RoboMongo

**RoboMongo**. Shell-centric cross-platform open source MongoDB management tool (i.e.... Admin GUI). [It] embeds the same JavaScript engine (based on Mozilla SpiderMonkey), that powers MongoDB's mongo shell. It means that you can reuse your existing skills of MongoDB Shell in **Robomongo**

We can download Robomongo from Robomongo Original Site named Robomongo.org. Here is the URL

<https://robomongo.org/>

Once download. Run the exe and Go to the File-->Connect (Make sure your mongodb server is up and running which we ran in last step through Mongod.exe)

MongoDB Terminology

Before going foreword we should know the terminology of MongoDB

| **Operation** | **In SQL** | **In MongoDB** |
| --- | --- | --- |
| Create | Insert | Insert |
| Read | Select | Find |
| Update | Update | Update |
| Delete | Delete | Remove |
| Table | Table | Collection |
| Row | Row | Document |

Key points of MongoDB

1. MongoDB Stores Data in Json Format(We call it BSON(Binary JSON))

2. JSON stands for JavaScript Object Notations and looks like {“Name”:”Vijay”}

3. JSON documents store data in Key Value Pair like {“X”:1,”Y”:2,”Z”:3}

4. There are two basic structure inside JSON is

      a. Array : List of things is represented in List of Items [……..]

      b. Dictionaries : Associate Maps {key:Value}

For example {Name : ‘Vijay’,City : ‘Shamli’,interest : ["Sports" ,"Music"  ]}

Name and city is dictionary and Interest is an Array.

5. **Mongo Db is Schema less , Schema Less means two documents don’t need to be same schema.**

   First Document in a collection can be :  {Name:"Vijay",Email:"VijayRana1091@gmail.com"}

   Second Document in same collection can be : {Name:"Vijay",Email:"VijayRana1091@gmail.com",Address : "Delhi"}

6. **MongoDB does not support Joins.**

7. **Mongo DB does not support Transactions.**

Its Query Time

We will see here some of the query which we run in Sql server on daily basis and equivalent query in Mongo DB.

For this i created a table in SQL server with below schema and a same collection(Remember table is a collection in MongoDB) in MongoDB.

https://www.codeproject.com/KB/NoSQL/1087008/SQLTable.jpg

Same table we will create in MongoDB and different operations on this table using MongoDB.

| **Operation** | **SQL** | **MongoDB** |
| --- | --- | --- |
| Select Database | use Test | use Test |
| https://www.codeproject.com/KB/NoSQL/1087008/use.jpg  We will get a message "switched to db Test" | | |
| Insert Record | Insert into  Student Values(1,'Vijay',  'VijayRana1091@gmail.com',  '9711965544','Delhi') | db.Student.insert(  {"\_id":1,"Name":"Vijay","Email":"VijayRana1091@gmail.com",  "PhoneNo":"9711965544","Address":"Delhi"}  ) |
| https://www.codeproject.com/KB/NoSQL/1087008/insertRecord.jpg | | |
| **Note :** \_id works as a primary key in MongoDb. if we will not insert any value in this column then MongoDB will automatically insert a unique ID in Table.  Inserting some more records in both Sql Server and in MongoDB (Attaching Sql script and MongoDB script)  [Download InsertRecord.zip](https://www.codeproject.com/KB/NoSQL/1087008/InsertRecord.zip)  After Running scripts we have below data in Student table (Sql server and in MongoDB) databases.  https://www.codeproject.com/KB/NoSQL/1087008/Records.jpg | | |
| Select | **Select all Columns :**  select \* from student  **Select few Column :**  select ID,Name from Student | **Select all Columns :**  db.Student.find()  **Select few Column :**  db.Student.find({},{"Name":true})  We use find() Method to pull all the records from the table. |
| https://www.codeproject.com/KB/NoSQL/1087008/Find.jpg | | |
| Where Clause | select \* from student where Name='Vijay' | db.Student.find({Name:"Vijay"}) |
| https://www.codeproject.com/KB/NoSQL/1087008/Where.jpg | | |
| Greater Than and Less Than | **select \* from student where ID>2**  **select \* from student where ID>=2**  **select \* from student where ID<2**  **select \* from student where ID<=2** | **db.Student.find({\_id:{$gt:2}})**  **db.Student.find({\_id:{$gte:2}})**  **db.Student.find({\_id:{$lt:2}})**  **db.Student.find({\_id:{$lte:2}})**  We use $gt/$gte for Greater Than/Greater Than Equal and $lt/$lte for Less Than/Less Than Equal |
| https://www.codeproject.com/KB/NoSQL/1087008/gt.jpg | | |
| Like | Below query will Find all the records where letter P exist somewhere in Name Column  **select \* from student where Name like '%P%'**  Suppose we want to find all the records where Name ends with letter 'a'  **select \* from student where Name like '%a'**  Suppose we want to fetch all the records where Name starts with Letter 'P'  **select \* from student where Name like 'p%'** | In Mongo DB we use $regex operator to check whether letter 'P' exist somewhere in Name Column.  **db.Student.find({Name:{$regex:"P"}})**  Below query will Fetch all the records where Name ends with letter 'a'  **db.Student.find({Name:{$regex:"a$"}})**  Below query will Fetch all the records where Name starts with letter 'P'  **db.Student.find({Name:{$regex:"^P"}})** |
| https://www.codeproject.com/KB/NoSQL/1087008/Regex.jpg | | |
| And/Or | **And :**select \* from Student  where Name ='Vijay'  and phoneNo='9711965544'  **Or :**select \* from Student  where Name ='Vijay'  or phoneNo='9711997119' | **And :** db.Student.find({$and :[{Name :'Vijay'},{PhoneNo : '9711965544'}]})  **Or :**db.Student.find({$or:[{Name :'Vijay'},{PhoneNo : '9711997119'}]}) |
| https://www.codeproject.com/KB/NoSQL/1087008/and.jpghttps://www.codeproject.com/KB/NoSQL/1087008/or.jpg | | |
| in | select \* from Student where id in(1,3,5) | db.Student.find({\_id:{$in:[1,3,5]}}) |
| https://www.codeproject.com/KB/NoSQL/1087008/in.jpg | | |
| Count/Sort | **Count all the records :**  select count(\*) from Student  **Sort Records in ascending order:**  select \* from student order by Name  **Sort in descending order :**  select \* from student order by Name desc | **Count all the records :**  db.Student.find().count()  **Sort Records in ascending order:**  db.Student.find().sort({Name:1})    **Sort in descending order :**  db.Student.find().sort({Name:-1}) |
| https://www.codeproject.com/KB/NoSQL/1087008/Sort.jpghttps://www.codeproject.com/KB/NoSQL/1087008/Count.jpg | | |
| Update | update student set phoneNo='2222222222', Address='USA'  where ID=4 | db.Student.update({\_id:4},{$set:{PhoneNo:'2222222222',Address:'USA'}})  **Upsert :**  db.Student.update({\_id:9},{$set:{PhoneNo:'2222222222',Address:'USA'}},{upsert:true})  above query will search a document where \_id is 9. it there will be a \_id with 9 it will update phoneNo and Address otherwise it will insert a new document where \_id is 9.  **Multi :**  Suppose we execute below query  **db.Student.update({Name:"Vijay"},{$set:{PhoneNo:'2222222222',Address:'USA'}})**  It will update only one record(remember only first Match where name is "Vijay"). But if we want to update all the records where Name is "Vijay" then we will pass multi argument also  db.Student.update({Name:"Vijay"},{$set:{PhoneNo:'2222222222',Address:'USA'}},{multi:true}) |
| https://www.codeproject.com/KB/NoSQL/1087008/update.jpg  https://www.codeproject.com/KB/NoSQL/1087008/Upsert.jpg | | |
| Delete/Remove/Drop | **Delete with Condition :**  Delete from student where ID=5  **Delete all the records :**  Delete from Student  **Drop :**  drop table Student | **Delete with Condition :**  db.Student.remove({\_id:5})  **Delete all the records :**  db.Student.remove({})  **Drop:**  db.Student.drop() |
| https://www.codeproject.com/KB/NoSQL/1087008/Remove.jpghttps://www.codeproject.com/KB/NoSQL/1087008/RemoveDrop.jpg | | |
| Top | select Top 2\* from Student | **Limit:**  db.Student.find().limit(2)  **Skip :**  db.Student.find().skip(2).limit(2)  As name suggest skip will skip the number of documents and limit will limit the number of records. |
|  | | |
| distinct | select distinct Name from Student | db.Student.distinct("Name") |
|  | | |
| Backup | BACKUP DATABASE Test TO DISK = 'C:\Vijay\Test.Bak'    WITH FORMAT,       MEDIANAME = 'Z\_SQLServerBackups',       NAME = 'Full Backup of Test Database'; | Go to the command Prompt and run the below command (assuming your mongodump exe is inside bin folder, if it is a different location then change this location accordingly)  **C:\Program Files\MongoDB\Server\3.0\bin\mongodump --db Test** |
|  | | |

MongoDB Functions

In MongoDB databases we can create functions like in Sql Server. MongoDB provides us a collection named as System.js for this.

System.js collection contains two keys

1. **\_id :**\_id is the function name.

2. **value :** value contains actual function defination.

For example, suppose we want to create a function which will accept two parameters named  **firstname**and **lastname**and will return **full name.**

Hide   Copy Code

db.system.js.save

(

{

\_id: "FullName",

value : function(FirstName,LastName) { return FirstName + ' ' + LastName; }

}

)

In order to call this function we need to load server scripts first, and than we can call this function as below

Hide   Copy Code

db.loadServerScripts();

FullName('Vijay','Rana')

**Vijay Rana** will be the result when we will call this function.

Auto Increment ID( Identity in Sql Server) in MongoDB

In MongoDB \_id works as a primary key. if we don't insert any value in \_id field then MongoDB automatically insert a unique hexadecimal value in this column.But if we want to enter auto increment integer value (Like Identity in Sql Server or Sequence in Oracle) in this field then there is no direct way to do so. For this we need to follow below steps

**Step 1:**Create a collection( Identity in my case) which will hold the counter

Hide   Copy Code

db.createCollection("Identity") *// Identity is my collection Name*

**Step 2:**Insert a document in this collection with intial counter value

Hide   Copy Code

db.Identity.insert({\_id:"incrementID",sequence\_value:0})

**Step 3:**Create a function which will increment this sequence\_value and add that function in system.js as below

Hide   Copy Code

db.system.js.save

(

{

\_id: "getNextIdentity",

value : function getNextIdentity(sequenceName)

{

var sequenceDocument = db.Identity.findAndModify

({

query:{\_id: sequenceName },

update: {$inc:{sequence\_value:1}},

new:true

});

return sequenceDocument.sequence\_value;

}

}

)

**Step 4:**While inserting a document we can call this function, which will return a incremented value as below

Hide   Copy Code

db.Employee.insert({

"\_id":getNextIdentity("incrementID"),

"EmpName":"Vijay",

"Age":"30"

})

## MongoDB is Schema less

We always talk about that MongoDB Schema Less behavior and this is the key behavior for which MongoDB is so popular now a days. So let's understand the meaning of Schema less.

Suppose we have a collection named Employeein our Test database (Please go through my previous article to know how to create database and collection in MongoDBDatabase as below):

Insert a document in Employee Collection:

Hide   Copy Code

db.Employee.insert({Name:"Vijay",Age:30,Email:"VijayRana1091@gmail.com"})

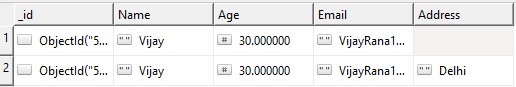
In the above document, we have Name, Emailas Stringand Ageas Number.

Now let me insert one more document and this time I will add one more column named address.

Hide   Copy Code

db.Employee.insert({Name:"Vijay",Age:30,Email:"VijayRana1091@gmail.com",Address:"Delhi"})

Here is the result:



Now just think about RDMS. In case of RDMS, we had to Alter the table first to add a new column but in case of MongoDB, we don't need to add a new column because all the documents can have different schema in MongoDB.

Let me insert one more document in EmployeeCollection.

Hide   Copy Code

db.Employee.insert

({

Name:"Vijay",Age:30,Email:"VijayRana1091@gmail.com",Address:"Delhi",Interest:["Cricket","Music"]

})

Wow, we can insert Array as well in a document in MongoDB. Actually, there are two basic structures inside JSON:

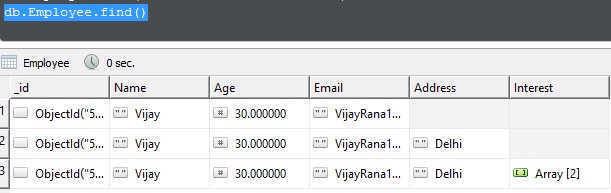
1. Array: List of things are represented in List of Items [……..]
2. Dictionaries: Associate Maps {key:Value}

So now, we are good with the concept of Schema less to go ahead.

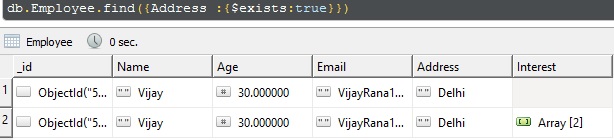
As in the above example in some document, we can have Addresscolumn and in another document in same collection, it's not necessary to have the same column. In short, in a Collection, different documents can have different columns/Schema. But suppose **you want to find out all the documents in a collection where Address column exists.**In MongoDB, we have $exist for such queries.

## $exists

We have Employee collection as below:



In the above collection as below, suppose I have some documents where we don't have Address and some documents where we have Address column along with other columns. Now suppose I want to retrieve all the records where Address exists.



Let me insert one more document in Employee Collection as below:

Hide   Copy Code

db.Employee.insert

({

Name:111,Age:30,Email:"VijayRana1091@gmail.com",Address:"Delhi"

})

This time, I inserted an integer value in Name column. Yes, we can do this as well. This is magic of Schema less nature. I can insert value of any type in any column.

But suppose I want to find out all the documents where name is in string, how we will find out?

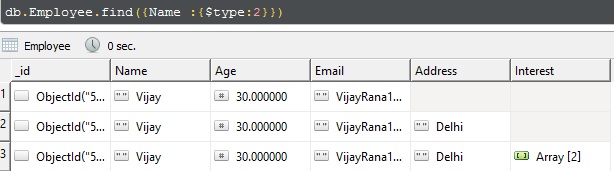
We have $type for this.

## $type

So far, we have the below records in my employee collection. I have 3 documents where I have name as a string and 1 document where name is in number.



Now, I want to find out all the documents where name is in string:



In mongodb, we use 2 for string, 1 for number. Complete list of all the data types are shown in the below table.

We can use the below data types in $type.

## Data Types in MongoDB

Some of the commonly used data types in MongoDB are as follows:

| **Data Type** | **Number** | **Meaning** |
| --- | --- | --- |
| Double | 1 | For float values |
| String | 2 | String is most commonly used DataType. In mongodb, string must be UTF-8 valid. |
| Object | 3 | For embedded documents |
| Array | 4 | For list or multiple values into one key |
| Binary Data | 5 | To store binary data |
| Undefined | 6 |  |
| Object Id | 7 | To store the document’s ID |
| Boolean | 8 | To store a boolean (true/false) value |
| Date | 9 | To store date and Time |
| Null | 10 | To store a Null value |
| Regular Expression | 11 | To store regular expression |
| 32-bit integer | 16 | To store 32 bit Integer |
| Timestamp | 17 | To store date and Time |
| 64-bit integer | 18 | To store 64 bit Integer |

Now what's next? Let me insert some more records

Hide   Copy Code

db.Employee.insert

({

Name:"Preeti",Age:26,Email:"Preeti@gmail.com",Address:"Delhi",Interest:["cooking","Music"]

})

db.Employee.insert

({

Name:"Ajay",Age:26,Email:"Preeti@gmail.com",Address:"Delhi",Interest:["Driving","Music"]

})

Now suppose we want to find out all the documents where interest is Music, i.e., this time we want to search inside an array.

We can search inside the array as well as below:

Hide   Copy Code

db.Employee.find({Interest : "Music"})

So we can say our matching is Polymorphic in MongoDb.

## $in, $all and $nin

If we want to find out all the documents where we have both cooking and Music as an interest, then we can use $all:

Hide   Copy Code

db.Employee.find({Interest : {$all:["cooking","Music"]}})



If we want to find out all the documents where Interest contains either music or driving, then we will use $inoperator:

Hide   Copy Code

db.Employee.find({Interest : {$in:["Driving","Music"]}})

If we want to find out all the records where interest is not cooking, then we will use $nin as below:

Hide   Copy Code

db.Employee.find({Interest : {$nin:["cooking"]}})

The above query will return all the documents where Interest is not cooking.

## Embedded Document and Dot Notation

We can have embedded documents as well in MongoDB as below:

Hide   Copy Code

db.Employee.insert

({

Name:{firstName:"Preeti",LastName:"Rana"},Age:26,Email:"Preeti@gmail.com",

Address:"Delhi",Interest:["cooking","Music"]

})

db.Employee.insert

({

Name:{firstName:"Vijay",LastName:"Rana"},Age:30,Email:"Vijay@gmail.com",

Address:"Delhi",Interest:["cooking","Music"]

})

To specify a condition inside Embedded document, we can use dot notation as below:

Hide   Copy Code

db.Employee.find({"Name.firstName":"Vijay"})

The above query will search all the documents where we have firstName as Vijayinside Name.

In the next article of this series, I will cover Aggregation(groupBy),Indexes, $Text and $lookup. So stay tuned.

## ****Indexes****

We can think Indexes like a Book Index. Suppose we are searching a topic in a book and we don't have indexes than we need to scan every pages until we will not reach to that page. if your book contains 100 pages then you can manage( if you have enough free time) but if your book contains suppose 1 million pages then it will be very tedious job to search the topic by flip through every page. Same concept we have in MongoDB

If we don't have indexes then mongodb will do a complete collection scan to select the document that match the query statement.If document count is high in collection than it will be a death of performance.

Suppose we have below documents in Names Collection as below

Hide   Copy Code

db.Names.insert({"Name":"Ajay"})

db.Names.insert({"Name":"Manoj"})

db.Names.insert({"Name":"Preeti"})

db.Names.insert({"Name":"Anuj"})

db.Names.insert({"Name":"Tony"})

db.Names.insert({"Name":"Steve"})

db.Names.insert({"Name":"Smith"})

db.Names.insert({"Name":"David"})

db.Names.insert({"Name":"William"})

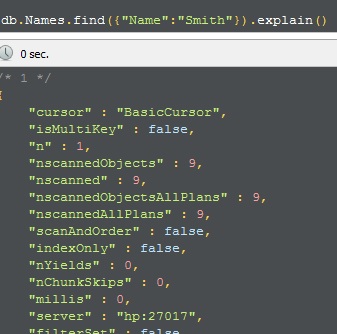
## https://www.codeproject.com/KB/NoSQL/1091645/ArbitartyOrder.jpg

Documents in Names collection will store in an arbitrary order.If there is no Index and If we will find a document like below then it will be complete collection scan and it will down the performance

Hide   Copy Code

db.Names.find({"Name":"Smith"})

If we want to find out what MongoDB is doing with the above Query. just use **explain()** like below



Here some points to notice

**cursor : BasicCursor** :  It means MongoDB will perform a complete collection scan.

**nscannedObjects** : MongoDB scan 9 objects to match this query.

So Now the Question Comes

**What is Index :**Index is an order set of items. Index stores the values in a specific order.

## ****Default Index****

When we create a collection in MongoDB then MongoDB creates a unique Index on \_id Field automatically.Becuae its a unique index that's why it prevents us to enter duplicate values in \_id field.We can not drop this index in MongoDB.

## ****Create Index****

To create an index in MongoDB we have two methods

**1. createIndex()**: Syntax of createIndex() method is

Hide   Copy Code

[\_\_strong\_\_]db.CollectionName.createIndex({"Key":1 or -1})

1 is for ascending and -1 is for descending order. So if we want to create a Index on Name key in Names collection then we will create a index as below

Hide   Copy Code

[\_\_strong\_\_]db.Names.createIndex({"Name":1})

**2. ensureIndex() :**Syntax of ensureIndex() method is

Hide   Copy Code

[\_\_strong\_\_]db.CollectionName.ensureIndex({"key": 1 or -1})

**ensureIndex()** method is deprecated since version 3.0.0.This method is an alias of **createIndex()**.

## Different Types of Indexes in MongoDB

### **1. Single Field Index :**

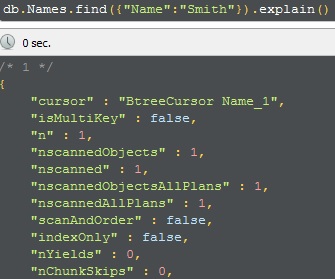
Apart from the default Index on \_id field created by MongoDB users can create ascending or descending index on a single field.

We create a single key index as below

Hide   Copy Code

db.Names.createIndex({"Name":1})

Now just run the explain()



ohh wow. Now we have an index (**BtreeCursor Name\_1**) defined instead of Basic Cursor and Most Importantly look the **nscannedObjects** which is 1 now it means MongoDB scanned only 1 document which we mentioned in the query.

Suppose we have another collection named users which have below documents

Hide   Copy Code

db.Users.insert({"Name":"Ajay","Age":30})

db.Users.insert({"Name":"Manoj","Age":60})

db.Users.insert({"Name":"Preeti","Age":20})

db.Users.insert({"Name":"Anuj","Age":70})

db.Users.insert({"Name":"Tony","Age":25})

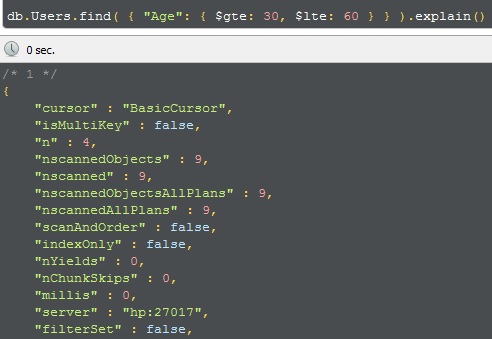
db.Users.insert({"Name":"Steve","Age":18})

db.Users.insert({"Name":"Smith","Age":33})

db.Users.insert({"Name":"David","Age":53})

db.Users.insert({"Name":"William","Age":65})

Now suppose we want to find out all the documents where Age is greater than 30 and less than 60.



So we have **BasicCursor**so it will be a complete table scan and total documents scanned by the query is 9. Now I am defining a index on Age

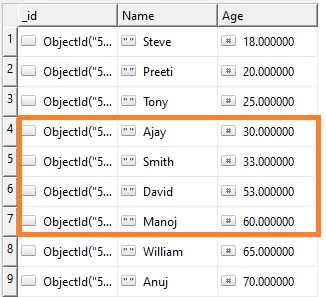
Hide   Copy Code

db.Users.createIndex({"Age":1})

Now again run the query which will find out all the documents where Age is greater than 30 and less than 60.



After Index, MongoDB will not do a complete table scan, it will only 4 rows.



### **2. Compound Index :**

Sometimes we want to search based on Name and Age both. In that case we will have to apply index on Name and Age both and this will called Compound Index

Syntax : **db.CollectionName({"Key1":1 or -1,"Key2": 1 or -1,"KeyN":1 or -1})**

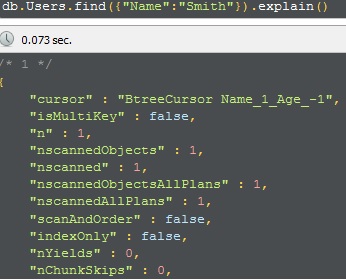
We will create an Index on Name and Age both in Users collection as below

Hide   Copy Code

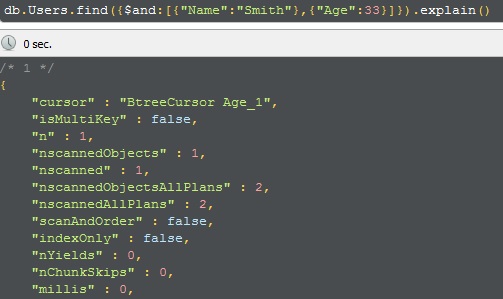
db.Users.createIndex({"Name":1,"Age":-1})

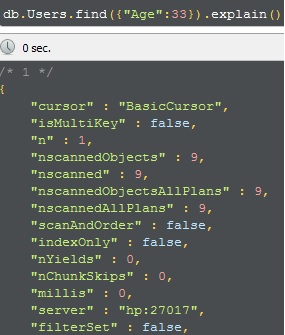
**Note : Compound Index will work only if we will search on Name or Name and Age. If we will search through Age only than Compound Index will not work.**

Suppose we are searching through Name then we can see that Compound Index is in use



If we will search through Name and Age than we can see again Compound Index is in use



But if search only on Age then we can see that Compound Index is not in use

So no Index is using in that case.

### **3. Multikey Index :**

Remove Existing documents from Users collection and insert some documents with **interest** in Users collection as below

Hide   Copy Code

db.Users.remove({})

db.Users.insert({"Name":"Ajay","Age":30,Interest : ["cricket","music"] })

db.Users.insert({"Name":"Manoj","Age":60,Interest : ["cricket","driving"]})

db.Users.insert({"Name":"Preeti","Age":20,Interest : ["music","driving"]})

db.Users.insert({"Name":"Anuj","Age":70,Interest : ["cooking","music"]})

db.Users.insert({"Name":"Tony","Age":25,Interest : ["swimming","cooking"]})

db.Users.insert({"Name":"Steve","Age":18,Interest : ["dancing","music"]})

db.Users.insert({"Name":"Smith","Age":33,Interest : ["tennis","tv"]})

db.Users.insert({"Name":"David","Age":53,Interest : ["music","swimming"]})

db.Users.insert({"Name":"William","Age":65,Interest : ["dancing","swimming"]})

Now if we want to Index the content of an array (**Interest**in my case) then we will use the Multikey Index.

**Syntax : db.CollectionName.createIndex({"Array": 1 or -1})**

We will create a MultiKey Index in Interest as below

Hide   Copy Code

db.Users.createIndex({Interest : 1})

### **4. Text Index :**

If we are performing a text search then for better performance we can apply the Text Index on string content.

We can create a text index only on string fields.

Syntax : **db.CollectionName.createIndex({Field Name:"text"})**

Suppose we want to create an Index on Name field then we will create a text index as below

Hide   Copy Code

db.Users.createIndex({Name : "text"})

**Note : A collection can have at most one text index.**

Apart from these Indexes MongoDB supports some more indexes which includes **Geospatial**and **Hashed**Index.

**Geospatial**Index is for better queries on geospatial coordinate data and **Hashed**Index Indexes the hash of the value of a field

## MongoDB Index Properties

### **1. Unique**

unique property on a index field allows MongoDB to not accept a duplicate value for index field.In other words unique property will restrict t insert duplicate value for Index field.

I am adding an another column named "SSN" in Users collection and adding a index on "SSN" Field with Unique Property as below

Hide   Copy Code

db.Users.drop()

db.Users.createIndex({SSN:1},{unique:true})

db.Users.insert({"Name":"Ajay","Age":30,Interest : ["cricket","music"] ,"SSN" : "12345"})

db.Users.insert({"Name":"Manoj","Age":60,Interest : ["cricket","driving"],"SSN" : "54321"})

I removed all the records from Users collection and create a index on SSN Filed with unique property. So if i will try to insert duplicate values in SSN then i will get the error. Let's try it

Hide   Copy Code

db.Users.insert({"Name":"Preeti","Age":20,Interest : ["music","driving"],"SSN" : "54321"})

and the error says

Hide   Copy Code

insertDocument :: caused by :: 11000 E11000 duplicate key error index: Test.Users.$SSN\_1 dup key: { : "54321" }

So we can't insert duplicate values in a field on which we have a index with unique property.

### **2. Sparse**

I am dropping the Users collection and will insert some document as below

Hide   Copy Code

db.Users.drop()

db.Users.insert({"Name":"Ajay","Age":30,Interest : ["cricket","music"] ,"SSN" : "12345"})

db.Users.insert({"Name":"Manoj","Age":60,Interest : ["cricket","driving"],"SSN" : "54321"})

db.Users.insert({"Name":"Preeti","Age":20,Interest : ["music","driving"]})

db.Users.insert({"Name":"Anuj","Age":70,Interest : ["cooking","music"]})

Now what will happen if i will try to create a Index on SSN Field with unique Property. If i will try to create a index with unique property as below then i will get an error because SSN contains null for last two documents and so SSN is not unique.

Hide   Copy Code

db.Users.createIndex({SSN:1},{unique:true})

and error is

Hide   Copy Code

E11000 duplicate key error index: Test.Users.$SSN\_1 dup key: { : null }

So what will be the solution? Can't i create a unique index on such records?

Hold on.. we have a solution.. we have sparse property for such scenarios.

Sparse will tell the database that those documents should not be included in index where SSN is missing.

So Cool.. Its time to create a index with sparse with unique property.

Hide   Copy Code

db.Users.createIndex({SSN:1},{unique:true,sparse:true})

and this time no error while creating index with unique property because sparse is there.

### **3. Partial Index**

This is a new born concept which comes with MongoDB 3.2.Sometimes we want to create index with some specific condition. If we are creating index with some condition than it is a partial Index.

Suppose i want to create an Index on Name field only when Age is greater than 30. We need to specify a condition while creating an Index as below

Hide   Copy Code

db.Users.createIndex(

{ Name: 1},

{ partialFilterExpression: { Age: { $gt: 30 } } }

)

To apply a condition we use **partialFilterExpression.**

### **4. TTL Index**

MongoDB has a special type of single Field Index named TTL Index.MongoDB uses this type of Index to remove document automatically after a certain period of Time. We use **expireAfterSeconds option**to provide the expiration Time.

I am creating a TTL Index on Age with 60 second expiring time as below

Hide   Copy Code

db.Users.createIndex( { "Age": 1 }, { expireAfterSeconds: 60 } )

It will delete this document after 60 seconds automatically. A background task runs in every 60 seconds that removes all the expire documents.so it might take some extra time to remove this document from collection. It depends also on workload of mongod instance so expired documents can be in collection beyond the specified amount of time.

## Some Key Points

### **1. getIndexes()**

If we want to see all the created indexes on a collection we use getIndexes() method.

Syntax : db.CollectionName.getIndexes()

### **2. dropIndex()**

To drop a Index we use dropIndex() method.

Syntax : db.CollectionName.dropIndex({"Key":1 or -1})

Pass the key with 1 or -1 which we passed while creating index.

### **3. dropIndexes()**

To remove all the Indexes from a collection we use dropIndexes() method.

Hide   Copy Code

db.CollectionName.dropIndexes()

### **4. To Get all Indexes from a collection**

Hide   Copy Code

db.getCollectionNames().forEach(function(collection) {

index = db[collection].getIndexes();

print("Indexes for " + collection + ":");

printjson(index);

});

### **5. Rebuild a Index**

To Rebuild a Index we use reIndex() method as below

Hide   Copy Code

db.CollectioName.reIndex()

## Limitations

1. A single collection can not have more than 64 indexes.

2. In Compound Index can not have more than 31 Fields.