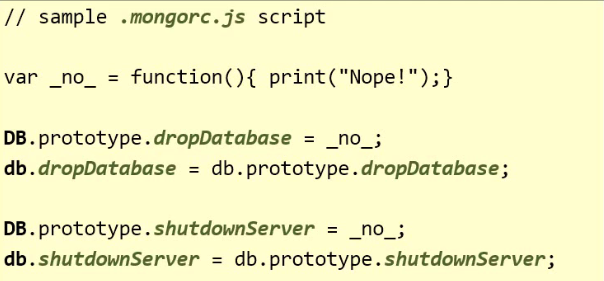
**Day 2:**

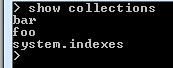
* Blind Command :
  + Let's say you had some administrative task that you needed to perform on a regular basis. You'd probably want to schedule that inside a batch file or something of this nature. The Mongo shell allows you to do just that using the **eval** command line option.
  + From the console, you will run the Mongo shell using the eval command line option and the shell will execute your command against the Mongo server and return immediately to the console without popping the interactive shell.
  + Not popping the interactive shell is important because you want to run it as a background or scheduled task.
  + **Example Rotating Log:**
    - By default, Mongo will continue and append to a single log file as long as you let it. That log file can get pretty large as you might imagine, so it's a very useful thing to schedule log rotation on a daily or weekly or hourly basis depending on how many transactions you have running.
    - Step 1 :Start Mongoserver >>
      * D:\>mongod -f D:\mongoconf\mongod.conf
    - Step 2 : Go to log location and check no of logfiles by issuing
      * D:\mymongodatalog>dir /b \*.log.\*
    - Step 3 : Do administrative action of Rotating logfile
      * D:\mymongodatalog>mongo localhost/admin --eval "db.runCommand({logRotate:1})" This command will create one more logfile
    - Use printJson() to see difference
      * D:\mymongodatalog> mongo localhost/admin --eval printjson("db.runCommand({logRotate:1})")
* **Running Js file without going in shell**
  + mongo userCount.js
* **Running the js file and not going out of shell**
  + **Safer.js** (Scripting to preventing accidently dropping of Database)
  + **Demo: Execute:**
    - D:\MongoDB 2.6 Standard\bin>mongo safer.js –shell
    - db.dropDatabase() ----u will see that database cant be dropped
* **Exploring shell:**
  + var stuff=function(name){ var doc={ Name:name, Created:date(), Type:'r'}; db.foo.save(doc); }
  + Stuff()
  + Stuff
* **External Editor**
  + You can associate external editor with MongoShell
  + set EDITOR="C:\Program Files (x86)\Notepad++\notepad++.exe"
  + Go to Mongo shell>> mongo
    - Type some script like :
      * var stuff=function(name){ var doc={ Name:name, Created:date(), Type:'r'}; db.foo.save(doc); }
    - Type edit stuff ..u will see that it will open in notepad++
  + Load('userCount.js') ..will load and execute .js file in Mongoshell
* **Understanding ".mogorc.js"**
  + Understanding mongo --norc
* Prevent disaster>>>sample .mongorc.js



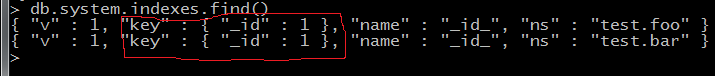
* Saving Data:
  + **mongo**
  + **db**--will show u current database
  + **show collections** --- show collections and indexes
  + **db.foo.save({\_id:1,x:10})**----will save data
  + **db.foo.find()** ...will show all ducuments in collection foo
  + again try show collections ..u will find this



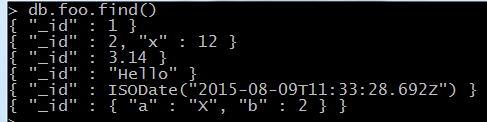
* + If I save in bar collection like ...**db.bar.save({\_id:1,x:10})** and execute "show collections " command we will see:



* + Now if we check index maintained for fast searching and updating we need to execute command **"db.system.indexes.find()"**



* Data Types for **\_id** field
* db.foo.save({\_id:1})
* >db.foo.save({\_id:3.14})
* >db.foo.save({\_id:"Hello"})
* >db.foo.save({\_id:ISODate()})
* > db.foo.save({\_id:{a:'X',b:2}})
  + And when u do db.foo.find() it shows



* \_id cant be array..i.e we cant do **db.foo.save({\_id:[1,2,3]})**
* **When u don’t assign \_id then :**
* db.users.save({Name:'Ali'}) and execute db.users.find() u get

{ "\_id" : ObjectId("55c73c70a9c69d31a5b80268"), "Name" : "Ali" }

* Here u saw mongo assigned id to the ducument
* Mongo has some very interesting properties with this id
  + ObjectId()
  + ObjectId().getTimestamp()...helps in ascending insertion
* **Difference between save() and insert()**
* While using save() u can save two documents with same \_id
  + db.foo.save({\_id:1,name:'Ali'})
  + db.foo.save({\_id:1,name:'Joshi'})

Later will override the first one

* To avoid this use insert:
  + db.bar.insert({\_id:1,name:'Ali'})
  + db.bar.save({\_id:1,name:'Joshi'})...you will get below error of
  + "errmsg" : "insertDocument :: caused by :: 11000 E11000 duplicate key error index: test.bar.$\_id\_ dup key: { : 1.0 }"
* There is no point is using insert() without specifying \_id
  + db.secbar.insert({name:'Ali'})
  + db.secbar.insert({name:'Joshi'})...after this when we will execute **db.secbar.find()**..we will get

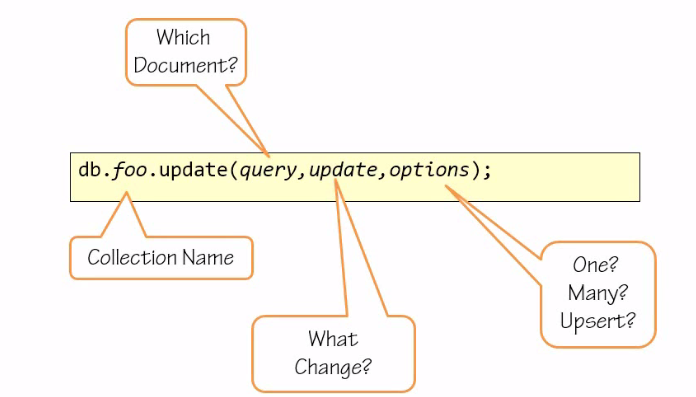


* **Case Study—for insert**
  + db.user.insert({\_id:"bob@gmail.com",Name:'bob',Adresss:{Street:'123 main',City:'paris'},Music:['Blues','Alt']})
  + Let’s say there is a form and some one dblclicks ….there should no issue in case of insert
  + db.user.find().pretty()
  + db.user.insert({\_id:"meg@meggmail.com",Name:'bob',Adresss:{Cruise:'fun',ship:'ss ahoy',port:'Marina'},Music:['Blues','Alt'],Logins:[{ip:123,t:ISODate("2012-11-10")}]})
  + Analyze **that in the same collection there are two documents with different schema**
* **Updating Documents**
* **Concurrency issues (Scenario 1...client 1)**
  + db.a.save({\_id:1,x:10}) ..let’s say I want to increment **x..**I might endup writing below code

var doc=db.a.findone({\_id:1});

doc.x=doc.x+1;

db.a.save(doc);

* + This approach is bad ..because if someone else is reading and modifying value of **x**..i have stale value of **x**
* **Concurrency issues (Scenario 2….client 2)**
* If another client gets a version of document and tries to save it with extra field
  + db.a.save({\_id:1,x:10,y:3})
  + db.a.find() will give ..**{ "\_id" : 1, "x" : 10, "y" : 3 }**
  + **But...lets say first client calls ..db.a.save(doc);**….it will be overridden by { "\_id" : 1, "x" : 11 }
* **Update Command:**
  + Mongo's update command to the rescue. The Mongo update command is **atomic** within a document. No two clients may update the same document at the same time. Two update commands issued concurrently will be executed one after the other. Here's the syntax of the update command. 
  + **First,** you need to specify which collection you're going to update.
  + **Second,** you will need to specify which document or documents you are targeting so that you don't update all the documents in that collection. You do that by issuing a Mongo query.
  + **Third,** you will need to specify what change you want to see enacted, that's the update parameter.
  + **Lastly,** you may specify other options such as do you want to change only one, the first document found matching the query, or multiple of them, any document matching the query. Do you want to **upsert**, meaning do you want to save a new record in case the query does not match any document, do you want to generate a document on the fly and inserted as the change specified. The options parameter in itself is optional. If you omit this parameter, Mongo will update only one document and Mongo will not create a new document if one does not exist that matches the query.
* **Demo for update** with **$inc and $set**
  + Considering the previous **scenario 1:**
    - Lets say we save by .. **db.a.save({\_id:1,x:10})** and want to increment x by one
    - use command **db.a.update({\_id:1},{$inc:{x:1}})**
  + Considering the previous **scenario 2**:Solution is:

db.a.save({\_id:1,x:10});

db.a.find()

{ "\_id" : 1, "x" : 10 }

db.a.update({\_id:1},{$set:{y:3}})

db.a.update({\_id:1},{$inc:{x:1}})

db.a.find()

{ "\_id" : 1, "x" : 11, "y" : 3 }

* + Removing value from ducument: by **$unset**

db.a.update({\_id:1},{$unset:{y:''}})

db.a.find()

{ "\_id" : 1, "x" : 11 }

* + Updating the key in document: using **$rename**

db.a.save({\_id:1,Naem:'bob'})

db.a.find()

db.a.update({\_id:1},{$rename:{'Naem':'Name'}})

* + Working with arry :**$push** and **#addToSet**

db.a.save({\_id:1})

db.a.find()

db.a.update({\_id:1},{$push:{things:'one'}})

db.a.update({\_id:1},{$push:{things:'two'}})

db.a.update({\_id:1},{$push:{things:'three'}})

db.a.find()

db.a.update({\_id:1},{$addToSet:{things:'four'}})

db.a.update({\_id:1},{$addToSet:{things:'four'}})

* + Removing from array :only work on array
    - db.a.update({\_id:1},{$pull:{things:'three'}})
    - db.a.update({\_id:1},{$pop:{things:1}}) ….removes last element in array
    - b.a.update({\_id:1},{$pop:{things:1}})…removes first element in array
  + Multiline update:
* db.a.save({\_id:1,things:[1,2,3]})
* db.a.save({\_id:2,things:[2,3]})
* db.a.save({\_id:3,things:[3]})
* db.a.find()
  + - * Now ..execute ..**db.a.update({},{$push:{things:4}})…{} means anything matched**

> db.a.find()

{ "\_id" : 1, "things" : [ 1, 2, 3, 4 ] }

{ "\_id" : 2, "things" : [ 2, 3 ] }

{ "\_id" : 3, "things" : [ 3 ] }

>...only one record affected ..because of default setting to fix this

* + - * **db.a.update({},{$push:{things:4}},{multi:true})**

db.a.find()

{ "\_id" : 1, "things" : [ 1, 2, 3, 4, 4 ] }

{ "\_id" : 2, "things" : [ 2, 3, 4 ] }

{ "\_id" : 3, "things" : [ 3, 4 ] }

* + - * If I want to document that have element "2"
        + **db.a.update({things:2},{$push:{things:42}},{multi:true})**

> db.a.find()

{ "\_id" : 1, "things" : [ 1, 2, 3, 4, 4, 42 ] }

{ "\_id" : 2, "things" : [ 2, 3, 4, 42 ] }

{ "\_id" : 3, "things" : [ 3, 4 ] }

>

* + - * **Understanding Find and Modify**

Machine generated alternative text: ctionme
db. foo . findAndModify(
query: <document>,
update: <document>,
upsert: <boolean> ,
remove: <boolean>,
new: <boolean>,
sort: <document>,
fields: <document>}
Delete ¡t? 
[ery orderL
Return new/olt?
‘‘

* + - * + Demo: **findandModify()**

db.a.save({\_id:1,things:[1,2,3]});

db.a.save({\_id:2,things:[2,3]});

db.a.save({\_id:3,things:[3]});

db.a.save({\_id:4,things:[1,3]});

var mod={

"query":{"things":1

},

"update":{

"$set":{

"touched":true

}

},

"sort":{

"\_id":-1

}

}

**db.a.findAndModify(mod)** ..this will give u document before modification was done

{ "\_id" : 4, "things" : [ 1, 3 ] }...touched field is missing

db.a.find()

{ "\_id" : 1, "things" : [ 1, 2, 3 ] }

{ "\_id" : 2, "things" : [ 2, 3 ] }

{ "\_id" : 3, "things" : [ 3 ] }

{ "\_id" : 4, "things" : [ 1, 3 ], "touched" : true }

* + - * Now lets get updated documents...........
        + set property **new =true** & update touched field to **false**
        + **mod.update.$set.touched=false**
        + > db.a.findAndModify(mod

{ "\_id" : 4, "things" : [ 1, 3 ], "touched" : false }....this is documentation after modification

* + - * + change the sort to find the first record ..rather than last record matching the Query

mod.sort.\_id=1

db.a.findAndModify(mod)

**Finding Document:**

* + - * **Find Command**

Machine generated alternative text: Which fields?
db.foo.find(query, projection)
Which documenJ

* + - * Save animals Collection with sample data
      * db.animals.save({\_id:1,name:"cat",tags:["land","cute"],info:{"type":"mamal",color:"red"}})
      * db.animals.find({\_id:1})...will give u document with id =1
      * **PROJECTION**

db.animals.find({\_id:1},{name:"cat"})

**Sample Data:**

db.animals.save({\_id:1,name:"cat",tags:["land","cute"],info:{type:"mamal",color:"red"}})

db.animals.save({\_id:2,name:"dog",tags:["security","watchman"],info:{type:"mamal",color:"brown"}})

db.animals.save({\_id:3,name:"elephant",tags:["huge","Intelligent"],info:{type:"mamal",color:"grey"}})

db.animals.save({\_id:4,name:"parrot",tags:["fly","light"],info:{type:"bird",color:"green"}})

db.animals.save({\_id:5,name:"duck",tags:["water","quack"],info:{type:"bird",color:"white"}})

db.animals.save({\_id:6,name:"lion",tags:["Wild","Hairy"],info:{type:"mamal",color:"brown"}})

db.animals.save({\_id:7,name:"Bee",tags:["Honey","small"],info:{type:"insect",color:"grey"}})

db.animals.save({\_id:8,name:"shark",tags:["canswim","huge"],info:{type:"insect",canFly:false,color:"black"}})

db.animals.save({\_id:9,name:"cock",tags:["canswim","feather"],info:{type:"bird",canFly:null,color:"black"}})

* + - * db.animals.find({\_id:{$gt:5}},{\_id:1}) ..greater than
      * db.animals.find({\_id:{$lt:5}},{\_id:1}) ……..less than

db.animals.find({\_id:{$lte:5}},{\_id:1})

db.animals.find({\_id:{$gte:5}},{\_id:1})

db.animals.find({\_id:{$gt:2,$lt:4}},{\_id:1})

db.animals.find({\_id:{$not:{$gt:2}}},{\_id:1})

db.animals.find({\_id:{$in:[1,3]}},{\_id:1})

db.animals.find({\_id:{$nin:[1,3]}},{\_id:1})

* + - * **Working with array...**
        + db.animals.find({\_id:1}).pretty()..contains tags
        + db.animals.find({tags:'cute'},{name:1})
        + db.animals.find({tags:{$in:['cute','huge']}},{name:1})...either huge or cute
        + db.animals.find({tags:{$all:['cute','huge']}},{name:1})..both(and operation)
        + db.animals.find({tags:{$nin:['cute']}},{name:1})
      * **Dot Notation** for Subdocument

**db.animals.find({\_id:1}).pretty()**

{

"\_id" : 1,

"name" : "cat",

"tags" : [

"land",

"cute"

],

"info" : {

"type" : "mamal",

"color" : "red"

}

}

**db.animals.find({"info.type":"mamal"}).pretty()**

db.animals.find({\_id:4}).pretty()

{

"\_id" : 4,

"name" : "Parrot",

"tags" : [

"fly",

"light"

],

"info" : {

"type" : "bird",

"color" : "green"

}

}

**db.animals.find({info:{type:'bird',color:'green'}},{name:1}) ...type is bird and color is green**

**db.animals.find({"info.type":'bird',"info.color":'green'},{name:1}) ...Dot Notation**

* **$Exists**
  + db.animals.find({"info.canFly":null},{name:1}) ..This will give u all documents (with null,without null)

{ "\_id" : 1, "name" : "cat" }

{ "\_id" : 2, "name" : "dog" }

{ "\_id" : 3, "name" : "elephant" }

{ "\_id" : 4, "name" : "Parrot" }

{ "\_id" : 5, "name" : "Duck" }

{ "\_id" : 6, "name" : "Lion" }

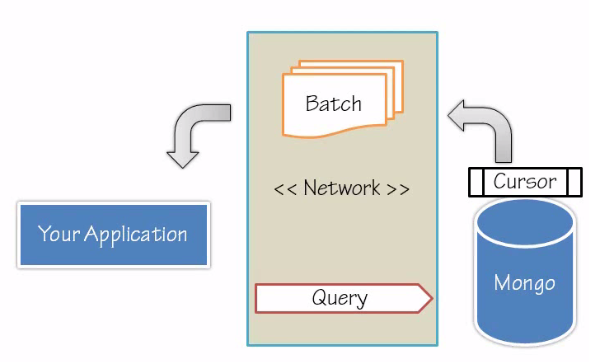
{ "\_id" : 7, "name" : "Bee" }

{ "\_id" : 9, "name" : "cock" }

db.animals.find({"info.canFly":{$exists:true}},{name:1}) .will give only thore documents where **canfly** exists

db.animals.find({"info.canFly":{$exists:false}},{name:1})...reverse

* + - * **AND**
        + db.animals.find({"info.type":'bird',tags:'quack'},{name:1}) ..will documents where type is bird and tags have value quack
        + ',' act as and
    - **More Projection**
      * db.animals.find({\_id:1},{\_id:1})
      * db.animals.find({\_id:1},{\_id:1,name:1})
      * db.animals.find({\_id:1},{\_id:0,name:0,info:0})
      * db.animals.find({\_id:1},{name:1,info:1}) ..this will give id also as
      * db.animals.find({\_id:1},{\_id:0,name:1,info:1}) ...this will exclude id
      * db.animals.find({\_id:1},{\_id:0,name:1,info:0}) ...u cannot mix including and excluding fields
    - **Cursor in Mongo**



* what exactly happens when you're shooting a query at the Mongo server? Your application will want to communicate over the network. It's going to shoot a query over the network to Mongo, and that query might cover many, many documents, more than you even need, more than fit in your memory.
* To support efficient retrieval of documents, Mongo uses a cursor. Mongo will populate a cursor and give you a batch of documents each time.
* On the client side, the Mongo shell and every other driver will retrieve a batch of documents from the cursor and finally close the cursor.
* A client can then signal the server to close the cursor before all the documents have been exhausted, thereby saving many, many bytes over the network and a lot of time, also freeing the server to service other calls.
* **Demo:**

db.animals.find({},{name:1})

var c=db.animals.find({},{name:1});

c.size()

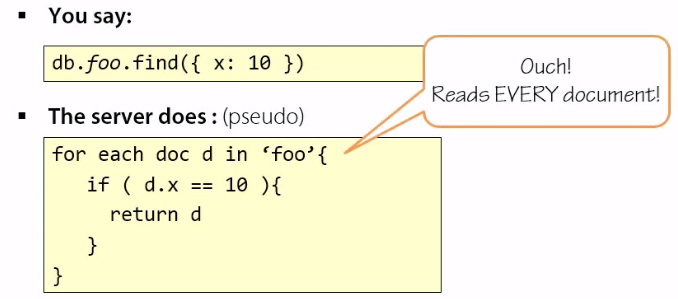
c.hasNext()

c.forEach(function(d){print(d.name)})

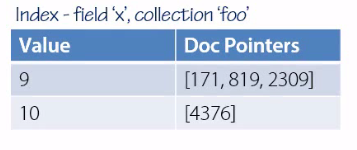
c.hasNext()

**Indexing :**

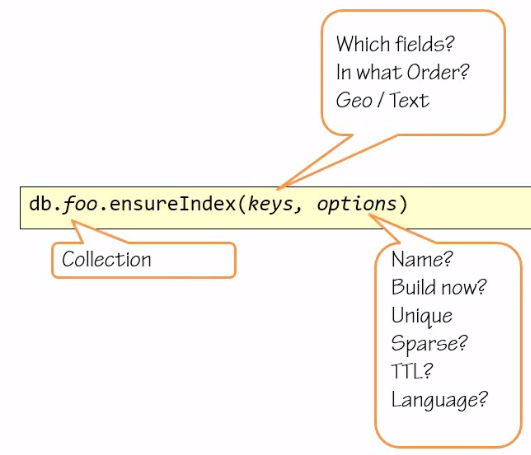
* **Problem :** Lets say we have collection score and we need to find a document with x=10
  + db.score.find({x:10}) ..documents are stored in contiguous manner
    - Sever will need to visit each and every document and check for x=10



* + - Very wasteful operation
  + **Solution :** Is to use Index



* + If indexes are in sorted order ..its advantage for **Mongo Query engine**
* Indexing Strategies in Mongo:
  + - Regualr( Btree index on )
    - Geo(proximity --find restaurant near redfort) To support efficient queries of geospatial coordinate data
    - Text-- MongoDB provides a text index type that supports searching for string content in a collection (search engine)
    - Hashed--- To support hash based sharding, MongoDB provides a hashed index type, which indexes the hash of the value of a field. (key-value ..Shrading)
    - TTL-- TTL indexes are special indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time.  ( documents get expired after some time).. This is ideal for certain types of information like machine generated event data, logs, and session information that only need to persist in a database for a finite amount of time.
* Create Index command:
  + db.foo.ensureIndex.(keys,options)



* **Demo:**

db.animals.find({name:'cat'})

{ "\_id" : 1, "name" : "cat", "tags" : [ "land", "cute" ], "info" : { "type" : "mamal", "color" : "red" } }

* + Checking Whether index was unsed in retrieving this
  + **db.system.indexes.find({ns:'test.animals'},{key:1})...**to check whether index exists of not
* But how to check whether index was used while retrieving record or not .
  + **db.animals.find({name:'cat'})** .explain() ..gives

{

**"cursor" : "BasicCursor", …..this is the clue that index was not used**

"isMultiKey" : false,

"n" : 1,

"nscannedObjects" : 9,

**"nscanned" : 9,**

"nscannedObjectsAllPlans" : 9,

"nscannedAllPlans" : 9,

"scanAndOrder" : false,

"indexOnly" : false,

"nYields" : 0,

"nChunkSkips" : 0,

"millis" : 0,

"server" : "ChandA:27017",

"filterSet" : false

}

**Lets Create Index on animals::**

* + db.animals.ensureIndex({name:1}) …1 means build index ascending(a...b)

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

* + db.system.indexes.find({ns:'test.animals'},{key:1})..will give u below info

{ "key" : { "\_id" : 1 } }

{ "key" : { "name" : 1 } }

* + Execute explain again….
  + **db.animals.find({name:'cat'})** .explain() ..this time it will show

{

**"cursor" : "BtreeCursor name\_1",**

"isMultiKey" : false,

"n" : 1,

"nscannedObjects" : 1,

**"nscanned" : 1,**

"nscannedObjectsAllPlans" : 1,

"nscannedAllPlans" : 1,

"scanAndOrder" : false,

"indexOnly" : false,

"nYields" : 0,

"nChunkSkips" : 0,

"millis" : 1,

"indexBounds" : {

"name" : [

[

"cat",

"cat"

]

]

},

"server" : "ChandA:27017",

"filterSet" : false

}

* + **Using other fields .**
    - db.animals.find({name:'cat',tags:'land'}).explain()..also used index
    - db.animals.find({name:{$lt:'dog'}},{name:1}).explain() …

{

"cursor" : "BtreeCursor name\_1",

"isMultiKey" : false,

"n" : 6,

"nscannedObjects" : 6,

"nscanned" : 6,

"nscannedObjectsAllPlans" : 6,

"nscannedAllPlans" : 6,

"scanAndOrder" : false,

"indexOnly" : false,

"nYields" : 0,

"nChunkSkips" : 0,

"millis" : 1,

**"indexBounds" : {**

**"name" : [**

**[**

**"",**

**"dog"**

**]**

**]**

**},**

"server" : "ChandA:27017",

"filterSet" : false

}

* + - db.animals.find({name:{$lt:'donkey'}},{name:1})

{ "\_id" : 7, "name" : "bee" }

{ "\_id" : 1, "name" : "cat" }

{ "\_id" : 9, "name" : "cock" }

{ "\_id" : 2, "name" : "dog" }

* + - **db.animals.find({name:{$lt:'donkey'},tags:'land'},{name:1})**

{ "\_id" : 1, "name" : "cat" }

* + - **db.animals.find({name:{$lt:'donkey'},tags:'land'},{name:1}).explain()**

**check for ..** "nscanned" : 4,

**check for .. "name" : [**

**[ "",**

**"donkey"**

**]**

* + **Dropping index:**
    - **db.system.indexes.find({ns:'test.animals'},{ns:1,name:1}) ...shows available** indexes

{ "name" : "\_id\_", "ns" : "test.animals" }---this index cannot be dropped

{ "name" : "name\_1", "ns" : "test.animals" } ..we can drop index on **name** key

* + - **db.animals.dropIndex("name\_1")**
* **Mongo Supports indexing arbitrary nested field..No true for other databases even ur RDBMS**
  + **db.animals.findOne({name:'cat'})**

{

"\_id" : 1,

"name" : "cat",

"tags" : [

"land",

"cute"

],

"info" : {

"type" : "mamal",

"color" : "red"

}

}

* + **db.animals.ensureIndex({"info:color":1})..creating index on nested field..**
  + **db.animals.find({"info.color":"grey"},{name:1}).explain()**
* **Indexing on tags array:**
  + db.animals.findOne()

{

"\_id" : 1,

"name" : "cat",

"tags" : [

"land",

"cute"

],

"info" : {

"type" : "mamal",

"color" : "red"

}

}

* **Create index on tag :**
  + **db.animals.ensureIndex({tags:1})**
  + db.animals.find({tags:'canswim'},{name:1})

{

"cursor" : "BtreeCursor tags\_1",

"isMultiKey" : true,

"n" : 2,

"nscannedObjects" : 2,

"nscanned" : 2,

"nscannedObjectsAllPlans" : 2,

"nscannedAllPlans" : 2,

**"scanAndOrder" : false,….here mongo uses order of doc in index**

"indexOnly" : false,

"nYields" : 0,

"nChunkSkips" : 0,

"millis" : 0,

"indexBounds" : {

"tags" : [

[

"canswim",

"canswim"

]

]

},

"server" : "ChandA:27017",

"filterSet" : false

}

* + **Now** ..db.animals.find({tags:'canswim'}).sort({name:1}).explain() ..index is not useful for sorting.
* **UniQue :Index**
* **db.animals.ensureIndex({name:1},{unique:true})**
* db.system.indexes.find({ns:'test.animals'}).pretty()
* db.animals.insert({name:'cat'}) ..Since cat already exist and

unique index is there on Name so this insert will not be allowed

* **Creating Index on multiple Field**
  + db.animals.ensureIndex({tags:1,name:1})
  + db.animals.find({name:'shark',tags:'ocean'}).explain()
  + db.animals.find({tags:'ocean',name:'shark'}).explain() ..reverse order
  + db.animals.find({name:'shark'}).explain() ..no index will be used .The index is deemed useful if the terms in the index match the index definition from left to right.
* **Usage of index with sort** 
  + **drop all indexes then create again**
  + **db.animals.ensureIndex({tags:1,name:1})**
    - When we created the index, we said what sort direction the terms should be with each field. In this case, ascending tags and then ascending name.
* db.animals.find().sort({tags:1,name:1}).explain()
* db.animals.find().sort({tags:-1,name:-1}).explain()
* db.animals.find().sort({tags:1,name:-1}).explain() ..mongo will not use Index
* db.animals.find().sort({tags:-1,name:1}).explain() ..mongo will not use Index
  + **Unwanted Index: Best way is to use explain()**
    - MongoDB is schema-less, so Mongo doesn't know what fields you might have in the future in the document. It also doesn't keep a central account of all fields present in all documents. So it's quite possible, and allowed, to create an index on a field name that doesn't match any field in any of your documents.
      * + db.animals.ensureIndex({zzz:1})
        + db.system.indexes.find({ns:'test.animals'},{ns:1,name:1})
* **Creating index in foreground and background**
  + When your database is quite large, you might want to add an index, you should be aware of a few things though. When you use the ensureIndex command without any options, the index is going to be built in the foreground. Mongo will bock all operations, all write and read operations until this index rebuilding is done.

Mongo will build the index in the background. While building in the background, read and write operations can continue.

* + - * + db.animals.ensureIndex({tags:1},{background:true})