**1 . Index:**

**Types of index**

**1. Single field index -> MongoDb can traverse in ascending and descending.**

**2. Compound Index -> db.collection.find().sort({"name":1,"city":1})**

**3.Multikey index -> it is used for indexing the values that are stored in the array.**

**4. Geospatial index -> it supports 2 types of indexing 1. 2D index 2. 2D sphere**

**5. Text index -> for searching the string in collection ex. db.collection.createIndex({name: "text"})**

**The createIndex() Method**

**Syntax**

**The basic syntax of createIndex() method is as follows().**

**>db.COLLECTION\_NAME.createIndex({KEY: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})**

**The dropIndex() method**

**You can drop a particular index using the dropIndex() method of MongoDB.**

**Syntax**

**The basic syntax of DropIndex() method is as follows().**

**>db.COLLECTION\_NAME.dropIndex({KEY:1})**

**Example:**

**db.mycol.dropIndex({"title":1})**

**The getIndexes() method**

**This method returns the description of all the indexes int the collection.**

**Syntax**

**Following is the basic syntax od the getIndexes() method −**

**db.COLLECTION\_NAME.getIndexes()**

**Example**

**Assume we have created 2 indexes in the named mycol collection as shown below −**

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

**Following example retrieves all the indexes in the collection mycol −**

**> db.mycol.getIndexes()**

**2. Index and SubIndex:**

**we need to insert the following document in the collection named users as shown below −**

db.users.insert(

{

"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.

**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.

To create an index on tags array, use the following code −

>db.users.createIndex({"tags":1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 2,

"numIndexesAfter" : 3,

"ok" : 1

}

>

**After creating the index, we can search on the tags field of the collection like this −**

> db.users.find({tags:"cricket"}).pretty()

{

"\_id" : ObjectId("5dd7c927f1dd4583e7103fdf"),

"address" : {

"city" : "Los Angeles",

"state" : "California",

"pincode" : "123"

},

"tags" : [

"music",

"cricket",

"blogs"

],

"name" : "Tom Benzamin"

}

>

**To verify that proper indexing is used, use the following explain command −**

>db.users.find({tags:"cricket"}).explain()

This gives you the following result −

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "mydb.users",

"indexFilterSet" : false,

"parsedQuery" : {

"tags" : {

"$eq" : "cricket"

}

},

"queryHash" : "9D3B61A7",

"planCacheKey" : "04C9997B",

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"tags" : 1

},

"indexName" : "tags\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"tags" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"tags" : [

"[\"cricket\", \"cricket\"]"

]

}

}

},

"rejectedPlans" : [ ]

},

"serverInfo" : {

"host" : "Krishna",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

>

The above command resulted in "cursor" : "BtreeCursor tags\_1" which confirms that proper indexing is used.

**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})

{

"numIndexesBefore" : 4,

"numIndexesAfter" : 4,

"note" : "all indexes already exist",

"ok" : 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"}).pretty()

{

"\_id" : ObjectId("5dd7c927f1dd4583e7103fdf"),

"address" : {

"city" : "Los Angeles",

"state" : "California",

"pincode" : "123"

},

"tags" : [

"music",

"cricket",

"blogs"

],

"name" : "Tom Benzamin"

}

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"}).pretty()

{

"\_id" : ObjectId("5dd7c927f1dd4583e7103fdf"),

"address" : {

"city" : "Los Angeles",

"state" : "California",

"pincode" : "123"

},

"tags" : [

"music",

"cricket",

"blogs"

],

"name" : "Tom Benzamin"

}

>