Mongodb is schema less, means each and every row or document is having different number of columns

# Mongo notes

1st row or 1st document having 2 columns with int,string

2nd row or 2 document is having 3 columns String,String,String

3rd row or document can be an embedded document

1. Unless there is a space in the key we don’t need to keep the key in double quotes

db.flights.insertOne({name:””}); -- here no double quotes in the key

1. {} empty braces without condition means select all ex:- db.flights.deleteMany({}); this will delete all
2. Double or singlequotes db.flights.insertOne({name:””}); here single quotes or double quotes doesn’t make any difference , because shell is based on java script
3. We can give any value to \_id field ex:- db.passengers.insertOne( {\_id : “p123654”} );
4. Note 1 document size could be max 16 MB even if u have inner docs or not , so don’t embedded millions of messages inside a single document
5. for any function we have to pass object only like for find method …and for every method we have to pass object only

db.employees.find( {key:value, operator: abcd} );

1. query- **Show dbs**

To see the list of dbs

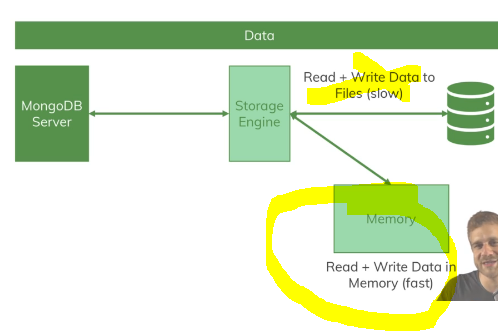
And to use certain db use this certain query **– use dbName**

1. query- **Show collections**

To see list of available collections

1. Use shops

To create a database and use it – if this database is not there it will just create one and switch to that



If mongo always interact with file system it will be literally be slow

Mongo Server will interact with storage engine and most latest used data will be maintained in the memory instead of file

|  |  |
| --- | --- |
| In RDBMS table | In mongodb it is called Collection |
| Row | document |
|  |  |

References

* Good one- Mongo db university course 15 hours java <https://learn.mongodb.com/catalog>

<https://learn.mongodb.com/learn/learning-path/mongodb-java-developer-path>

* Learn more about the MongoDB Drivers: <https://docs.mongodb.com/ecosystem/drivers/>
* Dive into the official Getting Started Docs: <https://docs.mongodb.com/manual/tutorial/getting-started/>
* You can find all limits (in great detail) here: <https://docs.mongodb.com/manual/reference/limits/>
* For the data types, MongoDB supports, you find a **detailed overview** on this page: <https://docs.mongodb.com/manual/reference/bson-types/>
* The MongoDB Limits: <https://docs.mongodb.com/manual/reference/limits/>
* The MongoDB Data Types: <https://docs.mongodb.com/manual/reference/bson-types/>
* More on Schema Validation: <https://docs.mongodb.com/manual/core/schema-validation/>

Helpful Articles/ Docs:

* insertOne(): <https://docs.mongodb.com/manual/reference/method/db.collection.insertOne/>
* insertMany(): <https://docs.mongodb.com/manual/reference/method/db.collection.insertMany/>
* Atomicity: <https://docs.mongodb.com/manual/core/write-operations-atomicity/#atomicity>
* Write Concern: <https://docs.mongodb.com/manual/reference/write-concern/>
* Using mongoimport: <https://docs.mongodb.com/manual/reference/program/mongoimport/index.html>
* <https://www.mongodb.com/docs/manual/reference/operator/query/>

Helpful Articles/ Docs:

* More on find(): <https://docs.mongodb.com/manual/reference/method/db.collection.find/>
* More on Cursors: <https://docs.mongodb.com/manual/tutorial/iterate-a-cursor/>
* Query Operator Reference: <https://docs.mongodb.com/manual/reference/operator/query/>

Shell specific

new Date()

new TimeStamp() are shell specific objects

generally for number field javascript accepts 64 bit floating point value like maximum 12345678901234567000

if u want to store super big numbers store as string

Predefined Functions

db.laptop.insertOne({lapName:”Lenovo”, lapInchSize: NumberInt(3)});

Arrays

Inside arrays you can keep string or Json objects

If u want to add an array to the existing document

db.updateOne(filter,data)

db.passengers.updateOne({"name": "Gordon Black"},{$set : {hobbies:["carom","Chess"]}});

After updating its looking like this

{

\_id: ObjectId("63aaeb3f03c92ae3e009dc80"),

name: 'Gordon Black',

age: 38,

**hobbies: [ 'carom', 'Chess' ]**

}

Searching arrays

It is also like searching inside inner document

db.users.find({"hobbies.title":"sports","hobbies.frequency":{$gte:17} });

[ {\_id: ObjectId("64167bd315ae4ae7e928cde8"), name: 'manideep',

hobbies: [

{ title: 'sports', frequency: 16 },

{ title: 'badminton', frequency: 26 }

]}]

Searching arrays with size

U can only search arrays with exact size, mongodb wont supports searching with arrays gte/lte

shop> db.users.find({hobbies:{$size:2}}); //this will work because you are searching with exact operator

[{ \_id: ObjectId("64167bd315ae4ae7e928cde7"), name: 'Santhoshi',

hobbies: [{ title: 'sports', frequency: 6 }, { title: 'cricket', frequency: 16 } ] },

db.users.find({hobbies:{$size:{$gte:3}}});

MongoServerError: Failed to parse $size. Expected a number in: $size: { $gte: 3 }

See here mongo can search only exact size ,it wont support lte or gte

Array operators

$all –search with all array contents with shuffling

The problem is if u want to search with array constraints

shop> db.users.find({hobbies:["badminton","shuttle"]});

the problem is if u search with badminton and shuttle u will get only this not shuttle and badminton so that’s why u should use $all operator

[ { \_id: ObjectId("64168b2415ae4ae7e928cdf1"),name: 'mohit', hobbies: [ 'badminton', 'shuttle' ] }}

Solution with $all operator

shop> db.users.find({hobbies:{$all: ["badminton","shuttle"]}});

[{ \_id: ObjectId("641683e315ae4ae7e928cdeb"), name: 'mani'**, hobbies: [ 'shuttle', 'badminton' ]** },

{ \_id: ObjectId("64168ae115ae4ae7e928cded"), name: 'mani',**hobbies: [ 'shuttle', 'badminton' ]},**

{\_id: ObjectId("64168af115ae4ae7e928cdef"),name: 'sai',**hobbies: [ 'shuttle', 'badminton' ]** },

{\_id: ObjectId("64168b2415ae4ae7e928cdf1"), name: 'mohit', ***hobbies: [ 'badminton', 'shuttle' ]***}

]

1. $elemMatch

This is for array elements having inner documents

Element match means all the conditions specified here must be satisfied by a single json object of an array

Definition:-

{<arrayField>:{$elemMatch:{condition 1:condition 2:condition:3}}}

At least one nested document inside an array must satisfy all the conditions , order of those conditions doesn’t matter

Example:1

db.users.find({hobbies:{$elemMatch:{title:'sports',frequency:{$gte:10}}}});

[ { \_id: ObjectId("64167bd315ae4ae7e928cde8"), name: 'manideep',

hobbies: [

{ title: 'sports', frequency: 16 },

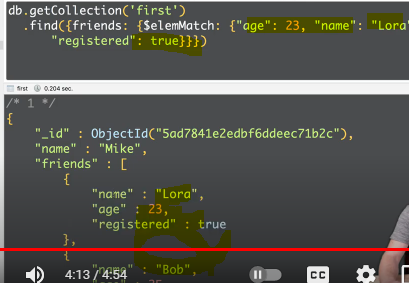
{ title: 'badminton', frequency: 26 }

] }]

//here also all the 2 conditions specified are matched by an single json document of an array

Though the array consists 2 docs where the other document didn’t satisfied still it fetched the entire main document

// element match says it will fetch a document which satisfies All the conditions mentioned under element Match must be satisfied by a single documents inside an array at least one array document must satisfy

here if u see that single document contains many json objects ,1 json objects satisfied all the 3 conditions and hence it fetched the entire documents which consists of entire array ,but I thought only that array single element will be fetched ,instead it fetc

It says compare all the elements of same json document/compare element/json object wise

Generally in an array, if we have an array of json objects see the below example,

Inside each json objects we will have many elements, the problem with and operator is it will compare along multiple documents

db.users.find({"hobbies.title":"sports","hobbies.frequency":{$gte:10}});

here I asked for title-sports and frequency-gte 10 ,but if u see below it fetch a document with frequency 6, what was happened was hobbies is an array and it has 2 json objects it searched for sports in 1 document and frequency in another document of that array instead of so this

element match will tell to mongo search my criteria in single json object of an array

[ { \_id: ObjectId("64167bd315ae4ae7e928cde7"), name: 'Santhoshi',

hobbies: [

{ title: 'sports', frequency: 6 },

{ title: 'cricket', frequency: 16 } ]},

{ \_id: ObjectId("64167bd315ae4ae7e928cde8"), name: 'manideep',

hobbies: [

{ title: 'sports', frequency: 16 },

{ title: 'badminton', frequency: 26 } ]}]

//Now no need to use and operator also u can use elemMatch

shop> db.users.find({hobbies:{$elemMatch:{"title":"sports","frequency":{$gte:10} }} });

now u will see u got an single element whse title is sports and frequency is 16 it searched the single json object of an array

[ {

\_id: ObjectId("64167bd315ae4ae7e928cde8"),

name: 'manideep',

hobbies: [

{ title: 'sports', frequency: 16 },

{ title: 'badminton', frequency: 26 }

] }]

Example:-2

shop> db.movies.count({genres:['Drama','Horror']});

2

shop> db.movies.find({},{genres:1,\_id:0}).limit(100);

{ genres: [ 'Drama', 'Science-Fiction', 'Thriller' ] },

{ genres: [ 'Drama', 'Action', 'Crime' ] },

{ genres: [ 'Drama', 'Horror', 'Romance' ] },

{ genres: [ 'Drama', 'Fantasy', 'Horror' ] },

{ genres: [ 'Drama', 'Action', 'Horror' ] },

{ genres: [ 'Drama', 'Horror', 'Thriller' ] },

{ genres: [ 'Drama', 'Horror', 'Thriller' ] }

{ genres: [ 'Drama', 'Action', 'Science-Fiction' ] },

{ genres: [ 'Drama', 'Crime', 'Thriller' ] },

{ genres: [ 'Action', 'Adventure', 'Science-Fiction' ] },

hed that entire document

Tasks

1)Import the attached data file into a new collection (e.g. exmoviestarts) in the boxOffice database

2)Find all movies with exactly two genres

Use boxoffice-extended.json for data

shop> db.exmoviestars.find({genre:{$size:2}});

[ { \_id: ObjectId("6416c4b1157114eab4e89fe3"), title: 'Supercharged Teaching',

meta: { rating: 9.3, aired: 2016, runtime: 60 },

visitors: 370000, expectedVisitors: 1000000, genre: [ 'thriller', 'action' ], ratings: [ 10, 9, 9 ] },

3)Find all movies which aired in 2018

db.exmoviestars.find({"meta.aired":2016});

4)Find all movies which have ratings greater than 8 but lower than 10

shop> db.exmoviestars.find( {ratings:{$elemMatch: {$gt:8,$lt:10} }} );

[ { \_id: ObjectId("6416c4b1157114eab4e89fe2"),

title: 'The Last Student Returns', meta: { rating: 9.5, aired: 2018, runtime: 100 },

visitors: 1300000, expectedVisitors: 1550000, genre: [ 'thriller', 'drama', 'action' ],

ratings: [ 10, 9 ]

},

{\_id: ObjectId("6416c4b1157114eab4e89fe3"),title: 'Supercharged Teaching',

meta: { rating: 9.3, aired: 2016, runtime: 60 }, visitors: 370000,

expectedVisitors: 1000000,

genre: [ 'thriller', 'action' ],

ratings: [ 10, 9, 9 ]

}

]

CreateCollection with constraints

In RDBMS we generally create collections with constraints like not null, unique key

Similarly in mongo dB also we can create collections with not null and column constraints like creating column we can define the datatype so that we can give only that type of data

Kindly check the file named script or “**creating collections with constraints.js”**

**In the below we created a Collection called Employee with 2 mandatory fields empId,empName**

db.createCollection('Employee',{

validator:{

$jsonSchema:{

bsonType:'object',

required:['empId','empName'],

properties:{

title:{

bsonType:'string',

description: 'Title must be a string cant be null'

}}}}}); //similarly declare for all fields

Example 2:-

db.createCollection('user',{

validator:{

$jsonSchema:{

bsonType:'object',

required: ['userName','sex'],

properties: {

userName:{

bsonType:'string',

description:'mandatorieee'

}

}

}

}

});

If any of above 2 columns are missing and even if we gave wrong data type then insertion will fail

As below Uncaught:

MongoServerError: Document failed validation

Additional information: {

failingDocumentId: 123,

details: {

operatorName: '$jsonSchema',

CRUD operations

Delete

deleteOne(filter,options);

deleteMany(filter,options)

Embedded documents

U can have a document inside a document, such total 1 document can have totally 300 embedded documents inside it, but maximum of 16MB only in total for a collection

Db.flights.updateMany(filter,data);

If u want to update general field without embedded document

db.flights.updateMany({} ,{$set: {engineDetails: “garuda”}});

db.flights.updateMany({} ,{$set: {engineDetails: {horsepower:1600, engineName: ‘garuda’} }});

Result as

{ "\_id": "i1", "flightName": "Indigo", "capcacity": 200, "manufacturedBy": "Tata", "wheels": 4,

**"engineDetails": {**

**"horsepower": 1600,**

**"engineName": "garuda"**

}

}

Delete collection

Db.collectionName.drop()

Ex:- db.Employee.drop() This table will be dropped

deleteMany

db.collectionName.deleteMany({}) means it will delete everything all the rows

Find

* More on find(): <https://docs.mongodb.com/manual/reference/method/db.collection.find/>

db.collection.find(query, projection, options)

1. To fetch all records use find method with no condition

db.products.find().pretty(); -- as no condition was given it will fetch all records

db.products.find({}).pretty(); -- as no condition was given it will fetch all records

lets say if this find method is giving 20,000 lakh records, but in general find will not give all records at once , it will just gives us the cursor

1. find(filter,Options)

findOne(filter,Options) // it will gives only 1st matching element

here options are nothing but projections, see projections for more details

shop> db.movies.find({},{weight:1,\_id:0}); //now in this case only weight column will be displayed as below

shop> db.movies.find({},{weight:1,\_id:0});// by default \_id field will be displayed and if we don’t want that then use \_id:0 so that this field will not be displayed

//results

{}, { weight: 91 },

{ weight: 96 }, { weight: 75 },

{ weight: 99 }, { weight: 99 },

db.flights.find({intercontinental:true});

output :-

[ { \_id: ObjectId("63a9c0a20d7ed583364551e0"), departureAirport: 'MUC',

arrivalAirport: 'SFO', aircraft: 'Airbus A380', distance: 12000, intercontinental: true }]

find with $gt

find all flights whose distance is greater than 1000

db.flights.find( {distance: {$gt:900} } ); //gives 2 docs

db.flights.findOne( {distance: {$gt:900} } ); //gives 1 doc

Accessing inner fields

Here we accessed inner field alone

db.flights.findOne({"\_id":"Air1"}).engineDetails;

|  |
| --- |
| { horsepower: 1600, engineName: 'garuda'} |

Now we can access horsepower field also

db.flights.findOne({"\_id":"Air1"}).engineDetails.horsepower;

Searching with array field

This is the actual document

db.flights.findOne({"\_id":"Air1"});

|  |
| --- |
| { \_id: **'Air1'**, |
| flightName: **'AirIndia'**, |
| capcacity: 250, |
| manufacturedBy: **'CTS'**, |
| designedBy: **'Accenture'**, |
| wheels: 4, |
| |  | | --- | | engineDetails: { horsepower: 1600, engineName: **'garuda'** }, | | stops: [ **'hyd'**, **'chennai'** ] } | |

}

Query array **contents : here sops is an array field**

**db.flights.findOne({stops: 'hyd'});**

|  |
| --- |
| { \_id: **'Air1'**, |
| flightName: **'AirIndia'**, |
| capcacity: 250, |
| manufacturedBy: **'CTS'**, |
| designedBy: **'Accenture'**, |
| wheels: 4, |
| engineDetails: { horsepower: 1600, engineName: **'garuda'** }, |
| stops: [ **'hyd'**, **'chennai'** ] }  searching with inner elements |

db.flights.findOne({"engineDetails.horsepower":1600});

Querying with inner fields

db.movies.findOne({"rating.average":7.8});

this will give the entire document

Searching with date

Searching with date

{createdTimeStamp:{$gte:ISODate(“2023-03-31”}}

Searching with date & timezone

{createdTimeStamp:{$gte:ISODate(“2023-03-31T04:00:00Z”)}}

1. Searching records where createdTimeStamp is today

{createdTimeStamp:{$gte:ISODate(“2023-03-31”}}

// this will fetch the records greater than 2023-03-21 T00:00:01 after 12:00 AM after 1 second

it will fetch all the records create on that day after 12:00AM after 12:00Am

Here the problem is while fetching the timezone ,

Main issue:- what ever u do the date will be converted into GMT time

Lets say record created now 25th march 1:00 PM mongo will calculate the current time in London GMT so it will be stored as 25th march 7:30AM will be stored but u want to filter based on

Lets say u want to fetch based on US Time EST which is UTC-4:00 then u can search like below

{createdTimeStamp:{$gte:ISODate(“2023-03-31T04:00:00Z”)}}

cursors

find() method could return millions of documents, hence generally that method returns a cursor

instead of returning millions of documents, generally even in a website also there would be lot many matching records but anyways we don’t display them at all , so better find method also returns a cursor

const c=db.movies.find();

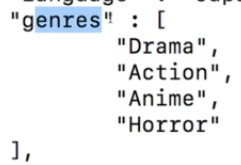
c.hasNext();// this will return true

### Querying array elements

If u are searching with array elements, you should not use equality operator, u may need to use $in operator

shop> db.movies.find({ genres:["Drama"] },{genres:1});

{ \_id: ObjectId("63cc1be06e68caf55b8641b7"), genres: [ 'Drama' ] },

{ \_id: ObjectId("63cc1be06e68caf55b8641e7"), genres: [ 'Drama' ] },

If u don’t use in operator it will internally use eq operator

Insert

Even though there is no collection named products – it will create one

1. insertOne(data,Options)
2. insertMany(data,Options)

Unless there is a space in the key we don’t need to keep the key in double quotes

InsertOne

No need to create collection, directly if u insert if document is collection is present it will add this document

Else it will create the collection and add this document

db.products.insertOne({product:”Book”,”cost”:12.99 })

here u can give \_id also – and u don’t need to give objectId type and type is fine 1 record can be of ObjectId type another maynot

db.flights.insertOne( {\_id:”i1”,flightName: “Indigo”,capcacity:200, manufacturedBy: “Tata” });

InsertMany

insertMany(Array arr) this function expects to pass an array

**db.flights.insertMany(**

[ { "departureAirport": "MUC", "arrivalAirport": "SFO", "aircraft": "Airbus A380", "distance": 12000, "intercontinental": true },

{ "departureAirport": "LHR", "arrivalAirport": "TXL", "aircraft": "Airbus A320", "distance": 950, "intercontinental": false }]**) ;**

InsertOne With Arrays of objects

Db.companies.insertOne({\_id:"c2", name:"Fresh Apple Inc","isStartUp":true, employees:33,funds:01234567890123456798,locations:[ {hyderabad:50000,mainCampus:"deccan park"} , {chennai:60000,mainCampus:"siruseri"}], foundedDate: new Date()});

|  |
| --- |
| { acknowledged: true, insertedId: 'c2' }  Result is |

{

\_id: 'c2',

name: 'Fresh Apple Inc',

isStartUp: true,

employees: 33,

funds: 1234567890123456800,

locations: [

{ hyderabad: 50000, mainCampus: 'deccan park' },

{ chennai: 60000, mainCampus: 'siruseri' }

],

foundedDate: ISODate("2022-12-28T06:25:04.409Z")

}

Inserting all records parallelly With ordered

Db.collName.insertMany(document , options)

While doing insertMany if u are trying to insert 10 records, if 1st record fails (due to any issue like duplicate key exception), remaining 9 records will not be inserted,

But if u want all records to be inserted parallelly irrespective of failing of previous records then use the key order:false option , so that even if the first record fails it doesn’t impact, 9 records will be inserted paralelly

shop> db.pen.insertMany([{\_id:'Renault', cost:3},{\_id:'Hero',cost:4}]);

db.pen.insertMany([{\_id:'Renault', cost:31}},{\_id:'HeroCycle',cost:41} ]); //Because same id

now as per above I will try to insert Renault pen it will throw error because same id, but the problem is it will not insert the remaining document

db.pen.insertMany([{\_id:'Renault', cost:31},{\_id:'HeroCycle',cost:41} ], { ordered:false});

writeConcernErrors: [],

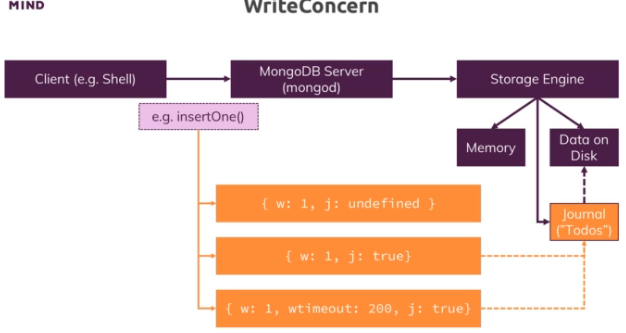
insertedIds: [ { index: 0, \_id: 'Renault' }, { index: 1, \_id: 'HeroCycle' } ],

nInserted: 1,

nUpserted: 0,

Now in this case also it will throw error but it will successfully insert second document eventhough first document failed

WriteConcern



Mongo Import

Where tvshows.json is the file to import

Mongoimport tvshows.json -d dbname –c collectionName –jsonArray –drop

Limit

limit(int i);

db.movies.find().limit(10); in this case only 10 will be retrived

Operators

|  |  |
| --- | --- |
| [$eq](https://www.mongodb.com/docs/manual/reference/operator/query/eq/#mongodb-query-op.-eq) | Matches values that are equal to a specified value. |
| [$gt](https://www.mongodb.com/docs/manual/reference/operator/query/gt/#mongodb-query-op.-gt) | Matches values that are greater than a specified value. |
| [$gte](https://www.mongodb.com/docs/manual/reference/operator/query/gte/#mongodb-query-op.-gte) | Matches values that are greater than or equal to a specified value. |
| [$in](https://www.mongodb.com/docs/manual/reference/operator/query/in/#mongodb-query-op.-in) | Matches any of the values specified in an array. |
| [$lt](https://www.mongodb.com/docs/manual/reference/operator/query/lt/#mongodb-query-op.-lt) | Matches values that are less than a specified value. |
| [$lte](https://www.mongodb.com/docs/manual/reference/operator/query/lte/#mongodb-query-op.-lte) | Matches values that are less than or equal to a specified value. |
| [$ne](https://www.mongodb.com/docs/manual/reference/operator/query/ne/#mongodb-query-op.-ne) | Matches all values that are not equal to a specified value. |
| [$nin](https://www.mongodb.com/docs/manual/reference/operator/query/nin/#mongodb-query-op.-nin) | Matches none of the values specified in an array. |
| [$and](https://www.mongodb.com/docs/manual/reference/operator/query/and/#mongodb-query-op.-and) | Joins query clauses with a logical AND returns all documents that match the conditions of both clauses. |
| [$not](https://www.mongodb.com/docs/manual/reference/operator/query/not/#mongodb-query-op.-not) | Inverts the effect of a query expression and returns documents that do not match the query expression. |
| [$nor](https://www.mongodb.com/docs/manual/reference/operator/query/nor/#mongodb-query-op.-nor) | Joins query clauses with a logical NOR returns all documents that fail to match both clauses. |
| [$or](https://www.mongodb.com/docs/manual/reference/operator/query/or/#mongodb-query-op.-or) | Joins query clauses with a logical OR returns all documents that match the conditions of either clause.   |  |  | | --- | --- | | [$exists](https://www.mongodb.com/docs/manual/reference/operator/query/exists/#mongodb-query-op.-exists) | Matches documents that have the specified field. | | [$type](https://www.mongodb.com/docs/manual/reference/operator/query/type/#mongodb-query-op.-type) | Selects documents if a field is of the specified type. | | [$text](https://www.mongodb.com/docs/manual/reference/operator/query/text/#mongodb-query-op.-text) | Performs text search. | | [$where](https://www.mongodb.com/docs/manual/reference/operator/query/where/#mongodb-query-op.-where) | Matches documents that satisfy a JavaScript expression. | |

Example:-

db.movies.find({runtime: {$lt :30}}).count();

Only thing is everything is an object for find method we should pass an Object, here key is string and value should also be an object

**db.movies.find({runtime: {$eq :30}}).count();**

**62**

**db.movies.find({runtime :30}).count(); both above statements are same , even if u don’t gove eq** operator those are same

db.movies.find({runtime:{$lte:30}}).count();

**in operator**

**db.movies.find({ runtime: {$in:[42,60]} },{runtime:1});**

**now it will** find elements which have runtime value as 42,60

{ \_id: ObjectId("63cc1be06e68caf55b86418a"), runtime: 60 },  
**or operator**

db.movies.find({ $or:[ {"rating.average":{$lt:5}}, {"rating.average":{$gt:9.3} } ] });

Here everything is an object for that find method we have to pass an object , it can be an array of objects also

Here every json object will be having a key and value here key is operator and value is an object

**Lte operator**

db.movies.find( { "rating.average":{$lt:5 } }).count();

**gte operator**

shop> db.movies.find( { "rating.average":{$gte:8 } }).count();

112

**And operator**

And is the default mechanism of mongo db

If u pass a json object with multiple fields by default it will internally uses and operator

{

“first name”: “manideep”,

“sur name”: “kumar”

}

Here we have a json object if we pass this to find method it will internally uses and operator

As “**Where “first name”: “manideep” and “sur name”: “kumar”** “

You can do it in 2 ways

1. **using $and operator**

everywhere we should pass json object key,value ,if value is multiple keep it inside an object

shop> db.movies.find ( { $and : [ {"rating.average": {$gt : 8} },{ "genres":"Drama" } ] } ) .count();

74

1. without using $and

shop> db.movies.find( { "rating.average" : {$gt:8} , "genres": "Drama"} ) .count();

74

Simple for find method we have to pass json object

1. **Using $not operator**

shop> db.movies.find( {runtime: 60} ).count();

170

shop> db.movies.find( {runtime: { $not: {$eq: 60} } } ).count();

71

The below is the easiest way to not equal to short hand operator

shop> db.movies.find( {runtime: { $ne:60 } } ).count();

71

Element operators

1. $Exists

Some documents may have some fields and some may not have that field,

Some docs may have phone number and some may not have

Lets say some documents have age field and some other may not have this age field

shop> db.movies.find( { updated: {$exists:true } } ).count( );

240 // here we are finding documents which contains the field **updated**

shop> db.movies.find( ).count( );

if u want to fetch documents who have phone number field

db.movies.insertMany([{name:"manideep",**age:29**,address:"Hyd"},{name:"manideep",address:"Hyd"}]); //here in second document age is not there

in the above collection 1 doc only have the age, so use below query to fetch that document

shop> db.persons.find({age:{$exists:true}});

shop> db.persons.find({address:{$exists:false} });

[ { \_id: ObjectId("6415b95540a9fadedfe65f2c"), name: 'charan', age: 21 },

1. $ne:null

shop> db.persons.find({age:{$ne:null}});

it will retrieve the documents where age not equal to null

age not equal to 60

shop> db.persons.find({age:{$ne:26} }); //result is below

[ { \_id: ObjectId("6415a48d40a9fadedfe65f28"), name: 'manideep', age: 29, address: 'Hyd' },

{ \_id: ObjectId("6415a48d40a9fadedfe65f29"), name: 'manideep', address: 'Hyd' },

1. $ne:null and $gt

Below query fetches the result records whose age>24 and address nit equal to null

shop> db.persons.find( { $and:[ {age:{$gte:24}},{address: {$ne:null}} ] } );

[ { \_id: ObjectId("6415a48d40a9fadedfe65f28"), name: 'manideep', age: 29, address: 'Hyd' },

1. $and + $eq + $gt

age>21 and persons from kavali

db.persons.find( { $and:[ {age:{$gte:21} },{address:"kavali"} ] } );

[ { \_id: ObjectId("6415b81c40a9fadedfe65f2b"), name: 'mohan', address: 'kavali', age: 28 },

shop> db.persons.find( {$and: [ {address:'Hyd'}, {age:{$exists:true}} ] });

[ { \_id: ObjectId("6415a48d40a9fadedfe65f28"), name: 'manideep', age: 29, address: 'Hyd' }

]

1. $and +$gte+ $lte

Use the json content present in the json “20.boxoffice-json data”

Inside every json object, we should pass key, value is json object, here $and is the key, Array is the value

1) Search all movies that have a rating higher than 9.2 and a runtime lower than 100 minutes

shop> db.boxoffice.find({ *$and*: **[{"meta.rating":{$gte:9.2}} ,{"meta.runtime":{$lte:100}}]** });

[ { \_id: ObjectId("6415de8abd2796ece6015bcb"), title: 'The Last Student Returns',

meta: { rating: 9.5, aired: 2018, runtime: 100 }, visitors: 1300000,

expectedVisitors: 1550000,genre: [ 'thriller', 'drama', 'action' ] },

1. Or operator $or

Search all movies that have a genre of "drama" or "action"

db.boxoffice.find({$or:[ {genre:"drama"}, {genre:"action"}]});

1. $regex

Now this will find names contain “sh” like it will fetch names –suresh,mahesh

shop> db.persons.find({name:{$regex: /sh/}});

[ { \_id: ObjectId("6415d80140a9fadedfe65f2f"), name: 'suresh',age: 45

1. $expr

If u want to compare 2 fields in a document then use this, here inside every document I wanted to compare visitors field with expected visitors field if it is greater than I wanted to fetch it so ..

db.boxoffice.find( {$expr:{ $gt:["$visitors","$expectedVisitors"]} } );

Search all movies where visitors exceeded expectedVisitors

[ { \_id: ObjectId("6415de8abd2796ece6015bcd"),title: 'Teach me if you can',

meta: { rating: 8.5, aired: 2014, runtime: 90 }, **visitors: 590378,expectedVisitors: 500000**,

genre: [ 'action', 'thriller' ]}]

1. $inc operator

Ex:- update the salaries by some 100, instead of fetching and modifying the count u can use this operator which will directly update the count

shop> db.users.updateOne({name:'Manuel'},{$inc:{phone:1},$set:{isSporty:false }});

working $inc,$set on same field like below is not possible,

db.users.updateOne({name:'Manuel'},{$inc:{phone:1},$set:{phone:false} });

MongoServerError: Updating the path 'phone' would create a conflict at 'phone'

1. $min

db.Employees.updateOne({name:'Anna'}, {$min:{age:2}});

if the value( age) here mentioned is lesser than that of existing document then it will be replaced

1. $Max -- $min --$mul (multiply)

If the value mentioned inside max is maximum, than that of existing document then it will be replaced

//here if the mentioned value is greater than that of existing document, hence it was replaced

shop> db.Employees.updateOne({'name':'Anna'},{$max:{age:25}});

{

acknowledged: true,

insertedId: null,

matchedCount: 1,

modifiedCount: 1,

upsertedCount: 0

}

1. $ unset –Get rid of fields

Lets say u don’t like a field called phone number and u don’t want that field to see in ur document

1. Setting to null –doesn’t work ,because field will still be there only value will be null

**db.Employees.updateOne({name:'Max'},{$set:{age:null } });**

now field will still be visible

1. Use unset operator –so that field it self will become invisible

**db.Employees.updateOne({name:'Max'},{$unset:{age:""}});**

Now the field will be invisible

1. $ rename- rename a field

**db.Employees.updateMany({},{$rename:{"age":"totalAge"}})**

it will rename all the fields matching with that criteria

{

acknowledged: true,

insertedId: null,

matchedCount: 4,

modifiedCount: 2,

upsertedCount: 0

}

### Pretty

U can call this function only on a cursor ex:-

db.find().pretty() – works because find returns a cursor

db.findOne().pretty(); - gives exception because findOne doesnot return a cursor

db.insertOne().pretty(); gives exception because insertOne method is not returning any data hence cursor will not be returned

cursor will be returned only in case when you are returning huge data

### Projections

Means selecting only few columns ,if database is having 100-200 columns if u don’t want all of them to be fetched then fetch only some columns, need means 1 ,If u don’t require that column give 0

Db.flights.find(filter,options);

Db.flights.find({},{name:1, \_id: 0 }); // since I don’t want to filter I didn’t gave any condition

By default \_id column will be fetched if u don’t want that to be fetched, hence I gave 0 wantedly

We can use projections on inner documents

Replaceone

While replacing it will maintain \_id alone and it will replace entire all fields with this field

Db.flights.replaceOne(condition,data);

db.flights.replaceOne(data);

//practice here

shutdownServer

db.shutdownServer(); //means shutdown the mongo db server

Sort

Sort is available only on cursor

Find() method give us a cursor whereas findOne()doesn’t return much data so it wont return cursor

Db.movies.find().sort({“ratings.average”:-1,runtime:-1}).pretty();

Here first sorting will happen

db.movies.find().sort({runtime:-1}).limit(1);

Skip

Skip(100) is used to skip the results in a result set, especially used in pagination

shop> db.movies.find().skip(100); it will skip starting 100 records and fetches remaining ones

shop> db.movies.find().sort({runtime:-1}).skip(10).limit(1)

here first it will sort by runtime and then it will skip first 10 results and limits /fetches only 1 record

Update

If u have partial modifications then prefer this

1. updateOne(filter,data,options)
2. replaceOne(filter,data,options)

Its Always recommended to use updateOne(), updateMany()

shop> db.flights.update({\_id:'i1'},{engine:'powerful'});

the problem was , since we are not using **$set** : {engine:'powerful'} here $set is missing so it was overriding all fields,means all fields are gone only this field will be set alone , whereas in updateOne(),updateMany() the $set symbol keyword is mandatory

as here the entire record is getting overridden seems this could have got deprecated.

DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.

MongoInvalidArgumentError: Update document requires atomic operators

UpdateOne

updateOne(filter,data,options)

1st argument is filter-where condition

db.flights.updateOne( {distance:12000}, {$set: { engine : ”double powered” } } );

db.flights.find();

**Before update**

There is no field called **designedBy**

{ \_id: 'Air1', flightName: 'AirIndia', capcacity: 250, manufacturedBy: 'CTS' },

**After running the query**

db.flights.updateOne({\_id:"Air1"}, {$set:{designedBy:"Accenture"}});

note:- if the field “designedBy “is present for that document it will update the key value else it will create that field key value pair for that document

[ { \_id: 'Air1', flightName: 'AirIndia', capcacity: 250, manufacturedBy: 'CTS', designedBy: Accenture' },

UpdateMany

1. updateMany(filter,data,options)

To select all documents without any condition then use empty flower braces {}

db.flights.updateMany({}, {$set: {wheels:4} } );

Here since first option is having empty braces {} means no condition it will select all records and it will update all records if there is a field called wheels in a document it will update else it will add that field to that document

{ acknowledged: true, insertedId: null, matchedCount: 3, modifiedCount: 3, upsertedCount: 0 }

In above already in that collection only 3 records are there before running that query, now it selected all docs it updated all docs

After running the query it will look like this

{ \_id: 'i1', flightName: 'Indigo', capcacity: 200, manufacturedBy: 'Tata', **wheels: 4** },

{ \_id: 'Air1', flightName: 'AirIndia', capcacity: 250, manufacturedBy: 'CTS', designedBy: Accenture',

**wheels: 4** },

{ \_id: 'king1', flightName: 'KingFisher-1', capcacity: 50, manufacturedBy: 'IQA', **wheels: 4** }

If u want a new field to all existing records then refer below

1st param is {} because we want to select all records

db.flights.updateMany({},{$set :{details:null}});

so for all documents new fields called details will be added with null value

Ex :2:-

db.Employees.updateMany( {"hobbies.title":"Sports"}, {$set:{experience:20}} );

//find with sports and add an element to it

[ { \_id: ObjectId("641ffff115ae4ae7e928cdf6"), name: 'Max',

hobbies: [

{ title: 'Sports', frequency: 3 },

{ title: 'Cooking', frequency: 6 }

], phone: 131782734, experience: 20 },

Datatypes

**Important data type limits are:**

* Normal integers (int32) can hold a maximum value of +-2,147,483,647
* Long integers (int64) can hold a maximum value of +-9,223,372,036,854,775,807
* Text can be as long as you want - the limit is the 16mb restriction for the overall document

It's also important to understand the difference between int32 (NumberInt), int64 (NumberLong) and a normal number as you can enter it in the shell. The same goes for a normal double and NumberDecimal.

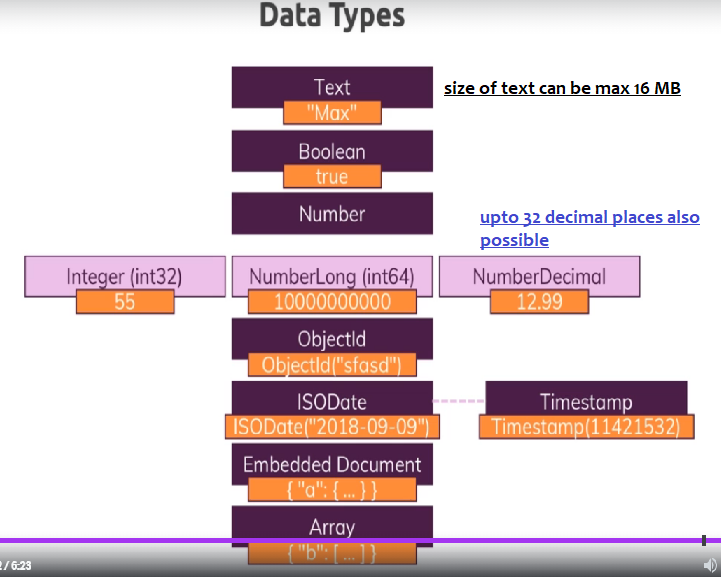
**NumberInt** creates a **int32** value => NumberInt(55)

**NumberLong** creates a **int64** value => NumberLong(7489729384792)

If you just use a number (e.g. insertOne({a: 1}), this will get added as a **normal double** into the database. The reason for this is that the shell is based on JS which only knows float/ double values and doesn't differ between integers and floats.

**NumberDecimal** creates a high-precision double value => NumberDecimal("12.99") => This can be helpful for cases where you need (many) exact decimal places for calculations.

When not working with the shell but a MongoDB driver for your app programming language (e.g. PHP, .NET, Node.js, ...), you can use the driver to create these specific numbers.



### Misc functions

##### 

This stats method can be called on database and on individual collection too

Db.stats(); entire database stats

Db.Flights.stats();

db.laptop.stats(); // here laptop is collection name

result is

{ ns: 'shop.laptop', size: 59, count: 1, avgObjSize: 59, numOrphanDocs: 0, storageSize: 4096,

freeStorageSize: 0, capped: false, wiredTiger: { metadata: { formatVersion: 1 },

operators

$eq

db.runtime.find({runtime: {$eq:60} });

db.runtime.find({runtime: 60}); this and above statement both are same

db.runtime.find({runtime: {$}});