**Show All cheatcode about mongodb**

**View all databases**

show dbs

**Create a new or switch databases**

use dbName

**View current Database**

db

**Delete Database**

db.dropDatabase()

**2. Collection Commands**

**Show Collections**

show collections

**Create a collection named 'comments'**

db.createCollection('comments')

**Drop a collection named 'comments'**

db.comments.drop()

**3. Row(Document) Commands**

**Show all Rows in a Collection**

db.comments.find()

**Show all Rows in a Collection (Prettified)**

db.comments.find().pretty()

**Find the first row matching the object**

db.comments.findOne({name: 'Harry'})

**Insert One Row**

db.comments.insert({

'name': 'Harry',

'lang': 'JavaScript',

'member\_since': 5

})

**Insert many Rows**

db.comments.insertMany([{

'name': 'Harry',

'lang': 'JavaScript',

'member\_since': 5

},

{'name': 'Rohan',

'lang': 'Python',

'member\_since': 3

},

{'name': 'Lovish',

'lang': 'Java',

'member\_since': 4

}])

**Search in a MongoDb Database**

db.comments.find({lang:'Python'})

**Limit the number of rows in output**

db.comments.find().limit(2)

**Count the number of rows in the output**

db.comments.find().count()

**Update a row**

db.comments.updateOne({name: 'Shubham'},

{$set: {'name': 'Harry',

'lang': 'JavaScript',

'member\_since': 51

}}, {upsert: true})

**Mongodb Increment Operator**

db.comments.update({name: 'Rohan'},

{$inc:{

member\_since: 2

}})

**Mongodb Rename Operator**

db.comments.update({name: 'Rohan'},

{$rename:{

member\_since: 'member'

}})

**Delete Row**

db.comments.remove({name: 'Harry'})

**Less than/Greater than/ Less than or Eq/Greater than or Eq**

db.comments.find({member\_since: {$lt: 90}})

db.comments.find({member\_since: {$lte: 90}})

db.comments.find({member\_since: {$gt: 90}})

db.comments.find({member\_since: {$gte: 90}})

///////////// my note

cls for clear window

for status //companyData> db.stats()

test2> db.flightdta.find({distance: 120}) // find all data

db.information.find({age: {$in: [30,42]}}).pretty()// all data between 30 to 40

db.information.find({age: {$nin: [30,42]}}).pretty() data missing between 30 to 40 age.

Using logical operator

db.tvchaneel.find({genres: "Drama", genres: "Horror" }) .count() // calculating count

db.tvchaneel.find() .count()

// (not equal to )

db.tvchaneel.find({runtime: {$ne: 60}}).count()

db.useres.find({age:{$exists: age, $ne :null}}).pretty()// use of exits

**Assignment Sections**

**Assignment 02**

**db.movies.find({"meta.rating":{$gt:9.2},"meta.runtime":{$lt:100}}).pretty() // how to find data inside the file document using (.) feield.whatever want**

**using or oprater find drama or action**

**BoxOffice> db.movies.find({$or: [ {genre:"drama" } , {genre: "action" }]}).pretty()**

**using and oprater find drama or action**

**BoxOffice> db.movies.find({$and: [ {genre:"drama" } , {genre: "action" }]}).pretty()**

**\*\***

**BoxOffice> db.exmoviestarts.find({genre:{$size:2}}).pretty()**

**[**

**{**

**\_id: ObjectId("6333f5fdf2a3d0240d692bba"),**

**title: 'Supercharged Teaching',**

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

**visitors: 370000,**

**expectedVisitors: 1000000,**

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

**ratings: [ 10, 9, 9 ]**

**},**

**{**

**\_id: ObjectId("6333f5fdf2a3d0240d692bbb"),**

**title: 'Teach me if you can',**

**meta: { rating: 8, aired: 2014, runtime: 90 },**

**visitors: 590378,**

**expectedVisitors: 500000,**

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

**ratings: [ 8, 8 ]**

**}**

**]**

**BoxOffice> db.exmoviestarts.find({rating: {$elemMatch:{$gt 8 , $lt:10}}})**

**Uncaught:**

**SyntaxError: Unexpected token, expected "," (1:48)**

**> 1 | db.exmoviestarts.find({rating: {$elemMatch:{$gt 8 , $lt:0}}})**

**| ^**

**2 |**

**BoxOffice> db.exmoviestarts.find({ratings: {$elemMatch:{$gt 8 , $lt:0}}})**

**Uncaught:**

**SyntaxError: Unexpected token, expected "," (1:49)**

**> 1 | db.exmoviestarts.find({ratings: {$elemMatch:{$gt 8 , $lt:0}}})**

**| ^**

**2 |**

**BoxOffice> db.exmoviestarts.find({ratings: {$elemMatch:{$gt: 8 , $lt:0}}})**

**BoxOffice> db.exmoviestarts.find({ratings: {$elemMatch:{$gt: 8 , $lt:10}}})**

**[**

**{**

**\_id: ObjectId("6333f5fdf2a3d0240d692bb9"),**

**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("6333f5fdf2a3d0240d692bba"),**

**title: 'Supercharged Teaching',**

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

**visitors: 370000,**

**expectedVisitors: 1000000,**

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

**ratings: [ 10, 9, 9 ]**

**}**

**]**

**Using Array in mongodb**

**users> db.users.find({hobbies:{$size: 3 }}).pretty() // use of size**

sorting Arry

**db.tvchaneel.find().sort({"rating.average":1}).pretty() //**

**db.tvchaneel.find().sort({"rating.average":1,runtime:1}).pretty() // shortby avarege and runtime 1 means increses order and average .**

**operator**

| **Name** | **Description** |
| --- | --- |
| [$eq](https://www.mongodb.com/docs/manual/reference/operator/query/eq/#mongodb-query-op.-eq) | Matches values that are equal to a specified value. |
| [$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. |

### Logical

| **Name** | **Description** |
| --- | --- |
| [$and](https://www.mongodb.com/docs/manual/reference/operator/query/and/#mongodb-query-op.-and) | Joins query clauses with a logical AND returns all documents that match the conditions of both clauses. |
| [$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. |

### Element

| **Name** | **Description** |
| --- | --- |
| [$exists](https://www.mongodb.com/docs/manual/reference/operator/query/exists/#mongodb-query-op.-exists) | Matches documents that have the specified field. |
| [$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. |

### Evaluation

| **Name** | **Description** |
| --- | --- |
| [$expr](https://www.mongodb.com/docs/manual/reference/operator/query/expr/#mongodb-query-op.-expr) | Allows use of aggregation expressions within the query language. |
| [$jsonSchema](https://www.mongodb.com/docs/manual/reference/operator/query/jsonSchema/#mongodb-query-op.-jsonSchema) | Validate documents against the given JSON Schema. |
| [$mod](https://www.mongodb.com/docs/manual/reference/operator/query/mod/#mongodb-query-op.-mod) | Performs a modulo operation on the value of a field and selects documents with a specified result. |
| [$regex](https://www.mongodb.com/docs/manual/reference/operator/query/regex/#mongodb-query-op.-regex) | Selects documents where values match a specified regular expression. |
| [$text](https://www.mongodb.com/docs/manual/reference/operator/query/text/#mongodb-query-op.-text) | Performs text search. |
| [$where](https://www.mongodb.com/docs/manual/reference/operator/query/where/#mongodb-query-op.-where) | Matches documents that satisfy a JavaScript expression. |

### Geospatial

| **Name** | **Description** |
| --- | --- |
| [$geoIntersects](https://www.mongodb.com/docs/manual/reference/operator/query/geoIntersects/#mongodb-query-op.-geoIntersects) | Selects geometries that intersect with a [GeoJSON](https://www.mongodb.com/docs/manual/reference/glossary/" \l "std-term-GeoJSON) geometry. The [2dsphere](https://www.mongodb.com/docs/manual/core/2dsphere/) index supports [$geoIntersects](https://www.mongodb.com/docs/manual/reference/operator/query/geoIntersects/#mongodb-query-op.-geoIntersects). |
| [$geoWithin](https://www.mongodb.com/docs/manual/reference/operator/query/geoWithin/#mongodb-query-op.-geoWithin) | Selects geometries within a bounding [GeoJSON geometry](https://www.mongodb.com/docs/manual/reference/geojson/" \l "std-label-geospatial-indexes-store-geojson). The [2dsphere](https://www.mongodb.com/docs/manual/core/2dsphere/) and [2d](https://www.mongodb.com/docs/manual/core/2d/) indexes support [$geoWithin](https://www.mongodb.com/docs/manual/reference/operator/query/geoWithin/#mongodb-query-op.-geoWithin). |
| [$near](https://www.mongodb.com/docs/manual/reference/operator/query/near/#mongodb-query-op.-near) | Returns geospatial objects in proximity to a point. Requires a geospatial index. The [2dsphere](https://www.mongodb.com/docs/manual/core/2dsphere/) and [2d](https://www.mongodb.com/docs/manual/core/2d/) indexes support [$near](https://www.mongodb.com/docs/manual/reference/operator/query/near/#mongodb-query-op.-near). |
| [$nearSphere](https://www.mongodb.com/docs/manual/reference/operator/query/nearSphere/#mongodb-query-op.-nearSphere) | Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The [2dsphere](https://www.mongodb.com/docs/manual/core/2dsphere/) and [2d](https://www.mongodb.com/docs/manual/core/2d/) indexes support [$nearSphere](https://www.mongodb.com/docs/manual/reference/operator/query/nearSphere/#mongodb-query-op.-nearSphere). |

### Array

| **Name** | **Description** |
| --- | --- |
| [$all](https://www.mongodb.com/docs/manual/reference/operator/query/all/#mongodb-query-op.-all) | Matches arrays that contain all elements specified in the query. |
| [$elemMatch](https://www.mongodb.com/docs/manual/reference/operator/query/elemMatch/#mongodb-query-op.-elemMatch) | Selects documents if element in the array field matches all the specified [$elemMatch](https://www.mongodb.com/docs/manual/reference/operator/query/elemMatch/#mongodb-query-op.-elemMatch) conditions. |
| [$size](https://www.mongodb.com/docs/manual/reference/operator/query/size/#mongodb-query-op.-size) | Selects documents if the array field is a specified size. |

### Bitwise

| **Name** | **Description** |
| --- | --- |
| [$bitsAllClear](https://www.mongodb.com/docs/manual/reference/operator/query/bitsAllClear/#mongodb-query-op.-bitsAllClear) | Matches numeric or binary values in which a set of bit positions all have a value of 0. |
| [$bitsAllSet](https://www.mongodb.com/docs/manual/reference/operator/query/bitsAllSet/#mongodb-query-op.-bitsAllSet) | Matches numeric or binary values in which a set of bit positions all have a value of 1. |
| [$bitsAnyClear](https://www.mongodb.com/docs/manual/reference/operator/query/bitsAnyClear/#mongodb-query-op.-bitsAnyClear) | Matches numeric or binary values in which any bit from a set of bit positions has a value of 0. |
| [$bitsAnySet](https://www.mongodb.com/docs/manual/reference/operator/query/bitsAnySet/#mongodb-query-op.-bitsAnySet) | Matches numeric or binary values in which any bit from a set of bit positions has a value of 1. |

## Projection Operators

| **Name** | **Description** |
| --- | --- |
| [$](https://www.mongodb.com/docs/manual/reference/operator/projection/positional/#mongodb-projection-proj.-) | Projects the first element in an array that matches the query condition. |
| [$elemMatch](https://www.mongodb.com/docs/manual/reference/operator/projection/elemMatch/#mongodb-projection-proj.-elemMatch) | Projects the first element in an array that matches the specified [$elemMatch](https://www.mongodb.com/docs/manual/reference/operator/projection/elemMatch/#mongodb-projection-proj.-elemMatch) condition. |
| [$meta](https://www.mongodb.com/docs/manual/reference/operator/aggregation/meta/#mongodb-expression-exp.-meta) | Projects the document's score assigned during [$text](https://www.mongodb.com/docs/manual/reference/operator/query/text/#mongodb-query-op.-text) operation. |
| [$slice](https://www.mongodb.com/docs/manual/reference/operator/projection/slice/#mongodb-projection-proj.-slice) | Limits the number of elements projected from an array. Supports skip and limit slices. |

## Miscellaneous Operators

| **Name** | **Description** |
| --- | --- |
| [$comment](https://www.mongodb.com/docs/manual/reference/operator/query/comment/#mongodb-query-op.-comment) | Adds a comment to a query predicate. |
| [$rand](https://www.mongodb.com/docs/manual/reference/operator/query/rand/#mongodb-query-op.-rand) | Generates a random float between 0 and 1. |

**db.tvchaneel.find().sort({"rating.average":1}).pretty() // using shorting method in embedded element**

**Update operator used in mongo dB**

1. **users> db.usr.updateOne({\_id: ObjectId("6334292716bd17bdc9bd6a17")},{$set:{hobbies:[{title:"sports",frequency:5},{"title":"Cooking", "frequency": 5 },{title: "Hiking" , frequency:1 }]}})**
2. **users> db.usr.updateOne({\_id: ObjectId("6334292716bd17bdc9bd6a17")},{$set:{age:40,phonenumber: 8619486272}})**

* **increment and decrement value in mongodb**

**increment of age by 1**

* **db.usr.updateOne({name: "Manuel" } ,{$inc: {age:1}})**
* **adding both prop**
* **users> db.usr.updateOne({name: "Manuel" },{$set:{isSporty:false}},{$inc: {age:-1}})**

**using min , max and mul**

**if you use the min opreter so we do not able to the change the value above more we can only change the value of min below value whatever we set the earlier .**

1. **db.usr.updateOne({name: "Chris"},{$min:{age:35}})**

**in case of max we can only update the value abobe then what ever set earlier**

1. **users> db.usr.updateOne({name: "Chris"},{$max:{age:101}})**

**In use of mul we have increrses the data by 10% in age**

* **users> db.usr.updateOne({name: "Chris"},{$mul:{age:1.1}})**

**using unset opreter we remove the the entry of element**

* **users> db.usr.updateMany({isSporty: false},{$unset: {phone:""}})**
* **db.usr.updateMany({}, {$rename: {age: " totaleAge"}}) // rename the field**

**Delete oprations**

1. **delete with id**

**Using type oprat**

**db.useres.find({phone: { $type:"phone"}}).pretty()**

using regex and projection

1. **db.tvchaneel.find({summary: {$regex:/musical/}}).pretty() // using regex fetch the data**
2. **db.sales.find({$expr: {$gt:["$volume", "$target"]}}).pretty()**// file which is grater then that the
3. tv> db.tvchaneel.find().sort({ "rating.average":1,runtime:1}).sort({"rating.average": 1 }).skip(10).pretty() // using skip 10 data .
4. db.tvchaneel.find().sort({"rating.average":1,runtime:1}).sort({"rating.average": 1 }).skip(200).pretty().count()
5. tv> db.tvchaneel.find().sort({"rating.average":1,runtime:1}).sort({"rating.average": 1 }).skip(100).limit(10).pretty()
6. db.tvchaneel.find({},{name: 1, genres:1 , runtime: 1,\_id:0}).pretty() // (using projection)
7. db.tvchaneel.find({},{name: 1, genres:1 , runtime: 1,\_id:0}).pretty() // (using projection embded)
8. db.tvchaneel.find({genres: "Drama" }, {"genres.$":1}).pretty() // using projection

**some important link**

* [MongoDB C#/.NET Driver — MongoDB Drivers](https://www.mongodb.com/docs/drivers/csharp/)
* [MongoDB C#/.NET Driver — MongoDB Drivers](https://www.mongodb.com/docs/drivers/csharp/)

**CRUD OPRATION COMMEND**

1. **db.ankurdatabase.updateMany({},{$set : {marker: ' delete' }})**
2. **db.ankurdatabase.deleteMany( {marker: ' delete' })**
3. **db.ankurdatabase.find({ distance :{$gt: 10000 }})**
4. **db.ankurdatabase.findOne({ distance :{$gt: 800 }}).pretty()**
5. **db.ankurdatabase.find().forEach((x) => {printjson(x)})// fetch all data**
6. **db.ankurdatabase.find().toArray()// fetech all data**
7. **\*\*db.information.updateOne({name: 'Albert Twostone'},{$set:{hobbie:["sports", "watching movies"]}} )// (how to inserted insert Array inside the table )**

**Advance mongo dB**

**Update through object id**

* **db.ankurdatabase.replaceOne({\_id: ObjectId("632d480ec0c363e18f502c13")},{"departureAirport": "LHR",})**

**using filter**

1. **db.ankurdatabase.find({},{name:1(//return),\_id:0}).pretty() // using projection**
2. **// how to fetch details**
3. **test2> db.information.findOne({ name: "Albert Twostone"}).hobbie**
4. **[ 'sports', 'watching movies' ]**
5. **test2> db.information.findOne({ name: "Albert Twostone"}).age**
6. **68**
7. **test2> db.information.findOne({ name: "Albert Twostone"}).\_id**
8. **ObjectId("632d8ab40c61aec0f02b4fe2")**

**Prop 1**

{

    "firstName": "Max",

    "lastName": "Sch",

    "age": 29,

    "histroy": [

      {

        "desises": "cold",

        "treatmnt": 95

      }

    ]

  }

]

**Sol 03 update**

**db.patition.updateOne({firstName: "Max"} , {$set:{lastname: "Mall" , age : 59 , history: [{desies: "verysad" , treatment: 61}]}})**

* Important: We will regularly start with a clean database server (i.e. all data was purged) .
* To get rid of your data, you can simply load the database you want to get rid of (use databaseName) and then execute db.dropDatabase().
* Similarly, you could get rid of a single collection in a database via db.myCollection.drop().

**Data type**

1. **companyData> db.companyData.insertOne({name: "Fresh Apples Inc " , isStartup : true , empolyees : 33.12, funding : 1234567890123456789 , details:{ ceo : "ank" }, tags : [{title : "super"}, {title : "perfect" } ] , foundingDate: new Date () , insertedAt : nnew Timestamp()})**
2. **// resullt**
3. **id: ObjectId("63314064698be8d71842ce9e"),**
4. **name: 'Fresh Apples Inc ',**
5. **isStartup: true,**
6. **empolyees: 33.12,**
7. **funding: 1234567890123456800,**
8. **details: { ceo: 'ank' },**
9. **tags: [ { title: 'super' }, { title: 'perfect' } ],**
10. **foundingDate: ISODate("2022-09-26T06:02:12.359Z"),**
11. **insertedAt: Timestamp({ t: 1664172132, i: 1 })**
12. **\*\*\***

**support> db.citizen.insertMany([{ name : "Max sch" , cityId: ObjectId("633172cc698be8d71842cea5")}, { name: "shivam Mall", cityId: ObjectId("633172cc698be8d71842cea5")}])**

**{**

**acknowledged: true,**

**insertedIds: {**

**'0': ObjectId("633173b1698be8d71842cea6"),**

**'1': ObjectId("633173b1698be8d71842cea7")**

**}**

**}**

**support> db.citizen.find()**

**[**

**{**

**\_id: ObjectId("633173b1698be8d71842cea6"),**

**name: 'Max sch',**

**cityId: ObjectId("633172cc698be8d71842cea5")**

**},**

**{**

**\_id: ObjectId("633173b1698be8d71842cea7"),**

**name: 'shivam Mall',**

**cityId: ObjectId("633172cc698be8d71842cea5")**

**}**

**R**

**Join with $lookup**

1. **test2> db.hobbies.insertMany([{\_id: "sprts", name: "sport" }, {\_id:"coocking", name: "Coocking"}, { \_id: "cars" , name: "bmw"}])**

MongoDB has a couple of hard limits - most importantly, a single document in a collection (including all embedded documents it might have) must be <= **16mb**. Additionally, you may only have **100 levels of embedded documents**.

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/>

**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.

Example for Node.js: <http://mongodb.github.io/node-mongodb-native/3.1/api/Long.html>

This will allow you to build a NumberLong value like this:

1. const Long = require('mongodb').Long;
3. db.collection('wealth').insert( {
4. value: Long.fromString("121949898291")
5. });

By browsing the API docs for the driver you're using, you'll be able to identify the methods for building int32s, int64s etc.

* 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>