**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()

**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**

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.