

Explore Developer Center's New Chatbot! MongoDB AI Chatbot can be accessed at the top of your navigation to answer all your MongoDB questions.

[VIEW MONGO AI](#)

## MongoDB Developer



[MONGODB DEVELOPER CENTER](#) > [DEVELOPER TOPICS](#) > [PRODUCTS](#) > [MONGODB](#) >

[QUICKSTARTS](#)

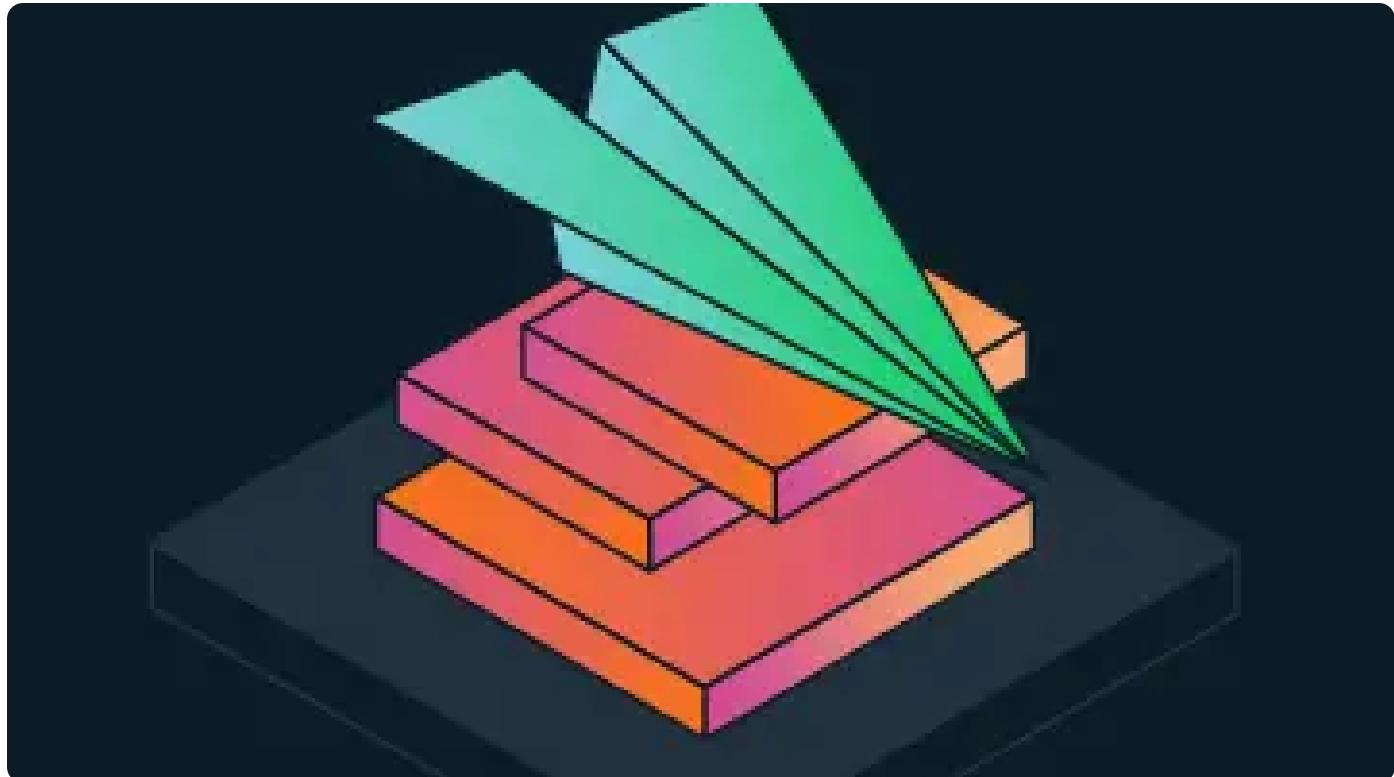
# MongoDB Cheat Sheet



Maxime Beugnet

5 min read • Published Jan 31, 2022 • Updated Sep 29, 2023

[MongoDB](#)



Rate this quickstart ☆ ☆ ☆ ☆ ☆

First steps in the MongoDB World? This cheat sheet is filled with some handy tips, commands, and quick references to get you connected and CRUD'ing in no time!

- Get a [free MongoDB cluster in MongoDB Atlas](#).
- Follow a course in [MongoDB University](#).

## Updates

- September 2023: Updated for MongoDB 7.0.

## Table of Contents

- [Connect MongoDB Shell](#)
- [Helpers](#)
- [CRUD](#)
- [Databases and Collections](#)
- [Indexes](#)
- [Handy commands](#)
- [Change Streams](#)
- [Replica Set](#)
- [Sharded Cluster](#)
- [Wrap-up](#)

Connect via `mongosh`

```
1 mongosh # connects to mongodb://127.0.0.1:27017 by default
2 mongosh --host <host> --port <port> --authenticationDatabase admin -u <user> -p
3 mongosh "mongodb://<user>:<password>@192.168.1.1:27017"
4 mongosh "mongodb://192.168.1.1:27017"
5 mongosh "mongodb+srv://cluster-name.abcde.mongodb.net/<dbname>" --apiVersion 1
```



- [mongosh documentation.](#)

↑ [TOP](#) Table of Contents [TOP](#) ↑

## Helpers

### Show Databases

```
1 show dbs  
2 db // prints the current database
```



### Switch Database

```
1 use <database_name>
```



### Show Collections

```
1 show collections
```



### Run JavaScript File

```
1 load("myScript.js")
```



↑ [TOP](#) Table of Contents [TOP](#) ↑

## CRUD

### Create

```
1 db.coll.insertOne({name: "Max"})  
2 db.coll.insertMany([{name: "Max"}, {name:"Alex"}]) // ordered bulk insert  
3 db.coll.insertMany([{name: "Max"}, {name:"Alex"}], {ordered: false}) // unorder  
4 db.coll.insertOne({date: ISODate()})  
5 db.coll.insertOne({name: "Max"}, {"writeConcern": {"w": "majority", "wtimeout":
```



### Read

```
1 db.coll.findOne() // returns a single document  
2 db.coll.find() // returns a cursor - show 20 results - "it" to display more  
3 db.coll.find().pretty()
```



```

4 db.coll.find({name: "Max", age: 32}) // implicit logical "AND".
5 db.coll.find({date: ISODate("2020-09-25T13:57:17.180Z")})
6 db.coll.find({name: "Max", age: 32}).explain("executionStats") // or "queryPla
7 db.coll.distinct("name")
8
9 // Count
10 db.coll.countDocuments({age: 32}) // alias for an aggregation pipeline - accur
11 db.coll.estimatedDocumentCount() // estimation based on collection metadata
12
13 // Comparison
14 db.coll.find({"year": {$gt: 1970}})
15 db.coll.find({"year": {$gte: 1970}})
16 db.coll.find({"year": {$lt: 1970}})
17 db.coll.find({"year": {$lte: 1970}})
18 db.coll.find({"year": {$ne: 1970}})
19 db.coll.find({"year": {$in: [1958, 1959]}})
20 db.coll.find({"year": {$nin: [1958, 1959]}})

21
22 // Logical
23 db.coll.find({name:{$not: {$eq: "Max"}}})
24 db.coll.find({$or: [{"year" : 1958}, {"year" : 1959}]})
25 db.coll.find({$nor: [{price: 1.99}, {sale: true}]})
26 db.coll.find({
27   $and: [
28     {$or: [{qty: {$lt: 10}}, {qty :{$gt: 50}}]}, 
29     {$or: [{sale: true}, {price: {$lt: 5 }}]}
30   ]
31 })
32
33 // Element
34 db.coll.find({name: {$exists: true}})
35 db.coll.find({"zipCode": {$type: 2}})
36 db.coll.find({"zipCode": {$type: "string"}})

37
38 // Aggregation Pipeline
39 db.coll.aggregate([
40   {$match: {status: "A"}},
41   {$group: {_id: "$cust_id", total: {$sum: "$amount"}}, 
42   {$sort: {total: -1}}
43 ])
44
45 // Text search with a "text" index
46 db.coll.find({$text: {$search: "cake"}}, {score: {$meta: "textScore"}}).sort({$textScore: -1})
47
48 // Regex
49 db.coll.find({name: /^Max/}) // regex: starts by letter "M"
50 db.coll.find({name: /^Max$/i}) // regex case insensitive
51
52 // Array

```

```

53 db.coll.find({tags: {$all: ["Realm", "Charts"]}})
54 db.coll.find({field: {$size: 2}}) // impossible to index - prefer storing the
55 db.coll.find({results: {$elemMatch: {product: "xyz", score: {$gte: 8}}}}})
56
57 // Projections
58 db.coll.find({"x": 1}, {"actors": 1}) // actors + _id
59 db.coll.find({"x": 1}, {"actors": 1, "_id": 0}) // actors
60 db.coll.find({"x": 1}, {"actors": 0, "summary": 0}) // all but "actors" and "s
61
62 // Sort, skip, limit
63 db.coll.find({}).sort({"year": 1, "rating": -1}).skip(10).limit(3)
64
65 // Read Concern
66 db.coll.find().readConcern("majority")

```

- [db.collection.find\(\)](#)
- [Query and Projection Operators](#)
- [BSON types](#)
- [Read Concern](#)

## Update

```

1 db.coll.updateOne({"_id": 1}, {$set: {"year": 2016, name: "Max"}})
2 db.coll.updateOne({"_id": 1}, {$unset: {"year": 1}})
3 db.coll.updateOne({"_id": 1}, {$rename: {"year": "date"} })
4 db.coll.updateOne({"_id": 1}, {$inc: {"year": 5}})
5 db.coll.updateOne({"_id": 1}, {$mul: {price: NumberDecimal("1.25"), qty: 2}})
6 db.coll.updateOne({"_id": 1}, {$min: {"imdb": 5}})
7 db.coll.updateOne({"_id": 1}, {$max: {"imdb": 8}})
8 db.coll.updateOne({"_id": 1}, {$currentDate: {"lastModified": true}})
9 db.coll.updateOne({"_id": 1}, {$currentDate: {"lastModified": {$type: "timesta
10
11 // Array
12 db.coll.updateOne({"_id": 1}, {$push :{"array": 1}})}
13 db.coll.updateOne({"_id": 1}, {$pull :{"array": 1}})}
14 db.coll.updateOne({"_id": 1}, {$addToSet :{"array": 2}})}
15 db.coll.updateOne({"_id": 1}, {$pop: {"array": 1}}) // last element
16 db.coll.updateOne({"_id": 1}, {$pop: {"array": -1}}) // first element
17 db.coll.updateOne({"_id": 1}, {$pullAll: {"array" :[3, 4, 5]}})}
18 db.coll.updateOne({"_id": 1}, {$push: {"scores": {$each: [90, 92]}}})
19 db.coll.updateOne({"_id": 2}, {$push: {"scores": {$each: [40, 60], $sort: 1}}})
20 db.coll.updateOne({"_id": 1, "grades": 80}, {$set: {"grades.$": 82}})
21 db.coll.updateMany({}, {$inc: {"grades.$[]": 10}})
22 db.coll.updateMany({}, {$set: {"grades.$[element]": 100}}, {multi: true, array

```

```

23
24 // FindOneAndUpdate
25 db.coll.findOneAndUpdate({name: "Max"}, {$inc: {"points": 5}}, {returnNewDoc: true})
26
27 // Upsert
28 db.coll.updateOne({_id: 1}, {$set: {item: "apple"}}, $setOnInsert: {defaultQuantity: 1})
29
30 // Replace
31 db.coll.replaceOne({name: "Max"}, {"firstname": "Maxime", "surname": "Beugne"})
32
33 // Write concern
34 db.coll.updateMany({}, {$set: {"x": 1}}), {"writeConcern": {"w": "majority", "wtimeout": 1000}}

```

## Delete

```

1 db.coll.deleteOne({name: "Max"})
2 db.coll.deleteMany({name: "Max"}, {"writeConcern": {"w": "majority", "wtimeout": 1000}})
3 db.coll.deleteMany({}) // WARNING! Deletes all the docs but not the collection
4 db.coll.findOneAndDelete({name: "Max"})

```

[Table of Contents](#)

## Databases and Collections

### Drop

```

1 db.coll.drop() // removes the collection and its index definitions
2 db.dropDatabase() // double check that you are *NOT* on the PROD cluster... :-( )

```

### Create Collection

```

1 // Create collection with a $jsonschema
2 db.createCollection("contacts", {
3   validator: {$jsonSchema: [
4     bsonType: "object",
5     required: ["phone"],
6     properties: {
7       phone: {
8         bsonType: "string",
9         description: "must be a string and is required"
10      },
11      email: {
12        bsonType: "string",
13        pattern: "@mongodb\\.com$",
14        description: "must be a string and match the regular expression pattern"
15      },
16    }
17  ]
18 })

```

```

16     status: {
17         enum: [ "Unknown", "Incomplete" ],
18         description: "can only be one of the enum values"
19     }
20 }
21 }
22 })

```

## Other Collection Functions

```

1 db.coll.stats()
2 db.coll.storageSize()
3 db.coll.totalIndexSize()
4 db.coll.totalSize()
5 db.coll.validate({full: true})
6 db.coll.renameCollection("new_coll", true) // 2nd parameter to drop the target

```

[TOP](#) Table of Contents [TOP](#)

## Indexes

### List Indexes

```

1 db.coll.getIndexes()
2 db.coll.getIndexKeys()

```

### Create Indexes

```

1 // Index Types
2 db.coll.createIndex({"name": 1}) // single field index
3 db.coll.createIndex({"name": 1, "date": 1}) // compound index
4 db.coll.createIndex({foo: "text", bar: "text"}) // text index
5 db.coll.createIndex({"$**": "text"}) // wildcard text index
6 db.coll.createIndex({"userMetadata.$**": 1}) // wildcard index
7 db.coll.createIndex({"loc": "2d"}) // 2d index
8 db.coll.createIndex({"loc": "2dsphere"}) // 2dsphere index
9 db.coll.createIndex({"_id": "hashed"}) // hashed index
10
11 // Index Options
12 db.coll.createIndex({"lastModifiedDate": 1}, {expireAfterSeconds: 3600})
13 db.coll.createIndex({"name": 1}, {unique: true})
14 db.coll.createIndex({"name": 1}, {partialFilterExpression: {age: {$gt: 18}}})
15 db.coll.createIndex({"name": 1}, {collation: {locale: 'en', strength: 1}})
16 db.coll.createIndex({"name": 1 }, {sparse: true})

```

## Drop Indexes

```
1 db.coll.dropIndex("name_1")
```



## Hide/Unhide Indexes

```
1 db.coll.hideIndex("name_1")
2 db.coll.unhideIndex("name_1")
```



- [Indexes documentation](#)

[TOP](#) Table of Contents [TOP](#)

## Handy commands

```
1 use admin
2 db.createUser({ "user": "root", "pwd": passwordPrompt(), "roles": [ "root" ] })
3 db.dropUser("root")
4 db.auth( "user", passwordPrompt() )
5
6 use test
7 db.getSiblingDB("dbname")
8 db.currentOp()
9 db.killOp(123) // opid
10
11 db.fsyncLock()
12 db.fsyncUnlock()
13
14 db.getCollectionNames()
15 db.getCollectionInfos()
16 db.printCollectionStats()
17 db.stats()
18
19 db.getReplicationInfo()
20 db.printReplicationInfo()
21 db.hello()
22 db.hostInfo()
23
24 db.shutdownServer()
25 db.serverStatus()
26
27 db.getProfilingStatus()
28 db.setProfilingLevel(1, 200) // 0 == OFF, 1 == ON with slowms, 2 == ON
```



```

29
30 db.enableFreeMonitoring()
31 db.disableFreeMonitoring()
32 db.getFreeMonitoringStatus()
33
34 db.createView("viewName", "sourceColl", [{$project:{department: 1}}])

```

[TOP](#) Table of Contents [TOP](#)

## Change Streams

```

1 watchCursor = db.coll.watch( [ { $match : { "operationType" : "insert" } } ] )
2
3 while (!watchCursor.isExhausted()){
4     if (watchCursor.hasNext()){
5         print(tojson(watchCursor.next()));
6     }
7 }

```



[TOP](#) Table of Contents [TOP](#)

## Replica Set

```

1 rs.status()
2 rs.initiate({"_id": "RS1",
3     members: [
4         { _id: 0, host: "mongodb1.net:27017" },
5         { _id: 1, host: "mongodb2.net:27017" },
6         { _id: 2, host: "mongodb3.net:27017" }]
7 })
8 rs.add("mongodb4.net:27017")
9 rs.addArb("mongodb5.net:27017")
10 rs.remove("mongodb1.net:27017")
11 rs.conf()
12 rs.hello()
13 rs.printReplicationInfo()
14 rs.printSecondaryReplicationInfo()
15 rs.reconfig(config)
16 rs.reconfigForPSASet(memberIndex, config, { options } )
17 db.getMongo().setReadPref('secondaryPreferred')
18 rs.stepDown(20, 5) // (stepDownSecs, secondaryCatchUpPeriodSecs)

```



[TOP](#) Table of Contents [TOP](#)

# Sharded Cluster

```
1 db.printShardingStatus()
2
3 sh.status()
4 sh.addShard("rs1/mongodb1.example.net:27017")
5 sh.shardCollection("mydb.coll", {zipcode: 1})
6
7 sh.moveChunk("mydb.coll", { zipcode: "53187" }, "shard0019")
8 sh.splitAt("mydb.coll", {x: 70})
9 sh.splitFind("mydb.coll", {x: 70})
10
11 sh.startBalancer()
12 sh.stopBalancer()
13 sh.disableBalancing("mydb.coll")
14 sh.enableBalancing("mydb.coll")
15 sh.getBalancerState()
16 sh.setBalancerState(true/false)
17 sh.isBalancerRunning()
18
19 sh.startAutoMerger()
20 sh.stopAutoMerger()
21 sh.enableAutoMerger()
22 sh.disableAutoMerger()
23
24 sh.updateZoneKeyRange("mydb.coll", {state: "NY", zip: MinKey}, { state: "NY",
25 sh.removeRangeFromZone("mydb.coll", {state: "NY", zip: MinKey}, { state: "NY"
26 sh.addShardToZone("shard0000", "NYC")
27 sh.removeShardFromZone("shard0000", "NYC")
```



↑ [Table of Contents](#) ↑

## Wrap-up

I hope you liked my little but - hopefully - helpful cheat sheet. Of course, this list isn't exhaustive at all. There are a lot more commands, but I'm sure you will find them in the [MongoDB documentation](#).

If you feel like I forgot a critical command in this list, please [send me a tweet](#) and I will make sure to fix it.