MongoDB Cheat Sheet - Essential MongoDB Shell Commands

Posted on 05th December 2016

Here are the most commonly used Mongo Shell commands with example usage.

Basic Commands		
To do this	**Run this command**	**Example**
Connect to local host on default port 27017	mongo	mongo
Connect to remote host on specified port	mongo --host *<hostname or ip address>* --port *<port no>*	mongo --host 10.121.65.23 --port 23020
Connect to a database	mongo *<host>/<database>*	mongo 10.121.65.58/mydb
Show current database	db	db
Select or switch database [1]	use *<database name>*	use mydb

Execute a JavaScript file	load(*<filename>*)	load (myscript.js)
Display help	help	help
Display help on DB methods	db.help()	db.help()
Display help on Collection	db.mycol.help()	db.mycol.help()
Show Commands		
Show all databases	show dbs	show dbs
Show all collections in current database	show collections	show collections
Show all users on current database	show users	show users
Show all roles on current database	show roles	show roles
CRUD Operations		
Insert a new document in a collection [2]	db.collection.insert(*<document>*)	db.books.insert({"isbn": 9780060859749, "title": "After Alice: A Novel", "author": "Gregory Maguire", "category":

		"Fiction", "year":2016})
Insert multiple documents into a collection	db.collection.insertMany([*<document1>, <document2>, ...*]) -or- db.collection.insert([*<document1>, <document2>, ...*])	db.books.insertMany([{"isbn": 9780198321668, "title": "Romeo and Juliet", "author": "William Shakespeare", "category": "Tragedy", "year": 2008}, {"isbn": 9781505297409, "title": "Treasure Island", "author": "Robert Louis Stevenson", "category": "Fiction", "year":2014}]) -or- db.books.insert([{ "isbn":"9781853260001", "title": "Pride and Prejudice", "author": "Jane Austen", "category": "Fiction"}, {"isbn":

		"9780743273565", "title": "The Great Gatsby", "author": "F. Scott Fitzgerald"}])
Show all documents in the collection	db.collection.find()	db.books.find()
Filter documents by field value condition	db.collection.find(*<query>*)	db.books.find({"title":"Treasure Island"})
Show only some fields of matching documents	db.collection.find(*<query>, <projection>)*	db.books.find({"title":"Treasure Island"}, {title:true, category:true, _id:false})
Show the first document that matches the query condition	db.collection.findOne(*<query>, <projection>*)	db.books.findOne({}, {_id:false})
Update specific fields of a single document that match the query condition	db.collection.update(*<query>, <update>*)	db.books.update({title : "Treasure Island"}, {$set : {category :"Adventure Fiction"}})
Remove certain fields of a single document the query condition	db.collection.update(*<query>, <update>*)	db.books.update({title : "Treasure Island"}, {$unset : {category:""}})

Remove certain fields of all documents that match the query condition	db.collection.update(*<query>*, *<update>*, {multi:true})	db.books.update({category : "Fiction"}, {$unset : {category:""}}, {multi:true})
Delete a single document that match the query condition	db.collection.remove(*<query>*, {justOne:true})	db.books.remove({title :"Treasure Island"}, {justOne:true})
Delete all documents matching a query condition	db.collection.remove(*<query>*)	db.books.remove({"category" :"Fiction"})
Delete all documents in a collection	db.collection.remove({})	db.books.remove({})
Index		
Create an index	db.collection.createIndex({indexField:type}) Type 1 means ascending; -1 means descending	db.books.createIndex({title:1})
Create a unique index	db.collection.createIndex({indexField:type} , {unique:true})	db.books.createIndex({isbn:1}, {unique:true})

Create a index on multiple fields (compound index)	db.collection.createIndex({indexField1:type1, indexField2:type2, ...})	db.books.createIndex({title:1, author:-1})
Show all indexes in a collection	db.collection.getIndexes()	db.books.getIndexes()
Drop an index	db.collection.dropIndex({indexField:type})	db.books.dropIndex({author:-1})
Show index statistics	db.collection.stats()	db.books.stats()
Cursor Methods		
Show number of documents in the collection	cursor.count()	db.books.find().count()
Limit the number of documents to return	cursor.limit(*<n>*)	db.books.find().limit(2)
Return the result set after skipping the first *n* number of documents	cursor.skip(*<n>*)	db.books.find().skip(2)
Sort the documents in a result set in ascending or descending order of field	cursor.sort(*<{field : value}>*) value = 1 for ascending, -1 for descending	db.books.find().sort({title : 1})

values		
Display formatted (more readable) result	cursor.pretty()	db.books.find({}) .pretty()
Comparison Operators		
equals to	{<field>: { $eq: <value> }}	db.books.find({year: {$eq: 2016}})
less than	{<field>: { $lt: <value> }}	db.books.find({year: {$lt: 2010}})
less than or equal to	{<field>: { $lte: <value> }}	db.books.find({ year: {$lte: 2008}})
greater than	{<field>: { $gt: <value> }}	db.books.find({year: {$gt: 2014}})
greater than or equal to	{<field>: { $gte: <value> }}	db.books.find({year: {$gte: 2008}})
not equal to	{<field>: { $ne: <value> }}	db.books.find({year: {$ne: 2008}})
value in	{<field>: { $in: [<value1>, <value2>, ... }}	db.books.find({year: {$in: [2008, 2016]}})
value not in	{<field>: { $nin: [<value1>, <value2>, ... }}	db.books.find({year: {$nin: [2008, 2016]}})
Logical Operators		

OR	{ $or: [<expression1>, <expression2>,...]}	db.books.find({ $or: [{year: {$lte: 2008}}, {year: {$eq: 2016}}]})
AND	{ $and: [<expression1>, <expression2>,...]}	db.books.find({ $and: [{year: {$eq: 2008}}, {category: {$eq: "Fiction"}}]})
NOT	{ $not: {<expression>}}	db.books.find({$not: {year: {$eq: 2016} }})
NOR	{ $nor: [<expression1>, <expression2>,...]}	db.books.find({ $nor: [{year: {$lte: 2008}}, {year: {$eq: 2016}}]})
Element Operators		
Match documents that contains that specified field	{<field>: {$exists:true}}	db.books.find({category: {$exists: true }})
Match documents whose field value is of the specified BSON data type	{<field>: {$type:value}}	db.books.find({category: {$type: 2 }})

[1] Databases are created on the fly and will actually be created when you insert something into it.

[2] Collections are created on the fly when you insert first

document into it.

Related Articles

Post a comment

Your Comment....

Name Email

Post Comment

Comments

@89rtE | October 2, 2017 3:30 PM |

Lots of HW, GZ. THX.

Reply

Shy Tamir | December 29, 2016 10:46 AM |

You might want to add this little gem for validation of data files. I use it on our restored daily backup to verify it's usable:
db.getSiblingDB("admin").runCommand("listDatabases").databases.forEach(function(d) {db.getSiblingDB(d.name).getCollectionNames().forEach(function(c) {assert(db.getSiblingDB(d.name).getCollection(c).validate().valid==true)})})

Reply