**MongoDB**

[Getting Started 1](#_Toc184913562)

[Terms to remember 2](#_Toc184913563)

[Commands 2](#_Toc184913564)

[Terminal commands 2](#_Toc184913565)

[CRUD Commands 2](#_Toc184913566)

[Create 2](#_Toc184913567)

[Databases and Collections 2](#_Toc184913568)

[Documents 3](#_Toc184913569)

[Read 3](#_Toc184913570)

[Sorting, Limiting and Count 3](#_Toc184913571)

[Delete 3](#_Toc184913572)

[Databases and Collections 3](#_Toc184913573)

[Documents 4](#_Toc184913574)

[Update 4](#_Toc184913575)

[Operators and Complex Queries 4](#_Toc184913576)

[Greater than/ Less Than 4](#_Toc184913577)

[Or 5](#_Toc184913578)

[In and Not In (Nin) 5](#_Toc184913579)

[Arrays of simple values and of objects 5](#_Toc184913580)

[Indexing 5](#_Toc184913581)

[Nested Documents 6](#_Toc184913582)

[Aggregation 7](#_Toc184913583)

[Accumulator Operators (sum, avg) 7](#_Toc184913584)

[Unary Operators 8](#_Toc184913585)

# Getting Started

1. Install MongoDB Community server
2. Install MongoDB Shell
3. Install MongoDB Compass (GUI)

# Terms to remember

1. Cluster – Database architecture
2. Collection - Table
3. Document – record

Note, Mongodb is case sensitive.

# Commands

## Terminal commands

mongosh – To open the mongodb shell in another terminal.

cls – clear screen

// Note: We do not need to use a ";" to terminate a command.

show dbs - // shows all the databases in the server.

use <database\_name> - // switch to the database specified. Works even if the database specified does not exist.

db - // to show the current database

show collections - // Show the tables in the database.

var name = "Daniel" - // To create a variable and initialise it.

name = // to return the value of the variable created

help - // List all the commands you can use and what they mean

exit - // To exit the shell

## CRUD Commands

### Create

#### Databases and Collections

use <database\_name> // Create a database if the database specified does not exist

db.createCollection("books") // Create a collection, called "books" in the database specified above

#### Documents

db.books.insertOne({title: "The River and the source", author: "Margaret Ogola", pages: 281, genres: [   "Fiction", "Historical"], rating: 4.8})

-- To insert one document/record into the collection/table

db.books.insertMany([{title: "Kidagaa Kimemwozea", author: "Ken Walibora", pages: 281, genres: ["Fiction", "Romance", "Politics"], rating: 4.8}, {title: "Caucasian Chalk Circle", author: "Bertolt Bretch", pages: 281, genres: ["Fiction", "Romance", "Politics"], rating: 4.8}])

-- To insert many documents/records into the collection/table

### Read

db.books.find() -- Returns the first 20 documents in the collection

it – To iterate over the next 20 documents

db.books.find({author: "Margaret Ogola"}) – To filter the documents by one property

db.books.find({ author: "Margaret Ogola", pages: 281 })"}) – To filter the documents by more than one property

db.books.find({author: "Margaret Ogola"}, {title: 1, pages: 1, rating: 1}) -- Returns the specified "title", "pages" and "rating" properties of the documents in the collection, that match the the filter passeed

db.books.findOne({title: 'The River and the source'}) -- Returns the first document that passes the filter specified

#### Sorting, Limiting and Count

db.books.find({ genres: "Romance" }).sort({ title: -1 }).skip(2).limit(5); // Finds the list of documents that have "Romance" as one of the genres, sorts the list on descending order by the titles of the documents, skips the first 2 (pagination as part of the query parameters of an endpoint URL), then out the remaining documents, limits the response list to 5 documents. In mysql, the equivalent of the skip() command is “OFFSET”.

### Delete

#### Databases and Collections

db.books.drop() // Drops the "books" table/ collection.

db.dropDatabase() // Drops the current database (specified when you use the "use <database\_name>")

#### Documents

db.books.deleteOne({\_id: ObjectId('66ffadfb81599ee5f817de45')}) // Deletes the document with the id shown

db.books.deleteMany({rating: 4.8}) // Delete the documents with the rating shown

### Update

db.books.updateOne({\_id: ObjectId('66ffadfb81599ee5f817de3e')}, {$set: {pages: 300}}) // Updates the document specified by setting the pages field with the static value 300.

db.books.updateMany({rating: 4.8}, {$set: {pages: 300}}) // Updates the documents with the rating specified by setting the pages field with the static value 300.

db.books.updateMany({rating: 4.8}, {$inc: {pages: 20}}) // Updates the documents with the rating specified by incrementing their already-existing value with 20

db.books.updateMany({rating: 4.8}, {$inc: {pages: -20}}) // Updates the documents with the rating specified by decrementing their already-existing value with 20

db.books.updateOne({\_id: ObjectId('66ffadfb81599ee5f817de3e')}, {$push: {genres: "Dystopia"}}) // Updates the document specified by adding the string specified to the document's array of genres.

db.books.updateOne({\_id: ObjectId('66ffadfb81599ee5f817de3e')}, {$pull: {genres: "Classic"}}) // Updates the document specified by removing the string specified from the document's array of genres.

db.books.updateOne({\_id: ObjectId('66ffadfb81599ee5f817de3e')}, {$push: {genres: {$each: ["Thriller", "Comedy"]}}}) // Updates the document specified by adding the strings specified to the document's array of genres.

db.books.updateOne({\_id: ObjectId('66ffadfb81599ee5f817de3e')}, {$pull: {genres: {$in: ["Thriller", "Comedy"]}}}) // Updates the document specified by removing the strings specified from the document's array of genres.

## Operators and Complex Queries

Operators in MongoDb are denoted by “$”.

### Greater than/ Less Than

db.books.find({ rating: {$gt: 4.5}}) // Returns all the documents with a rating greater than 4.5

db.books.find({ rating: {$lt: 4.5}}) // Returns all the documents with a rating less than 4.5

db.books.find({ rating: {$gte: 4.5}}) // Returns all the documents with a rating greater than or equal to 4.5

db.books.find({ rating: {$lte: 4.5}}) // Returns all the documents with a rating less than or equal to 4.5

db.books.find({ rating: {$ne: 4.5}}) // Returns all the documents whose rating is not 4.5

#### Or

db.books.find({ $or: [{author: /Margaret/i }, {rating: 4.5}, { pages: {$gte: 300 }}]}) // Returns all the documents with a rating of 4.5, OR those with pages greater than or equal to 300, OR that have a author contaning the substring "Margaret", checking case-insensitively.

### In and Not In (Nin)

db.books.find({rating: {$in: [4.0, 4.2, 4.4, 4.6, 4.8]}}) // Returns all the documents with a rating that is among the values specified

db.books.find({rating: {$nin: [4.0, 4.2, 4.4, 4.6, 4.8]}}) // Returns all the documents with a rating that is non among the values specified

### Arrays of simple values and of objects

// genres is an array of strings and is a field in every document.

db.books.find({genres: "Classic"}) // Returns all the documents with "Classic" as part of the genres

db.books.find({genres: ["Fiction", "Historical"]}) // Returns all the documents with the exact same array of genres as specified, in the same order as specified.

db.books.find({genres: {$all: ["Fiction", "Historical"]}})  // This query finds all documents in the "books" collection where the "genres" field contains both "Fiction" and "Historical".

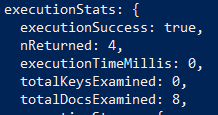
// The $all operator ensures that the "genres" array includes both specified values, regardless of their order in the array.

// reviews is an array of objects and is a field in every document.

db.books.find({"reviews.body": "Tamu sana. Naipenda"}) // Returns all the documents with "reviews" field that whose nested objects have their "body" property value as specified

## Indexing

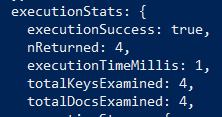
db.books.find({rating: {$gt: 4.7}}).explain('executionStats') // Finds the list of documents with a rating greater than 4.7, looping through all the documents to find the same.



db.books.getIndexes()  // Get all the indexes created for the collection, there is one, "\_id\_" created, from the documents' ids, by deafult

db.books.createIndex({rating: -1})  // Creates an index on the "rating" field in descending order.

db.books.find({rating: {$gt: 4.7}}).explain('executionStats') // Finds the list of documents with a rating greater than 4.7, using the index created from the rating field. Sppeds up the data retrieval because now, there are much fewer documents examined.



db.books.dropIndex({rating: -1})  // Deletes the newly created index

# Nested Documents

This is where documents are stored within documents. This enhances the speed of data retrieval because ordinarily, a normalized database would have at least two tables linked to each other by a foreign key. Querying, in this case, is, therefore, slower compared to if the entities are stored in a single table/ collection. E.g.

{

  "title": "Caucasian Chalk Circle",

  "author": "Bertolt Bretch",

  "pages": 281,

  "genres": [

    "Fiction",

    "Romance",

    "Politics"

  ],

  "rating": 4.8,

  "reviews": [

    { "name": "Daniel", "body": "Pretty good" },

    { "name": "Clemence", "body": "Tamu sana" },

    { "name": "Violet", "body": "Hmm" }

  ]

}

// The reviews are an array of objects

# Aggregation

db.getCollection("persons").aggregate([

    { $match: { "company.location.country": { $nin: ["USA"]}, age: { $gt: 30}} },

    { $sort: { age: 1, name: 1, isActive: 1}}

]);

## Accumulator Operators (sum, avg)

use('personsDatabase');

db.getCollection("persons").aggregate([

{$unwind: "$tags"},

{

    $group: { \_id: {tags: "$tags"}, count: {$sum: NumberInt(1)}}

}])

db.getCollection("persons").aggregate([

    //{$unwind: "$tags"}, // Uncomment if you need to unwind the "tags" field

    {

        $group: {

            \_id: { country: "$company.location.country" },

            averageAge: { $avg: "$age" }

        }

    },

    {

        $project: {

            \_id: 1,

            averageAge: { $toInt: "$averageAge" } // Convert the average to an integer

        }

    }

]);

db.getCollection("persons").aggregate([

    //{$unwind: "$tags"}, // Uncomment if you need to unwind the "tags" field

    {

        $group: {

            \_id: { country: "$company.location.country" },

            averageAge: { $avg: "$age" }

        }

    },

    {

        $project: {

            \_id: 1,

            averageAge: { $round: ["$averageAge", 2] } // Round to 0 decimal places

        }

    }

]);

db.getCollection("persons").aggregate([

    //{$unwind: "$tags"}, // Uncomment if you need to unwind the "tags" field

    {

        $group: {

            \_id: { country: "$company.location.country" },

            minAge: { $min: "$age" }

        }

    }

]);

## Unary Operators

Usually used in the $project operator.

db.getCollection("persons").aggregate([

    { $project : {

        name: 1,

        eyeColorType: { $type: "$eyeColor"},

        age: { $type: "$age"},

        company: { $type: "$company"},

        tags: { $type: "$tags"},

    }}

]);

db.getCollection("persons").aggregate([

    { $project : {

        name: 1,

        eyeColorType: { $type: "$eyeColor"},

        age: { $type: "$age"},

        company: { $type: "$company"},

        tags: { $type: "$tags"},

    }},

    { $out: "aggregationResults"}

]);

# MongoDB CommandLine Tools

## Mongodump

***mongodump --out backup*** – Backs up the databases in your local mongo db server into a folder called “backup”. The default name is “dump”.

***mongodump –d housesearchke -- singleDatabase*** – Backs up the a specific database (housesearchke in this case) into a folder called “singleDatabase”.

***mongodump -d housesearchke -c property\_owners --out singleCollection*** – Backs up the a specific collection (property\_owners) in a specific database (housesearchke in this case) into a folder called “singleCollection”.

***mongodump --uri="mongodb+srv://root:root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" --out backup*** - Backs up the databases in your remote mongo db server into a folder called “backup”. The default name is “dump”.

## Mongorestore

***mongorestore --nsInclude housesearchke.property\_owners singleCollection*** – Restores a collection (property\_owners) from the dump created of the same collection in the folder specified (*singleCollection*).

***mongorestore --uri="mongodb+srv://root:root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" --nsInclude="housesearchke.properties" --dir="C:\Users\Daniel Karongo\Documents\dump***" – Restores a collection (properties) from the dump created of the same collection in the directory specified.

## Mongoexport

***mongoexport --uri="mongodb+srv://root:root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" --db "jobpostings" --collection "jobs" --out "jobs.json" –jsonArray*** – Exports the data in the remote collection jobs to a file on the local machine as an array of json objects.

***mongoexport --uri="mongodb+srv://root:root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" --db="jobpostings" --collection="jobs" --query="{\"exp\": 15}" --out="jobs.json" --jsonArray***– Exports the data in the remote collection “jobs” that matches the query (profile = 15) to a file “jobs.json” on the local machine as an array of json objects.

## Mongoimport

***mongoimport -d housesearchke -c properties --file="housesearchke.properties.json" --jsonArray*** - Imports data, from a local json file with an array of documents, into a local collection.

***mongoimport --uri="mongodb+srv://root:root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" --db="housesearchke" --collection="properties" --file="housesearchke.properties.json" –jsonArray*** – Imports data, from a local json file with an array of documents, into a remote collection.

## Mongostat

***mongostat --uri="mongodb+srv://root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE"*** – Helps monitor mongodb in real time. You will be prompted to provide the password after running the command.

## Mongotop

***mongotop --uri="mongodb+srv://root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" 5*** - Monitor and reports on how much time is spent reading and writing data for each collection in the mongo db cluster provided every 5 seconds.

## Mongofiles

mongofiles is a MongoDB utility tool used to work with files stored in GridFS, MongoDB's specification for storing and retrieving large files, such as images, videos, and documents. mongofiles allows you to interact with GridFS collections to upload, download, list, and delete files. These files are usually in the test database, unless explicitly specified otherwise using the –db option

***mongofiles --uri="mongodb+srv://root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" --db="housesearchke" put housesearchke.properties.json -*** Adds the json file specified to the GridFS collections in MongoDB.

***mongofiles --uri="mongodb+srv://root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" --db="housesearchke" get housesearchke.properties.json -*** Downloads the json file from the GridFS collections in MongoDB.

***mongofiles --uri="mongodb+srv://root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" --db="housesearchke" list –*** Lists all the files in the GridFS collections in MongoDB.

***mongofiles --uri="mongodb+srv://root@housesearchke.y4yxv.mongodb.net/?retryWrites=true&w=majority&appName=HouseSearchKE" --db="housesearchke" delete housesearchke.properties.json -*** deletes the json file from the GridFS collections in MongoDB.