**Mongo DataBase**

**Documentation and Source**

* <https://www.mongodb.com/docs/manual/?_ga=2.100636637.275067343.1700902523-1425033203.1698233789>

**What is Mongo Database?**

MongoDB is a document database with the scalability and flexibility that you want with the querying and indexing that you need

**Document Database**

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.



The advantages of using documents are:

* Documents correspond to native data types in many programming languages.
* Embedded documents and arrays reduce need for expensive joins.
* Dynamic schema supports fluent polymorphism.

Download & Installation

<https://www.mongodb.com/try/download/community>

<https://www.mongodb.com/products/tools/compass>

<https://www.mongodb.com/try/download/shell>

<https://www.mongodb.com/docs/mongodb-shell/run-commands/>

show dbs 🡺 list all databases

use dbname 🡺 switch to database

show collectinos 🡺 list all collections/tables

create new database

use school

db.createCollection(‘students’

db.dropDatabase();

**Add a new record in collection/table**

db.movies.insertOne(

{

title: "The Favourite",

genres: [ "Drama", "History" ],

runtime: 121,

rated: "R",

year: 2018,

directors: [ "Yorgos Lanthimos" ],

cast: [ "Olivia Colman", "Emma Stone", "Rachel Weisz" ],

type: "movie"

}

)

**Response**

{

acknowledged: true,

insertedId: ObjectId("65631a0f975dd5bb0cda7944")

}

**Add multiple records in collection/table**

use sample\_mflix

db.movies.insertMany([

{

title: "Jurassic World: Fallen Kingdom",

genres: [ "Action", "Sci-Fi" ],

runtime: 130,

rated: "PG-13",

year: 2018,

directors: [ "J. A. Bayona" ],

cast: [ "Chris Pratt", "Bryce Dallas Howard", "Rafe Spall" ],

type: "movie"

},

{

title: "Tag",

genres: [ "Comedy", "Action" ],

runtime: 105,

rated: "R",

year: 2018,

directors: [ "Jeff Tomsic" ],

cast: [ "Annabelle Wallis", "Jeremy Renner", "Jon Hamm" ],

type: "movie"

}

])

**Find all records from a table/collection**

db.movies.find()

SELECT \* FROM movies

**Order by desc/asc**

db.movies.find().sort({name:1}) // sort ascending order

db.movies.find().sort({name:-1}) // sort descending order

**Limit documents**

db.movies.find().limit(1);

db.movies.find().limit(3);

**Limit with offset**

db.inventory.find().skip(2).limit(2)

**Order with Limit**

db.movies.find().sort({name:1}).limit(1)

**Find a specific record from a table/collection**

db.movies.find( { "title": "Titanic" } )

SELECT \* FROM movies WHERE title = "Titanic"

db.movies.find({}, { title: true});

db.movies.find({}, { \_id: false, title: true});

db.movies.find({}, { \_id: false, title: true, genres: true});

**More about get documents**

<https://www.mongodb.com/docs/manual/tutorial/query-embedded-documents/>

**Update a document**

db.movies.updateOne( { title: "Twilight" },

{

$set: {

plot: "A teenage girl risks everything–including her life–when she falls in love with a vampire."

},

$currentDate: { lastUpdated: true }

})

**Update Multiple Documents**

db.movies.updateMany(

{ security\_deposit: { $lt: 100 } },

{

$set: { security\_deposit: 100, minimum\_nights: 1 }

}

)

**Update Where exist or not**

db.customers.updateMany( {\_\_v: {$exists: false}}, {$set: {lastName: 'newName'}} )

**Delete column from a document**

db.movies.updateOne({\_id: objectID(‘sdfdsfsd34543irkfjsdk’)}, {$unset: { key:”” } })

**Replace document**

db.accounts.replaceOne(

{ account\_id: 371138 },

{ account\_id: 893421, limit: 5000, products: [ "Investment", "Brokerage" ] }

)

**Delete document**

db.movies.deleteOne( { cast: "Brad Pitt" } )

**Delete Multiple documents**

db.movies.deleteMany( { title: "Titanic" } )

**Comparison Operators**

db.movies.find( { title: {$ne: “Titanic” }} ) // not equal

db.movies.find( runtime: {$lt: 100}} ) // less than

db.movies.find( runtime: {$lte: 100}} ) // less than equal

db.movies.find( runtime: {$gt: 100}} ) // greater than

db.movies.find( runtime: {$gte: 100}} ) //greater than equal

db.movies.find( runtime: { $gt: 50, $lt: 100 } ) // greater than 50 and less than 100

db.movies.find( { rated: { $in: [ "PG", "PG-13" ] } } )

SELECT \* FROM movies WHERE rated in ("PG", "PG-13")

db.movies.find( { countries: "Mexico", "imdb.rating": { $gte: 7 } } )

SELECT \* FROM movies WHERE countries = “Mexico” and imdb.rating >= 7

**Logical Operators and, or**

db.movies.find( $and: [ {title: “Titanic”}, {genres: “Drama”} ] )

Where title = “Titanic” and genres = “Drama”

db.movies.find( $or: [ {title: “Titanic”}, {genres: “Drama”} ] )

Where title = “Titanic” or genres = “Drama”

db.movies.find( {

year: 2010,

$or: [ { "awards.wins": { $gte: 5 } }, { genres: "Drama" } ]

} )

SELECT \* FROM movies WHERE year = 2010 or ( awards.wins >= 5 and genres = "Drama" )

db.movies.find( runtime: {$not { $gt: 50 }} )

where not greater than 50

**To get execution status**

db.movies.find().explain("executionStats")

**Creating indexes**

db.movies.createIndex({title: 1})

db.movies.getIndexes()

db.movies.dropIndex(“title”)

**Aggregate quries**

db.inventory.aggregate([{$match:{qty:25}}]) 🡺 match all the records based on condition

db.inventory.aggregate([{$group: {\_id: '$qty'}}]) 🡺 group all the records based on condition

db.inventory.aggregate([{$match:{qty: 25}},{$count: "total records are"}])

db.inventory.aggregate([{$match:{qty: 25}},{$group: {\_id: "$qty", total: {$sum: "$qty"}}}])