Mongo DB

MongoDB is an Opensource and free dbms which follows document oriented structure.

It uses JSON like key-value structure to store / retrieve data.

Database is a physical container of collections.

Each Database can have multiple collections – its own set of files on physical space.

MongoDB hosts multiple Database

Collection = group of MongoDB document

MongoDB does not have concept of JOINS

Collections do not have any schema defined

Document is a set of key-value pair

Each document comes with a unique value – “\_id” which is made up of timestamp and other factors.

Document have flexible and dynamic schema – it is user defined, not static

Any valid data can be hold by document

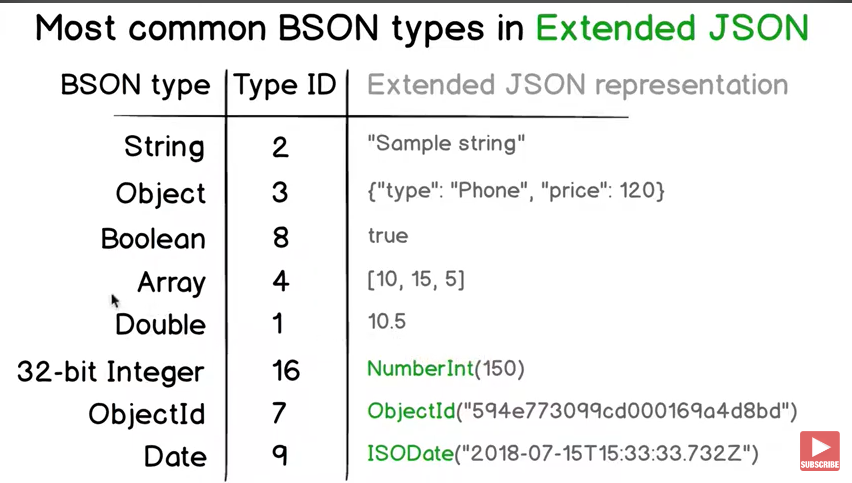
Inside a collection, there can be heterogeneous documents, means multiple rows with different schema

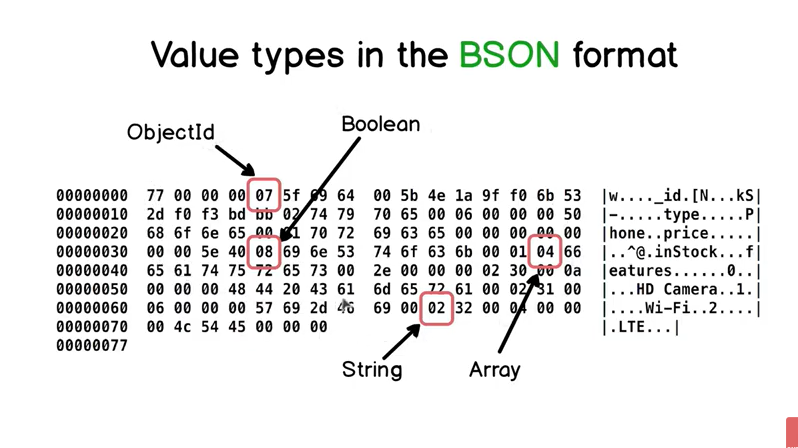
Generally the document with a collection are related data - belonging to a particular subject

MongoDB Shell

* Run Mongo.exe using terminal
* This will open terminal

BSON Data Types:





Creating Collections –

show dbs;

use <database name>;

db.createCollection(‘<collection-name>’);

Deleting Database / collections –

Delete a collection –

db.collection\_name.drop()

Delete a Database –

db.dropDatabase()

Inserting Data –

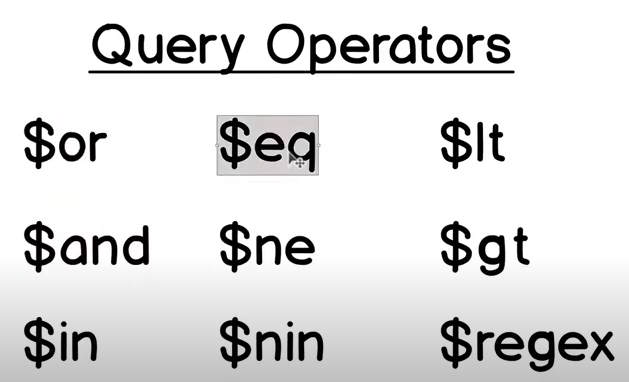
db.collection.insertOne({…});

db.collection.insertMany([{…}, {…},.. {…}]);

Find Data –

db.collection.find({query})

db.collection.findOne({query})



MongoDB Cheat Sheet

Terminology

Database

A container for collections. This is the same as a database in SQL and

usually each project will have its own database full of different collections.

Collection

A grouping of documents inside of a database. This is the same as a table in

SQL and usually each type of data (users, posts, products) will have its own

collection.

Document

A record inside of a collection. This is the same as a row in SQL and usually

there will be one document per object in the collection. A document is also

essentially just a JSON object.

Field

A key value pair within a document. This is the same as a column in SQL.

Each document will have some number of fields that contain information

such as name, address, hobbies, etc. An important difference between SQL

and MongoDB is that a field can contain values such as JSON objects, and

arrays instead of just strings, number, booleans, etc.

Basic Commands

mongosh

Open a connection to your local MongoDB instance. All other commands

will be run within this mongosh connection.

show dbs

Show all databases in the current MongoDB instance

use <dbname>

use myDatabase

Switch to the database provided by dbname

Switch to myDatabase

db

Show current database name

cls

Clear the terminal screen

show collections

Show all collections in the current database

db.dropDatabase() Delete the current database

exit Exit the mongosh session

Create

Each of these commands is run on a specific collection

db.<collectionName>.<command>

insertOne

db.users.insertOne({ name: “Kyle” })

Create a new document inside the specified collection

Add a new document with the name of Kyle into the users collection

insertMany

db.users.insertMany([{ age: 26 }, { age: 20 }])

Create multi new documents inside a specific collection

Add two new documents with the age of 26 and 20 into the users collection

Read

Each of these commands is run on a specific collection

db.<collectionName>.<command>

find

db.users.find()

Get all documents

Get all users

find(<filterObject>)

db.users.find({ name: “Kyle” })

db.users.find({ “address.street”: “123 Main St” })

Find all documents that match the filter object

Get all users with the name Kyle

Get all users whose adress field has a street field with the value 123 Main St

find(<filterObject>, <selectObject>)

db.users.find({ name: “Kyle” }, { name: 1, age: 1 })

db.users.find({}, { age: 0 })

Find all documents that match the filter object but only

return the field specified in the select object

Get all users with the name Kyle but only return their name, age, and \_id

Get all users and return all columns except for age

findOne

db.users.findOne({ name: “Kyle” })

The same as find, but only return the first document that

matches the filter object

Get the first user with the name Kyle

countDocuments

db.users.countDocuments({ name: “Kyle” })

Return the count of the documents that match the filter

object passed to it

Get the number of users with the name Kyle

Update

Each of these commands is run on a specific collection

db.<collectionName>.<command>

updateOne

db.users.updateOne({ age: 20 }, { $set: { age: 21 } })

Update the first document that matches the filter object

with the data passed into the second parameter which is the

update object

Update the first user with an age of 20 to the age of 21

updateMany

db.users.updateMany({ age: 12 }, { $inc: { age: 3 } })

Update all documents that matches the filter object with the

data passed into the second parameter which is the update

object

Update all users with an age of 12 by adding 3 to their age

replaceOne

db.users.replaceOne({ age: 12 }, { age: 13 })

Replace the first document that matches the filter object

with the exact object passed as the second parameter. This

will completely overwrite the entire object and not just

update individual fields. Replace the first user with an age of 12 with an object that has the age of 13 as

its only field

Delete

Each of these commands is run on a specific collection

db.<collectionName>.<command>

deleteOne

db.users.deleteOne({ age: 20 })

Delete the first document that matches the filter object

Delete the first user with an age of 20

deleteMany

db.users.deleteMany({ age: 12 })

Delete all documents that matches the filter object

Delete all users with an age of 12

Complex Filter Object

Any combination of the below can be use inside a filter object to make complex queries

$eq

db.users.find({ name: { $eq: “Kyle” } })

Check for equality

Get all users with the name Kyle

$ne

db.users.find({ name: { $ne: “Kyle” } })

Check for not equal

Get all users with a name other than Kyle

$gt / $gte

db.users.find({ age: { $gt: 12 } })

db.users.find({ age: { $gte: 15 } })

Check for greater than and greater than or equal to

Get all users with an age greater than 12

Get all users with an age greater than or equal to 15

$lt / $lte

db.users.find({ age: { $lt: 12 } })

db.users.find({ age: { $lte: 15 } })

Check for less than and less than or equal to

Get all users with an age less than 12

Get all users with an age less than or equal to 15

$in

db.users.find({ name: { $in: [“Kyle”, “Mike”] } })

Check if a value is one of many values

Get all users with a name of Kyle or Mike

$nin

db.users.find({ name: { $nin: [“Kyle”, “Mike”] } })

Check if a value is none of many values

Get all users that do not have the name Kyle or Mike

$and

db.users.find({ $and: [{ age: 12 }, { name: “Kyle” }] })

db.users.find({ age: 12, name: “Kyle” })

Check that multiple conditions are all true

Get all users that have an age of 12 and the name Kyle

This is an alternative way to do the same thing. Generally you do not need $and.

$ordb.users.find({ $or: [{ age: 12 }, { name: “Kyle” }] })

Check that one of multiple conditions is true

Get all users with a name of Kyle or an age of 12

$not

db.users.find({ name: { $not: { $eq: “Kyle” } } })

Negate the filter inside of $not

Get all users with a name other than Kyle

$existsdb.users.find({ name: { $exists: true } })

Check if a field exists

Get all users that have a name field

$exprdb.users.find({ $expr: { $gt: [“$balance”, “$debt”] } })

Do comparisons between different fields

Get all users that have a balance that is greater than their debt

Complex Update Object

Any combination of the below can be use inside an update object to make complex updates

$set

db.users.updateOne({ age: 12 }, { $set: { name: “Hi” } })

Update only the fields passed to $set. This will not affect

any fields not passed to $set.

Update the name of the first user with the age of 12 to the value Hi

$inc

db.users.updateOne({ age: 12 }, { $inc: { age: 2 } })

Increment the value of the field by the amount given

Add 2 to the age of the first user with the age of 12

$rename

db.users.updateMany({}, { $rename: { age: “years” } })

Rename a field

Rename the field age to years for all users

$unset

db.users.updateOne({ age: 12 }, { $unset: { age: “” } })

Remove a field

Remove the age field from the first user with an age of 12

$push

db.users.updateMany({}, { $push: { friends: “John” } })

Add a value to an array field

Add John to the friends array for all users

$pull

db.users.updateMany({}, { $pull: { friends: “Mike” } })

Remove a value from an array field

Remove Mike from the friends array for all users

Read Modifiers

Any combination of the below can be added to the end of any read operation

sort

db.users.find().sort({ name: 1, age: -1 })

Sort the results of a find by the given fields

Get all users sorted by name in alphabetical order and then if any names are the

same sort by age in reverse order

limit

db.users.find().limit(2)

Only return a set number of documents

Only return the first 2 users

skip

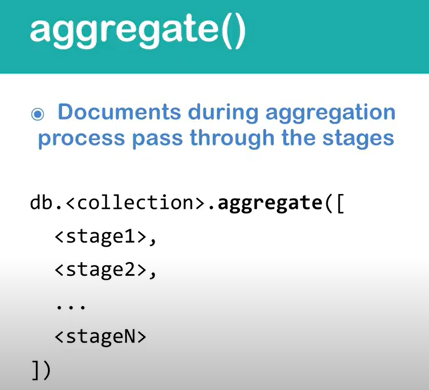
db.users.find().skip(4)

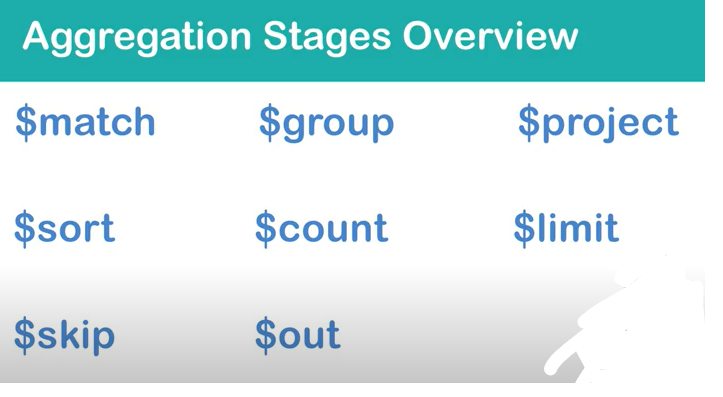
Skip a set number of documents from the beginning

Skip the first 4 users when returning results. This is great for pagination when

combined with limit.

Aggregation –





$match : Will filter the docs

Syntax:

db.Person.aggregate(

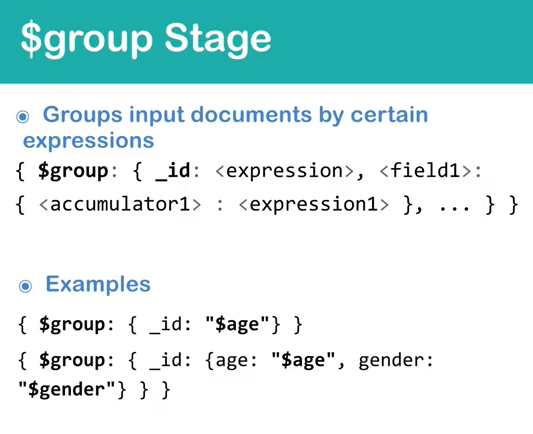
[

{ $match: { age: {$gt : 20}}} // Can be any find query

]

)

$group: Consider a common group by statement –



Select name as EmpName, sum(salary) as TotalSalary from employee where name like ‘A%’ group by name order by 2 desc;

Here the where condition is the $match

Group by is $group -> \_id

sum(salary) as TotalSalary is nothing but -> TotalSalary: {$sum: “$salary”} (and EmpName is \_id basically)

Then order by is -> $sort:{“$TotalSalary”: -1}

Here are some additional commands also and lets discover them one by one.

$count will count all the docs coming till previous stage –

{

$count: “FieldName-to-be-displayed”

}

Example –

db.Person.aggregate(

[

{$match: {"company.location.country":{$exists: true}}},

{$group: {

\_id: {

country: "$company.location.country",

gender: "$gender"

}

}}

,

{$count:"Counttillstage2"}

]

)

Result –

{

"Counttillstage2" : 8

}

$Sort – will sort the docs based on fields passed till previous stage

Example –

db.Person.aggregate(

[

{$match: {"company.location.country":{$exists: true}}},

{$sort: {age:1, gender:-1}}

]

)

== Order by age, gender desc;