**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

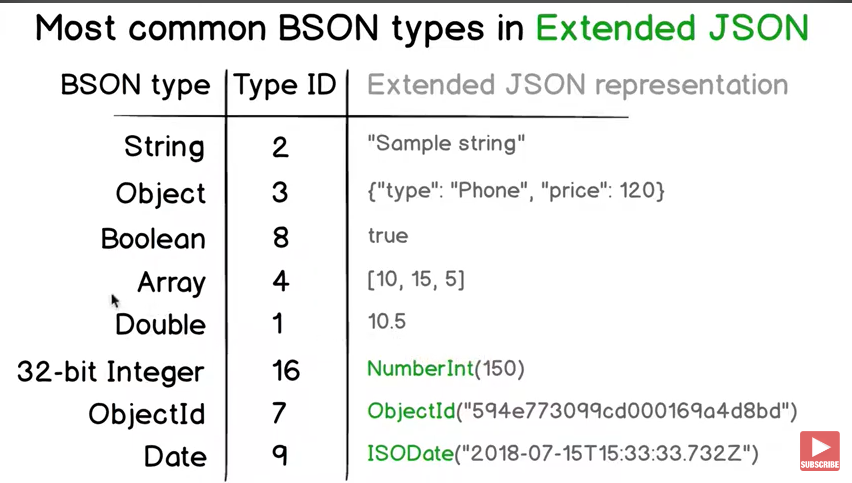
Inside a collection, there can be heterogeneous documents, means multiple rows with different schema

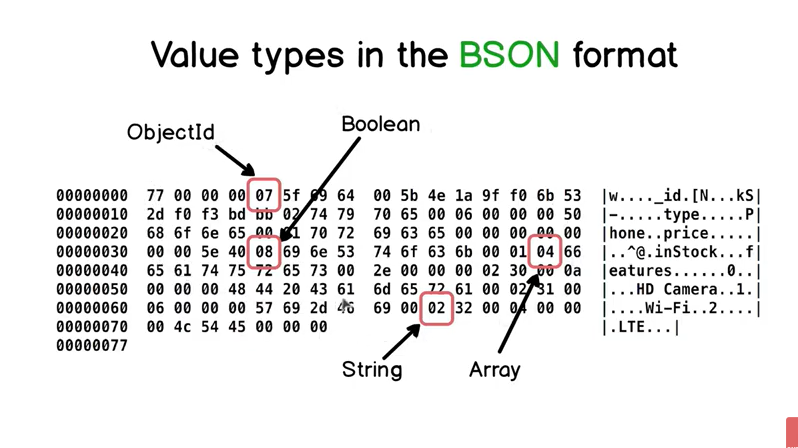
Generally the document with a collection are related data - belonging to a particular subject

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 –**

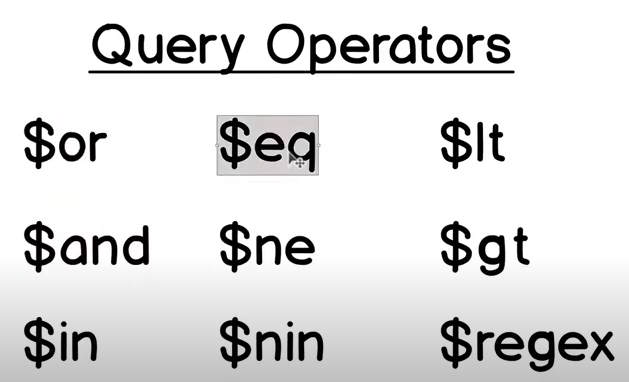
db.collection.insertOne({…});

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

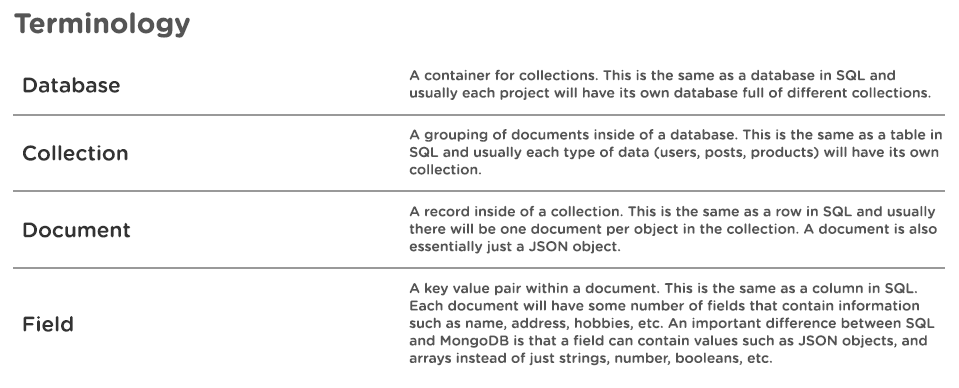
**Find Data –**

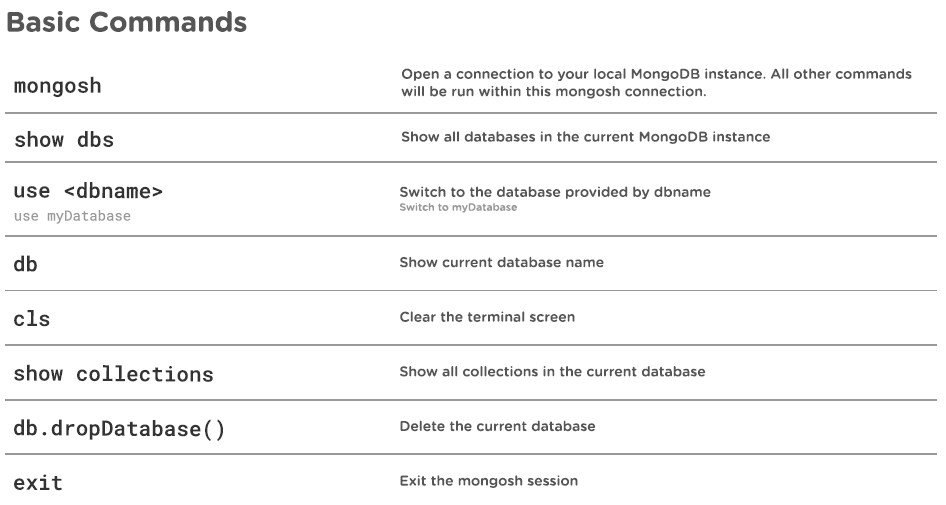
db.collection.find({query})

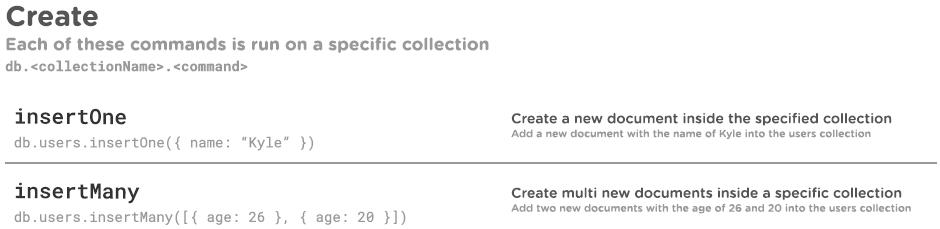
db.collection.findOne({query})



MongoDB Cheat Sheet

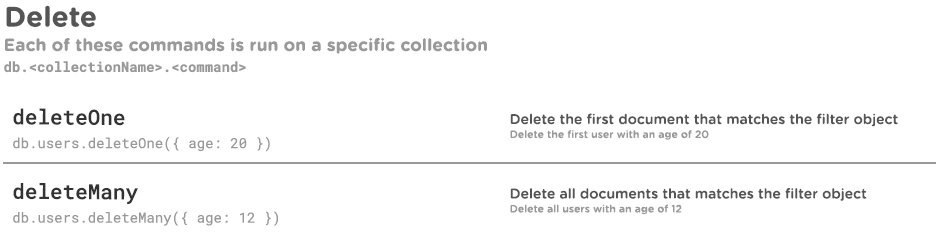


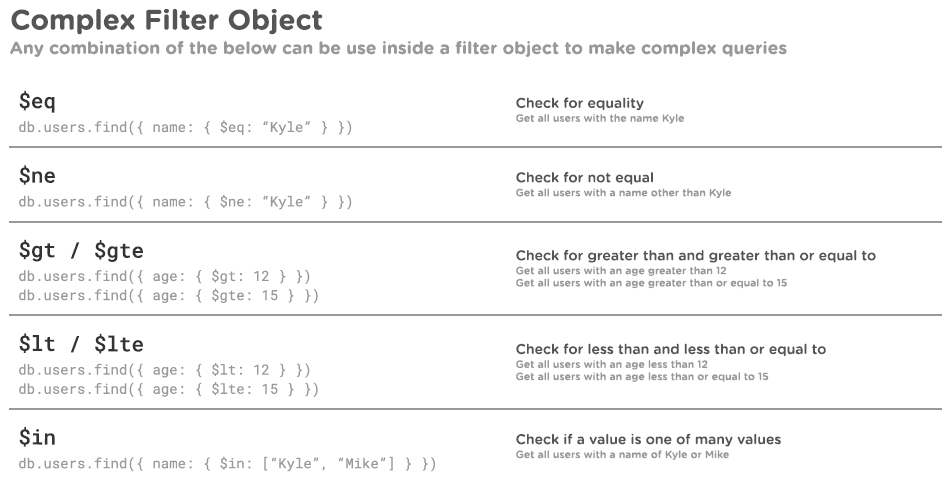


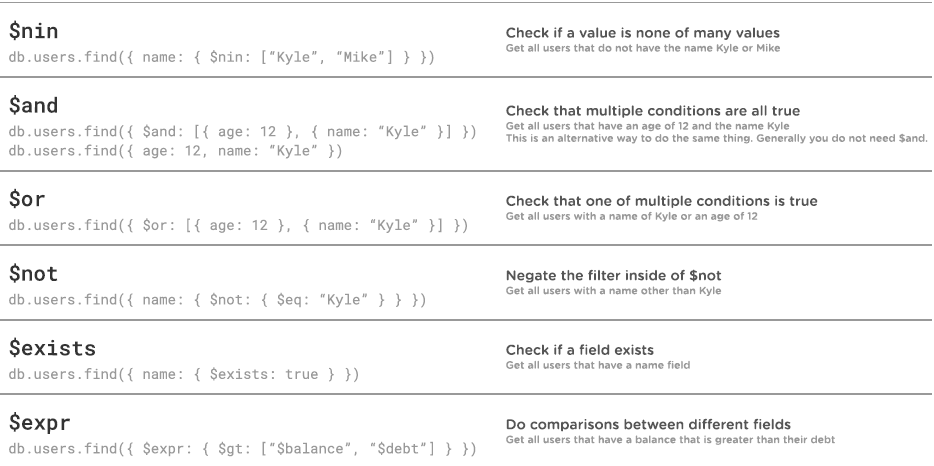


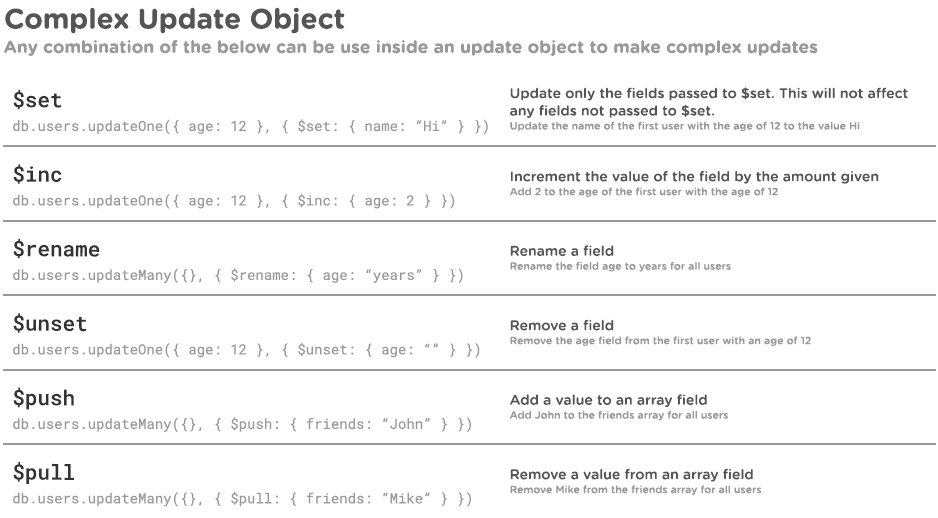


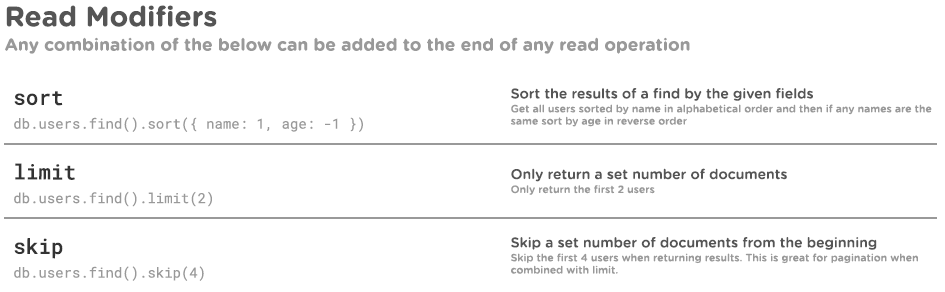












Notes:

Query on Nested Docs:

We can access nested docs using periods (.) and apply the same operators we have used in normal docs –

db.inventory.find( { **"size.uom"**: **"in"** } )

Query on Array:

Suppose we want to match the entire Array –

db.inventory.find( { tags: [**"red"**, **"blank"**] } )

This will make sure that array has **exactly 2 elements, red and blank, in the same order.**

But if we want to ignore the order and just want to check the values we can use the $all operator

db.inventory.find( { tags: { $all: [**"red"**, **"blank"**] } } )

If we simply want to check if an element is present in the array or not, we can specify the element itself –

db.inventory.find( { tags: **"red"** } )

This will simply run the query against each element and return if at least one match found.

Similar can be done for other operators as well. For example, the following operation queries for all documents where the array dim\_cm contains at least one element whose value is greater than 25.

db.inventory.find( { dim\_cm: { $gt: 25 } } )

db.inventory.find( { dim\_cm: { $gt: 15, $lt: 20 } } )

**Note that above will be satisfying even if there are distinct element satisfying these conditions, eg [10,25] -> 10 is less than 20 and 25 is greater than 15**

To make sure that the condition is checked on single element, we have to use **$elemMatch**

db.inventory.find( { dim\_cm: { $elemMatch: { $gt: 22, $lt: 30 }}} )

Query on specific index – we can use the index number (0-based) to query on specific index –

db.inventory.find( { **"dim\_cm.3"**: { $gt: 25 } } ) // 4th element>25

Length of Array –

db.inventory.find( { **"tags"**: { $size: 3 } } )

Query on Array of Embedded Docs –

Just like a normal array, we can add period to access the properties.

For example, assume a document Schema like –

{ item: **"journal"**, instock: [ { warehouse: **"A"**, qty: 5 }, { warehouse: **"C"**, qty: 15 } ] }

db.inventory.find( { **'instock.qty'**: { $lte: 20 } } ) // Will match any doc having a doc in instock array with qty<=20

Notable query is, when we pass entire doc –

db.inventory.find( { **"instock"**: { warehouse: **"A"**, qty: 5 } } )

At first it may seem like it will return all docs having warehouse=A and qty=5. But here is the twist, we are passing an Array here… and it will return all the docs with exact match, even the order of the fields.

db.inventory.find( { **'instock.0.qty'**: { $lte: 20 } } )//check if doc at 0th position having qty <= 20

db.inventory.find( { **"instock"**: { $elemMatch: { qty: 5, warehouse: **"A"** } } } ) // Check if there exists a doc whose instock array contains an element having both qty=5 AND warehouse=A

db.inventory.find( { **"instock"**: { $elemMatch: { qty: { $gt: 10, $lte: 20 } } } } )// where the instock array has at least one embedded document that contains the field qty that is greater than 10 and less than or equal to 20

db.inventory.find( { **"instock.qty"**: { $gt: 10, $lte: 20 } } )//all docs where atleast one embedded doc having qty>10 and atleast one having qty<=20

The following example queries for documents where the instock array has at least one embedded document that contains the field qty equal to 5 and at least one embedded document (but not necessarily the same embedded document) that contains the field warehouse equal to A:

db.inventory.find( { **"instock.qty"**: 5, **"instock.warehouse"**: **"A"** } )

Projection Queries –

We can show/hide the fields using 0/1 in the second document we pass in find Query. Below are some additional points –

Suppose we want to suppress a specific field in nested doc –

|  |
| --- |
| db.inventory.find( |
| { status: **"A"** }, |
| { **"size.uom"**: 0 } |
| ) |

OR, { size: { uom: 0 } }.

To Manipulate the array during projection we can use $elemMatch, $ and $splice

Example with $elemMatch –

db.players.find( {}, { games: { $elemMatch: { score: { $gt: 5 } } }, joined: 1, lastLogin: 1 } ) // in games array, include only those sub-docs having score>5

Example, to get the last element of an array –

db.inventory.find( { status: **"A"** }, { item: 1, status: 1, instock: { $slice: -1 } } )

<https://www.mongodb.com/docs/manual/reference/operator/projection/slice/#mongodb-projection-proj.-slice> for understanding $slice syntax.

**Basically, slice can take two kinds of arguments –**

**$slice: <n> // if n is positive, return first ‘n’ elements, else last n elements (like range function)**

**$slice: [n, m] // Skip n elements and return m elements (index, number\_of\_elements)**

Eg, to get elements from index 2 to 7, we will write –

$slice : [2, 6] {n=2, m=(7-2)+1 (number of elements)}

$ operator –

It will match and include only first element that matches the conditions specified in the former doc (first param of find method)

|  |
| --- |
| db.students.find( { semester: 1, grades: { $gte: 85 } }, |
| { **"grades.$"**: 1 } ) |

Result -

|  |
| --- |
| { **"\_id"** : 1, **"grades"** : [ 87 ] } |
| { **"\_id"** : 2, **"grades"** : [ 90 ] } |
| { **"\_id"** : 3, **"grades"** : [ 85 ] } |

Although the array field grades may contain multiple elements that are greater than or equal to 85, the [$](https://www.mongodb.com/docs/manual/reference/operator/projection/positional/#mongodb-projection-proj.-) projection operator returns only the first matching element from the array.

In the following query, the projection { "grades.$": 1 } returns only the first element with the mean greater than 70 for the grades field:

|  |
| --- |
| db.students.find( |
| { **"grades.mean"**: { $gt: 70 } }, |
| { **"grades.$"**: 1 } |
| ) |

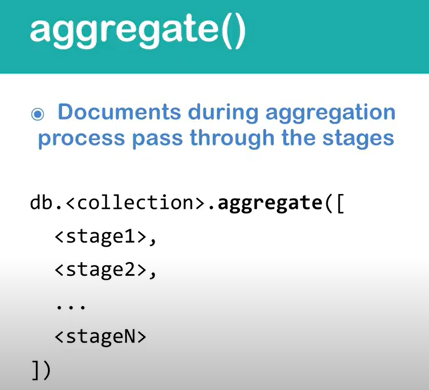
The operation returns the following documents: (first doc with mean>70)

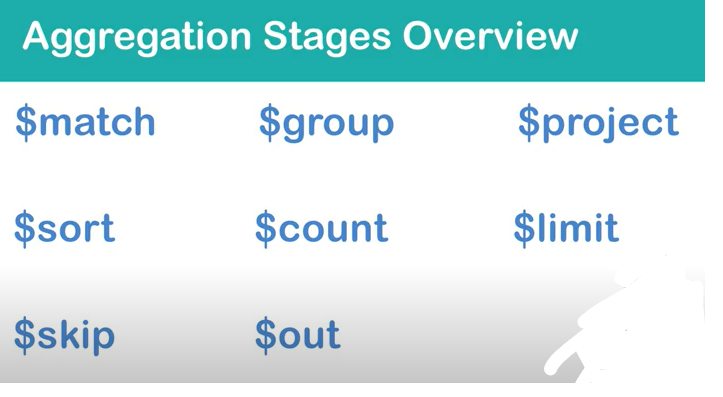
|  |
| --- |
| { **"\_id"** : 7, **"grades"** : [ { **"grade"** : 80, **"mean"** : 75, **"std"** : 8 } ] } |
| { **"\_id"** : 8, **"grades"** : [ { **"grade"** : 92, **"mean"** : 88, **"std"** : 8 } ] } |

Additional Update Methods – The following methods can also update documents from a collection:

* [db.collection.findOneAndReplace().](https://www.mongodb.com/docs/manual/reference/method/db.collection.findOneAndReplace/#mongodb-method-db.collection.findOneAndReplace)
* [db.collection.findOneAndUpdate().](https://www.mongodb.com/docs/manual/reference/method/db.collection.findOneAndUpdate/#mongodb-method-db.collection.findOneAndUpdate)
* [db.collection.findAndModify().](https://www.mongodb.com/docs/manual/reference/method/db.collection.findAndModify/#mongodb-method-db.collection.findAndModify)
* [db.collection.bulkWrite().](https://www.mongodb.com/docs/manual/reference/method/db.collection.bulkWrite/#mongodb-method-db.collection.bulkWrite)

**Aggregation –**





$match : Will filter the docs

Syntax:

db.Person.aggregate(

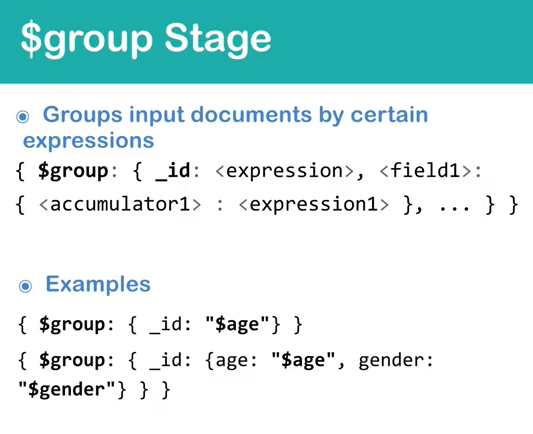
[

{ $match: { age: {$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;