Day-3 (Doc 06)

Employee ID: 201933938

Login ID: iamasif

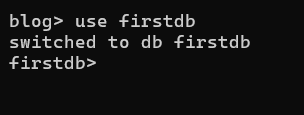
Name: Shaik Asif

MongoDb Scripts

Insert

use firstdb

# this will create db firstdb



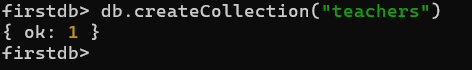
#### create collection by 2 ways

db.students.insertOne({name: "kim", age: 12 , standard: "seventh"})

# This is going to create collection students with one document in it.

db.createCollection("teachers")

# This is going to create collection teachers specifically



db.teachers.insertMany([

{

name:"roger",

subject:"maths"

},

{

name:"anita",

subject:"chemistry"

},

{

name:"thomas",

subject:"physics"

},

{

name:"tina",

subject:"maths"

},

{

name:"jonty",

subject:"english"

}

]

)



# This command will create multiple documents in teachers collection. We are passing an array of json documents.

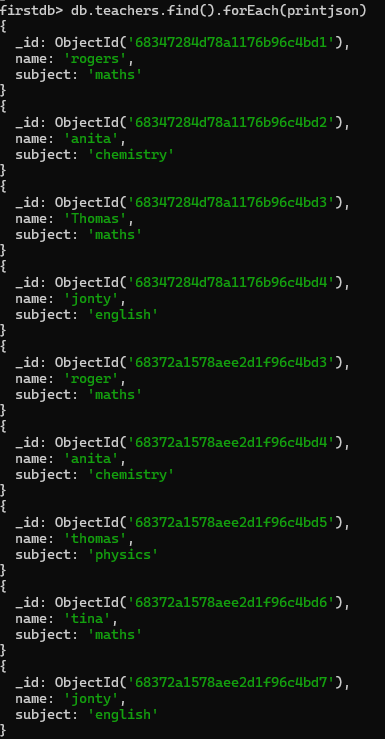
db.teachers.find()

#this will show all the documents present in the teachers collection



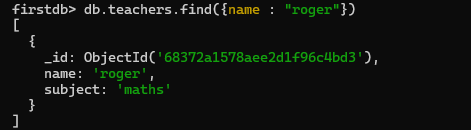
db.teachers.find().forEach(printjson)

# This will print result in json format.



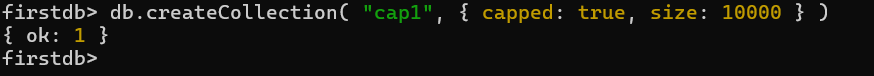
db.teachers.find({name : "roger"})

# this will find the document where name field is "roger"



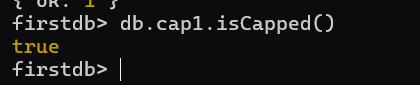
db.createCollection( "cap1", { capped: true, size: 10000 } )

#this is going create capped collection of 10000 bytes. Capped collection has to created explicitly

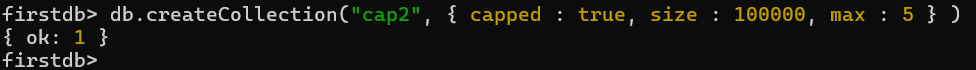


db.cap1.isCapped()

# command checks if collection is capped.



db.createCollection("cap2", { capped : true, size : 100000, max : 5 } )



db.cap2.insertMany(

[

{

\_id:10,

name:"neena",

age:12,

standard:"fifth"

},

{

\_id:11 ,

name:"prince",

age:7,

standard:"third"

},

{

\_id:12,

name:"leeta",

age:8,

standard:"fourth"

},

{

\_id:13,

name:"jhon",

age:4,

standard:"first"

}

]

)

# this is going to insert 4 entries in our capped collection cap2.



db.cap2.insertMany(

[

{

\_id:14,

name:"sheena",

age:16,

standard:"junior"

},

{

\_id:15 ,

name:"kris",

age:18,

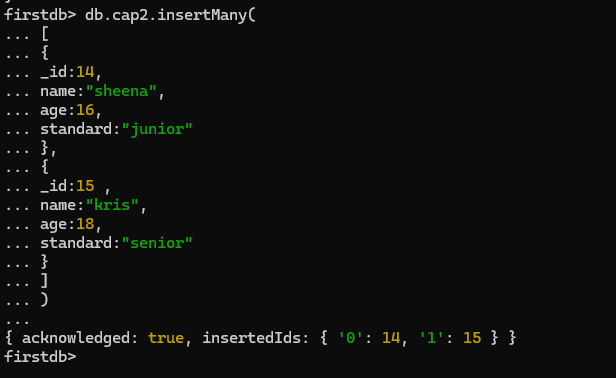
standard:"senior"

}

]

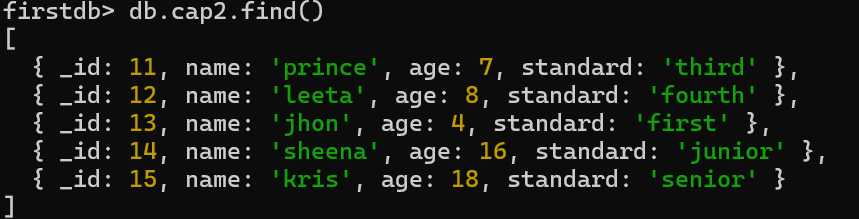
)

# This is going to insert 2 entries and since size is capped to 5 entries last entry from collection will get removed.



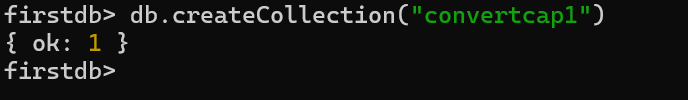
db.cap2.find()

# this will show only five entries and \_id is removed.



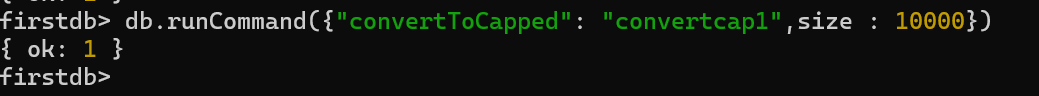
db.createCollection("convertcap1")

# create non capped collection



db.runCommand({"convertToCapped": "convertcap1",size : 10000})

#convert a collection to capped one



#Ordered bulk

var bulkOrdered = db.bulk1.initializeOrderedBulkOp()

bulkOrdered.insert({\_id:1, name:"Murugan"});

bulkOrdered.insert({\_id:2, name:"Niyas"});

bulkOrdered.insert({\_id:3, name:"Nitin"});

bulkOrdered.insert({\_id:1, name:"Murugan"});

bulkOrdered.insert({\_id:4, name:"Adarsh"});

bulkOrdered.execute();

or

--------------------------------------------------

db.runCommand({"convertToCapped": "convertcap1",size : 10000})

{ ok: 1 }

var myVar = db.bulk1.initializeOrderedBulkOp()

myVar.insert({\_id:2, name:"Niyas"});

myVar.insert({\_id:3, name:"Nitin"});

myVar.insert({\_id:1, name:"Murugan"});

myVar.insert({\_id:4, name:"Adarsh"});

myVar.execute();

--------------------------------------------------

db.bulk1.find()

#Unordred bulk

var bulkUnOrdered = db.bulk2.initializeUnorderedBulkOp()

bulkUnOrdered.insert({\_id:1, name:"Murugan"});

bulkUnOrdered.insert({\_id:2, name:"Niyas"});

bulkUnOrdered.insert({\_id:3, name:"Nitin"});

bulkUnOrdered.insert({\_id:1, name:"Murugan"});

bulkUnOrdered.insert({\_id:4, name:"Adarsh"});

bulkUnOrdered.execute();

db.bulk2.find()

—-----------------------------------------------------------------------------------------------------------

0000000000000000000000000000000000000000000000000000000000000

—------------------------------------------------------------------------------------------------------

Delete

This document would help understanding all the command related to deleting the data.

1. Let insert the data in

db.deleteplist.insertMany( [

{ product: "register", quantity: 25, size: { height: 14, weight: 21, unit: "cm" }, status: "A" },

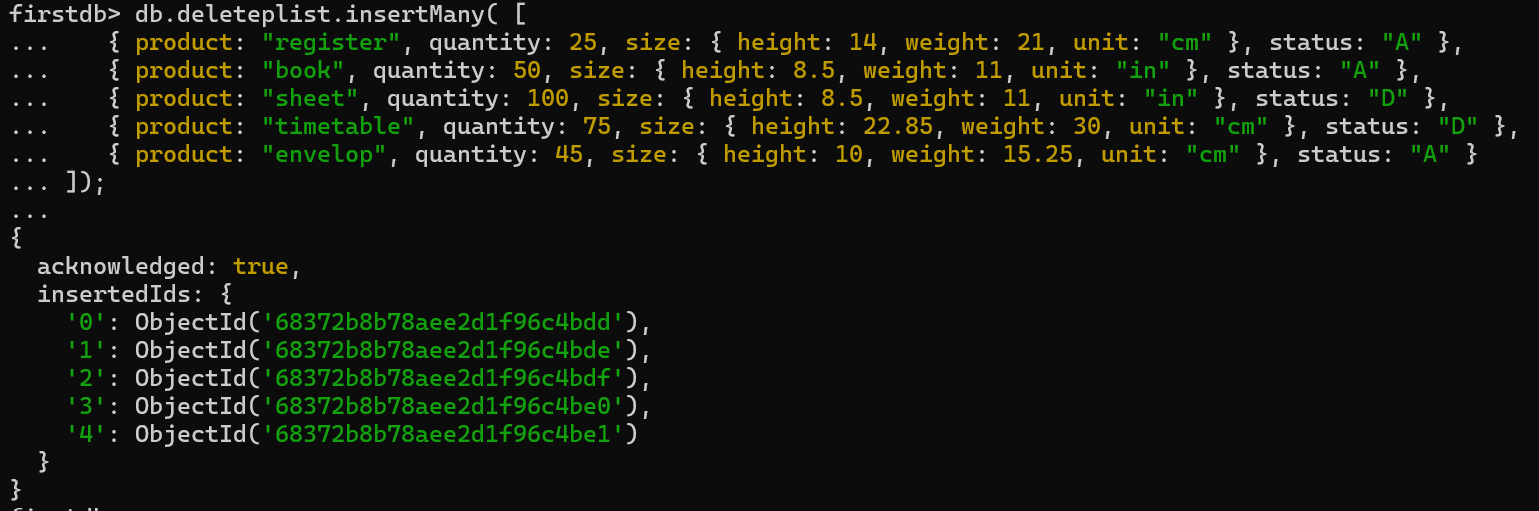
{ product: "book", quantity: 50, size: { height: 8.5, weight: 11, unit: "in" }, status: "A" },

{ product: "sheet", quantity: 100, size: { height: 8.5, weight: 11, unit: "in" }, status: "D" },

{ product: "timetable", quantity: 75, size: { height: 22.85, weight: 30, unit: "cm" }, status: "D" },

{ product: "envelop", quantity: 45, size: { height: 10, weight: 15.25, unit: "cm" }, status: "A" }

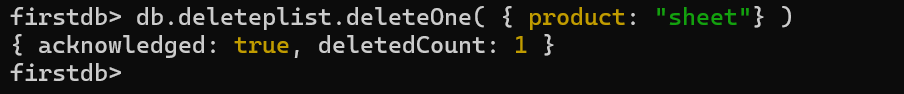
]);



Above data of products inserted

2. let delete the data using deleteOne command with simple condition.

db.deleteplist.deleteOne( { product: "sheet"} )



let check if we can find the deleted item.

db.deleteplist.find( { product: "sheet"} )

3. we can also delete a document based on the embedded document.

db.deleteplist.deleteOne( { size: { height: 14, weight: 21, unit: "cm" } } )

lets check if we have any such document.

db.deleteplist.find( { size: { height: 14, weight: 21, unit: "cm" } } )

so far we have deleted 2 documents from the collection. lets verify that.

db.deleteplist.find()

4. we can also delete document based on the embedded document fields.

db.deleteplist.deleteOne( { "size.unit": "in" } )

5. we can also delete document based on the embedded document fields which can compared such as numbers.

db.deleteplist.deleteOne( { "size.height": { $lt: 15 } } )

6. Lets add more data to our current collection

db.deleteplist.insertMany([

{ product: "register", quantity: 25, tags: ["blank", "red"], dim\_cm: [ 14, 21 ] },

{ product: "book", quantity: 50, tags: ["red", "blank"], dim\_cm: [ 14, 21 ] },

{ product: "sheet", quantity: 100, tags: ["red", "blank", "plain"], dim\_cm: [ 14, 21 ] },

{ product: "timetable", quantity: 75, tags: ["blank", "red"], dim\_cm: [ 22.85, 30 ] },

{ product: "envelop", quantity: 45, tags: ["blue"], dim\_cm: [ 10, 15.25 ] }

]);

7. lets apply more complext conditions in deleteOne method. we are passing exact array to delete document which matching the array.

db.deleteplist.deleteOne( { tags: ["red", "blank"] } )

db.deleteplist.deleteMany( { tags: ["red", "blank"] } )

8. we can delete more the one document by passing single element of array in it.

db.deleteplist.deleteMany( { tags: "red" } )

let see how many items are remaining in our collection.

db.deleteplist.find()

9. We will be inserting data with arrays of embedded documents.

db.deleteplist.insertMany( [

{ product: "register", instock: [ { warehouse: "A", quantity: 5 }, { warehouse: "C", quantity: 15 } ] },

{ product: "book", instock: [ { warehouse: "C", quantity: 5 } ] },

{ product: "sheet", instock: [ { warehouse: "A", quantity: 60 }, { warehouse: "B", quantity: 15 } ] },

{ product: "timetable", instock: [ { warehouse: "A", quantity: 40 }, { warehouse: "B", quantity: 5 } ] },

{ product: "envelop", instock: [ { warehouse: "B", quantity: 15 }, { warehouse: "C", quantity: 35 } ] }

]);

10. we can simply delete many documents by passing an embedded document.

db.deleteplist.deleteMany( { "instock": { warehouse: "A", quantity: 5 } } )

11. there is one more command for removing the document.

db.collection.remove() or rather db.deleteplist.remove()

we are not going to learn about it since this method is depricated thus it will removed in higher versions of mongodb. it works as same as deletemany except it return

other result.

db.deleteplist.remove({ 'instock.quantity': { $lte: 20 } })

12. lets insert more data in out collection.

db.deleteplist.insertMany( [

{ product: "register", status: "A", size: { height: 14, weight: 21, unit: "cm" }, instock: [ { warehouse: "A", quantity: 5 } ] , creationts: ISODate("2015-11-01T12:30:15Z")},

{ product: "book", status: "A", size: { height: 8.5, weight: 11, unit: "in" }, instock: [ { warehouse: "C", quantity: 5 } ], creationts: ISODate("2015-11-01T12:30:15Z") },

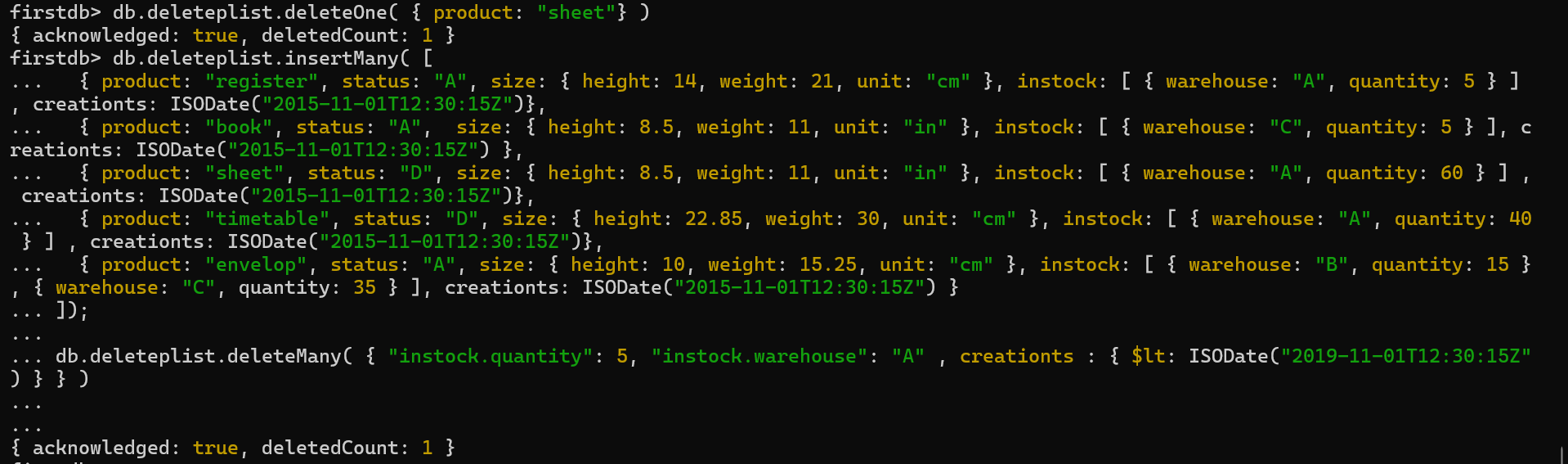
{ product: "sheet", status: "D", size: { height: 8.5, weight: 11, unit: "in" }, instock: [ { warehouse: "A", quantity: 60 } ] , creationts: ISODate("2015-11-01T12:30:15Z")},

{ product: "timetable", status: "D", size: { height: 22.85, weight: 30, unit: "cm" }, instock: [ { warehouse: "A", quantity: 40 } ] , creationts: ISODate("2015-11-01T12:30:15Z")},

{ product: "envelop", status: "A", size: { height: 10, weight: 15.25, unit: "cm" }, instock: [ { warehouse: "B", quantity: 15 }, { warehouse: "C", quantity: 35 } ], creationts: ISODate("2015-11-01T12:30:15Z") }

]);

db.deleteplist.deleteMany( { "instock.quantity": 5, "instock.warehouse": "A" , creationts : { $lt: ISODate("2019-11-01T12:30:15Z") } } )



—----------------------------------------------------------------------------------------------------------------

((((((((((((((((((((((((((((((((((((((((((((((((((((((((((((())))))))))))))))))))))))))))))))))))))))))))))))))))))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Aggregate:**

This tutorial about the data aggregation.

Aggregation operations process data records and return computed results.

Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.

There are three types of aggregation

aggregation pipeline,

the map-reduce function

single purpose aggregation methods.

1. lets insert the data first. lets call this aggregateproduct

db.aggregateproduct.insertMany( [

{ product: "register", quantity: 25, size: { height: 14, weight: 21, unit: "cm" }, status: "A" },

{ product: "book", quantity: 50, size: { height: 8.5, weight: 11, unit: "in" }, status: "A" },

{ product: "sheet", quantity: 100, size: { height: 8.5, weight: 11, unit: "in" }, status: "D" },

{ product: "timetable", quantity: 75, size: { height: 22.85, weight: 30, unit: "cm" }, status: "D" },

{ product: "envelop", quantity: 45, size: { height: 10, weight: 15.25, unit: "cm" }, status: "A" },

{ product: "register", quantity: 25, size: { height: 14, weight: 21, unit: "cm" }, status: "A" },

{ product: "book", quantity: 50, size: { height: 8.5, weight: 11, unit: "in" }, status: "A" },

{ product: "sheet", quantity: 100, size: { height: 8.5, weight: 11, unit: "in" }, status: "D" },

{ product: "timetable", quantity: 75, size: { height: 22.85, weight: 30, unit: "cm" }, status: "D" },

{ product: "envelop", quantity: 45, size: { height: 10, weight: 15.25, unit: "cm" }, status: "A" },

{ product: "register", quantity: 25, size: { height: 14, weight: 21, unit: "cm" }, status: "A" },

{ product: "book", quantity: 50, size: { height: 8.5, weight: 11, unit: "in" }, status: "A" }

]);

2 lets query the data based aggregate pipelining

db.aggregateproduct.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$product", total: { $sum: "$quantity" } } }

])

First Stage: The $match stage filters the documents by the status field and passes to the next stage those documents that have status equal to "A".

Second Stage: The $group stage groups the documents by the product field to calculate the sum of the quantity for each unique product.

3. we can add sort to pipe line as well.

db.aggregateproduct.aggregate([

{ $match: { status: "A" } },

{ $group: { \_id: "$product", total: { $sum: "$quantity" } } },

{$sort : { total : 1 }

}

])

4. we can apply other aggregation properties along with match $project or $unset or $addFields or $set. lets see sum queries.

db.aggregateproduct.aggregate([

{ $match: { status: "A" } },

{ $addFields: {

maxWeight: { $max: "$size.weight" },

minSizeHeight: { $min: "$size.height" }

} }

])

db.aggregateproduct.aggregate([

{ $match: { status: "A" } },

{ $addFields: {

maxQ: { $max: "$quantity" },

minSizeHeight: { $min: "$size.height" }

} }

])

5.

db.aggregateproduct.aggregate([

{ $project : { product : 1,quantity : 1 } } , //optional line

{ $group: { \_id: "$product", total: { $sum: "$quantity" } } },

{$sort : { total : 1 }

}

])

firstdb> db.aggregateproduct.aggregate([

... { $project : { product : 1,quantity : 1 } } ,

... { $group: { \_id: "$product", total: { $sum: "$quantity" } } },

... {$sort : { total : 1 }

... }

... ])

[

{ \_id: 'register', total: 75 },

{ \_id: 'envelop', total: 90 },

{ \_id: 'book', total: 150 },

{ \_id: 'timetable', total: 150 },

{ \_id: 'sheet', total: 200 }

]

firstdb> db.aggregateproduct.aggregate([{ $group: { \_id: "$product", total: { $sum: "$quantity" } } }, { $sort: { total: 1 } }] )

[

{ \_id: 'register', total: 75 },

{ \_id: 'envelop', total: 90 },

{ \_id: 'book', total: 150 },

{ \_id: 'timetable', total: 150 },

{ \_id: 'sheet', total: 200 }

]

6. actually aggregation has so many stages that we can not go through all as you can see in below link.

https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe.\_S\_project

7. lets look at MapReduce , no it is not Hadoops map reduce :) but exactly same concept.

Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results.

For map-reduce operations, MongoDB provides the MapReduce database command.

db.collection.mapReduce(

mapfunction() # this give document with key value pairs

reducefunction() # reduces earlier key and value pair to single key and single value pair.

{

query # this is basically same query as we throw it in the find function.

output # end result the way we want to print

})

lets see an example

db.aggregateproduct.mapReduce(

function() {

emit(this.product,1);

},

function(key, values) {

return Array.sum(values);

},

{

query:{status:"A"},

out: { merge: "map\_reduce\_result" },

}

)

db.map\_reduce\_result.find();

--------------------------------------------------------

--ignore--

For Satyam, Subham mapReduce is working in MongoDB 6 version

For Durga, Nagender mapReduce is working in online compiler

DeprecationWarning: Collection.mapReduce() is deprecated. Use an aggregation instead.

See https://docs.mongodb.com/manual/core/map-reduce for details.

Note

Map-reduce is unsupported for MongoDB Atlas free clusters and MongoDB Atlas serverless instances.

------------------------------------------------------

8. The aggregation pipeline provides an alternative to map-reduce and may be the preferred solution for aggregation tasks where the complexity of map-reduce may be unwarranted.

9. Pipeline stages have a limit of 100 megabytes of RAM. If a stage exceeds this limit, MongoDB will produce an error.

To allow for the handling of large datasets, use the allowDiskUse option to enable aggregation pipeline stages to write data to temporary files.

10. single command aggration

db.aggregateproduct.count()

11.

db.aggregateproduct.estimatedDocumentCount()

12

db.aggregateproduct.distinct("status")

db.aggregateproduct.distinct("product")