Mongodb is sechamaless document database(json).

What us BSON?

“Binary Json ”

Bson is binary structure encodes type and length information which allows it to be traversed much mo re quickly compared to JSON .

Cmd

Run as administration

Net start mongoDB

--to open in cmd by using exe file

Mono shell download

Package msi

Mongodb.exe

Monodb://localhost:27017

Show dbs

1)To check the version of mogodb

🡪mongod --version

2)To in mongodb

🡪mongo

3)to show database

🡪show dbs

4)to use the database

🡪use databasename

5)to show collection

🡪show collections

6)to show the all record

🡪db.collection.find();

Db.collection.find().pretty();

7)for showing more row

-🡪 it

8)update one if not present then insert

db.students.updateOne({name:”ram”},{$set:{idcards:{hasidcard:false,hasadharcard:true}}})

modified count🡪1🡪 when modification take place

9) update many

Db.students.updateMany({},{$set:{hobbies:[“trekking”,”cooking”]}})

Mathchcount 🡪

Modifiedcount🡪

10) to find count

Db.students.find({hobbies:”cooking”}).count()

11)To find nested

Db.students.find({‘idcards.haspancard’:true})

Nested limit 16 mb

CRUD operation

1.Create

Insertone()

insertmany()

2.Read

find(filter,options)

findOne(filter ,options)

3.Update

updateOne(filter ,data ,options)

updateMany(filter ,data ,options)

replaceOne(filter ,data ,options)

4.delete

Deleteone(filter,options)

DeleteMany(filter,options)

FindOne Vs FindMany

Db.students,find();

Db.students.find({age:11})

findOne🡪

Db.students.findOne({age:11})

Find🡪we use foreach and limit return cursor

Db.students.find().foreach((x)=>{ printjson(x)})

Db.students.find().limit(2);

Db.students.find().toarray()

$ sign means reservered keyword for mongodb

Find student whose age >10

Db.students.find({age:{$gt:10}})

Find student count whose age =<10

Db.students.find({age:{$lte:10}}).count();

Find student whose age >5and less than 11

Db.students.find({age:{}$gt:5, $lt:11})

InertOne and Insertmany

Inert one record

db.students.insertOne({name:”sarthak”,age:12})

Inert many record

db.students.insertMany([{name:”ak”,age:23},{name:”pk” ,age:34},{name:”sk”,age:6}])

UpdateOne Vs Updatemany

Update one student whose age = 15

Db.students.updateOne({name:”ek”},{$set:{age:15}});

Update all students whose age is 12

Db.students.updateMany({age:12},{$set:{age:15}})

Update collections students iseligible set false if not presnt then insert whose age 13

Db.students.updatemany({age:13},{$set:{isEligible:false}})

Update collections students iseligible set false if not presnt then insert whose age>=14

Db.students.updatemany({age:{$gte:14}},{$set:{iseligible:false}})

DeleteOne and DeleteMany

Delete student whose age 13

Db.student.deletemany({age:13})

Delete student whose name is vk

Db.students.deletemany({name:”vk”})

Delete all student from collections students

Db.student.deletemany({})

Db.student.deletemany()//error

Display only name of all students

Db.students.find({},{\_id:0,name:1})//by default \_id is 1

DataType in Monodb

Text

Boolean

Number ->Integer 🡪32

NumberLong 🡪64

NumberDecimal🡪

objectId

ISODate

TimeStamp

Array

Emb.document

Insert record company collections

db.company.insert({ name:’newton’,isfunded:true,funding:12345678901234567890,employee:[{name:’atish’,age:12},{name:’pk’,age:43}],foundeon:new Date(),foundeonTimeStamp:new Timestamp()})

How to delete databases and collection in mongodb

To delete all collection from particular database

db.dropDatabase()

to delete collection product

db.product.drop()

insert multiple value in collections

db.books.insertMany([{name:”pk”,price:23},{name:”ek”,price:77}])

IMP

db.books.insertMany([{\_id:”a”,name:”pk”,price:23},{\_id:”b”,name:”ek”,price:77}])

db.books.insertMany([{\_id:”c”,name:”pk”,price:23},{\_id:”a”,name:”ek”,price:77},},{\_id:”d”,name:”sk”,price:77}])

// this query give the error at time inserting \_id=a because the duplicate therefore insert \_id=c but not insert \_id =d but when ordered is false then insert both record and give error query is given below

db.books.insertMany([{\_id:”c”,name:”pk”,price:23},{\_id:”a”,name:”ek”,price:77},},{\_id:”d”,name:”sk”,price:77}],{ordered:false})

Validation in MongoDb

db.createCollection(“nonfiction”)

db.createCollection(

“nonfiction”,{

validator:{

$jsonSchema:{

required:[‘name’,’price’],

properties:{

name:{

bsonType:’string’,

description:’must be a string and required’

},

price:{

bsonType:’number’,

description:’must be a number and required’

}

}

}

},

validationAction:’error’

})

db.nonfiction.insertOne({name:”way of men”}) //give the error price field required

db.nonfiction.insertOne({name:"way of men",price:”100.35”})//error because datatype of price not number

db.nonfiction.insertOne({name:"way of men",price:100.35,author:"abc"})//insert without error

Write Concern 🡪to do operation fast

w🡪acknowledgment

db.books.insertOne({name:”ek”,price:1})//excuate slow as compare to below query because wait for ackowledgment

db.books.insertOne({name:”ek”,price:1},{witeConcern:{w:0}})//execuate fast

db.books.insertOne({name:”ek”,price:1},{witeConcern:{w:0,j:true}})//journal

db.books.insertOne({name:”ek”,price:1},{witeConcern:{w:0,j:true,wtimeout:100}})//timeout

Acid

Atomicity: either whole neither nothing

In mongodb atrocity occure in document level

MongoImport

Mongoimport “file path” -d college -c students --jsonArray –drop

Comparison operator($eq,$ne,$lt,$gt,$lte,$gte,$in and $nin)

db.students.find({age:5} )

//or

db.students.find({age:{$eq:5}} )

//in this way $eq,$ne,$lt,$gt,$lte,$gte

$in

Find students age 5 and 12

db.students.find({age:{$iq:[5,12]}} )

Logical Operator ( $and , $or $nor, $not)

Find the student whose age <=10 and age>=12

db.students.find({$or:[{age:{$lte:10}},{age:{$gte:12}}]})

Find the student whose not age <=10 and age>=12 means(11)

db.students.find({$nor:[{age:{$lte:10}},{age:{$gte:12}}]})

$and

Students of age <=11 and hobbies=walking

db.students.find({$and:[{age:{$lte:11}},{hobbies:’walking’}]})

db.students.find({age:{$lte:11},hobbies:’walk’})

//above both query give same output

But

We find student bet

db.students.find({age:{$lt:11},age:{$gt:5}})//int his case only last filter is fallowed age >5

db.students.find({$and:[{age:{$lt:11}},{age:{$gt:5}}])

Element query Operators($exist,$type)

db.students.find({ hasbook:{$exists:true,$eq:true}})

db.students.find({ hasbook:{$exists:true}})

db.students.find({ hasbook:{$exists:true,$type:B}})

db.students.find({ hasbook:{$exists:true,$type:’bool’}})

db.students.find({ hasbook:{$type:’bool’}})

Evaluation Operators

$expr🡪Allows use of aggration expression with in the query lanuaage

db.collection.find(

$expr:{

$gt:[“$field1”,”$Field2”]

})

db.collection.find(

{

$expr:{$gt:[“$price”,{“$avg”:”$price”}]}

}

)

$regexp 🡪 regural expression

$text🡪

db.students.createIndex({bio:”text”})

db.product.find({

$text:{

$search:”red shirt”

}

})

$mod🡪

db.collection.find({

quantity:{

$mod:[3,0]

}

})

size()🡪count

experince is gt than 3 company

db.students.find({$size:{$gte:3}})

db.students.find({$and:[{$experience:{$exists:true}},{$exp:{$gte:[{$size:”$experience”},3]}}]})

$All

db.students.find({Hobbies:{$in:[“walking”,”Reading”]}})

Sorting

db.teacher.find().sort({age:1})//ascending

db.teacher.find().sort({age:-1})//desc

db.teacher.find().sort({age:1,exp:1})//first age asc and exp desc

db.teacher.find().sort({age:1,exp:1}).skip(20)//skip first 20

$inc ,$min ,$max ,$unset, $rename upsert

db.student.updatemany({},{$inc:{$age:1}})//increament by 1

db.student.updatemany({},{$inc:{$age:-1}})//decrement by 1

$min// decrese the existing value

db.students.updatemany({name:”sita”},{$min:{age:50}})

//if sita age >50 then update and set 50 but is age <50 then no change

$max//increase the exists values

db.students.updatemany({name:”sita”},{$max:{age:50}})

//if sita age < 50 then update and set 50 but is age >50 then no change

$mul

db.students.updateMany({name:”sita”},{$mul:{age:2}}) //age multiply by 2

$unset

db.students.updatemany({name:”sita”},{$unset:{age:12333}})//remove age entity

$rename

db.students.updateOne({name:”sita”},{$rename:{age:”sage”}})//rename age to sage

$upsert

db.students.updateMany({name:”golu”},{$set:{age:100}},{upsert:true})//if matching record not found then insert

Arrays update

$push , $pull,$pop,$addToSet