Cls/ Exit- ctrl C

show dbs

use shop

db.products.insertOne({ :"",  :""})

*no white spaces in key. Else use””*

db.products.insertOne({"":"", "" :""})

*key-value are case sensitive*

db.products.insertOne({ :"",  :"",\_id:""})

*same-id = duplicate key error*

db.products.find()

db.products.find().pretty()

>db.flightData.updateOne({ distance:12000},

{ $set:{ marker:"delete" } })

>db.flightData.updateMany({},

{ $set:{ marker:"toDelete" }  })

db.flightData.deleteOne({airport:"LAX"})

db.flightData.deleteMany({})

db.flightData.deleteMany({marker:"toDelete"})

db.flightData.find({intercontinental: true})

db.flightData.find({distance: {$gt : 10000}})

>db.flightdata.updateOne(

{\_id :ObjectId("5dc252f246f5c6b3f4011d0c")},

{$set : {delayed : true}})

>db.flightdata.update(

{\_id :ObjectId("5dc252f246f5c6b3f4011d0c")},

{$set : {delayed : false}})

>db.flightdata.update({\_id :ObjectId("5dc252f246f5c6b3f4011d0c")},

{delayed : false})*--replace*

db.flightdata.replaceOne({\_id: ObjectId("5dc252f246f5c6b3f4011d0c")},

{"departureAirport": LHR",

"arrivalAirport": "TXL",

"aircraft": "Airbus A320",

"distance": 950,

"intercontinental": false})

*-- replace all the document with new one when use $set*

db.passengers.insertMany

([{"name": "Max Schwarz", "age": 29},

{“name": "Manu Lorenz",” age": 30},

{“name": "Chris Hayton",” age": 35},

{“name": "Sandeep Kumar",” age": 28},

{“name": "Maria Jones",” age": 30},

{“name": "Alexandra Maier",” age": 27},

{“name": "Dr. Phil Evans",” age": 47},

{“name": "Sandra Brugge",” age": 33},

{“name": "Scott Tolib",” age": 44},

{“name": "Klaus Arber",” age": 53},

{“name": "Albert Twostone",” age": 68},

{“name": "Gordon Black",” age": 38}])

>it

>db.passengers.find().toArray()

*-- display all data in form of array*

db.passengers.find().

forEach((passengerData) => {printjson(passengerData)})

*fetch all document - one at a time. Execute function - pass document into function*

Find() *gives you cursor*

.pretty()*- works on cursor*

findOne() *does not provide cursor*

*insert/update/delete cursors do not exist,as they do not fetch data. They manipulate.*

**Projection**

*2/10 column display, avoid wastage bandwidth*

db.passengers.find({},{name:1}).pretty()

db.passengers.find({},{name:1, \_id:0})

**Array of embedded doc-**

db.flightdata.insertMany (

[

{

"departureAirport":"MUC",

"arrivalAirport":"SFO",

"aircraft":"Airbus A380",

"distance":12000,

"intercontinental":true

},

{

"departureAirport":"LHR",

"arrivalAirport":"TXL",

"aircraft":"Airbus A320",

"distance":950,

"intercontinental":false

}

]

)

db.flightdata.updateMany( { },

{ $set:{

status:{

description:"on-time",

lastUpdated:"1 hour ago"

}

}

} )

db.flightdata.updateMany( { },

{ $set:{

status:{

description:"on-time",

lastUpdated:"1 hour ago",

details:{

responsible:"User1"

}

}

}

} )*-- multilevel document*

db.passengers.updateOne( { name:"Albert" },

{ $set:{

hobbies:[

"sports",

"cooking"

]

}

} ) *--Array*

>db.passengers.findOne

({name:"AlbertTwostone"}).hobbies

>db.passengers.findOne({ hobbies:"sports" })

db.flightdata.find

({ "status.description":"on-time"} ).pretty()

db.flightdata.find

({ "status.details.responsible":"User 1" })

**Collections**

Use hospitalData

show collections

db.createCollection('Patient')

db.Patient.insertOne(    
   {    
      firstName:'John',  
      lastname:'Muller',  
      age:35,  
      history:[    
         {    
            disease:'cold',  
            treatment:'medication'  
         },  
         {    
            disease:'migraine',  
            treatment:'cold water'  
         }  
      ]  
   }  
)

db.Patient.insertOne(    
   {    
      firstName:'Sam',  
      lasname:'Johnson',  
      age:35,  
      history:{    
         disease:'fever',  
         treatment:'medication'  
      }  
   }

)   
db.Patient.insertOne(   {    
      firstName:'Demon',  
      lasname:'Salvatore',  
      age:25,  
      history:{    
         disease: ‘vampire',  
         treatment: ‘pure blood'  
      }  
   }  
)

db.Patient.updateOne(    
 {    
   "\_id":ObjectId("5dc79969f3a4a4d6be0bf21d")  
 },  
   {    
      $set:{    
         history:{    
            disease:'e.coli',  
            treatment:'medicines'  
         }  
      }  
   }

)

db.Patient.updateOne({    
 "\_id":ObjectId("5dc79969f3a4a4d6be0bf21d")}, { $set:{  lastname:'Silva' } }   ) 

db.Patient.updateOne(

{"\_id":ObjectId("5dc79969f3a4a4d6be0bf21d")},  
   {$set:{age:48}})

db.Patient.find({age:{$gt:30}})

db.Patient.find({'history.disease':'cold'})

db.Patient.deleteMany

({'history.disease':'cold'})

db.Patient.updateOne(

{"\_id":ObjectId("5dc79969f3a4a4d")},

{$set:{firstName:'Ram', lastname:'Silva', age:48, history:[{disease:'e.coli', treatment:'medicines'}]}})*–update multiple*

db.product.find()

db.product.drop()

**Data storage**

*-- NON structured - storing completely different data-structure is possible*

db.products.insertOne({title:"T shirt", seller:{name:"Max", age:29}})

db.products.insertOne({name:"A Computer", price:1299, details:{CPU:"Intel i7 8770"}})

db.products.insertOne({name:"T shirt", price:12.99, details: null}) *--null SQL appr*

**DataType**

*text -16mb whole document*

*number - int 32, int64, numberDecimal,*

*object id –*

*Date -ISODate, Timestamp*

*Object - Embedded document -100level,16mb*

*Array*

use companyData

db.companies.insertOne(

{

name:'fresh Apple Inc.',

isStartup:true,

employees:33,

funding:12345678901234567890,

details:{

ceo:'Mark Super'

},

tags:[

{

title:'super'

},

{

title:'perfect'

}

],

fundingDate:new Date(),

insertedAt:new Timestamp()

}

)

db.numbers.insertOne({a:1}) *--size 33*

db.numbers.insertOne({a:NumberInt(1)})*-- 29*

db.stats()*-- db sizes notes*

**Data Schemas & Data Modeling**

*which data,*

*where do I need data,*

*which kind of data,*

*how often fetch,*

*how often write/change data*

**Embedded/ Nested -** Patient - Summary A

use hospital

db.patient.insertOne({name:'Max', age:29, diseases:[ 'cold','broken leg']})

db.patients.insertOne({name:'Max', age:29, diseaseSummary:'summary-max-1'})

db.diseaseSummaries.inserOne({\_id:'summary-max-1', diseases: ['cold','broken leg']})

db.patients.findOne().diseaseSummary

var dsid = db.patient.findOne().diseaseSummary

db.diseaseSummaries.findOne({\_id:dsid})

**Relation**

**one to one -embedded--**Person -Car

use carData

db.persons.insertOne({name:'Max', Car:{model: 'BMW', Price:40000}})

db.persons.insertOne({name:'Max', age:29, salary:3000})

db.cars.insertOne({model:'BMW', price:40000, owner:"Max"})

db.cars.insertOne({model:'BMW',Price:40000, owner:ObjectId("5dc89bab9e8089b5f0902613")})

**One to Many -**1Question - 2 Answer

use support

>db.questionThread.insertOne

({creator:"Max", question:"Howdoes it work?", answer:[{text:"like that"},{ text:"Thanks!"}]})

>db.questionThread.insertOne

({creator:"Max", question:"How does it work?", answer:["q1a1", "q1a2"]})

>db.answers.insertMany([{\_id:"q1a1",text:"like that"},{\_id:"q1a2",text:"Thanks!"}])

**Many to many-**

City - citizen

city A - citizen1,citizen2

city B - citizen1

use cityData

>db.cities.insertOne({name:'NY', coordinates:{lat:21, lng:55}})

>db.citizen.insertOne({name:"Max", cityId:ObjectId("5dc8a06f9e8089b5f0902618")})

db.citizen.insertMany([{name:"John", cityId:ObjectId("5dc8a06f9e8089b5f0902618")},

{name:"Manuel", cityId:ObjectId("5dc8a06f9e8089b5f0902618")}])

**Many to Many**

Customer – product

>db.products.insertOne({name:"T shirt", price:12.99})

>db.customers.insertOne({name: "Max", age:18})

>db.orders.insertOne({productId:ObjectId("5dc0ecc1f0e64f0fec9fcacc"),customerId:ObjectId("5dc8abb39e8089b5f090261c")})*--SQL Approach*

>db.customers.updateOne({},

{$set:{orders:[{productId: ObjectId("5dc0ecc1f0e64f0fec9fcacc"), quantity:2}]}})

>db.customers.updateOne({},{$set:{orders:[{ name:"A Computer", price:1299, quantity:2}]}})

*--data duplication, data update*

**Many to Many with reference**

Books -Authors

use bookRegistry

db.books.insertOne({name:"My book" , authors: [{name: 'Max', age: 29},{name:'John', age: 35}]})

db.authors.insertMany([{name:"Max", age:29, address:{street: 'main'}},{name:'John', age: 35, address: {street: 'tree'}}])

db.books.updateOne({},{$set:{authors:[ ObjectId("5dc8b4e09e8089b5f090261f"), ObjectId("5dc8b4e09e8089b5f0902620")]}})

db.books.update ({"\_id" : ObjectId("5dc8b2609e8089b5f090261e")},

{

"\_id": ObjectId("5dc8b2609e8089b5f090261e"),

"name": "My book",

"authors": [ObjectId("5dc8b4e09e8089b5f090261f"),ObjectId("5dc8b4e09e8089b5f0902620")]

})

**fetch split data with aggregate and lookup**

>db.books.aggregate

([{$lookup:{from:"authors",localField:"authors",foreignField:"\_id",as:"creators"}}]).pretty()

*--double quotes, camel case*

**Example:**

> use blog

>db.user.insertMany([{name:'Max', age: 28, email:'max@test.com'},

{name:'Jon', age:40, email:'jon@test.com'}])

db.post.insertOne(    
   {    
      title:'First Post',  
      text:'this is first post',  
      tags:[    
         'new',  
         'tech',  
         '2019'  
      ],  
      creator:ObjectId("5dca0cbca5a65d037a23b363"),  
      comments:[    
         {    
            text:'like the post',  
            author:ObjectId("5dca0cbca5a65d037a23b364")  
         },  
         {    
            text:'thank you',  
            author:ObjectId("5dca0cbca5a65d037a23b363")  
         }  
      ]  
   }  
)

**Schema Validation**

Maintain Schema

-insert into collection

- validation schema

- accept/reject

validation level

- which document gets validated

- strict(ins/updt), moderate(ins)

validation action

- error(deny),

- warn(warning and proceed)

db.createCollection(   
"posts",  
{   
 validator:{   
 $jsonSchema:{   
bsonType:"object",  
 required:[   
 "title",  
 "text",  
 "creator",  
 "comments"  
 ],  
 properties:{   
 title:{   
bsonType:"string",  
 description:"string and is required"  
 },  
 text:{   
bsonType:"string",  
 description:"string and is required"  
 },  
 creator:{   
bsonType:"objectId",  
 description:"objectId and is required"  
 },  
 comments:{   
bsonType:"array",  
 description:"array and is required",  
 items:{   
bsonType:"object",  
 required:[   
 "text",  
 "author"  
 ],  
 properties:{   
 text:{   
bsonType:"string",  
 description:"string and is required"  
 },  
 author:{   
bsonType:"objectId",  
 description:"objectId and is required"  
 }  
 }  
 }  
 }  
 }  
 }  
 }  
}  
)

>db.runCommand(

{collMod: 'posts',

validator:{...}, validationAction:'warn'})

*-- default – error*

*--warning on the log file and insert,*

db.posts.insertOne(    
 {    
 title:'First Post',  
 text:'this is first post',  
 tags:['new','tech'],  
creator:ObjectId("5dca0cbca5a65d037a23b363"),  
 comments:[    
 {    
 text:'like the post', author:ObjectId("5dca0cbca5a65d037a23b364")  
 }  
 ]  
 }  
)

*--error*

db.posts.insertOne(    
 {    
 title:'First Post',  
 text:'this is first post',  
 tags:['new','tech'],  
creator:ObjectId("5dca0cbca5a65d037a23b363"),  
 comments:[    
 {    
 text:'like the post',  
 author:123  
 }  
 ]  
 }  
)

**Mongodb options - configuration**

*mongo - shell*

*mongod– Server*

Sever help

mongod --port2708 *set port 27017 default*

mongod –help

*mongodb/bin/ . create mongodb/db*

*mongodb/bin/ . create mongodb/log*

mongod –dbpath path/db*--dbpatharg*

mongod –dbpath path/log*--logpatharg*

mongod –dbpath path/db –logpath path/log

*all output gets written in this location*

--directoryperdb

--fork -logpath *mac,linux -as child process*

windows - cmd as an admin

>net start MongoDB

>net stop MongoDB

>db.shutdownserver()

**Using config file**

Mongodb.cfg*->visual studio ->example google*

*Shell help*

Mongo --help

--nodb*shell based on java*

--quiet

--port arg *27017 default*

--host arg *set server*

> help

use test

db.help()

db.collection.help()

**MongoDB Compass GUI**

**CREATE**

use contactData

show collections

db.persons.insertOne({name: 'Max', age: 30 , hobbies:['sports','cooking']})

db.persons.insertMany([{name: 'Anna', age: 29 , hobbies:['Sports','yoga']},

{name:'Maria', age: 25}, {name: 'Chris', age :28}])

db.persons.insert({name: 'Phil', age: 35}) *--does not show id immediately*

db.persons.insert([{name: 'Andy', age: 29 },{name:'Hans', age: 25}])

**un-ordered insert**

db.hobbies.insertMany([{\_id:"sports", name:'sports'},

{\_id:"cooking", name: 'cooking'},

{\_id:'cars', name:'cars'}])

db.hobbies.insertMany([{\_id:"yoga", name:'yoga'},

{\_id:"cooking", name: 'cooking'},

{\_id:'hiking', name:'hiking'}]) *--ordered insert default*

db.hobbies.insertMany([{\_id:"yoga", name:'yoga'},

{\_id:"cooking", name: 'cooking'},

{\_id:'hiking', name:'hiking'}]**,{ordered: false}**) *-- cannot rollback*

**writeConcern**

>db.persons.insertOne({name:'Kristy' , age:41}, {writeConcern:{w: 0}})

*{"acknowledged"`: false} -- data does not end up database*

>db.persons.insertOne({name:'Kristy' , age:41}, {writeConcern:{w: 1}}) -default

db.persons.insertOne({name:'Mike' , age:51}, {writeConcern:{w: 1, j: false}}) *--default*

db.persons.insertOne({name:'Alia' , age:51}, {writeConcern:{w: 1, j: true}}) *--higher security*

db.persons.insertOne({name:'Mia' , age:51}, {writeConcern:{w: 1, j: false, wtimeout:200}})

***Atomicity***

*operation - error - Rolled back*

*-success - Saved whole*

Example:

use companyData

db.companies.insertOne({\_id:'Google', name:'Google', Stock: 250.5})

db.companies.insertMany({\_id:'Apple', name:'Apple', Stock: 350.5},

{\_id:'Amazon', name:'Amazon', Stock:300})

db.companies.insertMany([{\_id:'Apple', name:'Apple', Stock: 350.5},

{\_id:'TeraData', name:'teraData', Stock:150}])

db.companies.insertMany([{\_id:'Apple', name:'Apple', Stock: 350.5},

{\_id:'TeraData', name:'teraData', Stock:150}],{ordered: false})

db.companies.insertMany

([{\_id:'Microsoft', name:'Microsoft', Stock:350}],{writeConcern:{w:1 , j:true}})

**mongoImport**

*cmd prompt - store json file C:\Uday\mongoDB*

mongoimport tv-shows.json -d movieData -c movies --jsonArray --drop

*file db coll Array Drop if existing database*

>mongo *--to access the data*

**Reading Data**

*db.collection.method(document-key:value)*

use movieData

db.movies.findOne({})

db.movies.findOne()

db.movies.find({})

db.movies.find({name:'The Last Ship'})

db.movies.find({runtime:60})

**operators**

**Comparison operator**

>db.movies.findOne({runtime: {$eq:60}}) *---equals db.movies.find({runtime:60})*

>db.movies.findOne({runtime: {$ne:60}})

>db.movies.findOne({runtime: {$lt:40}})

>db.movies.findOne({runtime: {$lte:40}})

>db.movies.findOne({runtime: {$gt:60}})

>db.movies.findOne({runtime: {$gte:60}})

db.movies.findOne({"rating.average": 7.5})

db.movies.findOne({"rating.average":{$gt: 7.5}})

db.movies.findOne({genres:"Drama"})

db.movies.findOne({genres:["Drama"]}) *--exactly this array*

>db.movies.findOne({runtime : {$in:[30,42]}})

>db.movies.findOne({runtime : {$nin:[30,42]}})

**Logical Operators**

db.movies.find({'rating.average' : {$lt:5}}).count()

>db.movies.find({'rating.average' : {$gt:9.3}}).count()

db.movies.find({$or:[{'rating.average':{$lt:5}},{'rating.average':{$gt:9.3}} ]}).pretty() *-- 4*

db.movies.find({$nor:[{'rating.average':{$lt:5}},{'rating.average':{$gt:9.3}} ]}).pretty() *--236*

>db.movies.find({$and: [{'rating.average':{$gt:9}}, {genres:'Drama'}]}).count()

>db.movies.find({'rating.average':{$gt:9}, genres:'Drama'}).count() *by default ands everything*

>db.movies.find({genres:'Drama', genres:'Horror'}).count() *--23*

>db.movies.find({$and:[{genres:'Horror'}]}).count() *--23 as 2nd query over rights the first*

>db.movies.find({$and:[{genres:'Drama'},{genres:'Horror'}]}).count() *--17 -- Drama and Horror*

>db.movies.find({runtime: {$not: {$eq: 60}}}).count()

>db.movies.find({runtime: {$ne: 60}}).count() *--same*

**Element Operator**

Use users

db.users.insertMany(    
   [ {  name:'Max',  
         hobbies:[    
            {    
               title:'Sports',  
               frequency:3  
            },  
            {    
               title:'cooking',  
               frequency:6  
            }  
         ],  
         phone:013125229169  
      },  
      {  name:'John',  
         hobbies:[    
            {    
               title:'Cars',  
               frequency:2  
            },  
            {    
               title:'cooking',  
               frequency:5  
            }  
         ],  
         phone:"013153458765",*--phone number different fmt*         age:30  
      }  
   ]  
)

>db.users.find({age:{$exists:true}}).pretty()

>db.users.find({age:{$exists:true, $gt: 20}}).pretty()

>db.users.insertOne(    
   {    
      name:'Anna',  
      hobbies:[    
         {    
            title:'yoga',  
            frequencey:3  
         },  
         {    
            title:'sport',  
            frequency:4  
         }  
      ],  
      phone:014235437564,  
      age:null  
   }  
)

>db.users.find({age:{$exists:true}}).pretty()*–- pulls the null record*

>db.users.find({age:{$exists:false}}).pretty()

>db.users.find({age:{$exists:true, $ne: null}}).pretty()

$type:

>db.users.find({phone: {$type: "number"}})

>db.users.find({phone: {$type: "double"}})*--int is double too*

>db.users.find({phone: {$type:["number","string"]}})

**Evaluation Operator**

$regex: *-- helps in search text*

>db.movies.find({summary: "musical"}).count()

>db.movies.find({summary: {$regex: /musical/}}).count()

$expr: *-- compare 2 fields and return result*

use financialData

>db.sales.insertMany([{volume: 100, target:120},

{volume:89, target:80},

{volume:200, target:177}])

db.sales.find({$expr:{$gt: ["$volume", "$target"]}}).pretty()*---where volume > target*

*--volume is above 190, difference to target must be at least 10*

>db.sales.find(    
   {    
      $expr:{    
         $gt:[    
            {    
               $cond:{   if:{   $gte:[   "$volume",190] },  
                   then:{  $subtract:[  "$Volume", 10 ] },  
                   else:"$volume"  
                }  
            },  
            "$target"  
         ]  
      }  
   }  
).pretty()

Example:

mongoimportboxoffice.json -d boxofficeData -c boxoffice --jsonArray --drop

mongo

use boxofficeData

db.boxoffice.find()

db.boxoffice.find({"meta.rating":{$gt:9.2}, "meta.runtime":{$lt:100}})

>db.boxoffice.find({$or:[{genre:"drama"},{genre:"action"}]})

>db.boxoffice.find({$expr:{$gt:["$visitors","$expectedVisitors"]}})

**Querying Array**

use user

>db.users.find({hobbies:"Sports"}) *--doesnt work*

>db.users.find({hobbies:{title:"Sports"}}) *--doesnt work*

>db.users.find({hobbies:{title:"Sports", frequency : 3}}) *---works but frequency attached*

**>db.users.find({"hobbies.title":"Sports"})**

**Array Query Selectors**

$size

>db.user.insertOne({name:"Teresa", hobbies: ["Sports", "Cooking", "Hiking"]})

>db.users.find({hobbies:{$size: 3}}) --looks for exact match, not gt, lt

$all

>db.boxoffice.find({genre:["action","thriller"]})*---order matters*

>db.boxoffice.find({genre:{$all: ["action","thriller"]}}) *---does not care the order of genre*

*--both action or thriller*

$elemMatch

>db.users.find({$and:[{"hobbies.title":"Sports"},{"hobbies.frequency":{$gte:3}}]})

*--pulls throughout the documents both frequency:2 and title: somewhere*

>db.users.find({hobbies:{$elemMatch:{title: "Sports", frequency: {$gte: 3}}}})

Example:

mongoimportboxoffice-extended.json -d exmovieData -c exmoviestarts --drop --jsonArray

>db.exmoviestarts.find({genre:{$size:2}})

>db.exmoviestarts.find({"meta.aired":2018})

>db.exmoviestarts.find({ratings:{$elemMatch: {$gt:8, $lt:10}}})*--at least one rating gt 8, lt 10*

**Cursors**

>it

*Find() returns a cursor*

>db.movies.find().**next()***--1 document at a time*

> const dataCursor = db.movies.find()

>dataCursor.next()

>dataCursor.forEach(doc => {printjson(doc)})

*input to function* *print method*

>dataCursor.**hasNext()**

sorting cursor results

>db.movies.find().**sort({"rating.average":-1})**.pretty() *desc*

>db.movies.find().**sort({"rating.average":1})**.pretty() *asc*

>db.movies.find().sort({"rating.average":-1, runtime:1}).pretty()*projection runtime*

skipping, limiting cursor results

*Application 100 result per page 10, if user moves to page 2 skip first 10 results*

>db.movies.find().sort({"rating.average":-1, runtime:1}).**skip(10).**pretty()

*limit - limit the number of elements, cursor should fetch at a time*

>db.movies.find().sort({"rating.average":-1, runtime:1}).**skip(100).limit(10)**.pretty()

*--order matters*

**Using projections to shape our results**

>db.movies.find({}, {name:1, genre:1, runtime:1, rating:1}).pretty()*–-id by default*

>db.movies.find({}, {name:1, genre:1, runtime:1, rating:1, \_id:0}).pretty()

>db.movies.find({}, {name:1, genre:1, runtime:1, "schedule.time":1}).pretty()

>db.movies.find({genres:"Drama"}, {"genres.$":1}).pretty()

{"\_id": ObjectId("5dcb87da1b171961bf07b158"), "genres": [ "Drama"]}

*give 1 genre you found. there might be other genres but fetch just 1*

>db.movies.find({genres:{$all: ["Horror", "Drama"]}}, {"genres":1})

{"\_id": ObjectId("5dcb87da1b171961bf07b160"), "genres": [ "Drama", "Horror", "Romance"]}

{"\_id": ObjectId("5dcb87da1b171961bf07b162"), "genres": [ "Drama", "Fantasy", "Horror"]}

>db.movies.find({genres:{$all: ["Drama", "Horror"]}}, {"genres.$":1}).pretty()

{"\_id": ObjectId("5dcb87da1b171961bf07b160"), "genres": [ "Horror"]}

>db.movies.find({genres:{$all: ["Horror", "Drama"]}}, {"genres.$":1}).pretty()

{"\_id": ObjectId("5dcb87da1b171961bf07b160"), "genres": [ "Drama"]}

>db.movies.find({genres: "Drama"}, {"genres.$":1}).pretty()

{"\_id": ObjectId("5dcb87da1b171961bf07b158"), "genres": [ "Drama"]}

>db.movies.find({genres: "Drama"}, {genres: {$elemMatch: {$eq: "Horror"}}}).pretty()

{"\_id": ObjectId("5dcb87da1b171961bf07b160"), "genres": [ "Horror"]}

{"\_id": ObjectId("5dcb87da1b171961bf07b161" }

>db.movies.find({"rating.average": {$gt: 9}}, {genres: {$elemMatch: {$eq: "Horror"}}}).pretty()

--filter and projection are totally different

{"\_id": ObjectId("5dcb87da1b171961bf07b16f"), "genres": [ "Horror"]}

{"\_id": ObjectId("5dcb87da1b171961bf07b1a5")}

**slice operator**

>db.movies.find({"rating.average": {$gt: 9}}, {genres: {$slice: 2}, name:1}).pretty()

*shows first 2 genres*

{

"\_id”: ObjectId("5dcb87da1b171961bf07b211"),

"name”: "Firefly",

"genres”: [

"Adventure",

"Science-Fiction"

]

}

{

"\_id":ObjectId("5dcb87da1b171961bf07b218"),

"name": "Stargate SG-1",

"genres": [

"Action",

"Adventure"

]

}

>db.movies.find({"rating.average": {$gt: 9}}, {genres: {$slice: [1, 2]}, name:1}).pretty()

*skip 1 limit item 2*

{

"\_id":ObjectId("5dcb87da1b171961bf07b16f"),

"name": "Berserk",

"genres": [

"Fantasy",

"Horror"

]

}

**Update**

db.users.updateOne(    
   {    
      \_id:ObjectId("5dccb6f6829ea7433d1fb536")  
   },  
   {$set:{hobbies:[    
            {title:"Sports",frequency:5},  
            { title:"cooking",frequency:3},  
            { title:"Hiking",frequency:1}  
         ]}})

db.users.updateMany({"hobbies.title":"Sports"},{$set:{isSporty:true}})

db.users.updateOne({"\_id" : ObjectId("5dcb9b71cdd93aba40f2ec84")},

{$set:{age:40, phone:4089764567}})

>db.users.updateOne({"name" : "Max"},{**$inc**:{age:1}})*---age:-1, age:2*

>db.users.updateOne({"name" : "Max"},{$inc:{age:1}, $set:{isSporty:false}})

>db.users.updateOne({"name" : "Max"},{$inc:{age:1}, $set:{age:30}})*--error same field*

db.users.updateOne({"name" : "Max"},{**$min**:{age:35}})*current age is 40*

db.users.updateOne({"name" : "Max"},{$min:{age:38}})*--doesn’t work if value is higher than value*

db.users.updateOne({"name" : "Max"},{**$max**:{age:34}})

db.users.updateOne({"name" : "Max"},{**$mul**:{age:1.1}})*-- multiply current age*

$unset

>db.users.updateMany({"isSporty" : true},{$set:{phone:null}})

>db.users.updateMany({"isSporty" : true},{**$unset**:{phone:""}})

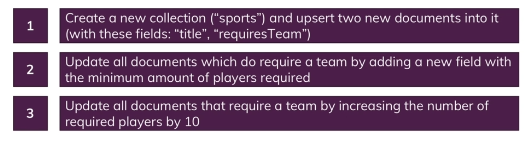
$rename

>db.users.updateMany({},{**$rename**:{age:"totalAge"}})*–-rename key values*

$upsert*--update or insert--overright or create*

>db.users.updateOne(  { "name":"Maria"},  
   {  $set:{age:29,hobbies:[ { title:"Good food",frequency:3}],isSporty:true}},**{upsert:true}**)

Example



>db.sports.updateOne({title:"softball"}, {$set:{requiresTeam:true}},{upsert:true})

>db.sports.updateOne({title:"tennis"}, {$set:{requiresTeam:false}},{upsert:true})

>db.sports.updateMany({requiresTeam:true},{$set:{minPlayers:9}})

>db.sports.updateMany({requiresTeam:true},{$inc:{minPlayers:10}})

>db.sports.updateMany({},{$set:{title:"softball", requiresTeam:true}},{upsert:true})

>db.sports.updateMany({title:"tennis"}, {$set:{requiresTeam:false}},{upsert:true})

**Arrays update**

>db.users.find({$and:[{"hobbies.title":"Sports"},{"hobbies.frequency":{$gte:3}}]})

>db.users.find({hobbies:{$elemMatch:{"title":"Sports", frequency:{$gte:3}}}})

>db.users.updateMany({hobbies:{**$elemMatch**:{title:"Sports", frequency:{$gte:3}}}},{$set:{title:"Sports", frequency:5, "hobbies.**$**.highFrequency":true}})

*--$ to find exact element found in query*

All array elements

>db.users.find({"hobbies.frequency":{$gte:2}})

>db.users.updateMany({"hobbies.frequency":{$gte:2}},{$set:{"hobbies.$.goodFrequency": true}})

*$ - only first element matching gets updated inside array*

>db.users.updateMany({totalAge:{$gt:20}},{$inc: {"hobbies.frequency":-1}})*--error*

>db.users.updateMany({totalAge:{$gt:20}},{$inc: {"hobbies.**$[]**.frequency":-1}})

*--$[] update all element. for each element -> frequency field.*

*--if no embedded document -stop after $[].*

Finding and Updating Specific Fields

*all hobbies where freq> 2*

>db.users.find({"hobbies.frequency":{$gt:2}})

*--one fulfilling document still pull entire document*

>db.users.updateMany({"hobbies.frequency":{$gt:2}},{$set:{"hobbies**.$[el].**goodFrequency":true}},

**{arrayFilters: [{"el.frequency":{$gt:2}}]}**)

Adding Elementto Array

db.users.updateOne({name:"Maria"},{**$push**: {hobbies:{title:"Sports", frequency:2}}})

db.users.updateOne({name:"Maria"},

{**$push**: {hobbies: {**$each**: [{title:"Good Wine", frequency:1},{title:"hiking", frequency:2}],

**$sort**:{frequency:-1}, **$slice**: 1}}})

*--sort existing hobbies too*

*--slice: add only one element*

Removing Element from Array

>db.users.updateOne({name:"Maria"}, {**$pull**: {hobbies: {title:"hiking"}}})

>db.users.updateOne({name:"Chris"}, {**$pop**: {hobbies: 1}}})*--1 removes last item, -1 first item*

$addToSet

db.users.updateOne({name:"Maria"},{**$push**: {hobbies:{title:"Hiking", frequency:2}}})

db.users.updateOne({name:"Maria"},{**$addToSet**: {hobbies:{title:"Hiking", frequency:2}}})

*--addToSet adds unique values only*

**DELETE**

db.users.deleteOne({name: "Chris"})

db.users.deleteMany({totalAge:{$exists: false},isSporty: true})

db.users.deleteMany({})

db.users.drop() *--sysadmin task collection drop*

db.dropDatabase()*--sysadmin task database drop*

**Index - Retrieve data efficiently**

*collection scan*

*ordered index column*

*cost at the time of insert*

mongoimportpersons.json -d contactData -c contacts --jsonArray --drop

>db.contacts.find({"dob.age"{$gt:60}})

>db.contacts.findOne({"dbo.age"{$gt:60}}).count() *--1222*

>db.contacts.**explain().**find({"dob.age":{$gt:60}})

{

"queryPlanner": {

"plannerVersion": 1,

"namespace": "contactData.contacts",

"indexFilterSet": false,

"parsedQuery": {

"dob.age": {

"$gt": 60

}

},

"queryHash": "FC9E47D2",

"planCacheKey": "FC9E47D2",

"winningPlan": {

"stage": "COLLSCAN",

"filter": {

"dob.age": {

"$gt": 60

}

},

"direction": "forward"

},

"rejectedPlans": [ ]

},

"serverInfo": {

"host": "UDAY-PC",

"port": 27017,

"version": "4.2.1",

"gitVersion": "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok": 1

}

>db.contacts.**explain("executionStats").**find({"dob.age":{$gt:60}})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.contacts",

"indexFilterSet" : false,

"parsedQuery" : {

"dob.age" : {

"$gt" : 60

}

},

"winningPlan" : {

"stage" : "COLLSCAN",

"filter" : {

"dob.age" : {

"$gt" : 60

}

},

"direction" : "forward"

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

**"nReturned" : 1222,**

**"executionTimeMillis" : 13,**

**"totalKeysExamined" : 0,**

**"totalDocsExamined" : 5000,**

"executionStages" : {

**"stage" : "COLLSCAN",**

"filter" : {

"dob.age" : {

"$gt" : 60

}

},

"nReturned" : 1222,

"executionTimeMillisEstimate" : 1,

"works" : 5002,

"advanced" : 1222,

"needTime" : 3779,

"needYield" : 0,

"saveState" : 39,

"restoreState" : 39,

"isEOF" : 1,

"direction" : "forward",

"docsExamined" : 5000

}

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

>db.contacts.**createIndex**({"dob.age": 1}) *--asc*

>db.contacts.createIndex({"dob.age": -1}) *--desc*

>db.contacts.explain("executionStats").find({"dob.age":{$gt:60}})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.contacts",

"indexFilterSet" : false,

"parsedQuery" : {

"dob.age" : {

"$gt" : 60

}

},

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

**"stage" : "IXSCAN",**

"keyPattern" : {

"dob.age" : 1

},

"indexName" : "dob.age\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"dob.age" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"dob.age" : [

"(60.0, inf.0]"

]

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 1222,

**"executionTimeMillis" : 47,**

**"totalKeysExamined" : 1222**,

**"totalDocsExamined" : 1222,**

"executionStages" : {

"stage" : "FETCH",

"nReturned" : 1222,

"executionTimeMillisEstimate" : 1,

"works" : 1223,

"advanced" : 1222,

"needTime" : 0,

"needYield" : 0,

"saveState" : 9,

"restoreState" : 9,

"isEOF" : 1,

"docsExamined" : 1222,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 1222,

"executionTimeMillisEstimate" : 0,

"works" : 1223,

"advanced" : 1222,

"needTime" : 0,

"needYield" : 0,

"saveState" : 9,

"restoreState" : 9,

"isEOF" : 1,

"keyPattern" : {

"dob.age" : 1

},

"indexName" : "dob.age\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"dob.age" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"dob.age" : [

"(60.0, inf.0]"

]

},

"keysExamined" : 1222,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

}

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

>db.contacts.explain("executionStats").find({"dob.age":{$gt:20}})

"executionTimeMillis" : 39,

**"totalKeysExamined" : 5000,**

**"totalDocsExamined" : 5000,**

>db.contacts.dropIndex({"dob.age": 1})

"executionTimeMillis" : 12,

Compound index

>db.contacts.createIndex({"gender": 1})

>db.contacts.explain("executionStats").find({gender: "male"})

*doesn’t make sense - 2435 records, like Boolean*

>db.contacts.dropIndex({"gender": 1})

db.contacts.createIndex({"dob.age":1, gender:1}) *-- one index on two fields*

>db.contacts.explain().find({"dob.age":35, gender: "male"})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.contacts",

"indexFilterSet" : false,

"parsedQuery" : {

"$and" : [

{

"dob.age" : {

"$eq" : 35

}

},

{

"gender" : {

"$eq" : "male"

}

}

]

},

"queryHash" : "6602FE25",

"planCacheKey" : "0EBB13B2",

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

**"stage" : "IXSCAN", --index scan**

"keyPattern" : {

"dob.age" : 1,

"gender" : 1

},

**"indexName" : "dob.age\_1\_gender\_1",**

"isMultiKey" : false,

"multiKeyPaths" : {

"dob.age" : [ ],

"gender" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"dob.age" : [

"[35.0, 35.0]"

],

"gender" : [

"[\"male\", \"male\"]"

]

}

}

},

"rejectedPlans" : [ ]

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

>db.contacts.explain().find({"dob.age":35}) *--uses same index*

*--use left /left-right (sorted left to right) age /age - gender*

**"stage" : "IXSCAN",**

"indexName" : "**dob.age\_1\_gender\_1**",

>db.contacts.explain().find({gender:"male"})

**"stage" : "COLLSCAN",** *--gender - not sorted, age sorted in this ex.*

indexes for sorting

*32mb for sorting*

*no index - sort in memory and therefore time out*

>db.contacts.explain().find({"dob.age":35}).sort({gender:1})

**"stage" : "IXSCAN",**

understanding default index

>db.contacts.**getIndexes()**

[

{

"v" : 2,

"key" : {

**"\_id" : 1** --default

},

"name" : "\_id\_",

"ns" : "contactData.contacts"

},

{

"v" : 2,

**"key" : {**

**"dob.age" : 1,**

**"gender" : 1**

**},**

"name" : "dob.age\_1\_gender\_1",

"ns" : "contactData.contacts"

}

]

Configuring index

>db.contacts.createIndex({email: 1},**{unique: true}**) *-- guarantee - unique values*

{

"ok" : 0,

"errmsg" : "E11000 duplicate key error collection: contactData.contacts index: email\_1 dup key: { email: \"abigail.clark@example.com\" }",

"code" : 11000,

"codeName" :**"DuplicateKey",**

"keyPattern" : {

"email" : 1

},

"keyValue" : {

"email" : "abigail.clark@example.com"

}

}

>db.contacts.find({email:"abigail.clark@example.com"}).count()

2

Understanding Partial filter

*bigger the index - more performance drop*

*partial index - add value regularly going to look*

*--can be added to compound index*

>db.contacts.createIndex({"dob.age":1}, **{partialFilterExpression: {gender: "male"}**})

>db.contacts.explain().find({"dob.age":{$gt:60}})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.contacts",

"indexFilterSet" : false,

"parsedQuery" : {

"dob.age" : {

"$gt" : 60

}

},

"queryHash" : "FC9E47D2",

"planCacheKey" : "D8015F8B",

"winningPlan" : {

**"stage" : "COLLSCAN",** *--as gender wasn't there in selection criteria*

"filter" : {

"dob.age" : {

"$gt" : 60

}

},

"direction" : "forward"

},

"rejectedPlans" : [ ]

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

applying partial index

db.users.insertMany([{name:'Max', email:'max@test.com'},{name:'Manu'}])

>db.users.createIndex**({email:1}, {unique: true})**

>db.users.insertOne({name:"Anna"})

2019-11-18T18:54:49.616-0800 E QUERY [js] WriteError({

"index" : 0,

"code" : 11000,

"errmsg" : "E11000 **duplicate key error** collection: contactData.users index: email\_1 dup key: { **email: null** }",

"op" : {

"\_id" :ObjectId("5dd35979aea4f1b6cd9859fb"),

"name" : "Anna"

}

}) :

WriteError({

"index" : 0,

"code" : 11000,

"errmsg" : "E11000 duplicate key error collection: contactData.users index: email\_1 dup key: { email: null }",

"op" : {

"\_id" :ObjectId("5dd35979aea4f1b6cd9859fb"),

"name" : "Anna"

}

})

WriteError@src/mongo/shell/bulk\_api.js:458:48

mergeBatchResults@src/mongo/shell/bulk\_api.js:855:49

executeBatch@src/mongo/shell/bulk\_api.js:919:13

Bulk/this.execute@src/mongo/shell/bulk\_api.js:1163:21

DBCollection.prototype.insertOne@src/mongo/shell/crud\_api.js:264:9

@(shell):1:1

>db.users.createIndex({email:1},

{**unique: true**, **partialFilterExpression: {email: {$exists: true}}**}) *--element where email exists*

Understanding TTL (Time to Live)

>db.sessions.insertOne({data: "asdasf", createdAt: new Date()})

>db.sessions.find().pretty()

{

"\_id" :ObjectId("5dd35aceaea4f1b6cd9859fc"),

"data" : "asdasf",

"createdAt" :ISODate("2019-11-19T03:00:30.764Z")

}

>db.sessions.createIndex({createdAt: 1})

>db.sessions.dropIndex({createdAt: 1})

>db.sessions.createIndex({createdAt: 1}, **{expireAfterSeconds: 10}**)

*--every element should remove after 10 secs*

*-- session data -- data clean up itself*

*-- only on date objects*

*-- does not work on compound index*

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

**Query diagnosis and Query Planning**

Covered Queries

>db.customers.insertMany([{name: 'Max', age:29, salary:3000},

{name: 'Manu', age:30, salary:4000}])

>db.customers.createIndex({name:1})

>db.customers.explain("executionStats").find({name: "Max"})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.customers",

"indexFilterSet" : false,

"parsedQuery" : {

"name" : {

"$eq" : "Max"

}

},

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

**"stage" : "IXSCAN",**

"keyPattern" : {

"name" : 1

},

"indexName" : "name\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"Max\", \"Max\"]"

]

}

}

},

**"rejectedPlans" : [ ]**

},

"executionStats" : {

"executionSuccess" : true,

**"nReturned" : 1,**

"executionTimeMillis" : 2,

"totalKeysExamined" : 1,

**"totalDocsExamined" : 1,**

"executionStages" : {

"stage" : "FETCH",

"nReturned" : 1,

"executionTimeMillisEstimate" : 0,

"works" : 2,

"advanced" : 1,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"docsExamined" : 1,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 1,

"executionTimeMillisEstimate" : 0,

"works" : 2,

"advanced" : 1,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"name" : 1

},

"indexName" : "name\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"Max\", \"Max\"]"

]

},

"keysExamined" : 1,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

}

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

>

>db.customers.explain("executionStats").find({name: "Max"},{\_id:0, name:1})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.customers",

"indexFilterSet" : false,

"parsedQuery" : {

"name" : {

"$eq" : "Max"

}

},

"winningPlan" : {

"stage" : "PROJECTION\_COVERED",

"transformBy" : {

"\_id" : 0,

"name" : 1

},

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"name" : 1

},

"indexName" : "name\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"Max\", \"Max\"]"

]

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

**"nReturned" : 1,**

"executionTimeMillis" : 0,

"totalKeysExamined" : 1,

**"totalDocsExamined" : 0,**

"executionStages" : {

"stage" : "PROJECTION\_COVERED",

"nReturned" : 1,

"executionTimeMillisEstimate" : 0,

"works" : 2,

"advanced" : 1,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"transformBy" : {

"\_id" : 0,

"name" : 1

},

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 1,

"executionTimeMillisEstimate" : 0,

"works" : 2,

"advanced" : 1,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"name" : 1

},

"indexName" : "name\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"Max\", \"Max\"]"

]

},

"keysExamined" : 1,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

}

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

Mongodb rejects plan

>db.customers.createIndex({age:1, name:1}) *--name sorted within age*

>db.customers.explain().find({name: "Max", age:30})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.customers",

"indexFilterSet" : false,

"parsedQuery" : {

"$and" : [

{

"age" : {

"$eq" : 30

}

},

{

"name" : {

"$eq" : "Max"

}

}

]

},

"queryHash" : "2A9321A4",

"planCacheKey" : "66DCA32B",

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

**"stage" : "IXSCAN",**

"keyPattern" : {

"age" : 1,

"name" : 1

},

**"indexName" : "age\_1\_name\_1",**

"isMultiKey" : false,

"multiKeyPaths" : {

"age" : [ ],

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"age" : [

"[30.0, 30.0]"

],

"name" : [

"[\"Max\", \"Max\"]"

]

}

}

},

**"rejectedPlans"** : [

{

"stage" : "FETCH",

"filter" : {

"age" : {

"$eq" : 30

}

},

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"name" : 1

},

**"indexName" : "name\_1",**

"isMultiKey" : false,

"multiKeyPaths" : {

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"Max\", \"Max\"]"

]

}

}

}

]

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

**winning plans**

>db.customers.explain**("allPlansExecution").**find({name: "Max", age:30})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.customers",

"indexFilterSet" : false,

"parsedQuery" : {

"$and" : [

{

"age" : {

"$eq" : 30

}

},

{

"name" : {

"$eq" : "Max"

}

}

]

},

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

"stage" : "**IXSCAN**",

"keyPattern" : {

"age" : 1,

"name" : 1

},

"indexName" : "**age\_1\_name\_1**",

"isMultiKey" : false,

"multiKeyPaths" : {

"age" : [ ],

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"age" : [

"[30.0, 30.0]"

],

"name" : [

"[\"Max\", \"Max\"]"

]

}

}

},

"rejectedPlans" : [

{

"stage" : "FETCH",

"filter" : {

"age" : {

"$eq" : 30

}

},

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"name" : 1

},

"indexName" : "**name\_1**",

"isMultiKey" : false,

"multiKeyPaths" : {

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"Max\", \"Max\"]"

]

}

}

}

]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 0,

"executionTimeMillis" : 0,

"totalKeysExamined" : 0,

"totalDocsExamined" : 0,

"executionStages" : {

"stage" : "FETCH",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 2,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"docsExamined" : 0,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"age" : 1,

"name" : 1

},

"indexName" : "age\_1\_name\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"age" : [ ],

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"age" : [

"[30.0, 30.0]"

],

"name" : [

"[\"Max\", \"Max\"]"

]

},

"keysExamined" : 0,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

},

"allPlansExecution" : [

{

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"totalKeysExamined" : 0,

"totalDocsExamined" : 0,

"executionStages" : {

"stage" : "FETCH",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"docsExamined" : 0,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "**IXSCAN**",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"age" : 1,

"name" : 1

},

"indexName" : "**age\_1\_name\_1**",

"isMultiKey" : false,

"multiKeyPaths" : {

"age" : [ ],

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"age" : [

"[30.0, 30.0]"

],

"name" : [

"[\"Max\", \"Max\"]"

]

},

"keysExamined" : 0,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

}

},

{

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"totalKeysExamined" : 1,

"totalDocsExamined" : 1,

"executionStages" : {

"stage" : "FETCH",

"filter" : {

"age" : {

"$eq" : 30

}

},

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 1,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 0,

"docsExamined" : 1,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 1,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 1,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 0,

"keyPattern" : {

"name" : 1

},

"indexName" : "name\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"name" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"name" : [

"[\"Max\", \"Max\"]"

]

},

"keysExamined" : 1,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

}

}

]

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

Multi-Key Index

*Multikey -- as array of value*

*-- stored differently. pulls all the values in array and stores them as separate element in index*

*array 4 value in 1000 rows = 4000 elements stored*

>db.contacts.drop()

>db.contacts.insertOne({name: "Max", hobbies:["cooking","Sports"], address:[{street:"Main Street"}, {street:"second street"}]})

>db.contacts.createIndex({hobbies:1})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

>db.contacts.find({hobbies:"Sports"}).pretty()

>db.contacts.explain("executionStats").find({hobbies:"Sports"})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.contacts",

"indexFilterSet" : false,

"parsedQuery" : {

"hobbies" : {

"$eq" : "Sports"

}

},

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"hobbies" : 1

},

"indexName" : "hobbies\_1",

"isMultiKey" : true,

"multiKeyPaths" : {

"hobbies" : [

"hobbies"

]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"hobbies" : [

"[\"Sports\", \"Sports\"]"

]

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 1,

"executionTimeMillis" : 0,

"totalKeysExamined" : 1,

"totalDocsExamined" : 1,

"executionStages" : {

"stage" : "FETCH",

"nReturned" : 1,

"executionTimeMillisEstimate" : 0,

"works" : 2,

"advanced" : 1,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"docsExamined" : 1,

"alreadyHasObj" : 0,

"inputStage" : {

**"stage" : "IXSCAN",**

"nReturned" : 1,

"executionTimeMillisEstimate" : 0,

"works" : 2,

"advanced" : 1,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"hobbies" : 1

},

"indexName" : "hobbies\_1",

**"isMultiKey" : true,**

"multiKeyPaths" : {

"hobbies" : [

"hobbies"

]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"hobbies" : [

"[\"Sports\", \"Sports\"]"

]

},

"keysExamined" : 1,

"seeks" : 1,

"dupsTested" : 1,

"dupsDropped" : 0

}

}

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

>db.contacts.createIndex({**addresses:1**})

>db.contacts.explain("executionStats").find({"addresses.street":"Main Street"})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.contacts",

"indexFilterSet" : false,

"parsedQuery" : {

"addresses.street" : {

"$eq" : "Main Street"

}

},

"winningPlan" : {

"stage" : "COLLSCAN",

"filter" : {

"addresses.street" : {

"$eq" : "Main Street"

}

},

"direction" : "forward"

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 0,

"executionTimeMillis" : 0,

"totalKeysExamined" : 0,

"totalDocsExamined" : 1,

"executionStages" : {

**"stage" : "COLLSCAN",**

"filter" : {

"addresses.street" : {

"$eq" : "Main Street"

}

},

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 3,

"advanced" : 0,

"needTime" : 2,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"direction" : "forward",

"docsExamined" : 1

}

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

>

*does not go to see the documents inside the addresses with "." dot operator*

>db.contacts.explain("executionStats").find({addresses: {street:"Main Street"}})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.contacts",

"indexFilterSet" : false,

"parsedQuery" : {

"addresses" : {

"$eq" : {

"street" : "Main Street"

}

}

},

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

"stage" : "IXSCAN",

"keyPattern" : {

"addresses" : 1

},

"indexName" : "addresses\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"addresses" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"addresses" : [

"[{street: \"Main Street\" }, {street: \"Main Street\" }]"

]

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 0,

"executionTimeMillis" : 2,

"totalKeysExamined" : 0,

"totalDocsExamined" : 0,

"executionStages" : {

"stage" : "FETCH",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"docsExamined" : 0,

"alreadyHasObj" : 0,

"inputStage" : {

**"stage" : "IXSCAN",**

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"addresses" : 1

},

"indexName" : "addresses\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"addresses" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"addresses" : [

"[{ street: \"Main Street\" }, { street: \"Main Street\" }]"

]

},

"keysExamined" : 0,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

}

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

>

>db.contacts.createIndex({**"address.street":1**})

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 3,

"numIndexesAfter" : 4,

"ok" : 1

}

>db.contacts.explain("executionStats").find({"address.street":"Main Street"})

{

"queryPlanner" : {

"plannerVersion" : 1,

"namespace" : "contactData.contacts",

"indexFilterSet" : false,

"parsedQuery" : {

"addresses.street" : {

"$eq" : "Main Street"

}

},

"winningPlan" : {

"stage" : "FETCH",

"inputStage" : {

**"stage" : "IXSCAN",**

"keyPattern" : {

"addresses.street" : 1

},

"indexName" : "addresses.street\_1",

**"isMultiKey" : true**

"multiKeyPaths" : {

"addresses.street" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"addresses.street" : [

"[\"Main Street\", \"Main Street\"]"

]

}

}

},

"rejectedPlans" : [ ]

},

"executionStats" : {

"executionSuccess" : true,

"nReturned" : 0,

"executionTimeMillis" : 2,

"totalKeysExamined" : 0,

"totalDocsExamined" : 0,

"executionStages" : {

"stage" : "FETCH",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"docsExamined" : 0,

"alreadyHasObj" : 0,

"inputStage" : {

"stage" : "IXSCAN",

"nReturned" : 0,

"executionTimeMillisEstimate" : 0,

"works" : 1,

"advanced" : 0,

"needTime" : 0,

"needYield" : 0,

"saveState" : 0,

"restoreState" : 0,

"isEOF" : 1,

"keyPattern" : {

"addresses.street" : 1

},

"indexName" : "addresses.street\_1",

"isMultiKey" : false,

"multiKeyPaths" : {

"addresses.street" : [ ]

},

"isUnique" : false,

"isSparse" : false,

"isPartial" : false,

"indexVersion" : 2,

"direction" : "forward",

"indexBounds" : {

"addresses.street" : [

"[\"Main Street\", \"Main Street\"]"

]

},

"keysExamined" : 0,

"seeks" : 1,

"dupsTested" : 0,

"dupsDropped" : 0

}

}

},

"serverInfo" : {

"host" : "UDAY-PC",

"port" : 27017,

"version" : "4.2.1",

"gitVersion" : "edf6d45851c0b9ee15548f0f847df141764a317e"

},

"ok" : 1

}

>db.contacts.createIndex({**name:1, hobbies:1**}) *--possible - compound* ***with 1 multi key***

*--error - as it creates Cartesian product of the values stored.*

>db.contacts.createIndex({**address:1, hobbies:1**})

{

"ok" : 0,

"errmsg" : "cannot index parallel arrays [hobbies] [address]",

"code" : 171,

"codeName" : "CannotIndexParallelArrays"

}

Text index

*$regex- low performance*

*text array converts text into array of words (key words only). removes - is, a, the*

>db.products.insertMany

([{title:' a book', description: 'this is an awesome book of a young artist!'},

{title: 'red t shirt', description: 'this t shirt is red and its pretty awesome'}])

>db.products.find().pretty()

{

"\_id": ObjectId("5dd36adbaea4f1b6cd985a01"),

"title": " a book",

"description": "this is an awesome book of a young artist!"

}

{

"\_id": ObjectId("5dd36adbaea4f1b6cd985a02"),

"title": "red t shirt",

"description": "this t shirt is red and its pretty awesome"

}

>

>db.products.createIndex({description: 1})

**>db.products.dropIndex({description: 1})**

**>db.products.createIndex({description: "text"})**

>db.products.find({$text: **{$search: "awesome"}**})

*--case not important. everything is stored as a lower case*

>db.products.find({$text: **{$search: "book"}**})

>db.products.find({$text: **{$search: "red book"}**})

{ "\_id" : ObjectId("5dd36adbaea4f1b6cd985a01"), "title" : " a book", "description" : "this is an awesome book of a young artist!" }

{ "\_id" : ObjectId("5dd36adbaea4f1b6cd985a02"), "title" : "red t shirt", "description" : "this t shirt is red and its pretty awesome" }

>db.products.find({$text: **{$search: "\"red book\""}**})

>

*--returns nothing*

>db.products.find({$text: **{$search: "\"awesome book\""}**})

{ "\_id" : ObjectId("5dd36adbaea4f1b6cd985a01"), "title" : " a book", "description" : "this is an awesome book of a young artist!" }

*--returns phrase*

Text index and sorting

>db.products.find({$text: {$search: "awesome shirt"}})

{ "\_id" : ObjectId("5dd36adbaea4f1b6cd985a01"), "title" : " a book", "description" : "this is an **awesome** book of a young artist!" }

{ "\_id" : ObjectId("5dd36adbaea4f1b6cd985a02"), "title" : "red t shirt", "description" : "this t **shirt** is read and its pretty **awesome**" }

>db.products.find({$text: {$search: "awesome shirt"}}, **{score: {$meta: "textScore"}}**).pretty()

{

"\_id" :ObjectId("5dd36adbaea4f1b6cd985a02"),

"title" : "red t shirt",

"description" : "this t **shirt** is read and its pretty **awesome**",

**"score" : 1.2**

}

{

"\_id" :ObjectId("5dd36adbaea4f1b6cd985a01"),

"title" : " a book",

"description" : "this is an **awesome** book of a young artist!",

**"score" : 0.625**

}

*--by default, sorts with the score.*

>db.products.find({$text: {$search: "awesome shirt"}}, {score: {$meta: "textScore"}})

**.sort({score: {$meta: "textScore"}}).**pretty() *--enforcing sorting*

Creating combined text index

>db.products.**createIndex({title: "text"})** *–limit one index per collection*

{

"ok" : 0,

"errmsg" : "Index: { v: 2, key: { \_ft

"description\_text\", ns: \"contactData.produc

"code" : 85,

"codeName" : "IndexOptionsConflict"

>db.products.dropIndex({description: "text"})

{

"ok" : 0,

"errmsg" : "can't find index with key: { description: \"text\" }",

"code" : 27,

**"codeName" : "IndexNotFound"**

}

>db.products.dropIndex({"name" : "description\_text"}) *--does not work*

>db.products.**dropIndex("description\_text")** -*dropindex text index with index name.*

{ "nIndexesWas" : 2, "ok" : 1 }

>db.products.createIndex**({title: "text", description: "text"})**

{

"createdCollectionAutomatically" : false,

"numIndexesBefore" : 1,

"numIndexesAfter" : 2,

"ok" : 1

}

>db.products.insertOne({title:"A Ship", description:"Floats perfectly!"})

>db.products.find({$text:{$search: "ship"} })

{ "\_id" : ObjectId("5dd37168aea4f1b6cd985a03"), "title" : "A Ship", "description" : "Floats perfectly!" }

Using text index to exclude words

>db.products.find({$text:{$search: "awesome"} })

{ "\_id" : ObjectId("5dd36adbaea4f1b6cd985a01"), "title" : " a book", "description" : "this is an awesome book of a young artist!" }

{ "\_id" : ObjectId("5dd36adbaea4f1b6cd985a02"), "title" : "red t shirt", "description" : "this t shirt is read and its pretty awesome" }

>db.products.find**({$text:{$search: "awesome -shirt"} })***–excluding word shirt*

{ "\_id" : ObjectId("5dd36adbaea4f1b6cd985a01"), "title" : " a book", "description" : "this is an awesome book of a young artist!" }

Setting default language & using weights

>db.products.createIndex({title: "text", description: "text"}, **{default\_language:"german"}**)

*--defined how stop words are removed*

>db.products.createIndex({title: "text", description: "text"}, **{default\_language:"english"}**)

>db.products.createIndex({title: "text", description: "text"}, **{default\_language:"english", weights: {title: 1, description:10}}**) *--description 10 times title*

>db.products.find({$text: {$search: "", $language:"english", **$caseSensitive: true**}})

>db.products.find({$text: {$search: "red"}}, {score: {$meta: "textScore"}}).pretty()

{

"\_id" :ObjectId("5dd36adbaea4f1b6cd985a02"),

"title" : "red t shirt",

"description" : "this t shirt is read and its pretty awesome",

"score" : 0.6666666666666666

}*--score different because of the weights*

Building index

*foreground -collection is locked and can’t edit, faster*

*background - collection is accessible, slower*

cmd

mongo credit-rating.js

mongo

show dbs

use credit

show collections

db.ratings.count() *–1000000 records*

db.ratings.findOne()

db.ratings.craeteIndex({age:1})

db.ratings.explain("executionStats").find({age: {$gt: 80}}) *–Ixscan, 100k keys examined, 225 ms*

db.ratings.dropIndex({age:1})

db.ratings.explain("executionStats").find({age: {$gt: 80}}) *--400ms*

db.ratings.craeteIndex({age:1})

db.ratings.findOne() *--takes a while*

db.ratings.insertOne({person\_id:'sadas', score: 55.211, age:90}) *--takes a while to create index*

*therefore create in background*

db.ratings.craeteIndex({age:1}, {background: true})

db.ratings.insertOne({person\_id:'sadas', score: 55.211, age:90})

*-- inserts immediately*

*-- not locking collection*

**Geospetial Data**

*Adding GeoJSON Data*

>db.places.insertOne({name: "California Academy of Sciences",

location: {**type:"Point",coordinates:** [-122.4738657, 37.7683167]}})

*Geospetial type longitude latitude*

>db.places.find(

{location: {$near: {$geometry:{type: "Point", coordinates:[-122.4748527, 37.7697458]}}}})

Error: error: {

"ok" : 0,

"errmsg" : "error processing query: ns=awesomeplaces.placesTree: GEONEAR field=location maxdist=1.79769e+308 isNearSphere=0\nSort: {}\nProj: {}\n planner returned error :: caused by :: **unable to find index** for $geoNear query",

"code" : 291,

"codeName" : "NoQueryExecutionPlans"

}

*--need geospetial index*

>db.places.createIndex({location: **"2dsphere"**})

{

"createdCollectionAutomatically": false,

"numIndexesBefore": 1,

"numIndexesAfter": 2,

"ok": 1

}

*--find places nearby from a random spot*

>db.places.find({location: **{$near:{$geometry:** {type: "Point", coordinates:[-122.4748527, 37.7697458]}}}}).pretty()

{ "\_id": ObjectId("5dd49c23fec34cab8042de50"),

"name": "California Academy of Sciences",

"location": {

"type": "Point",

"coordinates": [

-122.4738657,

37.7683167

]

}

}

>db.places.find({location: {$near:{$geometry: {type: "Point", coordinates:[-122.4748527, 37.7697458]}, **$maxDistance: 500, $minDistance:10**}}}).pretty()

{ "\_id": ObjectId("5dd49c23fec34cab8042de50"),

"name": "California Academy of Sciences",

"location": {

"type": "Point",

"coordinates": [

-122.4738657,

37.7683167

]

}

}

*--If you do not add min max it shows all the places in asc order of distance.*

*--distance is added in meters*

>db.places.insertOne({name: "Conservatory of Flowers",

location: {type: "Point", coordinates:[ -122.4792301, 37.7697458]}})

# >db.places.insertOne({name: "Robin Williams Meadow",

location: {type: "Point", coordinates:[ -122.4612807, 37.7682444]}})

# >db.places.insertOne({name: "San Francisco County Fair Building",

# location: {type: "Point", coordinates:[ -122.4663179, 37.7691816]}})

# >db.places.insertOne({name: " San Francisco Fire Department Station 12",

# location: {type: "Point", coordinates:[ -122.4569804, 37.7629446]}})

# *--Finding spots within the area*

# > const p1= [-122.454682, 37.774757]

# > p1

# [ -122.454682, 37.774757]

# > const p2= [-122.452979, 37.766375]

# > const p3= [-122.510350, 37.764060]

# > const p4= [-122.511053, 37.771340]

# >db.places.find({location: {$geoWithin: {$geometry: {type:"Polygon", coordinates: [[p1, p2, p3, p4, p1]]}}}})

# { "\_id" : ObjectId("5dd4a337fec34cab8042de53"), "name" : "San Francisco County Fair Building", "location" : { "type" : "Point", "coordinates" : [ -122.4663179, 37.7691816 ] } }

# { "\_id" : ObjectId("5dd4a222fec34cab8042de52"), "name" : "Robin Williams Meadow", "location" : { "type" : "Point", "coordinates" : [ -122.4612807, 37.7682444 ] } }

# { "\_id" : ObjectId("5dd49c23fec34cab8042de50"), "name" : "California Academy of Sciences", "location" : { "type" : "Point", "coordinates" : [ -122.4738657, 37.7683167 ] } }

# { "\_id" : ObjectId("5dd4a0f6fec34cab8042de51"), "name" : "Conservatory of Flowers", "location" : { "type" : "Point", "coordinates" : [ -122.4792301, 37.7697458 ] } }

# *--find if spot is inside the area- other way*

# >db.areas.insertOne({name: "Golden Gate Park" , area: {type: "Polygon", coordinates: [[p1, p2, p3, p4, p1]]}})

# >db.areas.createIndex({area: "2dsphere"})

# >db.areas.find({area:{$geoIntersects: {$geometry: {type: "Point", coordinates:[-122.4666982, 37.7682388]}}}})

# { "\_id" : ObjectId("5dd4a79ffec34cab8042de55"), "name" : "Golden Gate Park", "area" : { "type" : "Polygon", "coordinates" : [ [ [ -122.454682, 37.774757 ], [ -122.452979, 37.766375 ], [ -122.51035, 37.76406 ], [ -122.511053, 37.77134 ], [ -122.454682, 37.774757 ] ] ] } }

# Find places within certain radius

# *center radius (km)*

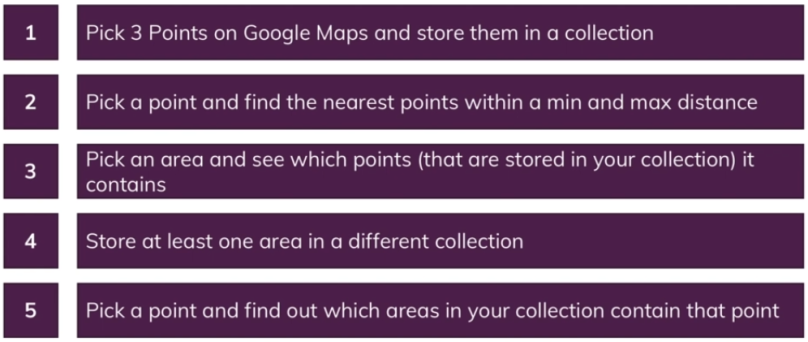
>db.places.find({location: **{$geoWithin: {$centerSphere:** [[-122.461720, 37.770787], 1/6378.1]}}})

{ "\_id" : ObjectId("5dd4a337fec34cab8042de53"), "name" : "San Francisco County Fair Building", "location" : { "type" : "Point", "coordinates" : [ -122.4663179, 37.7691816 ] } }

{ "\_id" : ObjectId("5dd4a615fec34cab8042de54"), "name" : " San Francisco Fire Department Station 12", "location" : { "type" : "Point", "coordinates" : [ -122.4569804, 37.7629446 ] } }

{ "\_id" : ObjectId("5dd4a222fec34cab8042de52"), "name" : "Robin Williams Meadow", "location" : { "type" : "Point", "coordinates" : [ -122.4612807, 37.7682444 ] } }

Example



>db.missionBay.insertOne({name:"Sea World", location:{type:"Point", coordinates:[-117.2633824,13, 32.7754859]}})

>db.missionBay.insertOne({name:"Hyatt", location:{type:"Point", coordinates:[-117.2459584, 32.7772876]}})

>db.missionBay.insertOne({name:"Sandbar", location: {type:"Point", coordinates: [-117.2538118,32.7737508]}})

>db.missionBay.insertOne({name:"Crown Point", location: { type:"Point", coordinates: [-117.2503348,32.7731668]}})

>db.missionBay.createIndex({location:"2dsphere"})

>db.missionBay.find({location:{$near:{$geometry: {type:"Point", coordinates: [-117.2503348, 32.7731668]}, $maxDistance: 800, $minDistance: 10}}})

{ "\_id" : ObjectId("5dd4b4d7fec34cab8042de5b"), "name" : "Sandbar", "location" : { "type" : "Point", "coordinates" : [ -117.2538118, 32.7737508 ] } }

{ "\_id" : ObjectId("5dd4b080fec34cab8042de57"), "name" : "Hyatt", "location" : { "type" : "Point", "coordinates" : [ -117.2459584, 32.7772876 ] } }

>db.missionBay.find()

{ "\_id" : ObjectId("5dd4affbfec34cab8042de56"), "name" : "Sea World", "location" : { "type" : "Point", "coordinates" : [ -117.2633824, 13, 32.7754859 ] } }

{ "\_id" : ObjectId("5dd4b080fec34cab8042de57"), "name" : "Hyatt", "location" : { "type" : "Point", "coordinates" : [ -117.2459584, 32.7772876 ] } }

{ "\_id" : ObjectId("5dd4b4cefec34cab8042de5a"), "name" : "Crown Point", "location" : { "type" : "Point", "coordinates" : [ -117.2503348, 32.7731668 ] } }

{ "\_id" : ObjectId("5dd4b4d7fec34cab8042de5b"), "name" : "Sandbar", "location" : { "type" : "Point", "coordinates" : [ -117.2538118, 32.7737508 ] } }\

# > const mbp1= [-117.253081, 32.771958]

# > const mbp2= [-117.233726, 32.773256]

# > const mbp3= [-117.228059, 32.796563]

# > const mbp4= [-117.255142, 32.790647]

# >db.missionBay.find({location:{$geoWithin: {$geometry: {type: "Polygon", coordinates:[[mbp1, mbp2, mbp3, mbp4, mbp1]]}}}})

# { "\_id" : ObjectId("5dd4b4cefec34cab8042de5a"), "name" : "Crown Point", "location" : { "type" : "Point", "coordinates" : [ -117.2503348, 32.7731668 ] } }

# { "\_id" : ObjectId("5dd4b080fec34cab8042de57"), "name" : "Hyatt", "location" : { "type" : "Point", "coordinates" : [ -117.2459584, 32.7772876 ] } }

>db.missionBayArea.insertOne({name: "Mission Bay Area", area: {type:"Polygon", coordinates:[[mbp1, mbp2, mbp3, mbp4, mbp1]]}})

# >db.missionBayArea.createIndex({area:"2dsphere"})

# >db.areas.find({area:{$geoIntersects: {$geometry: {type: "Point", coordinates:[-122.4666982, 37.7682388]}}}})

# >db.missionBayArea.find({area:{$geoIntersects: {$geometry: {type: "Point", coordinates:[-117.2457428,32.7872382]}}}})

# { "\_id" : ObjectId("5dd4b7e8fec34cab8042de5c"), "name" : "Mission Bay Area", "area" : { "type" : "Polygon", "coordinates" : [ [ [ -117.253081, 32.771958 ], [ -117.233726, 32.773256 ], [ -117.228059, 32.796563 ], [ -117.255142, 32.790647 ], [ -117.253081, 32.771958 ] ] ] } }

# > const mbp1= [-117.253081, 32.771958]

# > const mbp2= [-117.233726, 32.773256]

# > const mbp3= [-117.228059, 32.796563]

# > const mbp4= [-117.255142, 32.790647]

# db.places.find({loc:{$geoWithin: {$geometry: {type: ‘Polygon’, coordinates:[[mbp1,mbp2,mbp3,mbp4,mbp1]]}}}})

# > const polygonArea = [[mbp1,mbp2,mbp3,mbp4,mbp1]]

# > const polygonObject = {type:’Polygon’, coordinates: polygonArea}

# db.places.find({loc:{$geoWithin:{$geometry: polygonObject}}})

# db.areas.insertOne({name:”are1”, a:polyObject})

**Aggregation**

*pipeline of functions. input data - stages - result*

*--aggregate similar to find, has indexes, accepts array*

mongoimportpersons.json -d analytics -c persons --jsonArray

db.persons.aggregate([ **{$match**:{gender:"female"} }])

*--aggregate returns cursor*

db.persons.aggregate([ {$match:{gender:"female"}} ]).pretty()

db.persons.aggregate([

{$match:{gender:"female"}},

{ **$group:** {\_id: { state: "$location.state"}, totalPersons: {$sum: 1}}}

]).pretty() *--every person adds 1*

{ "\_id" : { "state" : "ohio" }, "totalPersons" : 1 }

{ "\_id" : { "state" : "hatay" }, "totalPersons" : 3 }

{ "\_id" : { "state" : "niedersachsen" }, "totalPersons" : 3 }

**{ "\_id" : { "state" : "florida" }, "totalPersons" : 3 }**

>db.persons.find({"location.state": 'florida', gender:'female'}).count()

3

db.persons.aggregate([

{$match:{gender:"female"}},

{$group: {\_id: { state: "$location.state"}, totalPersons: {$sum: 1}}}

{**$sort:** {totalPersons: -1}} *--output of 2nd state (previous stage --pipeline)*

]).pretty()

example:

*find only persons older than 50*

*group by gender - how many people by gender*

*what is the average age in gender*

*order by total persons per gender*

db.persons.aggregate([

{$match:{"dob.age":{$gt: 50}}},

{$group: {\_id:{gender: "$gender"}, total: {**$sum:1**}, average: {**$avg:"$dob.age"**}}},

{$sort: {total:-1}}

])

**Projection**

db.persons.aggregate([

{$project: {\_id:0, gender:1, fullname: { **$concat:** ["Hello", "World"]}}}

]).pretty()

{ "gender" : "male", "fullname" : "HelloWorld" }

{ "gender" : "male", "fullname" : "HelloWorld" }

db.persons.aggregate([

{$project: {\_id:0, gender:1, fullname: { $concat: ["$name.first", " ", "$name.last"]}}}

]).pretty()

{ "gender" : "male", "fullname" : "victor pedersen" }

{ "gender" : "male", "fullname" : "carl jacobs" }

db.persons.aggregate([

{$project: {

\_id:0,

gender:1,

fullname: { **$concat: [{$toUpper:** "$name.first"}, " ", {$toUpper: "$name.last"}]}}}

]).pretty()

{ "gender" : "male", "fullname" : "VICTOR PEDERSEN" }

{ "gender" : "male", "fullname" : "CARL JACOBS" }

db.persons.aggregate([

{$project: { \_id:0, gender:1,

fullname: { $concat: [

**{$toUpper: {$substrCP: ["$name.first", 0, 1]}},**

**{$substrCP: ["$name.first", 1, { $subtract: [ {$strLenCP: "$name.first"}, 1] }]},**

" ",

{$toUpper: {$substrCP: ["$name.last", 0, 1]}},

{$substrCP: ["$name.last", 1, { $subtract: [ {$strLenCP: "$name.last"}, 1] }]}

]}

}} ])

{ "gender" : "male", "fullname" : "Victor Pedersen" }

{ "gender" : "male", "fullname" : "Carl Jacobs" }

db.persons.aggregate([

{$project: {\_id:0, name: 1, email:1,

**location: {type:"Point", coordinates: ["$location.coordinates.longitude", "$location.coordinates.latitude"]}**}},

{$project: { email:1, location:1,

fulname: { $concat: [

{$toUpper: {$substrCP: ["$name.first", 0, 1]}},

{$substrCP: ["$name.first", 1, { $subtract: [ {$strLenCP: "$name.first"}, 1] }]},

" ",

{$toUpper: {$substrCP: ["$name.last", 0, 1]}},

{$substrCP: ["$name.last", 1, { $subtract: [ {$strLenCP: "$name.last"}, 1] }]}

]}

}}

])

{ "gender" : "male", "location" : { "type" : "Point", "coordinates" : [ "-31.0208", "-29.8113" ] }, "email" : "victor.pedersen@example.com", "fullname" : "Victor Pedersen" }

{ "gender" : "male", "location" : { "type" : "Point", "coordinates" : [ "-154.6037", "-29.6721" ] }, "email" : "carl.jacobs@example.com", "fullname" : "Carl Jacobs" }

db.persons.aggregate([

{$project: {\_id:0, name: 1, email:1, gender:1,

location: {type:"Point",

coordinates: [

**{$convert:{input: "$location.coordinates.longitude", to: "double", onError: 0.0, onNull:0.0}},**

{$convert:{input: "$location.coordinates.latitude", to: "double", onError: 0.0, onNull:0.0}}

]}}},

{$project: { gender:1, email:1, location:1,

fulname: { $concat: [

{$toUpper: {$substrCP: ["$name.first", 0, 1]}},

{$substrCP: ["$name.first", 1, { $subtract: [ {$strLenCP: "$name.first"}, 1] }]},

" ",

{$toUpper: {$substrCP: ["$name.last", 0, 1]}},

{$substrCP: ["$name.last", 1, { $subtract: [ {$strLenCP: "$name.last"}, 1] }]}

]}

}}

])

{ "gender" : "male", "location" : { "type" : "Point", "coordinates" : [ -31.0208, -29.8113 ] }, "email" : "victor.pedersen@example.com", "fullname" : "Victor Pedersen" }

{ "gender" : "male", "location" : { "type" : "Point", "coordinates" : [ -154.6037, -29.6721 ] }, "email" : "carl.jacobs@example.com", "fullname" : "Carl Jacobs" }

db.persons.aggregate([

{$project: {\_id:0, name: 1, email:1,

**birthdate: {$convert: {input: "$dob.date", to: "date"}},**

age: "$dob.age",

location: {type:"Point",

coordinates: [

{$convert:{input: "$location.coordinates.longitude", to: "double", onError: 0.0, onNull:0.0}},

{$convert:{input: "$location.coordinates.latitude", to: "double", onError: 0.0, onNull:0.0}}

]}}},

{$project: { gender:1, email:1, location:1, birthdate:1, age:1,

fulname: { $concat: [

{$toUpper: {$substrCP: ["$name.first", 0, 1]}},

{$substrCP: ["$name.first", 1, { $subtract: [ {$strLenCP: "$name.first"}, 1] }]},

" ",

{$toUpper: {$substrCP: ["$name.last", 0, 1]}},

{$substrCP: ["$name.last", 1, { $subtract: [ {$strLenCP: "$name.last"}, 1] }]}

]}

}}

])

{ "location" : { "type" : "Point", "coordinates" : [ -31.0208, -29.8113 ] }, "email" : "victor.pedersen@example.com", "birthdate" : ISODate("1959-02-19T23:56:23Z"), "age" : 59, "fulname" : "Victor Pedersen" }

{ "location" : { "type" : "Point", "coordinates" : [ -154.6037, -29.6721 ] }, "email" : "carl.jacobs@example.com", "birthdate" : ISODate("1984-09-30T01:20:26Z"), "age" : 33, "fulname" : "Carl Jacobs" }

db.persons.aggregate([

{$project: {\_id:0, name: 1, email:1,

**birthdate: {$toDate: "$dob.date"},**

age: "$dob.age",

location: {type:"Point",

coordinates: [

{$convert:{input: "$location.coordinates.longitude", to: "double", onError: 0.0, onNull:0.0}},

{$convert:{input: "$location.coordinates.latitude", to: "double", onError: 0.0, onNull:0.0}}

]}}},

{$project: { gender:1, email:1, location:1, birthdate:1, age:1,

fulname: { $concat: [

{$toUpper: {$substrCP: ["$name.first", 0, 1]}},

{$substrCP: ["$name.first", 1, { $subtract: [ {$strLenCP: "$name.first"}, 1] }]},

" ",

{$toUpper: {$substrCP: ["$name.last", 0, 1]}},

{$substrCP: ["$name.last", 1, { $subtract: [ {$strLenCP: "$name.last"}, 1] }]}

]}

}}

])

"location" : { "type" : "Point", "coordinates" : [ -31.0208, -29.8113 ] }, "email" : "victor.pedersen@example.com", "birthdate" : ISODate("1959-02-19T23:56:23Z"), "age" : 59, "fulname" : "Victor Pedersen" }

"location" : { "type" : "Point", "coordinates" : [ -154.6037, -29.6721 ] }, "email" : "carl.jacobs@example.com", "birthdate" : ISODate("1984-09-30T01:20:26Z"), "age" : 33, "fulname" : "Carl Jacobs" }

db.persons.aggregate([

{$project: {\_id:0, name: 1, email:1,

birthdate: {$toDate: "$dob.date"},

age: "$dob.age",

location: {type:"Point",

coordinates: [

{$convert:{input: "$location.coordinates.longitude", to: "double", onError: 0.0, onNull:0.0}},

{$convert:{input: "$location.coordinates.latitude", to: "double", onError: 0.0, onNull:0.0}}

]}}},

{$project: { gender:1, email:1, location:1, birthdate:1, age:1,

fulname: { $concat: [

{$toUpper: {$substrCP: ["$name.first", 0, 1]}},

{$substrCP: ["$name.first", 1, { $subtract: [ {$strLenCP: "$name.first"}, 1] }]},

" ",

{$toUpper: {$substrCP: ["$name.last", 0, 1]}},

{$substrCP: ["$name.last", 1, { $subtract: [ {$strLenCP: "$name.last"}, 1] }]}

]}

}},

**{ $group: {\_id: {birthYear:{$isoWeekYear: "$birthdate"}}, numPersons: {$sum: 1}}},**

**{ $sort: {numPersons: -1}}**

])

{ "\_id" : { "birthYear" : NumberLong(1955) }, "numPersons" : 113 }

{ "\_id" : { "birthYear" : NumberLong(1961) }, "numPersons" : 111 }

{ "\_id" : { "birthYear" : NumberLong(1993) }, "numPersons" : 110 }

*Group - n:1, sum,count,avg*

*Project 1:1, transform*

db.friends.insertMany([

{ "name": "Max",

"hobbies": ["Sports", "Cooking"],

"age": 29,

"examScores": [

{ "difficulty": 4, "score": 57.9 },

{ "difficulty": 6, "score": 62.1 },

{ "difficulty": 3, "score": 88.5 }

]

},

{ "name": "Manu",

"hobbies": ["Eating", "Data Analytics"],

"age": 30,

"examScores": [

{ "difficulty": 7, "score": 52.1 },

{ "difficulty": 2, "score": 74.3 },

{ "difficulty": 5, "score": 53.1 }

]

},

{ "name": "Maria",

"hobbies": ["Cooking", "Skiing"],

"age": 29,

"examScores": [

{ "difficulty": 3, "score": 75.1 },

{ "difficulty": 8, "score": 44.2 },

{ "difficulty": 6, "score": 61.5 }

]

} ])

>db.friends.find()

db.friends.aggregate([

{$group: {\_id: {age: "$age"}, allHobbies**:{$push: "$hobbies"}**}}

])

{ "\_id" : { "age" : 29 }, "allHobbies" : [ [ "Sports", "Cooking" ], [ "Cooking", "Skiing" ] ] }

{ "\_id" : { "age" : 30 }, "allHobbies" : [ [ "Eating", "Data Analytics" ] ] }

db.friends.aggregate([

{**$unwind:** "$hobbies"},

])

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3c9"), "name" : "Max", "hobbies" : "Sports", "age" : 29, "examScores" : [ { "difficulty" : 4, "score" : 57.9 }, { "difficulty" : 6, "score" : 62.1 }, { "difficulty" : 3, "score" : 88.5 } ] }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3c9"), "name" : "Max", "hobbies" : "Cooking", "age" : 29, "examScores" : [ { "difficulty" : 4, "score" : 57.9 }, { "difficulty" : 6, "score" : 62.1 }, { "difficulty" : 3, "score" : 88.5 } ] }

db.friends.aggregate([

{$unwind: "$hobbies"},

{$group: {\_id: {age: "$age"}, allHobbies:{$push: "$hobbies"}}}

])

{ "\_id" : { "age" : 29 }, "allHobbies" : [ "Sports", "Cooking", "Cooking", "Skiing" ] }

{ "\_id" : { "age" : 30 }, "allHobbies" : [ "Eating", "Data Analytics" ] }

db.friends.aggregate([

{$unwind: "$hobbies"},

{$group: {\_id: {age: "$age"}, allHobbies:{**$addToSet:** "$hobbies"}}}

])

{ "\_id" : { "age" : 29 }, "allHobbies" : [ "Sports", "Cooking", "Skiing" ] }

{ "\_id" : { "age" : 30 }, "allHobbies" : [ "Eating", "Data Analytics" ] }

**Projections with arrays - slice**

db.friends.aggregate([

{$project: {\_id:0, examScore: **{$slice: ["$examScores",1]}**}}

])

{ "examScore" : [ { "difficulty" : 4, "score" : 57.9 } ] }

{ "examScore" : [ { "difficulty" : 7, "score" : 52.1 } ] }

{ "examScore" : [ { "difficulty" : 3, "score" : 75.1 } ] }

>db.friends.aggregate([ {$project: {\_id:0, examScore: **{$slice: ["$examScores",-2]}**}} ])*--starts at the end*

{ "examScore" : [ { "difficulty" : 6, "score" : 62.1 }, { "difficulty" : 3, "score" : 88.5 } ] }

{ "examScore" : [ { "difficulty" : 2, "score" : 74.3 }, { "difficulty" : 5, "score" : 53.1 } ] }

{ "examScore" : [ { "difficulty" : 8, "score" : 44.2 }, { "difficulty" : 6, "score" : 61.5 } ] }

>db.friends.aggregate([ {$project: {\_id:0, examScore: **{$slice: ["$examScores",2,1]}**}} ])

--start at position 2, grab element 1

{ "examScore" : [ { "difficulty" : 3, "score" : 88.5 } ] }

{ "examScore" : [ { "difficulty" : 5, "score" : 53.1 } ] }

{ "examScore" : [ { "difficulty" : 6, "score" : 61.5 } ] }

*length of array*

>db.friends.aggregate([

... {$project: {\_id: 0, numScores: **{$size: "$examScores"}**}}

... ])

{ "numScores" : 3 }

{ "numScores" : 3 }

{ "numScores" : 3 }

$filter operator

>db.friends.aggregate([

... {$project: {\_id: 0,

... scores: **{$filter:{input:"$examScores", as: "sc", cond: {$gt:["$$sc.score", 60]}}}**}}

... ])

{ "scores" : [ { "difficulty" : 6, "score" : 62.1 }, { "difficulty" : 3, "score" : 88.5 } ] }

{ "scores" : [ { "difficulty" : 2, "score" : 74.3 } ] }

{ "scores" : [ { "difficulty" : 3, "score" : 75.1 }, { "difficulty" : 6, "score" : 61.5 } ] }

db.friends.aggregate([

{$unwind:"$examScores"},

{$sort: {"examScores.score": -1}}

])

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3c9"), "name" : "Max", "hobbies" : [ "Sports", "Cooking" ], "age" : 29, "examScores" : { "difficulty" : 3, "score" : 88.5 } }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3cb"), "name" : "Maria", "hobbies" : [ "Cooking", "Skiing" ], "age" : 29, "examScores" : { "difficulty" : 3, "score" : 75.1 } }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3ca"), "name" : "Manu", "hobbies" : [ "Eating", "Data Analytics" ], "age" : 30, "examScores" : { "difficulty" : 2, "score" : 74.3 } }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3c9"), "name" : "Max", "hobbies" : [ "Sports", "Cooking" ], "age" : 29, "examScores" : { "difficulty" : 6, "score" : 62.1 } }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3cb"), "name" : "Maria", "hobbies" : [ "Cooking", "Skiing" ], "age" : 29, "examScores" : { "difficulty" : 6, "score" : 61.5 } }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3c9"), "name" : "Max", "hobbies" : [ "Sports", "Cooking" ], "age" : 29, "examScores" : { "difficulty" : 4, "score" : 57.9 } }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3ca"), "name" : "Manu", "hobbies" : [ "Eating", "Data Analytics" ], "age" : 30, "examScores" : { "difficulty" : 5, "score" : 53.1 } }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3ca"), "name" : "Manu", "hobbies" : [ "Eating", "Data Analytics" ], "age" : 30, "examScores" : { "difficulty" : 7, "score" : 52.1 } }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3cb"), "name" : "Maria", "hobbies" : [ "Cooking", "Skiing" ], "age" : 29, "examScores" : { "difficulty" : 8, "score" : 44.2 } }

db.friends.aggregate([

{$unwind:"$examScores"},

{$project:{\_id:0, name:1, age:1, score: "$examScores.score"}},

{$sort: {score: -1}} ---not array anymore. no need to use $

])

{ "name" : "Max", "age" : 29, "score" : 88.5 }

{ "name" : "Maria", "age" : 29, "score" : 75.1 }

{ "name" : "Manu", "age" : 30, "score" : 74.3 }

{ "name" : "Max", "age" : 29, "score" : 62.1 }

{ "name" : "Maria", "age" : 29, "score" : 61.5 }

{ "name" : "Max", "age" : 29, "score" : 57.9 }

{ "name" : "Manu", "age" : 30, "score" : 53.1 }

{ "name" : "Manu", "age" : 30, "score" : 52.1 }

{ "name" : "Maria", "age" : 29, "score" : 44.2 }

>db.friends.aggregate([

... {$unwind:"$examScores"},

... {$project:{\_id:1, name:1, age:1, score: "$examScores.score"}},

... {$sort: {score: -1}} ,

... {$group: {\_id:"$\_id", maxScore:{$max: "$score"}}}

... ])

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3cb"), "maxScore" : 75.1 }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3ca"), "maxScore" : 74.3 }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3c9"), "maxScore" : 88.5 }

>db.friends.aggregate([

... {$unwind:"$examScores"},

... {$project:{\_id:1, name:1, age:1, score: "$examScores.score"}},

... {$sort: {score: -1}} ,

... {$group: {\_id:"$\_id", name:{$first: "$name"}, maxScore:{$max: "$score"}}},

... {$sort: {maxScore: -1}}

... ])

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3c9"), "name" : "Max", "maxScore" : 88.5 }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3cb"), "name" : "Maria", "maxScore" : 75.1 }

{ "\_id" : ObjectId("5dd61566dca0da90aa36a3ca"), "name" : "Manu", "maxScore" : 74.3 }

**distribution with bucket**

>db.persons.aggregate([

... {$bucket:

... {groupBy: "$dob.age",

... boundaries: [0, 18, 30, 50, 80, 120],

... output: {

... numPersons: {$sum:1},

... averageAge: {$avg: "$dob.age"},

... names: {$push: "$name.first"}

... }

... }}

... ])

{ "\_id" : 18, "numPersons" : 868, "averageAge" : 25.101382488479263, "names" : [ "katie", ...

>db.persons.aggregate([

... {$bucket:

... {groupBy: "$dob.age",

... boundaries: [0, 18, 30, 50, 80, 120],

... output: {

... numPersons: {$sum:1},

... averageAge: {$avg: "$dob.age"},

... }

... }}

... ])

{ "\_id" : 18, "numPersons" : 868, "averageAge" : 25.101382488479263 }

{ "\_id" : 30, "numPersons" : 1828, "averageAge" : 39.4917943107221 }

{ "\_id" : 50, "numPersons" : 2304, "averageAge" : 61.46440972222222 }

db.persons.aggregate([ {

$bucketAuto:

{

groupBy: "$dob.age",

buckets: 5,

output: {

numPersons: {$sum:1},

avgAge: {$avg: "$dob.age"}

}

}

}

])

{ "\_id" : { "min" : 21, "max" : 32 }, "numPersons" : 1042, "avgAge" : 25.99616122840691 }

{ "\_id" : { "min" : 32, "max" : 43 }, "numPersons" : 1010, "avgAge" : 36.97722772277228 }

{ "\_id" : { "min" : 43, "max" : 54 }, "numPersons" : 1033, "avgAge" : 47.98838334946757 }

{ "\_id" : { "min" : 54, "max" : 65 }, "numPersons" : 1064, "avgAge" : 58.99342105263158 }

{ "\_id" : { "min" : 65, "max" : 74 }, "numPersons" : 851, "avgAge" : 69.11515863689776 }

10 persons with oldest birth date

db.persons.aggregate([

{$project: {\_id:0, name:1, birthDate: { $toDate: "$dob.date"}}}

])

db.persons.aggregate([

{$project: {\_id:0, name:1, birthDate: { $toDate: "$dob.date"}}},

{$sort: {birthDate: 1}}

])

db.persons.aggregate([

{$project: {\_id:0, name:1, birthDate: { $toDate: "$dob.date"}}},

{$sort: {birthDate: 1}},

{$limit: 10}

])

>db.persons.aggregate([

... {$project: {\_id:0,

... name:{ $concat: ["$name.first"," ", "$name.last"]},

... birthDate: { $toDate: "$dob.date"}}},

... {$sort: {birthDate: 1}},

... {$limit: 10}

... ])

{ "name" : "victoria hale", "birthDate" : ISODate("1944-09-07T15:52:50Z") }

{ "name" : "عباسیاسمی", "birthDate" : ISODate("1944-09-12T07:49:20Z") }

{ "name" : "erundinaporto", "birthDate" : ISODate("1944-09-13T14:58:41Z") }

{ "name" : "پرهامجعفری", "birthDate" : ISODate("1944-09-16T16:03:28Z") }

{ "name" : "eli henry", "birthDate" : ISODate("1944-09-17T15:04:13Z") }

{ "name" : "kirk brown", "birthDate" : ISODate("1944-09-18T11:03:05Z") }

{ "name" : "alexisbélanger", "birthDate" : ISODate("1944-10-02T22:56:32Z") }

{ "name" : "gina beck", "birthDate" : ISODate("1944-10-04T07:41:31Z") }

{ "name" : "sebastianolsen", "birthDate" : ISODate("1944-10-13T15:29:05Z") }

{ "name" : "lucywilson", "birthDate" : ISODate("1944-10-25T16:27:56Z") }

--order is important

>db.persons.aggregate([

... {$project: {\_id:0,

... name:{ $concat: ["$name.first"," ", "$name.last"]},

... birthDate: { $toDate: "$dob.date"}}},

... {$sort: {birthDate: 1}},

... {$skip:10},

... {$limit: 10}

... ])

{ "name" : "evamurray", "birthDate" : ISODate("1944-10-29T02:05:56Z") }

{ "name" : "elena chevalier", "birthDate" : ISODate("1944-10-31T02:56:40Z") }

{ "name" : "gretchenschmidtke", "birthDate" : ISODate("1944-11-01T20:49:03Z") }

{ "name" : "joseph thomas", "birthDate" : ISODate("1944-11-06T11:08:45Z") }

{ "name" : "sarah lee", "birthDate" : ISODate("1944-11-07T07:53:47Z") }

{ "name" : "conradscheepbouwer", "birthDate" : ISODate("1944-11-08T02:15:17Z") }

{ "name" : "martinacharles", "birthDate" : ISODate("1944-11-08T07:38:49Z") }

{ "name" : "olgablanco", "birthDate" : ISODate("1944-11-17T09:16:50Z") }

{ "name" : "elisamorales", "birthDate" : ISODate("1944-11-22T22:51:47Z") }

{ "name" : "rafaelvelasco", "birthDate" : ISODate("1944-11-27T07:12:20Z") }

db.persons.aggregate([

{$match: { gender: "male"}},

{$project: {\_id:0,

name:{ $concat: ["$name.first"," ", "$name.last"]},

birthDate: { $toDate: "$dob.date"}}},

{$sort: {birthDate: 1}},

{$skip:10},

{$limit: 10}

])

{ "gender" : "male", "name" : "pierreboyer", "birthDate" : ISODate("1945-01-01T22:35:55Z") }

{ "gender" : "male", "name" : "emile noel", "birthDate" : ISODate("1945-01-10T03:05:21Z") }

{ "gender" : "male", "name" : "torgeirapeland", "birthDate" : ISODate("1945-01-13T17:04:33Z") }

{ "gender" : "male", "name" : "igorkvistad", "birthDate" : ISODate("1945-01-17T22:13:14Z") }

{ "gender" : "male", "name" : "mariuszgabler", "birthDate" : ISODate("1945-01-22T06:16:30Z") }

{ "gender" : "male", "name" : "lewis freeman", "birthDate" : ISODate("1945-01-28T20:15:28Z") }

{ "gender" : "male", "name" : "theodoremoore", "birthDate" : ISODate("1945-02-10T03:34:29Z") }

{ "gender" : "male", "name" : "florianmercier", "birthDate" : ISODate("1945-02-22T04:18:31Z") }

{ "gender" : "male", "name" : "dursunschellekens", "birthDate" : ISODate("1945-02-22T07:28:00Z") }

{ "gender" : "male", "name" : "marcel rey", "birthDate" : ISODate("1945-02-28T02:18:01Z") }

writing pipeline results into a new collection

db.persons.aggregate([

{$project: {\_id:0, name: 1, email:1,

birthdate: {$toDate: "$dob.date"},

age: "$dob.age",

location: {type:"Point",

coordinates: [

{$convert:{input: "$location.coordinates.longitude", to: "double", onError: 0.0, onNull:0.0}},

{$convert:{input: "$location.coordinates.latitude", to: "double", onError: 0.0, onNull:0.0}}

]}}},

{$project: { gender:1, email:1, location:1, birthdate:1, age:1,

fulname: { $concat: [

{$toUpper: {$substrCP: ["$name.first", 0, 1]}},

{$substrCP: ["$name.first", 1, { $subtract: [ {$strLenCP: "$name.first"}, 1] }]},

" ",

{$toUpper: {$substrCP: ["$name.last", 0, 1]}},

{$substrCP: ["$name.last", 1, { $subtract: [ {$strLenCP: "$name.last"}, 1] }]}

]}

}},

{$out: "transformedPersons"}

])

> show collections

friends

persons

transformedPersons

>db.transformedPersons.find()

{ "\_id" : ObjectId("5dd626a3ee6f6b367b283954"), "location" : { "type" : "Point", "coordinates" : [ -31.0208, -29.8113 ] }, "email" : "victor.pedersen@example.com", "birthdate" : ISODate("1959-02-19T23:56:23Z"), "age" : 59, "fulname" : "Victor Pedersen" }

{ "\_id" : ObjectId("5dd626a3ee6f6b367b283955"), "location" : { "type" : "Point", "coordinates" : [ -154.6037, -29.6721 ] }, "email" : "carl.jacobs@example.com", "birthdate" : ISODate("1984-09-30T01:20:26Z"), "age" : 33, "fulname" : "Carl Jacobs" }

{ "\_id" : ObjectId("5dd626a3ee6f6b367b283956"), "location" : { "type" : "Point", "coordinates" : [ 34.1689, 4.6625 ] }, "email" : "پریا.پارسا@example.com", "birthdate" : ISODate("1962-01-10T05:26:30Z"), "age" : 56, "fulname" : "پریاپارسا" }

{ "\_id" : ObjectId("5dd626a3ee6f6b367b283957"), "location" : { "type" : "Point", "coordinates" : [ 111.3806, -31.6359 ] }, "email" : "maeva.wilson@example.com", "birthdate" : ISODate("1962-08-11T20:51:07Z"), "age" : 56, "fulname" : "Maeva Wilson" }

{ "\_id" : ObjectId("5dd626a3ee6f6b367b283958"), "location" : { "type" : "Point", "coordinates" : [ -18.5996, -42.6128 ] }, "email" : "elijah.lewis@example.com", "birthdate" : ISODate("1986-03-29T06:40:18Z"), "age" : 32, "fulname" : "Elijah Lewis" }

{ "\_id" : ObjectId("5dd626a3ee6f6b367b283959"), "location" : { "type" : "Point", "coordinates" : [ -67.5738, -52.8348 ] }, "email" : "olav.oehme@example.com", "birthdate" : ISODate("1960-11-28T23:07:18Z"), "age" : 57, "fulname" : "Olav Oehme" }

$geoNear:

>db.transformedPersons.createIndex({location:"2dsphere"})

db.transformedPersons.aggregate([

{$geoNear: {

near: {type: "Point", coordinates: [-90.6, -65.2]},

maxDistance: 100000,

$limit: 10,

query: { age: {$gt:30} },

distanceField: "distance"

}}

])

{ "\_id" : ObjectId("5dd626a3ee6f6b367b283965"), "location" : { "type" : "Point", "coordinates" : [ -90.4049, -65.0877 ] }, "email" : "delia.durand@example.com", "birthdate" : ISODate("1966-08-03T09:22:41Z"), "age" : 52, "fulname" : "Delia Durand", "distance" : 15479.591845673658 }

{ "\_id" : ObjectId("5dd626a3ee6f6b367b283af2"), "location" : { "type" : "Point", "coordinates" : [ -91.6107, -65.0265 ] }, "email" : "nicklas.rasmussen@example.com", "birthdate" : ISODate("1978-01-19T05:30:01Z"), "age" : 40, "fulname" : "Nicklas Rasmussen", "distance" : 51134.13688625751 }

**Working with numbers**

Db.persons.insertOne({name: ‘Max, age:29})

29.0000002 store behind the scenes – 64 bit double

db.insertOne({num:29}) –size 35

db.insertOne({num: NumberInt(“29”)}) –size 31

db.companies.insertOne({valuation:NumberInt(“5000000000”)}) – try this

no error but stores different value if we go above the range

db.companies.insertOne({valuation: NumberInt(“2147483647”)})

2147483647

db.companies.insertOne({valuation: NumberInt(“2147483648”)})

-2147483648

db.companies.insertOne({valuation: 2147483648)})

stored correctly as 64 bitdouble

64 bit all are not available for integer values. Available for decimal

Best way to store long int

db.companies.insertOne({valuation: NumberLong(9223372036854775807)})

–error as double 64. Therefore pass as string

db.companies.insertOne({valuation: NumberLong(“9223372036854775807”)})

db.accounts.insertOne({name: “Max”, amount:”12323423423454235463465464676545463”}) – can be stored as text

db.accounts.insertOne({name: “Max”, amount:”12”}) – calculation will fail

db.zccounts.updateOne({name: ‘Max’}, {$inc:{amount:1}}) – error as not number but string.

db.accounts.insertOne({name: “Max”, amount:numberInt(”12”)})

db.zccounts.updateOne({name: ‘Max’}, {$inc:{amount:numberInt(“10”}})

db.companies.insertOne({funding: NumberLong(“123456789123456789”)})

db.companies.updateOne({}, {$inc: {valuation: 1}}) –NumberLong- rounds it wrong. As out of capacity. 123456789123456780

db.companies.updateOne({}, {$inc: {valuation: NumberLong(“1”)}}) --correct 123456789123456790

db.science.insertOne({a:0.3, b:0.1})

0.3000000000000000001

db.science.aggregate([ {$project: {result: {$subtract: [“$a”,”$b”]}}} ]) ---0.19999999999999998 approximation

db.science.insertOne({a: NumberDecimal(“0.3”), b: NumberDecimal(“0.1”)}) – mongoDB shell constructor

db.science.aggregate([ {$project: {result: {$subtract: [“$a”,”$b”]}}} ]) – result : NumberDecimal(“0.2”)

db.science.updateOne({}, {$inc: {a:0.1}}) --NumberDecimal(“0.400000000000000”)

db.science.updateOne({}, {$inc: {a: NumberDecimal(“0.1”)}}) -- guarantee not to add any precision

db.nums.insertOne({a: 0.1}) – size 33

db.nums.insertOne({a: NumberDecimal(“0.1”)}) –size 41

**MongoDB Security & user**

Authentication -valid users

Authorization -roles-privileges (Resources - collections, Actions- insert())

Administrator – manage database config, create users

Developer -CRUD

Data scientist – fetch data

Create and edit user

Sudomongod --auth

Mongo -u max -p max ---username, password

db.auth(‘maxuser’,’maxpassword’)

show dbs

show collections

no access so no dbs

use admin --database

db.createUser({user:’Max’, pwd:’max’, roles:[“userAdminAnyDatabase”]}) –right to administer any database

successfully added user.

Need to authenticate as well

Db.auth(‘max’,’max’)

Built in roles

Db user – Read, readWrite

dbAdmin – userAdmin, dbOwner

all dbroles – readAnyDatabase, readWriteAnyDatabase, userAdminAnyDatabase, dbAdminAnyDatabase

cluster Admin – clusterManager, clusterMonitor, hostManager, clusterAdmin -- multiple mongo server together

Backup/Restore – backup, restore

Superuser – dbOwner (admin), userAdmin(admin), userAdminAnyDatabase, root

Mongo -u max -p max ---error

Mongo -u max -p max --authenticationDatabase admin ----authentication database need to be mentioned

Use shop

Db.createUser({user: ‘appdev’, pwd: ‘dev’, roles: [“readWrite”] }) –global read write

Usuccessfully added

Db.auth(‘appdev’, ‘dev’)

Db.product.insertOne({name:’book’, price: 12.99})

--error – too many users authenticated

Db.logout() –logout from current user

Mongo -u appdev -p dev –authenticationDatabase admin – fail

Mongo -u appdev -p dev –authenticationDatabase shop – success

Db.product.insertOne({name:’book’, price: 12.99}) –error

Use shop

Db.product.insertOne({name:’book’, price: 12.99}) – just works with shop

Db.updateUser(“appdev”, {pwd:’nepwd’, roles: [“readWrite”, {role: “readwrite”, db: “blog”}]}) –error

Db.logout()

Db.auth(‘max’, ‘max’) –fail in shop db because max user is in admin database

Use admin

Db.auth(‘max’,’max’)

Db.updateUser(“appdev”, {pwd:’nepwd’, roles: [“readWrite”, {role: “readwrite”, db: “blog”}]}) –error as user in shop db

Use shop

Db.updateUser(“appdev”, {pwd:’nepwd’, roles: [“readWrite”, {role: “readwrite”, db: “blog”}]}) – works confirm with getUser

Db.getUser(“adddev”)

Db.logout

Use shop

Db.auth(‘adddev’ , ‘newpwd’)

Use blog

Db.posts.insertOne(title: “This works”}) –error -too many user authenticated.

--should have switched to admin to log out with my max user.

Db.posts.insertOne(title: “This works”}) –works

Example

Dbpath set, scratch setup, delete old db

dropUser()/dropDatabase()

New mongodb environment

keep 3 users –

database admin -create collection, index

User admin – Manage Users

Developer – Read Write Data in customer and sales db

mongod –dbpath path

mongod –auth

mongo

use admin

db.createUser({user:’maxUAdmin’, pwd:’maxUpwd’, role:[“userAdminAnyDatabase”]})

db.auth(‘maxUAdmin’,’maxUpwd’)

db.createUser({user:’maxdbAdmin’, pwd:’maxdbpwd’, role:[“dbAdminAnyDatabase”]})

db.auth(‘maxdbAdmin’,’maxdbpwd’)

mongo -u maxUAdmin -p maxUpwd

db.createUser({user:’maxDeveloper’, pwd:’maxDpsw’}, roles:[ {role: “readwrite”, db: “customer”} ,{role: “readwrite”, db: “sales”} ])

mongo -u maxdbAdmin -p maxdbpwd –authenticationDatabase admin

show dbs

use customers

db.createColelction(“customerData”)

mongo -u maxDevloper -p maxdpwd –authenticationDatabase admin

use customers

db.custoemrData.insertOne({name:’Max”})

use blog

db.post.insertOne({title:’Does it work?”}) ---fail

**Encryption**

**Encryption during transportaion**

SSL – public private key pair

Linux /mac – run command google mongodbssl

openssl – Exec – download and run executable and run same command there

generating 2048 bit RSA private key

writing private key to mongodb.cert.key

country Name: US

city

unit name

common Name: localhost ---address of webserver

email

2 files. :mongodb.cert, mongodb.key

Now concatenate both the keys

type mongodb-cert.key mongodb-cert.crt >mongodb.pem

mongodb.pem – to enable ssl encryption

.pem folder >mongod --sslMorearg ----set ssl configured port, requireSSL

--sslPEMKetFilearg -----PEM file for ssl –self sign certificate

--sslCAfilearg- -official paid version

>mongod –sslModerequireSSL–sslPEMKeyFilemongodb.pem

----Start server

----waiting for connections on port 27017 ssl

----to connect to server open different cmd prompt .. same poath of pemfolder

>mongo --it will fail or get connected to different mongodinstance. Shut down

>mongo –ssl – sslCAFilemongodb.pem –host localhost

--now all data sent from mongo shell to server is encrypted

**Encryption during REST**

Encrypt storage

As a developer storing password hash that password -driver module of course

**Performance, Fault Tolerance, Deployment**

What Influence Performance?

Developer/ DB Admin - Efficient queries/Operations, Indexes, Fitting Data Schema

DB Admin/system admin - h/w sharding, replica

Capped Collections – limit amount of data, old data automatically deleted/cached , use most recent logs

> use performance

>db.createCollection(“capped”, {capped: true, size:10000, max: 3})

--default size 4 byte, capped 3 documents most

>db.capped.insertOne({name:”Max”})

>db.capped.insertOne({name:”Manu”})

>db.capped.insertOne({name:”Anna”})

>db.capped.find() ---order in which it retrieved is always the order in which it is inserted

--normal collection mit may be the case not guarantee

>db.capped.find().sort({$natural: -1}) ----sorts the other way round

>db.capped.insertOne({name:”Maria”}) –Max is gone

**Replica Set** -managed as system administrator

Client(shell, Driver)

Insert MongoDB – add node –primary Node – replica Set –(Asynchronous Replica – not immediate)) --secondary Node

--backup/fault Tolerance / Improve Read Performance

Write to primary, reads can talk to secondary nodes

Sharding (horizontal scaling)

More read and write on server. Horizontal scale is multiple servers – data chunk split over server – data is distributed

Queries run across all Shards

Shards connected to Router mongos – with Shard key

Queries &sharding

Find() - mongos – 1. broadcast to shard -each shard returns response

2. mongos has shard key – direct shard contact and access data.

Deploying a MongoDB server

Localhost – mongod ---- > webserver –mongod

Tasks

Manage Shards

Manage Replica set

Secure user/Auth Setup

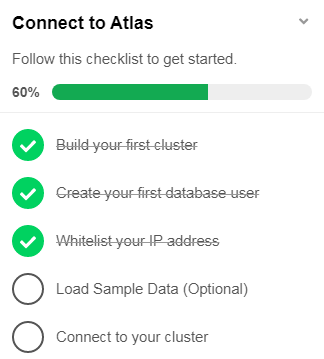
Protect web server/network

Encryption Regular backups

Update software

MongoDB Atlast -scalable and best practice mongodb server running through cloud. Does all the tasks

create cluster



add user - security/ database access - can set priviledges and roles - collection

IP whitelist : Security/ Network Access: Add IP address of server running application or in this case current IP address

backup setting atlas

Alerts: add new alert -email when something happens , avg read time long

cluster:collections

cluster:metrics

cluster:migrate data into cluster

cluster:connect - ip address whitelist, connection method: shell

quit running shell , server in Atlas. path as below /collection

mongo "mongodb+srv://cluster0-yijrp.mongodb.net/test" --username UdayTak

enter password. MsChicago2016 -----connected to Mongo cloud

show dbs

use products

db.products.insertOne({title:"A Book", price:12.99})

**Transactions**

Users -- post (1:m) if user delete.. delete posts together

if user deleted.. but network down.. posts still there. therefore transaction . so rollback

use blog

db.users.insertOne({name:'Max'})

db.posts.insertMany([{title:'first Post', userId: ObjectId("5dd8bf1ffe88b3d06b5cd059")},{title:'second Post', userId: ObjectId("5dd8bf1ffe88b3d06b5cd059")}])

db.users.deleteOne({\_id: ObjectId("5dd8bf1ffe88b3d06b5cd059"})

db.posts.deleteMany({userId: ObjectId("5dd8bf1ffe88b3d06b5cd059")})

transactions need session

session -requests group logically

const session = db.getMongo().startSession()

session.startTransaction()

const usersColl = session.getDatabase("test").users

const postsColl = session.getDatabase("test").posts

usersColl.deleteOne({\_id: ObjectId("5dd8bf1ffe88b3d06b5cd059"}) --acknowledged and saved but does not run on the database

db.users.find({}) --shows result

postsColl.deleteMany({userId: ObjectId("5dd8bf1ffe88b3d06b5cd059")})

session.commitTransaction()

session.abortTransaction()

**Mongo drivers**

shell commnads to driver commands

shell - configure db, create collection, create index

Driver - CRUD, aggregation pipeline

preparing project

atlas - cluster setup, one user security with read and write access

local ip add in whitelist

install nodejs

unzip nodejs project

navigate to project folder path

>npm install ----install dependencies of the project

>npm start ----serves up front-end react interface

--starts developement server

--error - need to start node REST API

new cmd> navigate to project folder>

npm run start:server

npm install mongodb ---install mongodb driver on terminal inside visual studio code

mongo do not connect directly to react or any front end - as security

Atlas - instead of connect through shell

-connect through application

- driver server

-view driver connection example - Mode - code how to connect to address

app.js

request to backend

code to connect to cluster copy paste enter username, pswd

after changing in application - restart the server connection from cmd prompt

mongo client connect code

insertMany, insertOne- pass object

Decimal128.fromString(string)

insertOne(new product).then( result => {console.log(result)}).catch(err => (client.close()))

--std --- > send response in the then bloxk of the code rather than sending in the end and error msg in catch block

client.close()

get(/ ) method to fetch data

.find().forEach().then().catch() -- then return product Array

can write separate js file for initDB and getDB

module.exports = { initDb, getDb} ---creates a key

call the const in app.js

update( {\_id: req.params.id} ,{$set:udpatedProduct}) -- edit product

delete({\_id: req.params.id}

sort({}).skip({}).limit({}) -- implimenting pegination

index - sorting on price again and again. better create index -- from shell

signing users up --encryption

email, pswd ---- .hash(pw, 12) .insert .then()

createIndex({unique}) --for email

signin functionality

findOne(email).then().catch()

MongoDb stitch

serverless platform for building application

understand authorisation and permissions

react to event