**Inserting documents in collection**

* Pentru a ne conecta o la o baza de date din cluser:

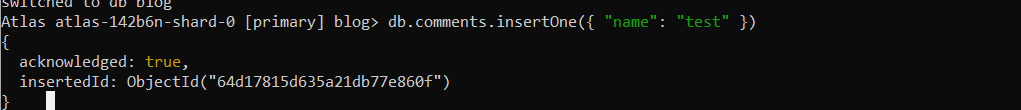
use DataBase

* **db.Collection.insertOne({“name” : “value”})**

- Il folosim asa:

db.<collection>.insertOne(“”)

- Daca primim asa ceva, inseaman ca insert s-a facut cu succes





- Daca asa colectie nu exista, se va crea una

* **db.Collection.insertMany([{"name":"test1"},{"name":"test2"}]))**

atentie la [], trimitem un array de documente

**Finding documents in a collection**

* **db.Collection.find()** – returneaza toate documentele dintr-o colectie

Daca sunt prea multe documente si nu incap, scriem **it** si se vor tot afisa mai multe

* **db.Collection.find({field1 : “value”,field2 : “value”})** – putem specifica unul sau mai multe fields, si ordinea nu conteaza

Pentru a gasi dupa id, folosim ObjectId(“”)

db.zips.find({ \_id: ObjectId("5c8eccc1caa187d17ca6ed16") })

**Fiecare operator are in fata lui {**

* **$in** – selecteaza documentele a caror field(s) au o valoarea din cele oferite in $in

**db.Collection.find({field : {$in: [“value1”,”value2”...]}})**

db.planets.find({name : {$in: ["Uranus","Mercury"]}})

**Comparator operators**

* **$gt – greater than**
* **$lt – less than**
* **$lte – less than or equals to**
* **$gte – greater than or equals to**
* **$eq – equals**
* **$ne – not equals**
* Exemplu:

db.planets.find({"surfaceTemperatureC.max" : {$gte : 200}})

Le putem si combina

db.planets.find({orderFromSun: {$gt : 0, $lte : 10}})

**Query on arrays**

* Putem folosi si un simplu query de find.
* De ex, fie ca avem asa date:

**\_id**: ObjectId('5ca4bbc7a2dd94ee5816238c')

**account\_id**: 371138

**limit**: 9000

**products**: Array

**0**: "Derivatives"

**1**: "InvestmentStock"

Vedem ca products este un Array. Acum, daca am folosi un asa query:

db.accounts.find({"products" : "Derivatives"})

chiar daca products e un array, noi am oferi un field cu o simpla valoare, si anume “Derivatives”. In asa caz, MongoDB va cauta fie un field “products” cu o singura valoare, fie un array in care valoarea “Derivatives” se gaseste, indiferent ca mai sunt si alte elemente in el

* **$elemMatch** – ne permite sa definim mai multe conditii, si in array trebuie sa fie cel putin un element care sa respecte conditiile:

{ <field>: { $elemMatch: { <query1>, <query2>, ... } } }

**\_id**: ObjectId('5ca4bbcea2dd94ee58162a68')

**username**: "fmiller"

**name**: "Elizabeth Ray"

**address**: "9286 Bethany Glens Vasqueztown, CO 22939"

**birthdate**: 1977-03-02T02:20:31.000+00:00

**email**: "arroyocolton@gmail.com"

**active**: true

**accounts**: Array

**0**: 371138

**1**: 324287

**2**: 276528

**3**: 332179

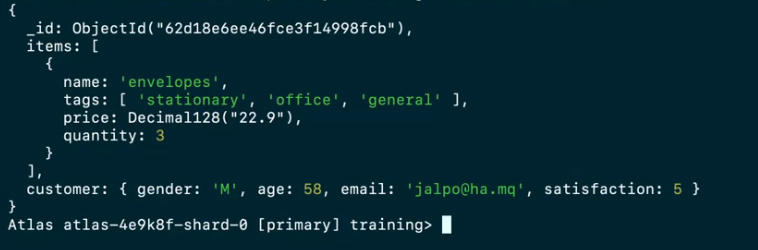
**4**: 422649

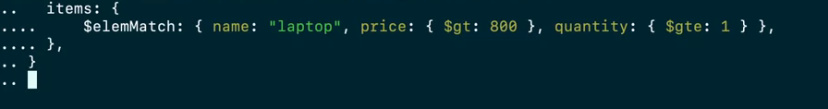
**5**: 387979

**tier\_and\_details**: Object

db.customers.find({"accounts" : {$elemMatch : {$lt : 4000000,$gt : 100000}}})

* Poate fi folosit si cand in array sunt tot obiecte:





Deci, va cauta obiectele ce au un field items cu array de obiecte,unde din array cel putin un obiect are name “laptop”, are price > 800 si si quantity >= 1

**Logical Operators**

* **$and –**

Acest operator e folosit automat chiar aici

db.Collection.find({field1 : “value”,field2 : “value”})

Cerem doar ca mai multe fielduri sa aiba anume aceste valori

Forma generala:

{ $and: [ { <expression1> }, { <expression2> } , ... , { <expressionN> } ] }

Are un array de conditii

db.companies.find({$and : [{founded\_year : {$gte : 2004}},{founded\_day : {$lte : 20}}]})

* **$or –**

{ $or: [ { <expression1> }, { <expression2> }, ... , { <expressionN> } ] }

db.companies.find({$or : [{founded\_year : {$gte : 2004}},{founded\_day : {$lte : 10}}]})

* **$not** – returneaza elementele ce nu corespund unei conditii

{ field: { $not: { <operator-expression> } } }

db.inventory.find( { price: { $not: { $gt: 1.99 } } } )

* **$nor** – returneaza elementele care nu corespund la toate conditiile

{ $nor: [ { <expression1> }, { <expression2> }, ... { <expressionN> } ] }

db.inventory.find( { $nor: [ { price: 1.99 }, { sale: true } ] } )

* Nested:

db.routes.find({

  $and: [

    { $or: [{ dst\_airport: "SEA" }, { src\_airport: "SEA" }] },

    { $or: [{ "airline.name": "American Airlines" }, { airplane: 320 }] },

  ]

})

Nu se permite folosirea la 2 de $or unul dupa altul fara $and!

**Replacing a document**

* db.collection.replaceOne(filter,replacement,options)

- filter – query pentru a arata care document il inlocuim. Putem scrie orice query ca in find()

- replacement – noile fielduri

- options este optional

**db.customers.replaceOne({\_id : ObjectId('5ca4bbcea2dd94ee58162a6a')},{email : "test@gmail.com"})**

Atentie! Daca nu punem vreun field cu o valoare noua, acel field se va sterge.

Gen, mai sus am modificat email la un client, care mai avea vreo 8 fields, dar acum, dupa replaceOne are doar 2 fields, si anume \_id si email

* Nu se va modifica id in noul document!

**UpdateOne()**

* **db.collection.updateOne(filter, update, options)**
* **$set**

- adauga noi fields si valori intr-un document

- inlocuieste valoarea unui field cu alta daca asa field exista deja

**db.grades.updateOne(**

**{\_id : ObjectId('56d5f7eb604eb380b0d8d8ce')},**

**{$set : {class\_id: 350}}**

**)**

* **$push**

- adauga o valoarea intr-un array

- daca array e absent, adauga un field cu un array cu valoarea data

**db.accounts.updateOne({\_id : ObjectId('5ca4bbc7a2dd94ee5816238e')},{$push : {products : "CurrencyService"}})**

products este un array in document

* **upsert** – se poate intampla ca nu se gaseste niciun document ca sa modifice valoarea, si poate vrem ca in asa caz sa se creeze unul cu asa valoare. Asta face upstream. Il punem la options, adica ca al 3 parametru

**db.comments.updateOne({\_id : ObjectId('64d17815d635a21db77e860c')},{$set : {name : "azazaaa"}},{upsert: true})**

**UpdateMany(filter, update)**

* Atentie, updateMany() nu merge dupa principiu all sau nothing
* Daca gen nu poate da update la un document, dar la 10 da, nu se va face rollback. Va da update la cele 10 si nu va da update la cel la care nu a reusit

**db.data.updateMany({elevation: 9999},{$set : {dataSource : 5}})**

**$regexp**

* MongoDB nu are un operator LIKE, in schimb folosesc regular expression

{name : {$regex : "^Test.\*$"}}

**findAndModify({**

**query: {},**

**update: {},**

**new: true**

**})**

* **findAndModify()** da update la un document si returneaza documentul modificat
* Deci, este ca o combinare dintre updateOne() si findOne()
* Problema cu a folosi updateOne() si apoi findOne() e ca se fac 2 queries, si apoi pana a face findOne(), alt user poate modifica datele si sa primit al rezultat, nu cel inserat de noi, deci o alta versiune la document
* findAndModify() garanteaza sa returneze versiunea corecta la document
* Contine un parametru doar, dar setam fieldurile in el, si dupa fielduri, ca query, update etc. el va face totul
* **query** – query de selectie
* **update** – noua valoare
* **new**: daca sa se returneze noua versiune a documentului sau nu
* **daca nu punem true, nu se va returna documentul modificat, ci cel care era nemodificat**

db.data.findAndModify({

query: {\_id: ObjectId('5553a998e4b02cf7151190bc')},

update: {$set : {elevation : 1}},

new : true

})

Mare grija sa fie parametrii intre {} si sa folosim $set, de altfel inlocuim documentul cu unul nou ce are doar un field si gata

**Delete documents**

* **deleteOne(filter)**

db.data.deleteOne({\_id : ObjectId('5553a998e4b02cf7151190bc')})

* **deleteMany(filter)**

**Sorting and Limiting**

* **Cursor –** pointer catre resultatul returnat de un query, deci catre una sau mai multe date
* Functiile de sorting si limiting se folosesc dupa ce se returneaza obiectele, gen cu findOne(), find() etc. caci ele returneaza pointeri catre documents
* **Cu cursor,** mai putem efectua operatii asupra la datele returnate
* **.sort({field1 : 1/-1, field2 : 1/-1}) –** fieldname va spune cum vrem sa sortam datele dupa acest field anume. 1 inseamna ascendentm -1 descendend

**db.planets.find().sort({orderFromSun: 1})**

* **.limit(nr)**

db.planets.find().limit(3)

**Projection**

* In MongoDB, orice query de find returneaza toate fields din document(e)
* Prin projection, putem specifica strict care fields sa fie returnate
* Tot ce trebuie sa facem este sa adaugam la find() sau findOne() un al 2 parametru pentru a spune care fields sa fie afisate sau care nu:

db.planets.findOne({\_id : ObjectId('621ff30d2a3e781873fcb65c')},{name : 1, orderFromSun: 1})

Vedem ca punem 1 daca vrem sa fie afisat, si 0 daca nu

* Chiar daca nu am cerut asta, \_id oricum mereu e default afisat. Pentru a-l exclude, doar punem \_id : 0

db.planets.findOne({\_id : ObjectId('621ff30d2a3e781873fcb65c')},{name : 1, orderFromSun: 1,\_id : 0})

* Putem de ex sa facem sa nu fie aratat un field ce se gaseste in alt field din document:

db.planets.findOne({\_id : ObjectId('621ff30d2a3e781873fcb65c')},{"surfaceTemperatureC.min" : 0})

Mare atentie sa punem “” cand folosim .(cand accesam field din alt field)

* Atentie! Nu putem adauga si exclude fields in acelasi timp:

db.planets.findOne({\_id : ObjectId('621ff30d2a3e781873fcb65c')},{"surfaceTemperatureC.min" : 0,name : 1})

**ERROR!!!!**

\_id e exceptie! El mereu poate fi exclus chiar si cand alegem fielduri

* Daca gen nu oferim un query de filter, in locul lui punem doar {} si gata

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

asa se vor selecta toate documents, dar numai cu 2 fields

* Daca includem fields, doar ele vor fi incluse in rezultatul final, si restul nu. Daca excludem fields, ele nu vor fi afisate, dar restul da.

**CountDocuments(<query>,<options>)**

* **query** – e query pentru a selecta documentele pe baza la un query. Putem sa il si lasam gol si se vor socoti toate
* **options** – e foarte rar folosit
* db.planets.countDocuments() – socotim toate documentele din planets data base
* db.planets.countDocuments({hasRings: false}) – socotim doar documentele ce corespund la query