

Introdução ao MongoDB



Prof. Dr. Vladimir Costa Alencar

vladimir.uepb@gmail.com

www.valencar.com

Introdução ao MongoDB



MongoDB é um Sistema Gerenciador de Banco de Dados Multi-plataforma orientado a documentos

Livre de esquemas (NoSQL)

Possui Alta Performance

Fácil Escalabilidade

Introdução ao MongoDB



Escrito em C++

Trabalha com o conceito de Coleção e Documento

Lida com documentos e não com registros como no modelo relacional

Introdução ao MongoDB



Um registro (tupla) em Mongo DB é um documento

A sua estrutura é composta de campos e valores.

Um documento é similar ao objeto JSON.

Os valores dos campos podem incluir outro documentos, arrays e arrays de documentos.

Introdução ao MongoDB



Document database

```
{  
  name: "sue",  
  age: 26,  
  status: "A",  
  groups: [ "news", "sports" ]  
}
```

← field: value
← field: value
← field: value
← field: value

Introdução ao MongoDB



Database é um container físico de collections.

Cada database tem o seu conjunto de arquivos no file system.

Um simples servidor MongoDB possui vários databases.



Introdução ao MongoDB

Collection é um grupo de documentos MongoDB

É equivalente a uma **tabela** num SGBD relacional

A collection existe dentro do banco de dados

Collections não exigem um esquema de dados

Documentos dentro de uma collection podem ter diferentes campos



Introdução ao MongoDB

Um documento é um conjunto de pares chave-valor

Um documento tem um schema dinâmico

Schema dinâmico significa que documentos da mesma collection não precisam ter o mesmo conjunto de atributos ou estrutura

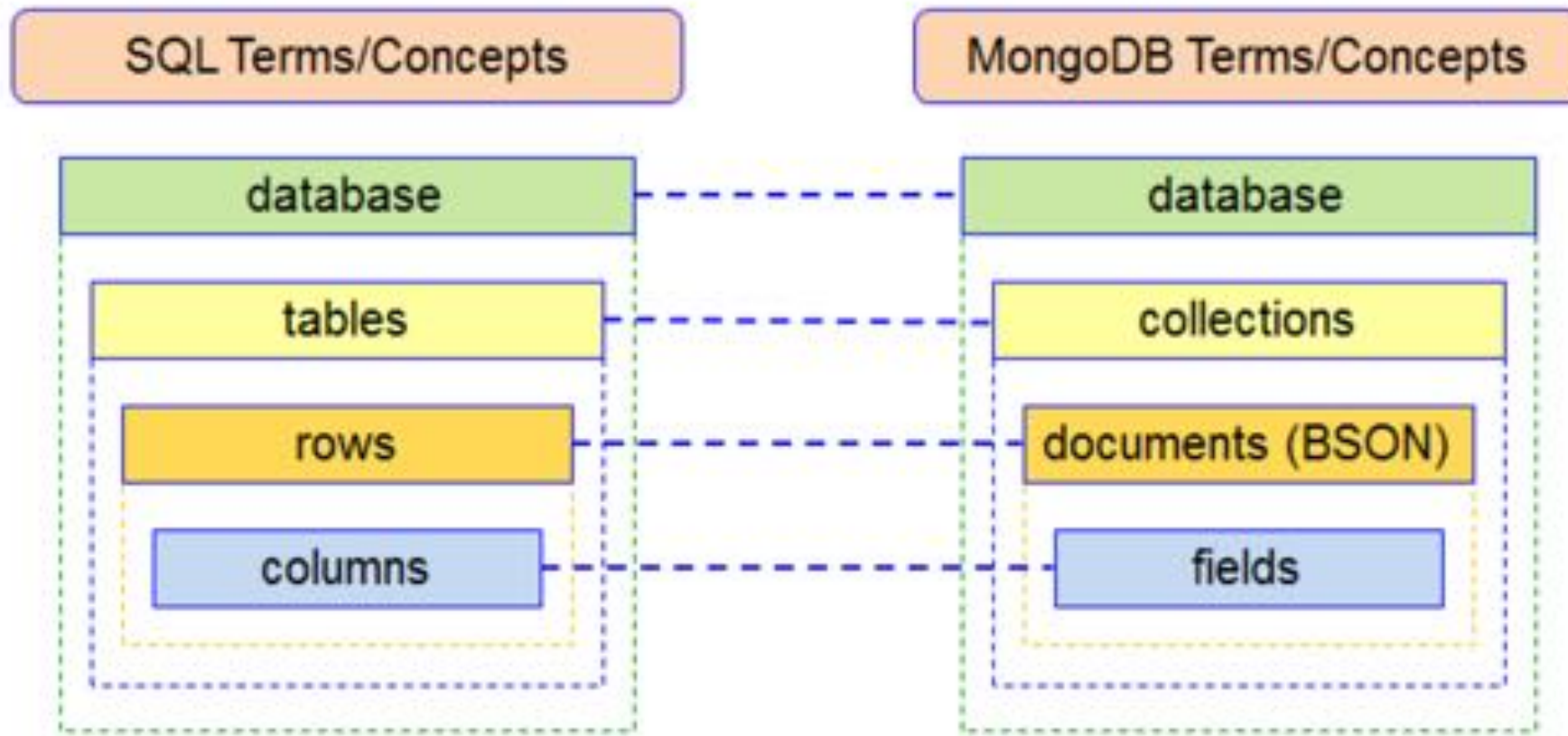
Cada documento pode ter diferentes tipos de dados.

SGDB Relacional X NoSQL - Analogia



RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by mongodb itself)

SGDB Relacional X NoSQL - Analogia



SGDB Relacional X NoSQL - Analogia



Database: RH

Collection: Funcionários (analogia a Tabela)

Documents (analogia a tupla, registro)

Documento 2

```
{
  "nome": "JOAO",
  "sobrenome": "SAMPAIO",
  "idade": "23"
}
```

Documento 2

```
{
  "nome": "IVAN",
  "sobrenome": "QUEIROZ",
  "idade": "35",
  "site": "http://www.ivanqueiroz.com"
}
```

SGDB Relacional X NoSQL - Analogia

SQL VS NoSQL Queries

NoSQL Query:

```
db.users.find(  
  { age: { $gt: 18 } },  
  { name: 1, address: 1 }  
) .limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

SQL Query:

```
SELECT _id, name, address  
FROM users  
WHERE age > 18  
LIMIT 5
```

← projection
← table
← select criteria
← cursor modifier



Vantagens de usar documentos

Documentos (i.e. objetos) correspondem a tipos de dados nativos em várias linguagens de programação

Documentos incorporados e arrays reduzem a necessidade de muitos joins

Um esquema dinâmico suporta polimorfismo

Características principais

Alta Performance



MongoDB proporciona alta performance na persistência dos dados.

Suporta modelos de dados encapsulados reduzindo a atividade de Entrada/Saída (I/O) no SGBD

Índices suportam consultas rápidas e podem incluir chaves de documentos incorporados e arrays

Características principais

Linguagem de consulta com recursos



MongoDB suporta:

Operações de leitura e escrita (CRUD)

Agregação de dados

Busca Textual

Queries Geospacial

Características principais

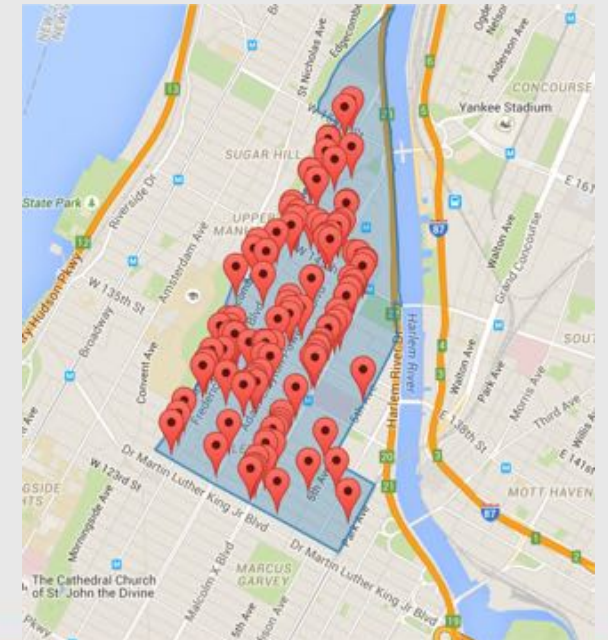
Query espacial



```
var neighborhood = db.neighborhoods.findOne( { geometry: { $geoIntersects: {  
$geometry: { type: "Point", coordinates: [ -73.93414657, 40.82302903 ] } } } } )
```

```
db.restaurants.find( { location: { $geoWithin: { $geometry:  
neighborhood.geometry } } } ).count()
```

Seleção de restaurantes próximos de
Um cliente



Características principais

Alta Disponibilidade



Recuperação de falhas automática

Redundância de dados

Um replica set é um grupo de servidores MongoDB que armazenam o mesmo dataset, permitindo redundância e aumentando a disponibilidade do dados

Escalabilidade Horizontal



A técnica de Sharding (fragmentar) distribui o dado através de máquinas do cluster

MongoDB suporta a criação de Zonas de dados baseados em em shard key (chave para acessar dados distribuídos em cluster)

MongoDB e Big Data



MongoDB foi criado com Big Data em mente.

Ele suporta tanto escalonamento horizontal quanto vertical usando replica sets (instâncias espelhadas) e sharding (dados distribuídos), tornando-o uma opção muito interessante para grandes volumes de dados, especialmente os desestruturados.

MongoDB e Big Data



Dados desestruturados são um problema para a imensa maioria dos bancos de dados relacionais, mas não tanto para o MongoDB.

Quando o seu schema é variável, é livre, usar MongoDB vem muito bem a calhar.

Os documentos BSON (JSON binário) do Mongo são schemaless e aceitam quase qualquer coisa que você quiser armazenar

MongoDB – Utilização



Big Data

Escrita Intensa (sensores, redes sociais, etc)

Buscas simples, porém pesada (muitos dados)

Alta escala e disponibilidade

Schema flexível

Utilizando o MongoDB - Aplicativos



mongod: inicializa o servidor de banco de dados

mongo: inicializa o cliente de banco de dados

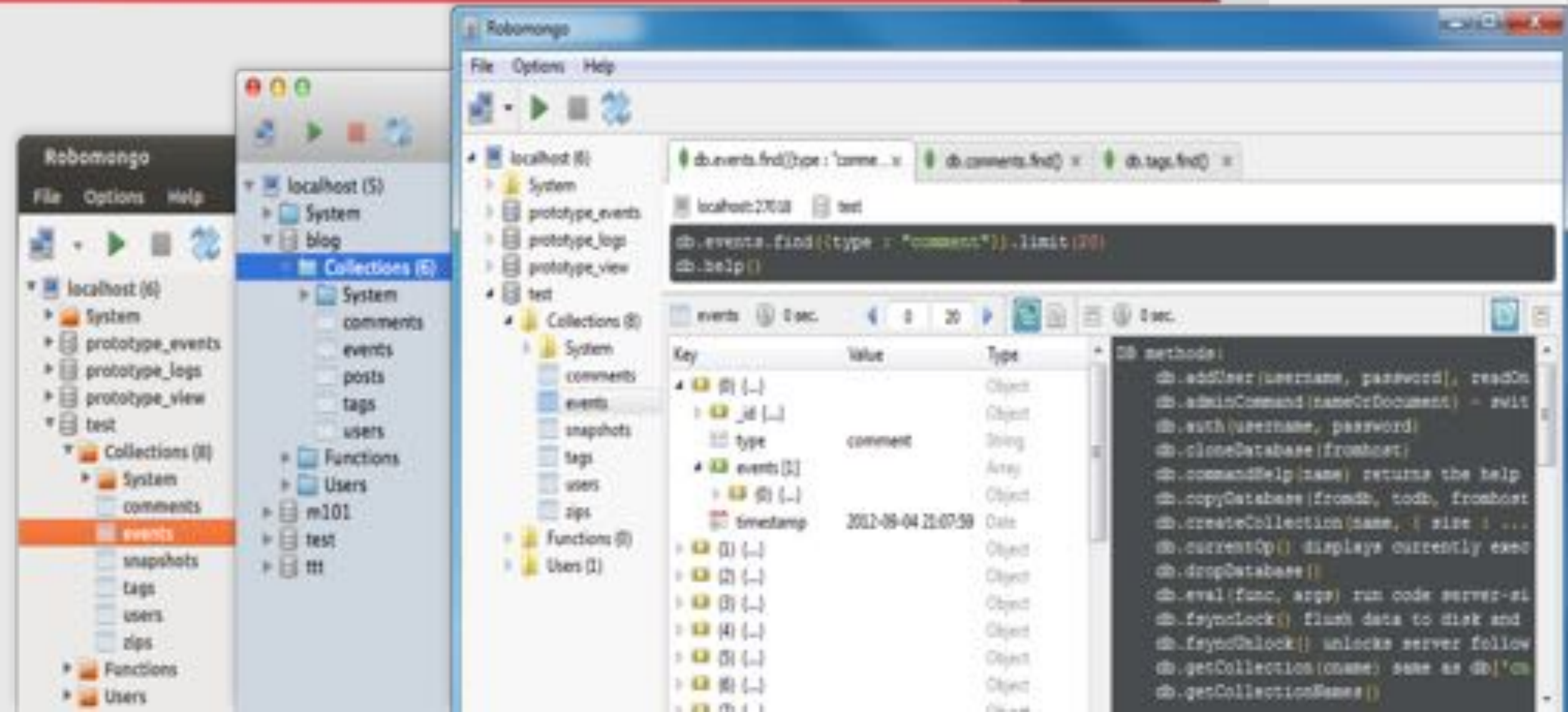
mongodump: realiza dump do banco (backup binário)

mongorestore: restaura dumps do banco (restore binário)

mongoimport: importa documentos JSON ou CSV pro seu banco

mongoexport: exporta documentos JSON ou CSV do seu banco

Robomongo - Interface Gráfica



Utilizando o MongoDB



Ativar o processo (daemon) do mongod

```
C:\mongodb\bin>mongod.exe --dbpath "\\mongodb\data"
```

Ativar o mongo

```
C:\mongodb\bin>mongo.exe
```

Ou usar o RoboMongo (interface gráfica)

Tipos de Dados - MongoDB



- **String** – This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** – This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** – This type is used to store a boolean (true/ false) value.
- **Double** – This type is used to store floating point values.
- **Arrays** – This type is used to store arrays or list or multiple values into one key.
- **Timestamp** – ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** – This datatype is used for embedded documents.
- **Null** – This type is used to store a Null value.

Tipos de Dados - MongoDB



- **Date** – This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** – This datatype is used to store the document's ID.
- **Binary data** – This datatype is used to store binary data.
- **Code** – This datatype is used to store JavaScript code into the document.
- **Regular expression** – This datatype is used to store regular expression.

Utilizando o MongoDB



> use DBprodutos -> cria o banco de dados DBprodutos,
ou retorna Dbprodutos, se já foi criado

>show dbs

admin	0.000GB
dbprodutos	0.000GB
local	0.000GB
test	0.000GB

criando uma collection



- > use DBprodutos
- > db.createCollection("DBProdutos")
- > show collections

clientes

products

- > Apagar uma collection
- > db.DBprodutos.drop()

Inserindo dados



Cria a **collection** clientes e insere o **documento** ("Vladimir", 42)

- `db.clientes.insertOne({ Nome: "Vladimir", Idade: 42})`
- `db.clientes.insertMany([`
 - `{ Nome: "Sarah", Idade: 19},`
 - `{ Nome: "Gabriel", sexo: "m"},`
 - `{ Nome: "Paula", Idade: 25, sexo: "f"}``])`



Lista todos os dados da collection

```
db.clientes.find()
{  "_id" : ObjectId("5aa03a06f8b976253147b10f"),
  "Nome" : "Vladimir",  "Idade" : 42.0}

{  "_id" : ObjectId("5aa03ce8f8b976253147b110"),
  "Nome" : "Sarah",  "Idade" : 19.0}

{  "_id" : ObjectId("5aa03ce8f8b976253147b111"),
  "Nome" : "Gabriel",  "sexo" : "m"}

{  "_id" : ObjectId("5aa03ce8f8b976253147b112"),
  "Nome" : "Paula",  "Idade" : 25.0,  "sexo" : "f"}
```


Operadores



operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:<value>}	db.mycol.find({"nome": "Computador"})	where nome = 'computador'
Less Than	{<key>:{\$lt:<value>}}	db.mycol.find({"likes": {\$lt:50}})	where likes < 50
Less Than Equals	{<key>:{\$lte:<value>}}	db.mycol.find({"likes": {\$lte:50}})	where likes <= 50
Greater Than	{<key>:{\$gt:<value>}}	db.mycol.find({"likes": {\$gt:50}})	where likes > 50
Greater Than Equals	{<key>:{\$gte:<value>}}	db.mycol.find({"likes": {\$gte:50}})	where likes >= 50
Not Equals	{<key>:{\$ne:<value>}}	db.mycol.find({"likes": {\$ne:50}})	where likes != 50

Operador And



```
>db.mycol.find( { $and:  
    [ {key1:value1}, {key2:value2} ]  
} )
```

```
>db.clientes.find({ $and: [ {Nome: "Vladimir"}, {Idade: 42.0} ] } )  
  
{ "_id" :  
  ObjectId("5aa03a06f8b976253147b10f"),  
  "Nome" : "Vladimir",  "Idade" : 42.0 }
```

Operador OR



```
>db.mycol.find( { $or:  
    [ {key1:value1}, {key2:value2} ]  
} )
```

```
>db.clientes.find({ $or: [ {Nome: "Vladimir"}, {Idade: 19.0} ] } )
```

```
{  "_id": ObjectId("5aa03a06f8b976253147b10f"),  
  "Nome" : "Vladimir",  "Idade" : 42.0}  
{  "_id" : ObjectId("5aa03ce8f8b976253147b110"),  
  "Nome" : "Sarah",  "Idade" : 19.0}
```

Update



```
>db.COLLECTION_NAME.update(SELECTION_CRITERIA,  
UPDATED_DATA)
```

```
>db.clientes.update( {Nome: "Paula"},  
{ $set : { Nome: "Ana Paula"} } )
```

Remove



```
>db.COLLECTION_NAME.remove(DELETION_CRITTERIA)
```

```
>db.clientes.remove({Nome: "Pedro"})
```

Caracteres Curinga



```
>db.clientes.find({Nome: /^Pa/}) ==>//like 'pa%'
```

Paulo

Patric

```
>db.clientes.find({Nome: /ro$/}) ==>//like '%ro'
```

Pedro

Expressões Regulares



```
db.collection.find({'name': {'$regex': 'sometext'}})
```

```
db.clientes.find({'Nome': {'$regex': 'vlad*', '$options': 'i'}})
```

Vladimir



Ignore Case

Projeção



```
>db.COLLECTION_NAME.find({}, {KEY:1})
```

```
db.clientes.find({}, {_id:0, Nome:1 })
```

```
{ "Nome" : "Vladimir"}  
{ "Nome" : "Sarah"}  
{ "Nome" : "Gabriel"}  
{ "Nome" : "Ana Paula"}  
{ "Nome" : "Paulo"}  
{ "Nome" : "Patric"}  
{ "Nome" : "Pedro"}
```

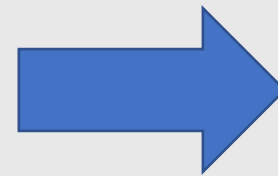



Aggregate (Group By do SQL)

```
db.clientes.aggregate( [ {  
  $group: {  
    _id: null,  
    Media_Idade: { $avg: "$Idade" }  
  } } ] )  
  
{  "_id" : null,  "Medialdade" : 28.666666666666667 }
```

Aggregate (Group By do SQL)

```
db.clientes.aggregate( [  
  {  
    $match: {  
      Idade: {  
        $exists: true    }  
      } },  
  {  
    $group: {  
      _id: null,  
      Medialdade: { $avg: "$Idade" }  
    } }  
])
```



```
{  "_id" : null,  
  "Medialdade" : 28.66666666666667}
```

Outras funções



```
db.getCollection('febre amarela').renameCollection("febre_amarela")
```

Backup



MongoDump -> faz o backup de todo o banco de dados
(coloca na pasta dump os databases)

```
mongodump --db twitter --db twitter --collection clientes  
--out /data/dumps
```

MongoRestore -> traz o backup para o MongoDB

```
restaurar bd: mongorestore --db twitter path_to_bson_file
```

Backup



```
mongoimport --stopOnError --db loja --collection clientes  
    < clientes.json
```

outras funções



<< ordenação >>

```
db.clientes.find ( { uf : { $exists : 1 } }, { _id : 0, nome : 1, uf : 1 } ).sort ( { uf : 1 } )
```

<< ordenação composta e decrescente >>

```
db.clientes.find ( { uf : { $exists : 1 } }, { _id : 0, nome : 1, uf : 1 } ).sort ( { uf : 1, nome : -1 } )
```

<< limit >>

```
db.clientes.find ( {}, { _id : 0, nome : 1, grauFidelidade : 1 } ).sort ( { grauFidelidade : -1 } ).limit (3)
```

<< skip: mostramos a partir do sexto >>

```
db.clientes.find ( { grauFidelidade : { $exists : 1 } },  
{ _id : 0, nome : 1, grauFidelidade : 1 } ).sort ( { grauFidelidade : -1 } ).skip(5)
```

outras funções



< seleção; usando primeiro argumento >>

```
db.clientes.find ( {uf : "RJ" }, { _id : 0, nome : 1, uf : 1 } )
```

<< seleção: \$or; ressaltar a presença do vetor >>

```
db.clientes.find ( { $or : [ {uf : "ES" }, {uf : "MG" } ] }, { _id : 0, nome : 1, uf : 1 } )
```

<< seleção: \$gte >>

```
db.clientes.find ( { grauFidelidade : { $gte : 50 } }, { _id : 0, nome : 1, grauFidelidade : 1 } )
```

<< seleção: \$and ; pressionar ENTER depois da projeção >>

```
db.clientes.find ( { $and : [ { grauFidelidade : { $gte : 20 } }, { grauFidelidade : { $lte : 80 } } ] },  
                  { _id : 0, nome : 1, grauFidelidade : 1 } )
```

<< seleção: \$exists >>

```
db.clientes.find ( { uf : { $exists : 1 } }, { nome : 1, uf : 1 } ).count()
```

<< seleção: busca em um vetor >>

```
db.clientes.find ( { "dependentes.nome" : "Francisco" }, { _id : 0, nome : 1 , dependentes : 1 } )
```

<< melhorando a saída com pretty>>

```
db.clientes.find ( { "dependentes.grau" : "e" }, { _id : 0, nome : 1 , dependentes : 1 } ).pretty()
```

outras funções



< seleção; usando primeiro argumento >>

```
db.clientes.find ( {uf : "RJ" }, { _id : 0, nome : 1, uf : 1 } )
```

<< seleção: \$or; ressaltar a presença do vetor >>

```
db.clientes.find ( { $or : [ {uf : "ES" }, {uf : "MG" } ] }, { _id : 0, nome : 1, uf : 1 } )
```

<< seleção: \$gte >>

```
db.clientes.find ( { grauFidelidade : { $gte : 50 } }, { _id : 0, nome : 1, grauFidelidade : 1 } )
```

<< seleção: \$and ; pressionar ENTER depois da projeção >>

```
db.clientes.find ( { $and : [ { grauFidelidade : { $gte : 20 } }, { grauFidelidade : { $lte : 80 } } ] },  
                  { _id : 0, nome : 1, grauFidelidade : 1 } )
```

<< seleção: \$exists >>

```
db.clientes.find ( { uf : { $exists : 1 } }, { nome : 1, uf : 1 } ).count()
```

<< seleção: busca em um vetor >>

```
db.clientes.find ( { "dependentes.nome" : "Francisco" }, { _id : 0, nome : 1 , dependentes : 1 } )
```

<< melhorando a saída com pretty>>

```
db.clientes.find ( { "dependentes.grau" : "e" }, { _id : 0, nome : 1 , dependentes : 1 } ).pretty()
```