MongoDB

Eric Lau de Oliveira

NoSQL: Banco de dados não relacionais

Mongo vem da expressão em inglês “Humongous”.

Significados: Gigantesco, Monstruoso.

11-03-2018

Índice

[NoSQL: 4](#_Toc509142077)

[O que é um documento? 4](#_Toc509142078)

[Download MongoDB: 4](#_Toc509142079)

[Instalação: 5](#_Toc509142080)

[Terminal de comandos. 7](#_Toc509142081)

[Comandos básicos: 7](#_Toc509142082)

[Criando uma base de dados. 7](#_Toc509142083)

[Criando uma coleção: 8](#_Toc509142084)

[Visualizando os dados de uma coleção: 8](#_Toc509142085)

[Visualizar todas as bases de dados: 9](#_Toc509142086)

[Selecionar uma base de dados: 9](#_Toc509142087)

[Mostrar todas as coleções. 10](#_Toc509142088)

[Visualizar o stats de uma base de dados: 10](#_Toc509142089)

[Visualizando stats de uma coleção: 11](#_Toc509142090)

[DATE. 12](#_Toc509142091)

[PRETTY. 12](#_Toc509142092)

[OBJETOS E VETORES. 13](#_Toc509142093)

[CRUD (Create, Read, Update, Delete) 15](#_Toc509142094)

[CREATE. 16](#_Toc509142095)

[UPDATE. 18](#_Toc509142096)

[DELETE. 24](#_Toc509142097)

[READ. 25](#_Toc509142098)

[Operadores Lógicos: 27](#_Toc509142099)

[$and: 27](#_Toc509142100)

[$or: 28](#_Toc509142101)

[$exists: 29](#_Toc509142102)

[$not: 30](#_Toc509142103)

[$nor: 31](#_Toc509142104)

[Operadores de Comparação: 31](#_Toc509142105)

[$eq: 31](#_Toc509142106)

[$ne: 32](#_Toc509142107)

[Operadores de Maior e Menor que. 33](#_Toc509142108)

[$gt: 33](#_Toc509142109)

[$lt: 33](#_Toc509142110)

[$lte: 34](#_Toc509142111)

[$gte: 35](#_Toc509142112)

[Projections. 35](#_Toc509142113)

[Visualizar atributos específicos: 35](#_Toc509142114)

[Como omitir colunas: 36](#_Toc509142115)

[Buscas Ordenadas: 37](#_Toc509142116)

[LIMIT. 39](#_Toc509142117)

[SKIP. 40](#_Toc509142118)

[Busca por palavras específicas: 42](#_Toc509142119)

[$regex: 42](#_Toc509142120)

[COUNT. 43](#_Toc509142121)

[Associações. 44](#_Toc509142122)

[Fim. 48](#_Toc509142123)

# **NoSQL:**

É um banco de dados não relacional.

Trabalha com coleções de documentos.

NoSQL são diferentes sistemas de armazenamento que vieram para suprir necessidades em demandas onde os bancos de dados tradicionais (relacionais) são ineficazes. Muitas dessas bases apresentam características muito interessantes como altas performances, escalabilidade, replicação, suporte a dados estruturados e sub colunas.

No caso dos bancos NoSQL, toda a informação necessária estará agrupada no mesmo registro, ou seja, em vez de você ter o relacionamento entre várias tabelas para formar uma informação, ela estará em sua totalidade no mesmo registro.

Artigo completo em:

<https://imasters.com.br/artigo/17043/banco-de-dados/nosql-voce-realmente-sabe-do-que-estamos-falando/>

## O que é um documento?

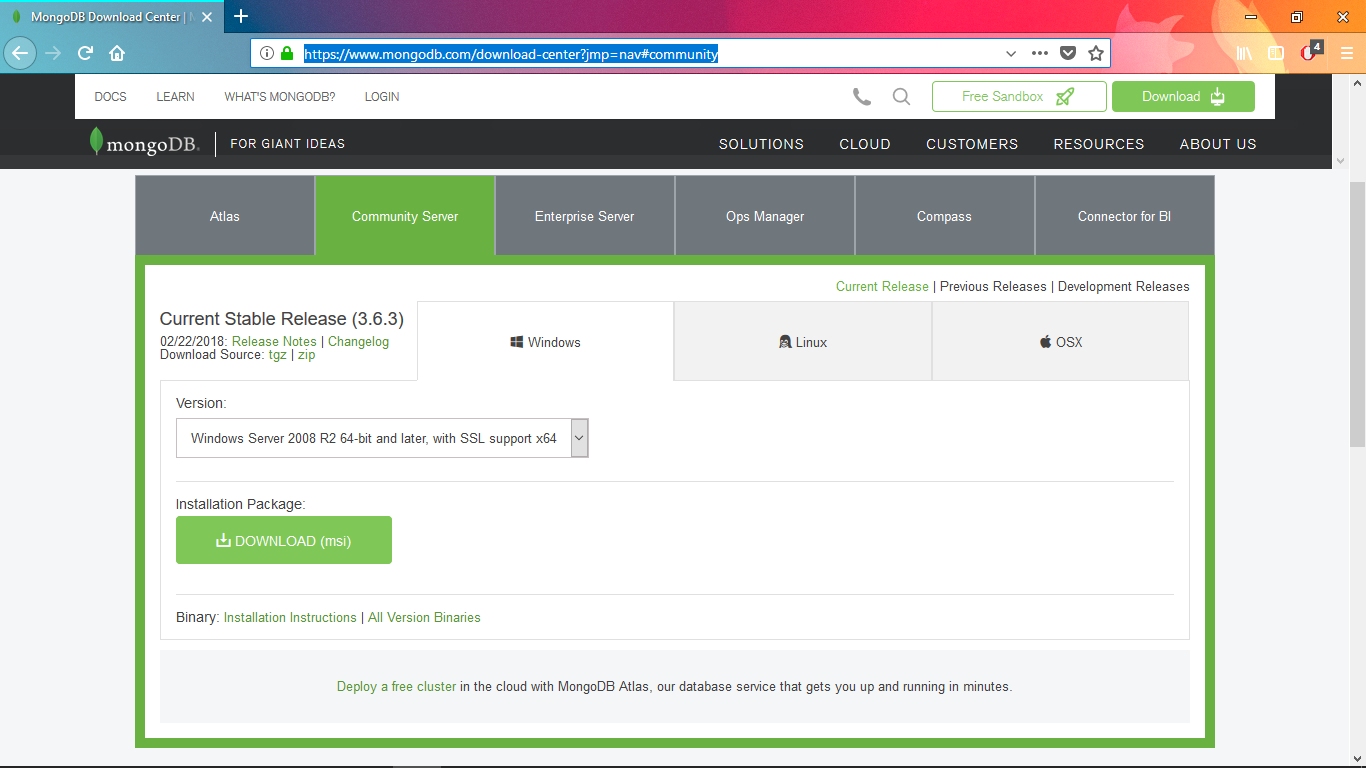
Um documento é um conjunto de chaves e valores. Documentos tem esquema dinâmico. Esquema dinâmico significa que os documentos na mesma coleção não precisam ter o mesmo conjunto de campos ou estrutura de campos comuns em documentos de uma coleção podendo conter diferentes tipos de dados.

Artigo completo:

https://mongodbwise.wordpress.com/2014/05/22/mongodb-guia-rapido/

## Download MongoDB:

<https://www.mongodb.com/download-center?jmp=nav#community>



## Instalação:

Após a instalação é necessário criar os seguintes diretórios:

Acesse o Disco Local C.

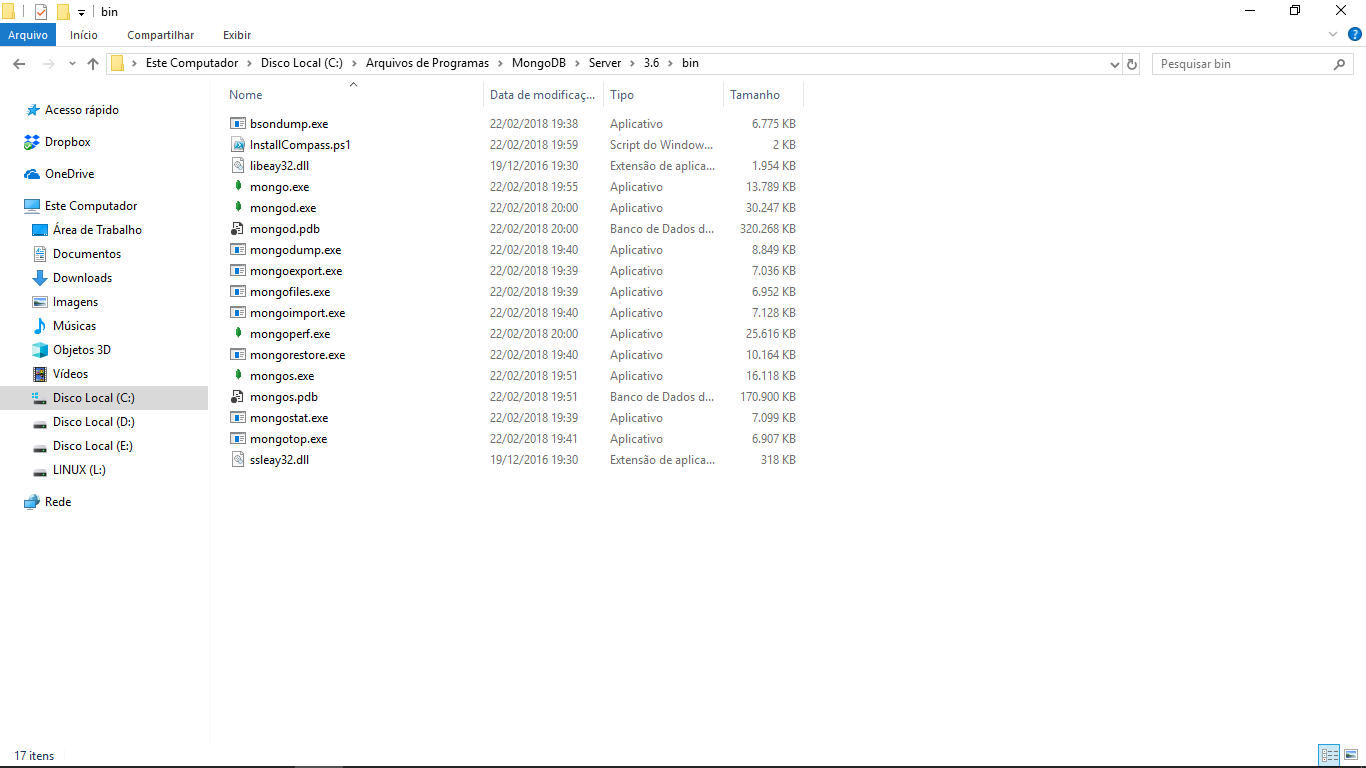
Crie uma pasta chamada “data”.

Dentro da pasta da “data” crie outra chamada “db”.

Feito isso...

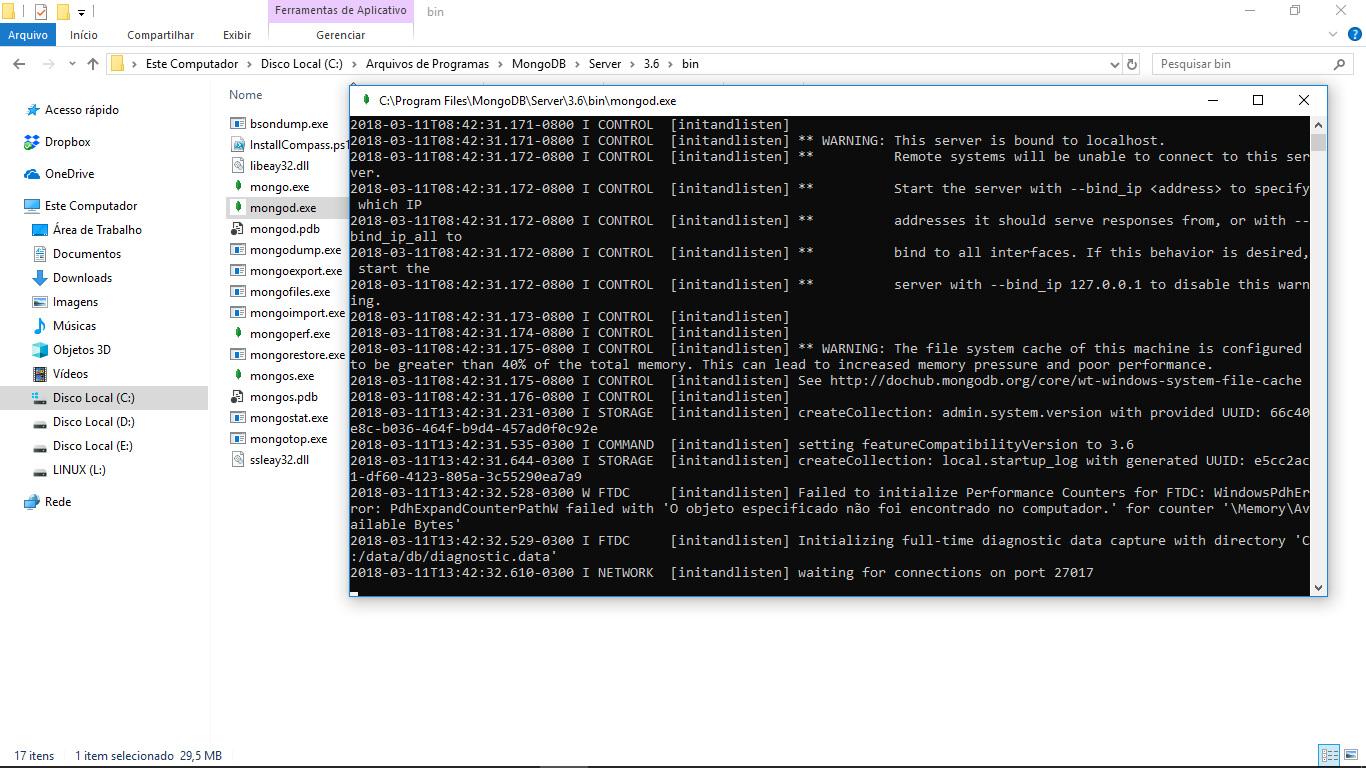
Agora será necessário ligar o servidor do mongodb.

Acesse a pasta de instalação.



Clique em “mongod.exe”.

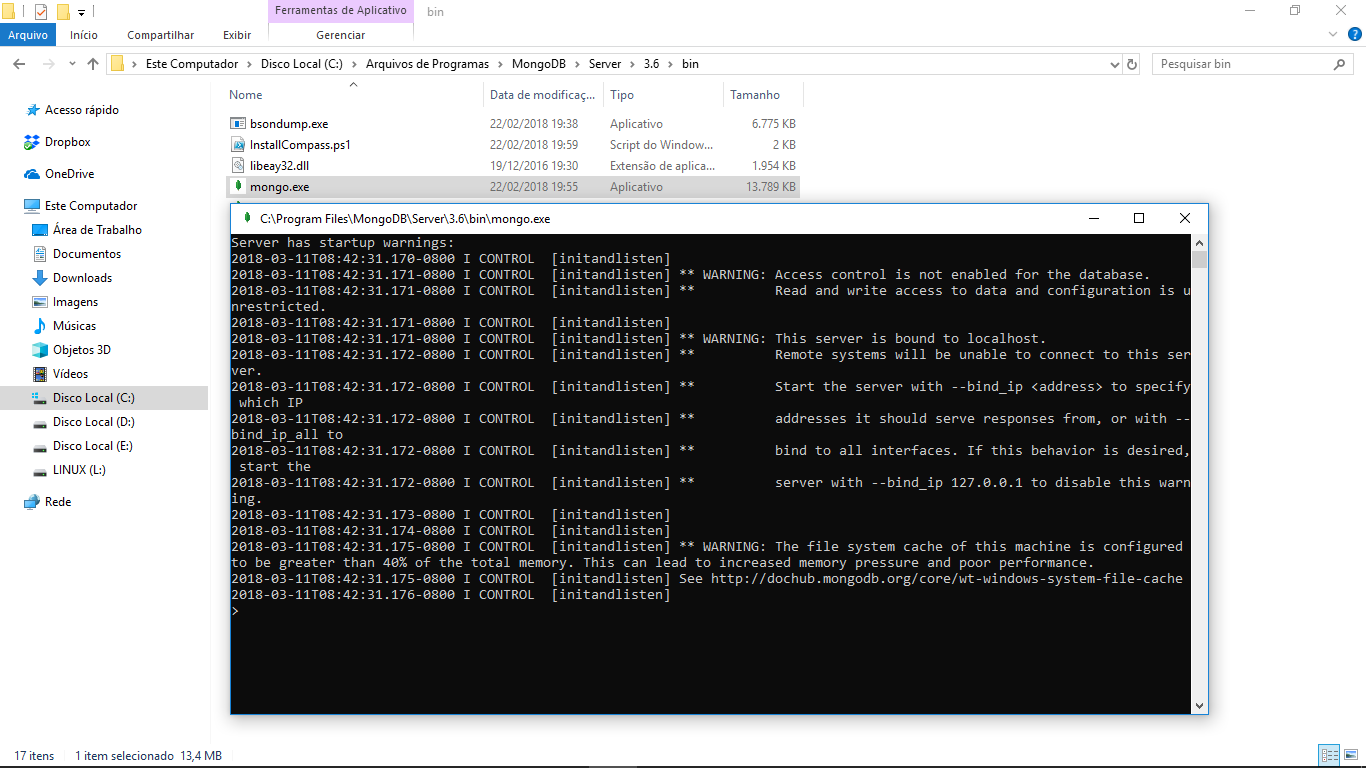
Servidor Start...



Caso este terminal seja fechado os comandos não poderão ser executados, pois o servidor é desligado.

Minimize a janela para abrir o novo terminal, onde será executado os comandos.

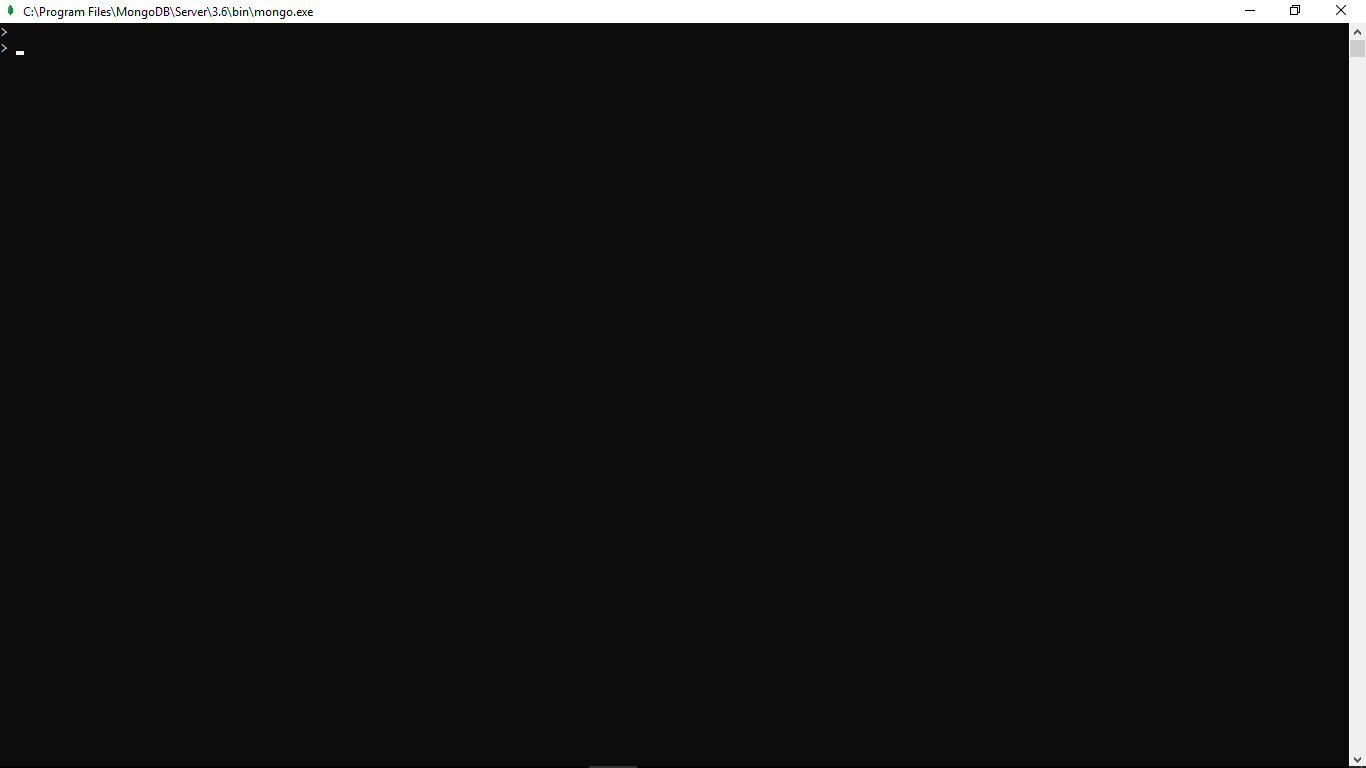
Clique em “mongo.exe”



# Terminal de comandos.

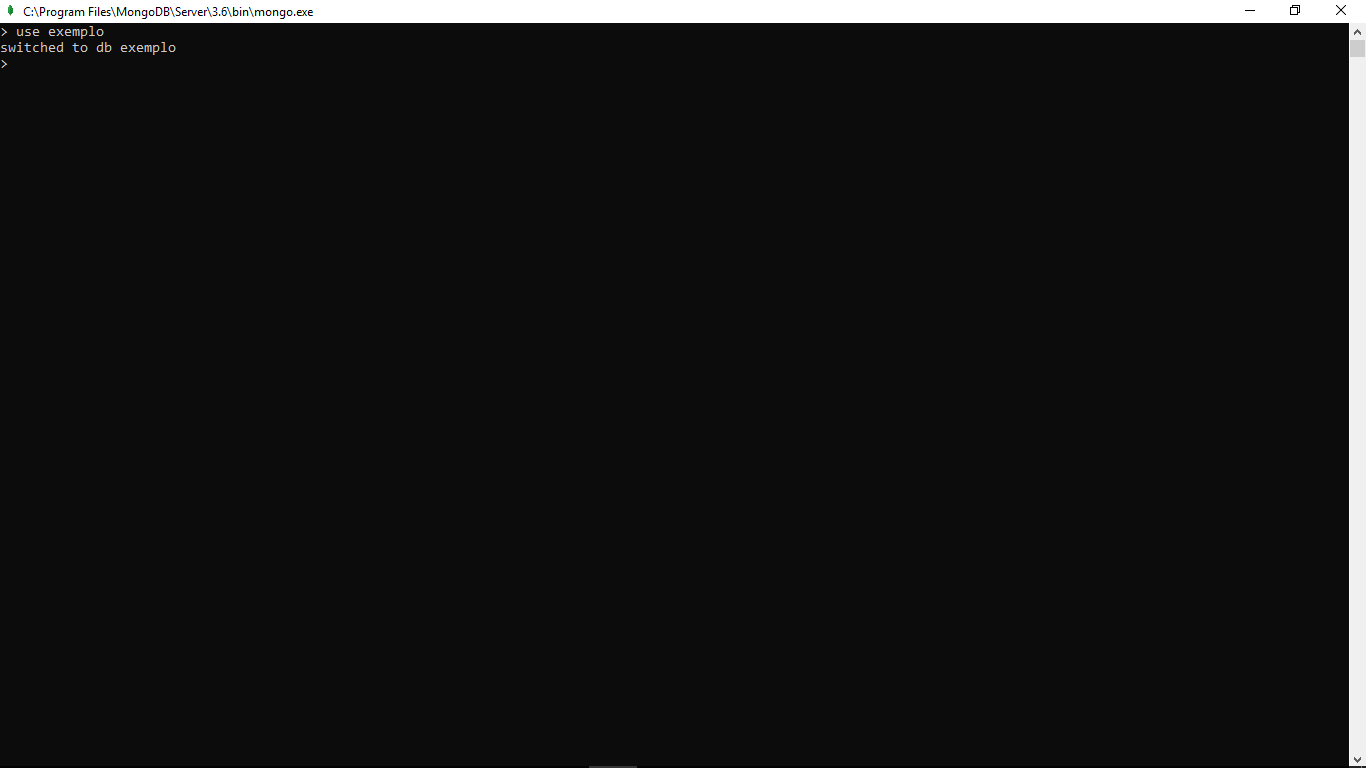
## Comandos básicos:

* *cls* : limpa a tela.



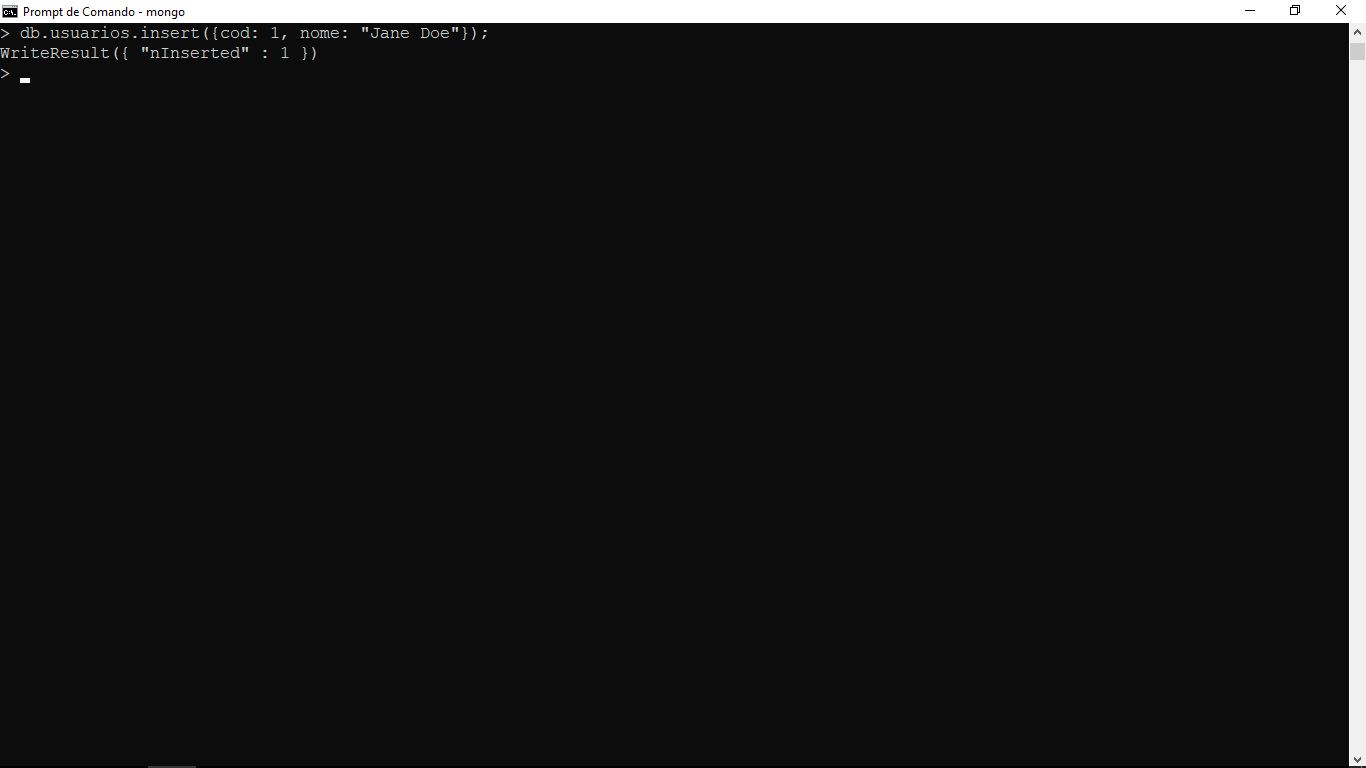
## Criando uma base de dados.

* use exemplo



## Criando uma coleção:

* db.usuarios.insert({cod: 1, nome: “Jane Doe”});



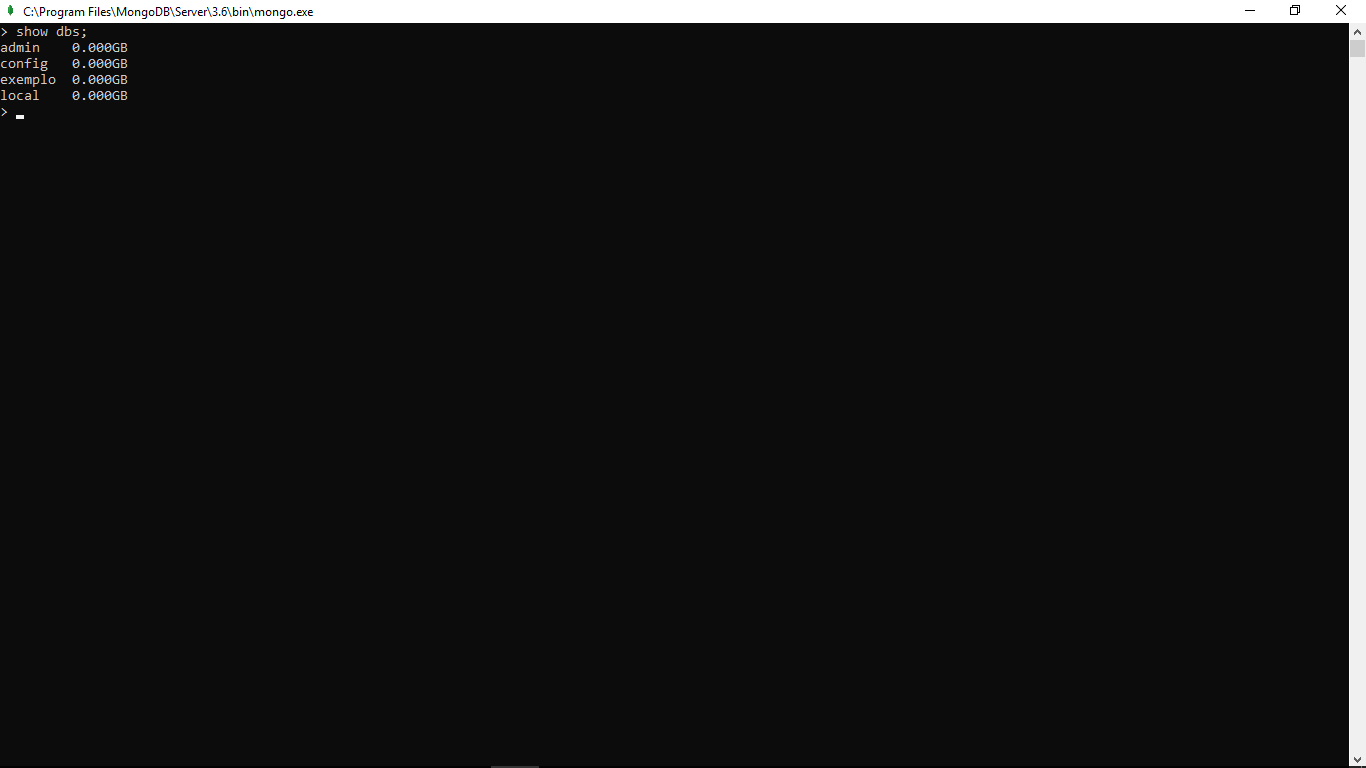
## Visualizando os dados de uma coleção:

* db.usuarios.find();



## Visualizar todas as bases de dados:

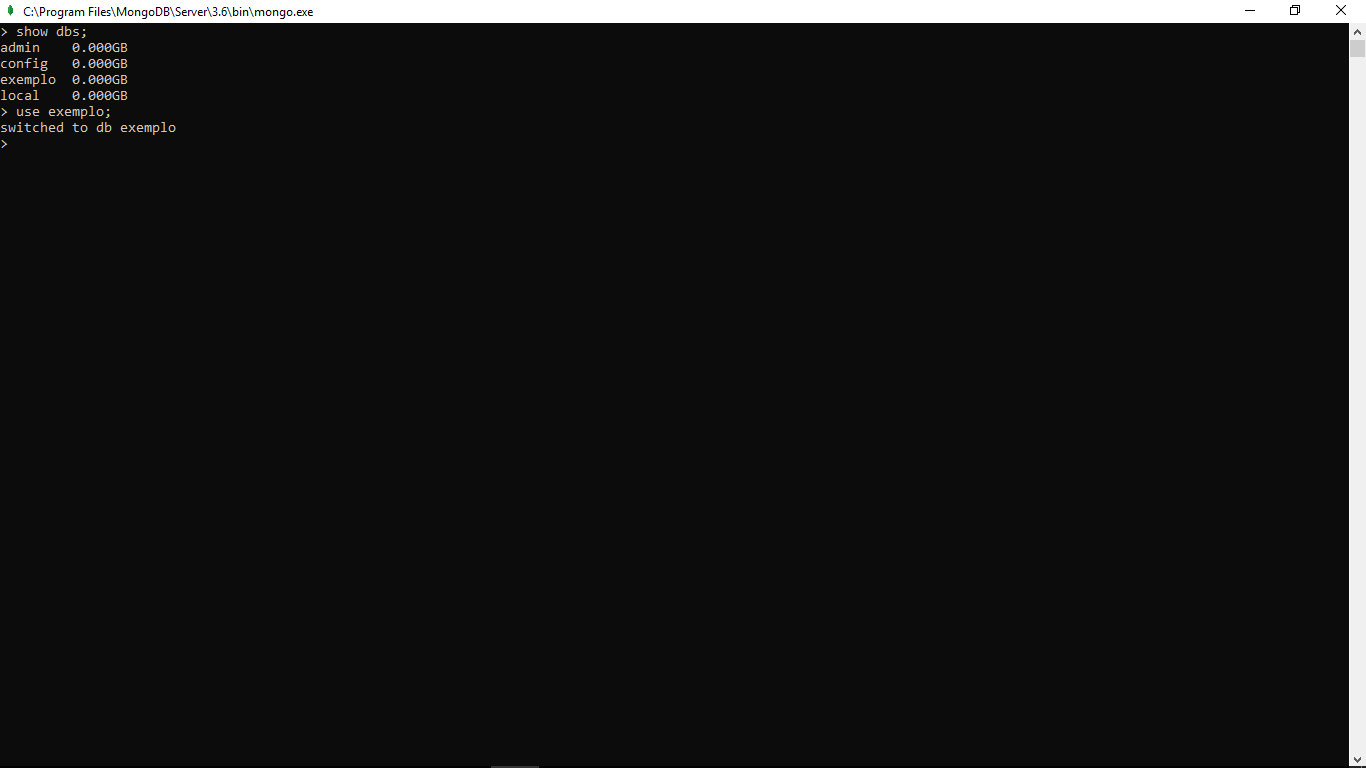
* show dbs;



## Selecionar uma base de dados:

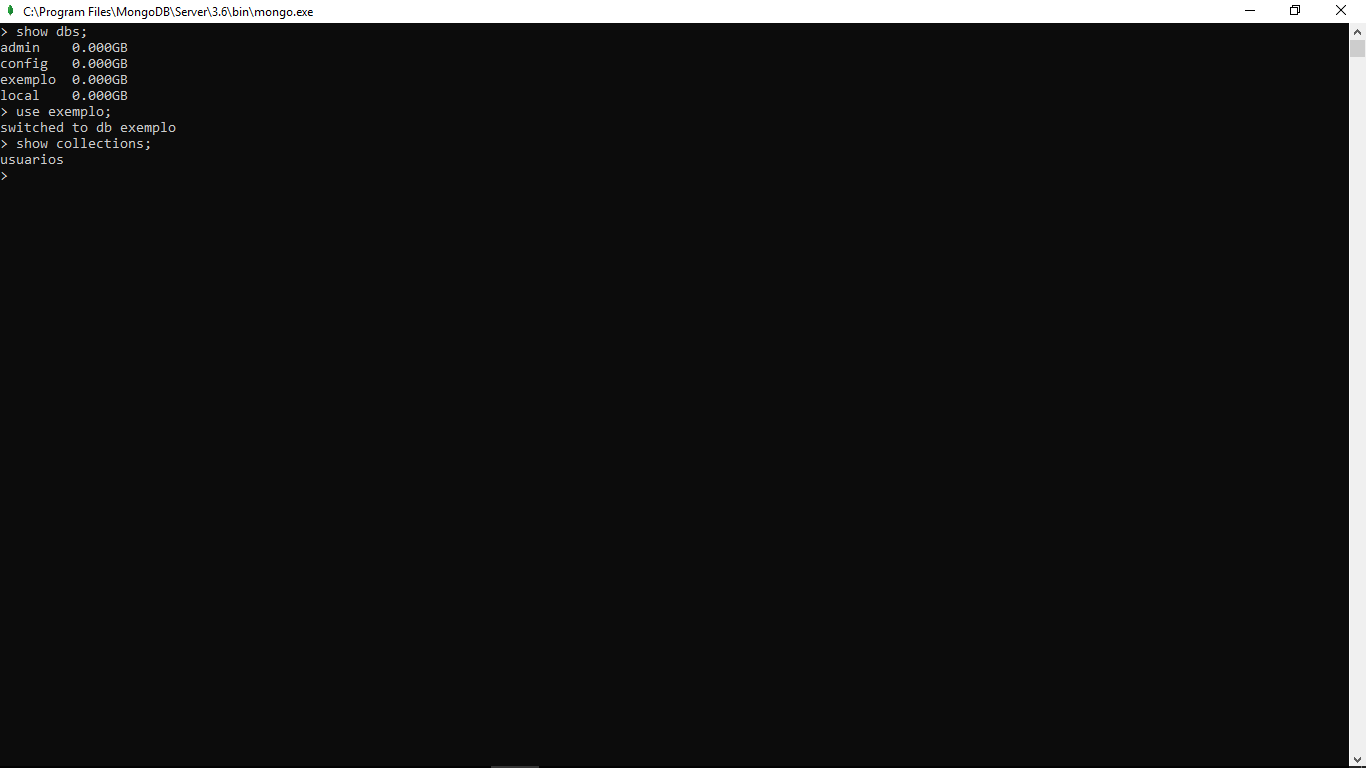
* use exemplo;

Caso a base não exista ela será criada com este mesmo comando.



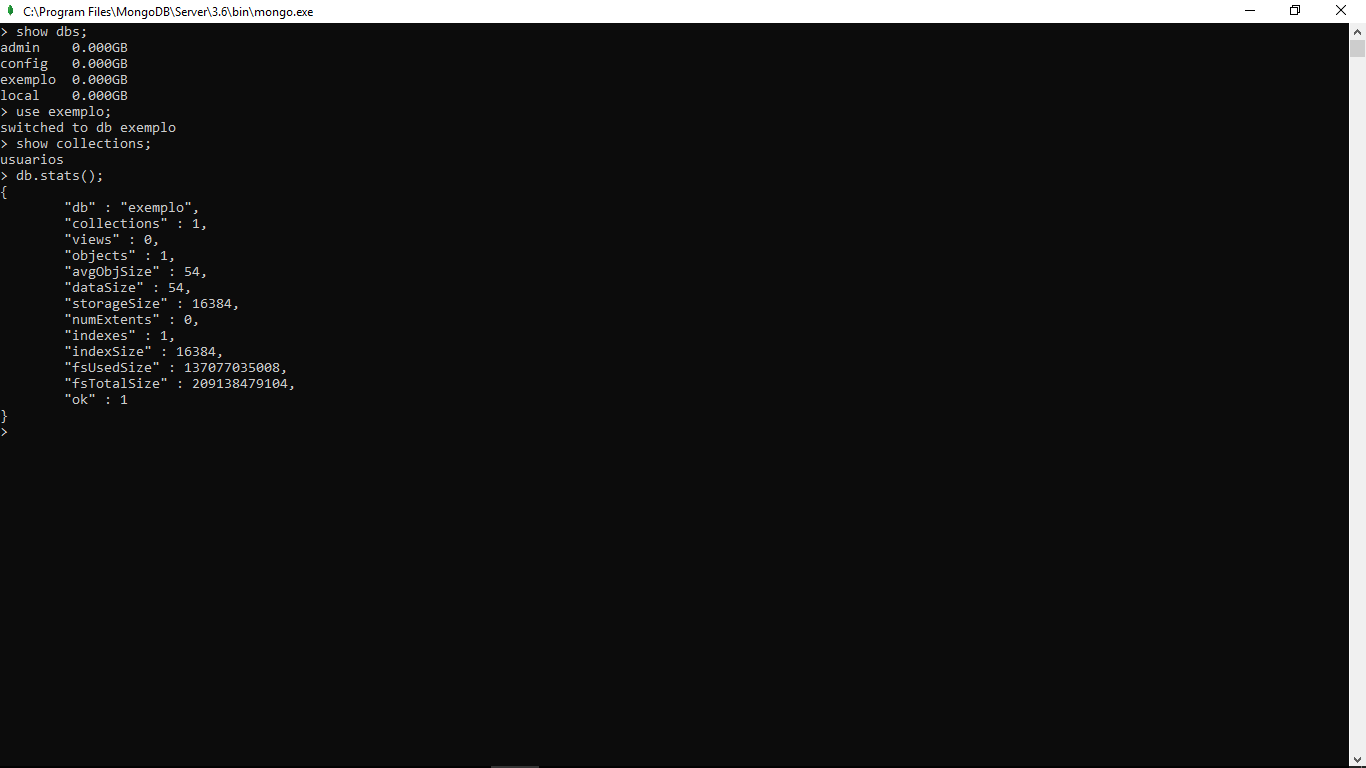
## Mostrar todas as coleções.

* show collections;



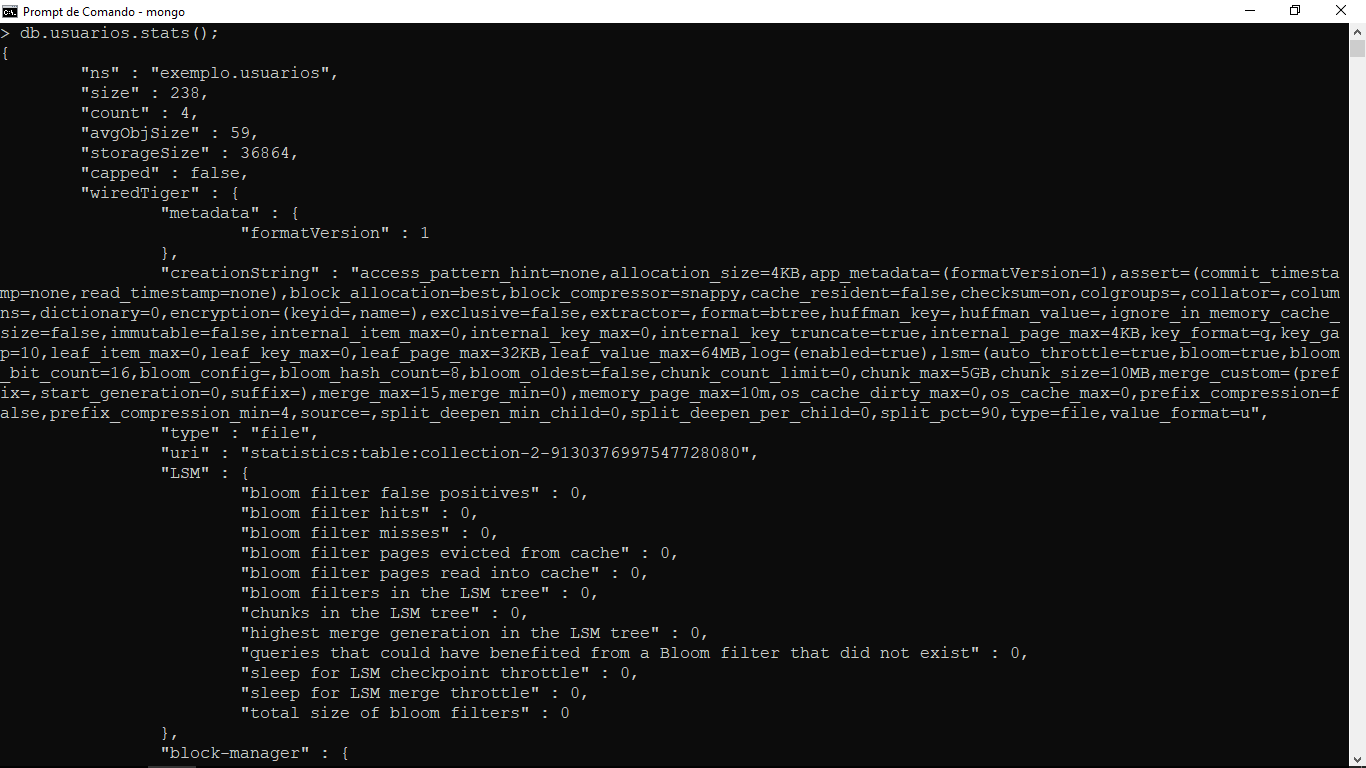
## Visualizar o stats de uma base de dados:

* db.stats();



## Visualizando stats de uma coleção:

* db.usuarios.stats();

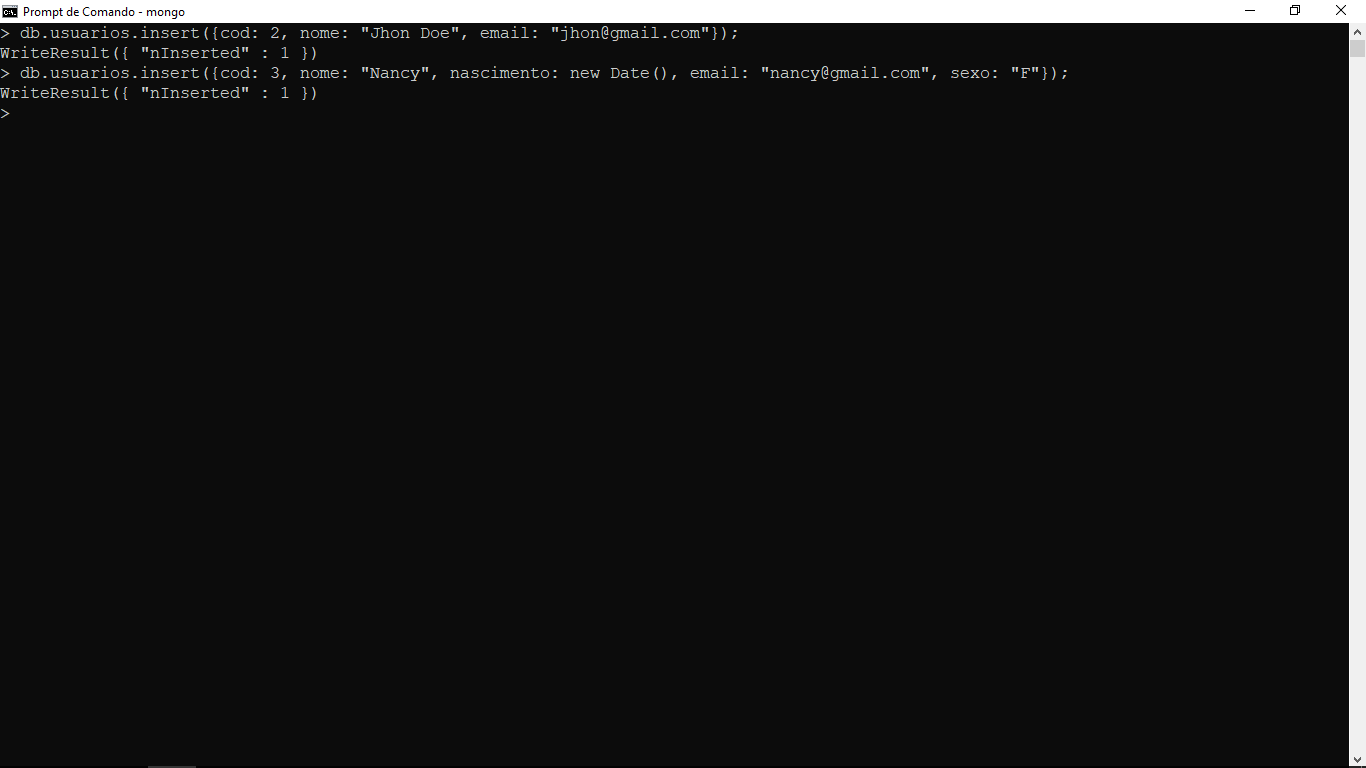


Inserindo dados com novas colunas em uma coleção existente:

* db.usuarios.insert({ cod: 2, nome: “Jhon Doe”, email: “jhon@gmail.com”});
* db.usuarios.insert({ cod: 3, nome: “Nancy”, nascimento: new Date(), email: “[*nancy@gmail.com*](mailto:nancy@gmail.com)”, sexo: “F”});

## DATE.

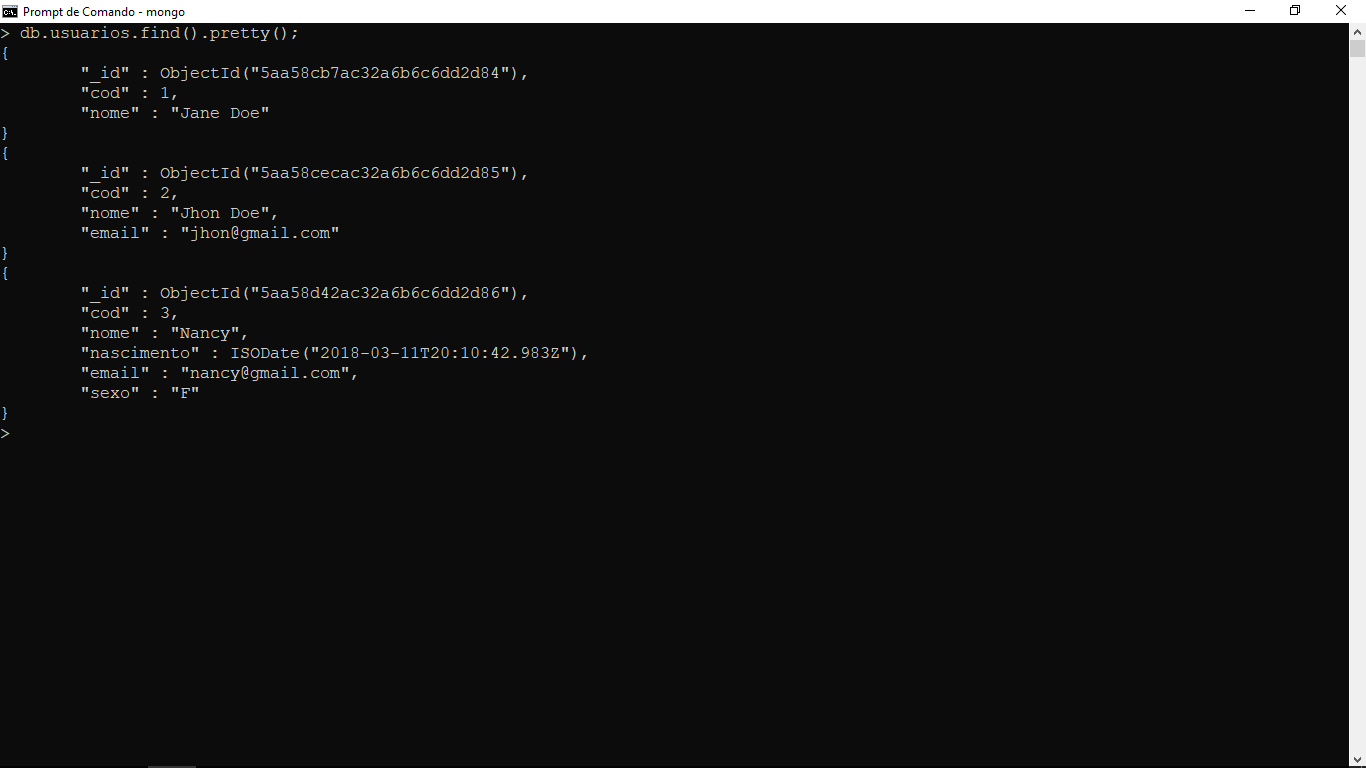
O parametro *“new Date()”* retorna a data e a hora do Sistema Operacional.



## PRETTY.

Visualizando dados no formato estruturado:

* db.usuarios.find().pretty();



## OBJETOS E VETORES.

Inserindo informações em atributos com vetores:

* db.usuarios.insert({

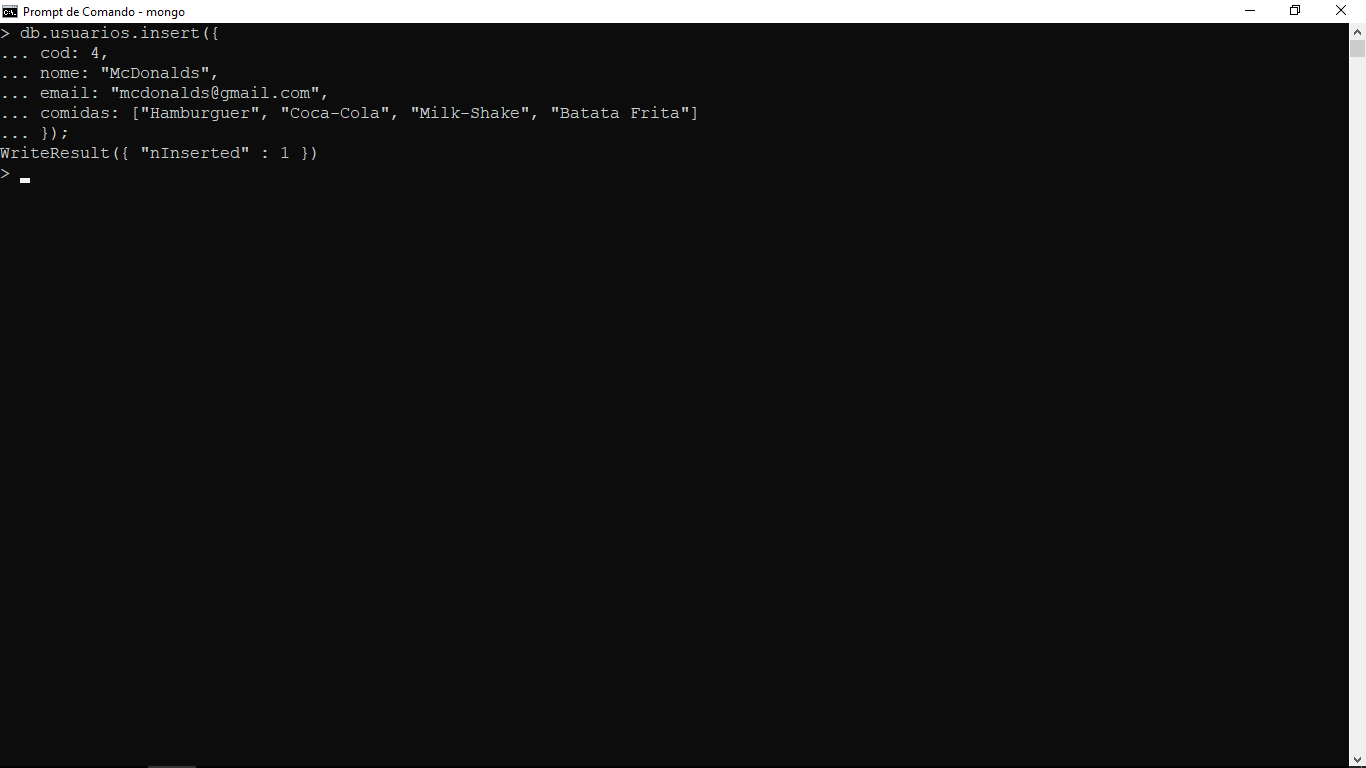
cod: 4,

nome: "McDonalds",

email: "mcdonalds@gmail.com",

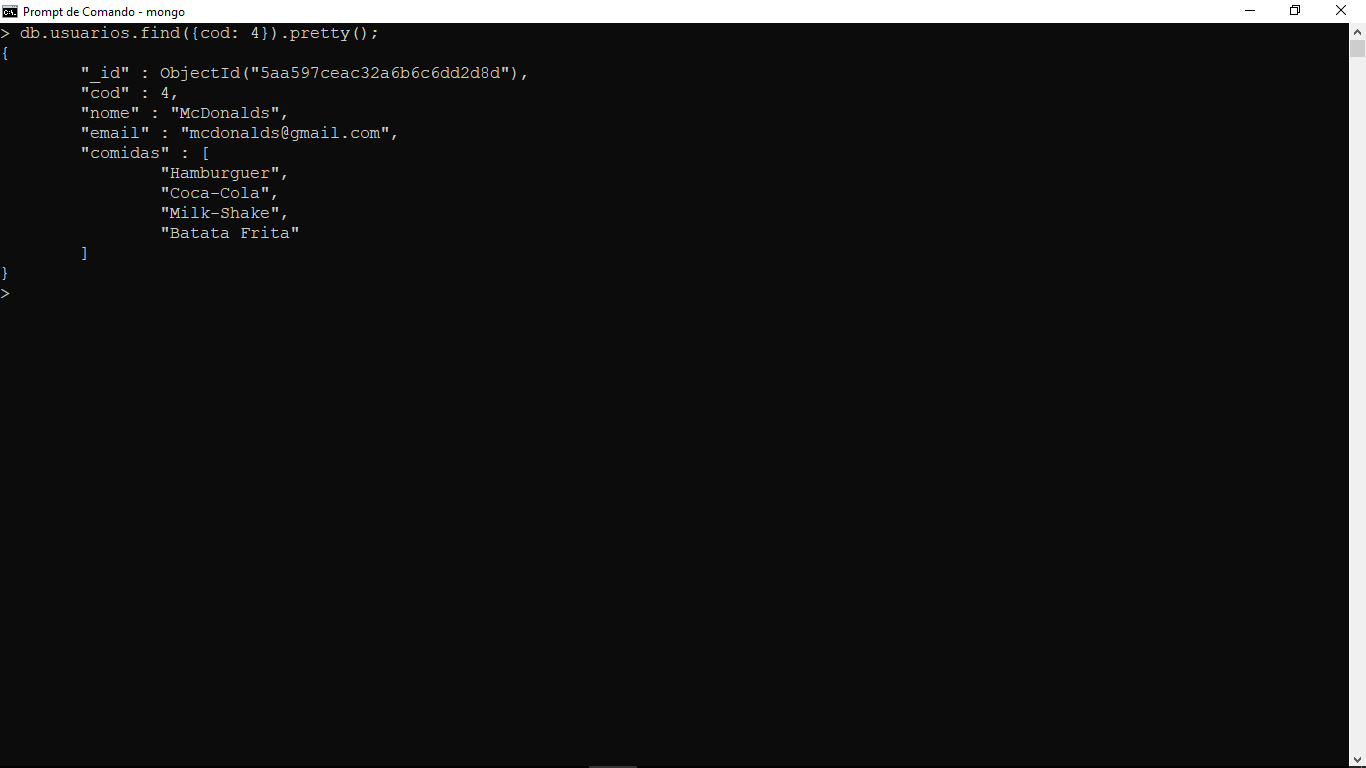
comidas: ["Hambúrguer", "Coca-Cola", "Milk-Shake", "Batata Frita"]

});



Visualizando os dados de uma chave especifica:

* db.usuarios.find({cod: 4}).pretty();



Inserido informações em atributos com objetos:

* db.usuarios.insert({

cod: 5,

nome: "Amy Adams",

profissao: "Atriz",

email: "adams@email.com",

filmes: {

lancamento: 2016,

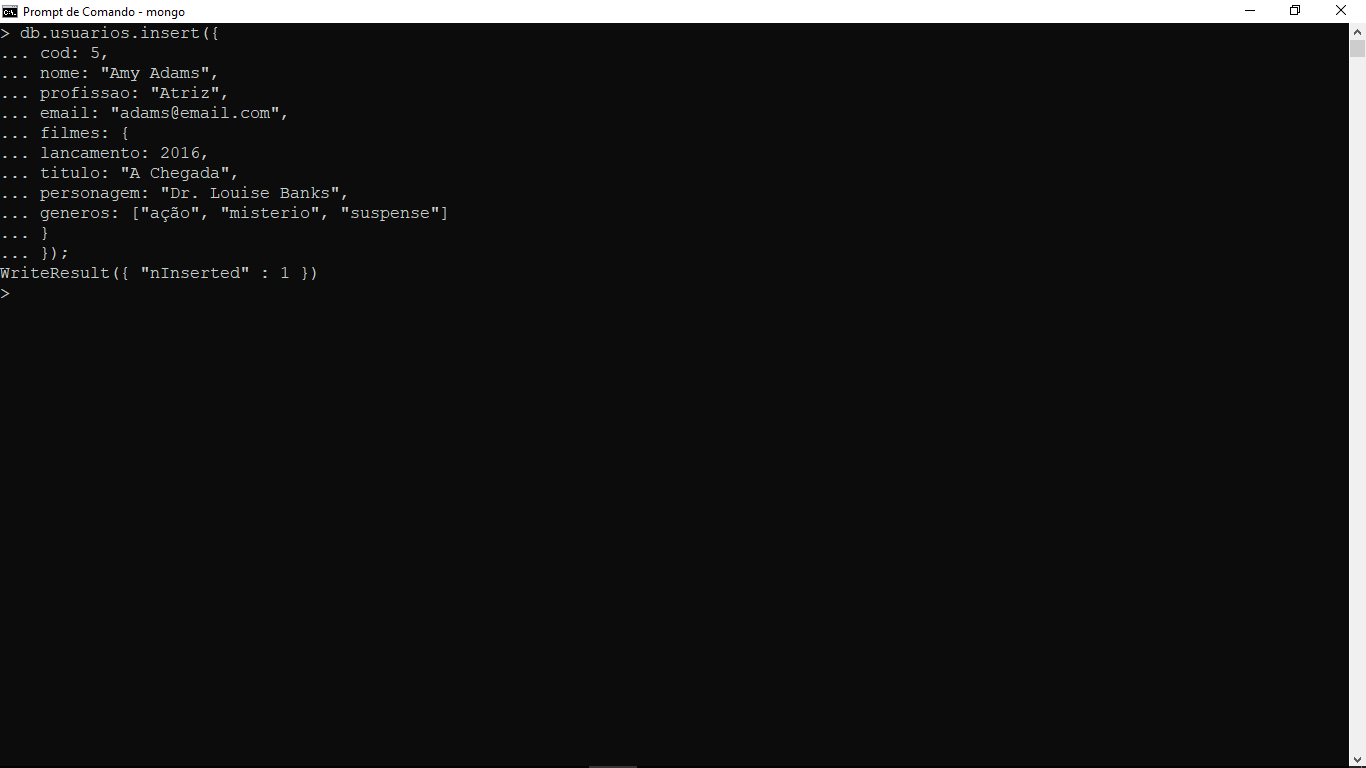
titulo: "A Chegada",

personagem: "Dr. Louise Banks",

generos: ["ação", "misterio", "suspense"]

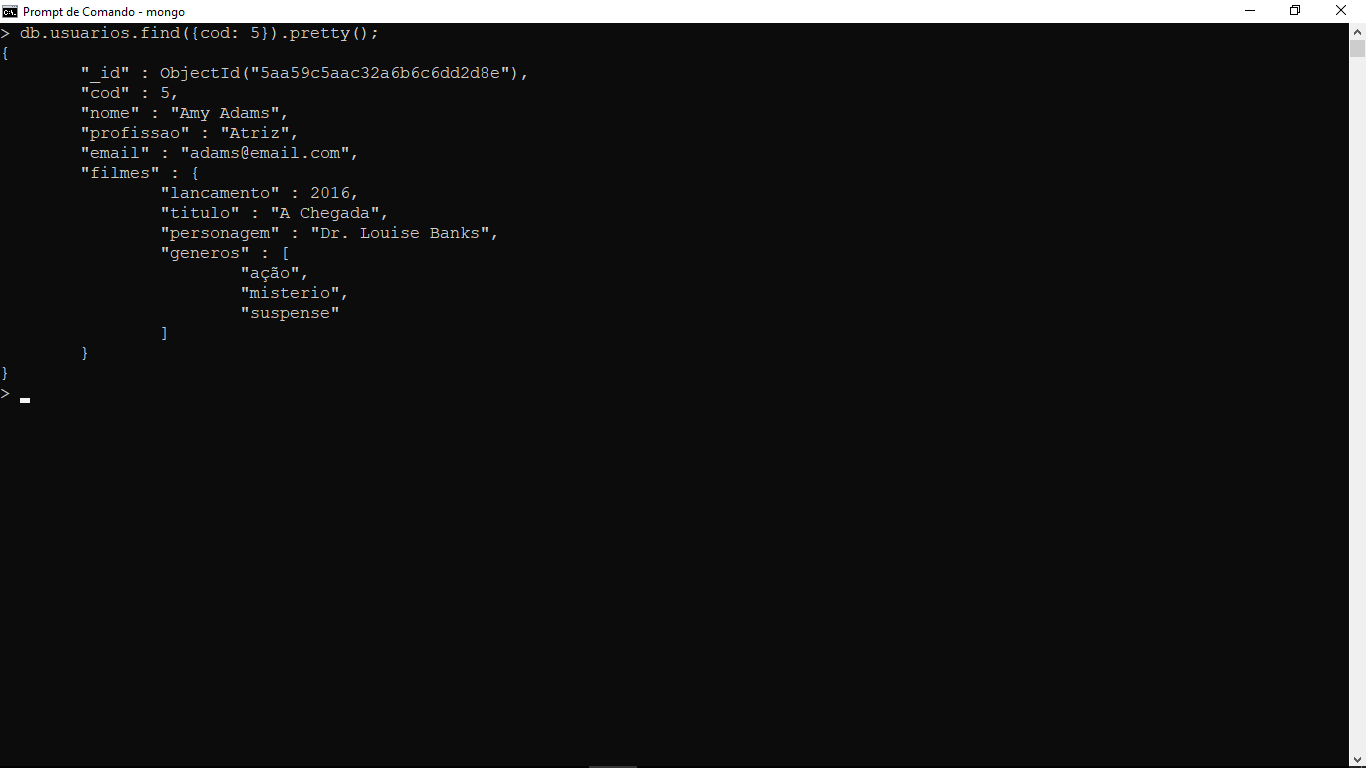
}

});



Visualizando as informações:

* db.usuarios.find({cod: 5}).pretty();



# **CRUD (Create, Read, Update, Delete)**

## CREATE.

Tipos de insert:

1. db.collection.insert();

Insere 1 ou mais documentos;

2. db.collection.insertOne();

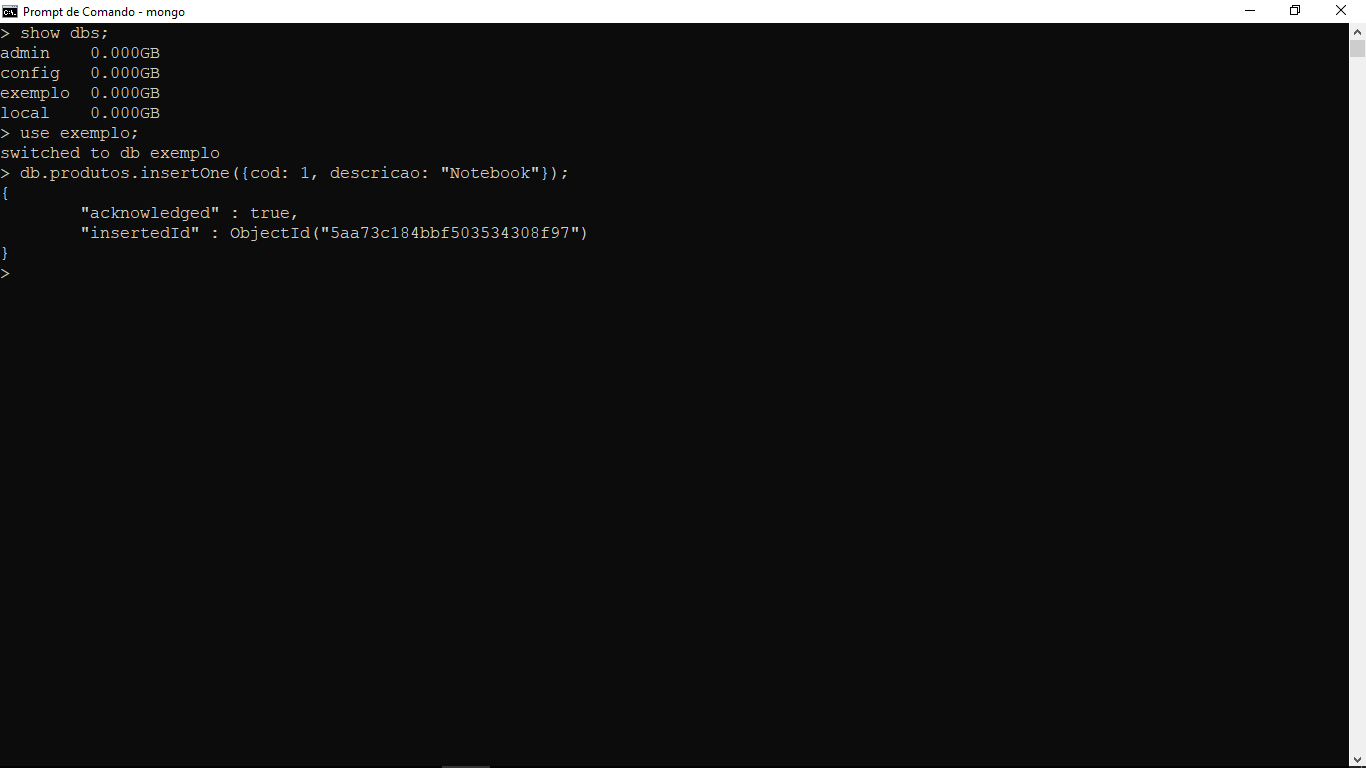
Insere apenas 1 documento;

3. db.collection.insertMany();

Insere mais de um documento.

Usando “insertOne()”:

* db.produtos.insertOne({cod: 1, descricao: "Notebook"});



Usando *“insert()”* com vários parâmetros:

É necessário abrir um vetor para inserir mais de um objeto.

* db.produtos.insert([

{

cod: 2,

descricao: "Placa de Video",

valor: 1700.00

},

{

cod: 3,

descricao: "Mouse"

},

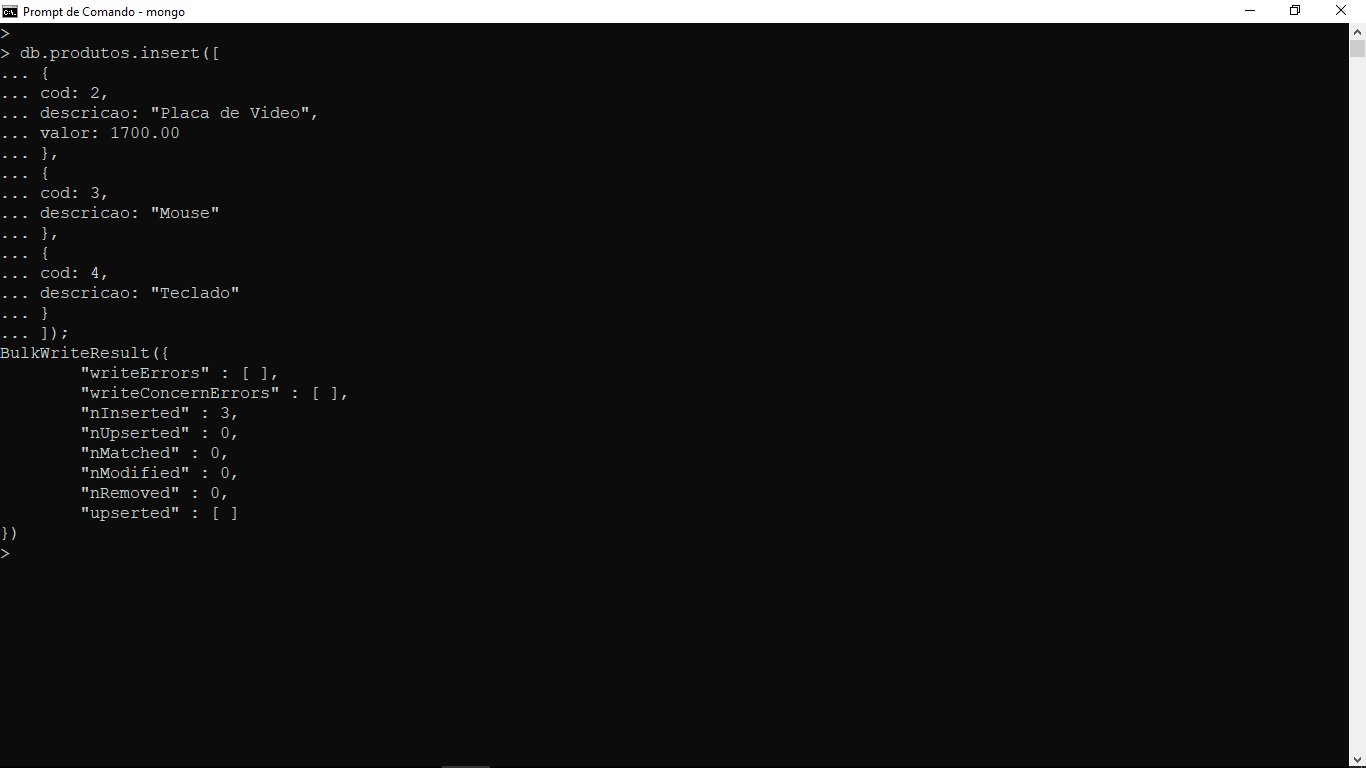
{

cod: 4,

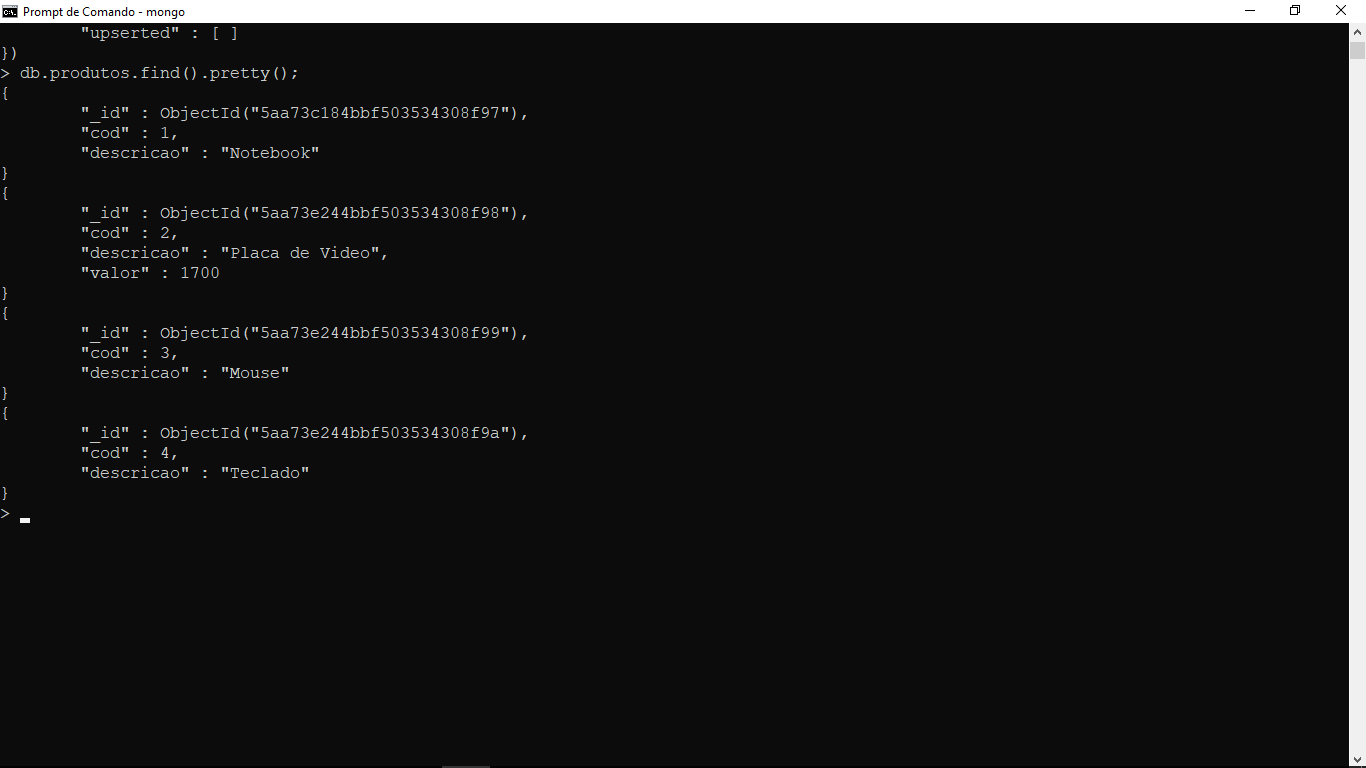
descricao: "Teclado"

}

]);



Visualizando dados inseridos:



Usando o *“insertMany()”*.

* db.produtos.insertMany([

{

cod: 5,

descricao: "Monitor",

tela: 15.4

},

{

cod: 6,

descricao: "Gabinete"

},

{

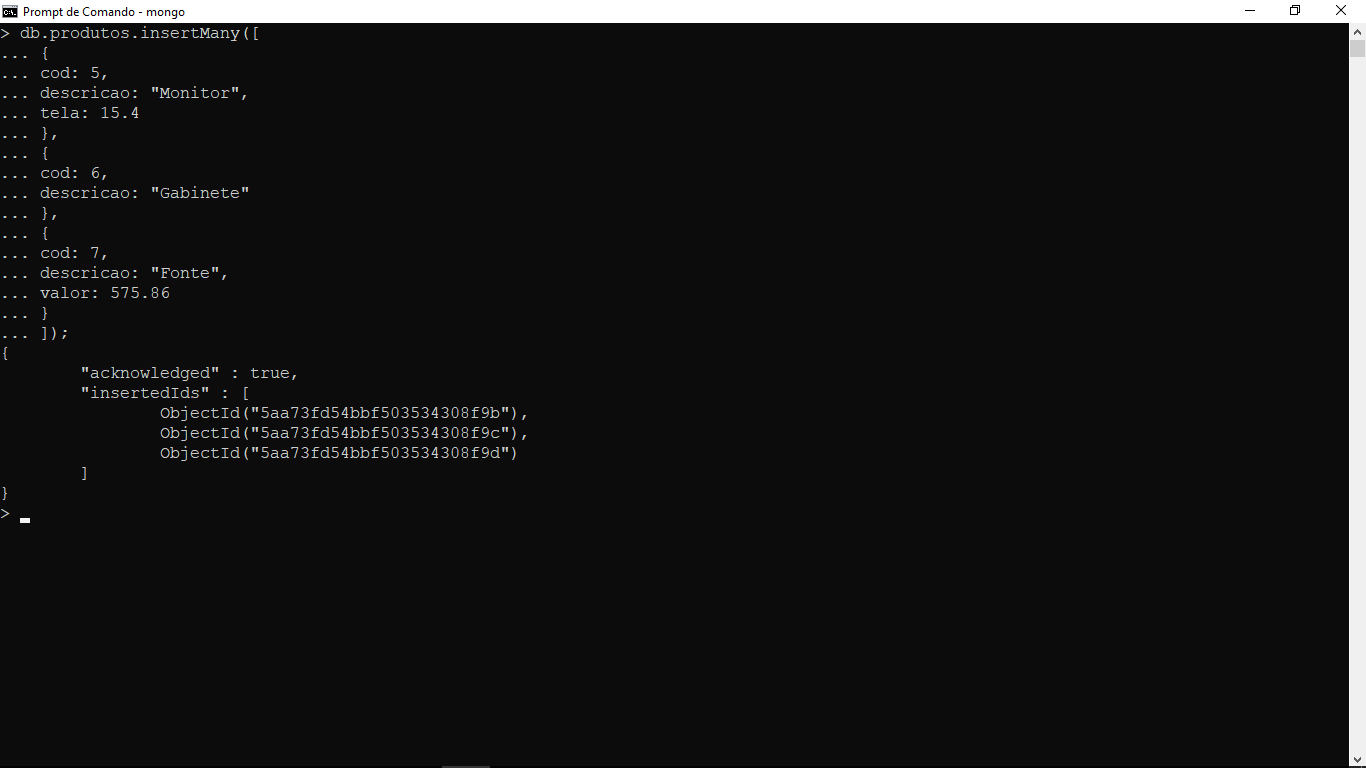
cod: 7,

descricao: "Fonte",

valor: 575.86

}

]);



## UPDATE.

1. db.collection.update();

Atualiza o primeiro documento encontrado.

2. db.collection.updateOne();

Atualiza o primeiro documento encontrado.

3. db.collection.updateMany();

Atualiza todos os documentos encontrados.

Usando “update()”

* db.usuarios.update(

{cod: 2},

{

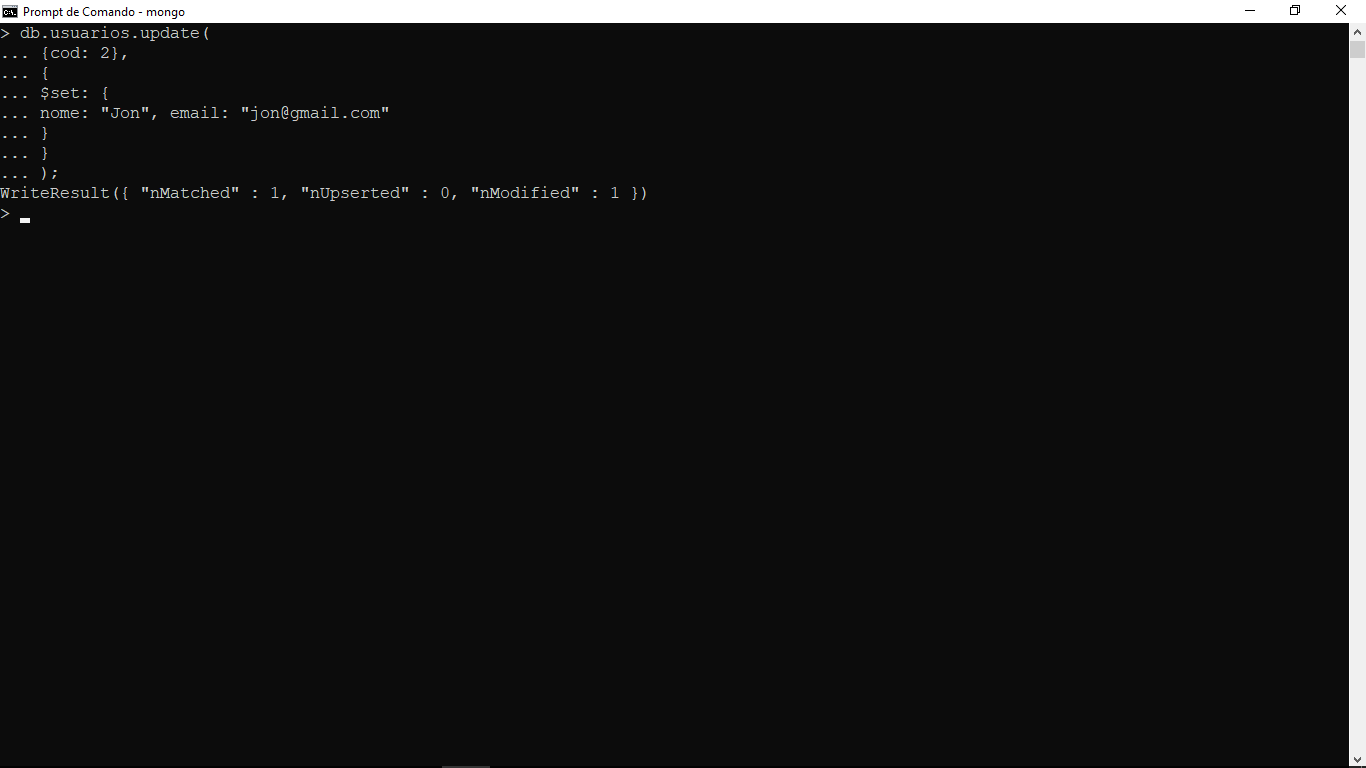
$set: {

nome: "Jon", email: "jon@gmail.com"

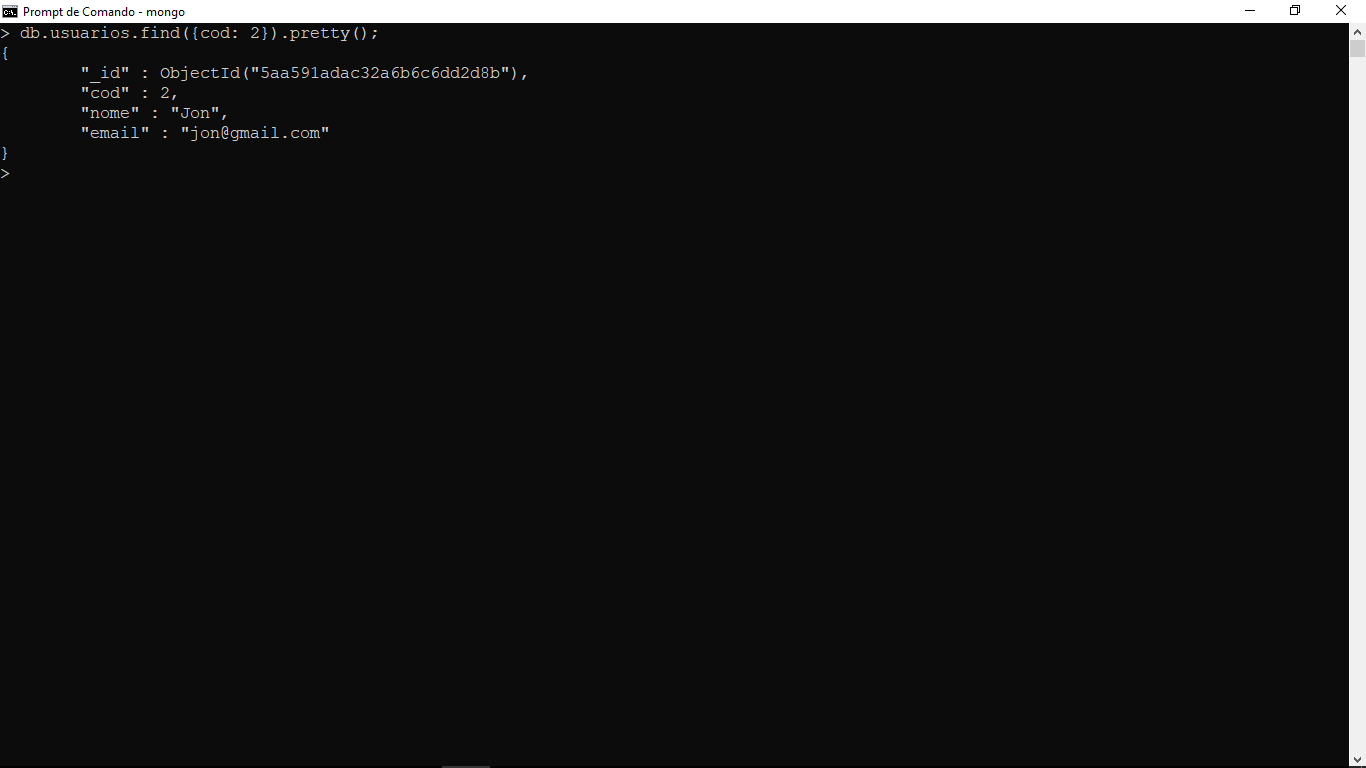
}

}

);



Visualizando as alterações:



Utilizando “updateOne()”:

* db.usuarios.updateOne(

{cod: 3},

{

$set: {

nome: "Dany",

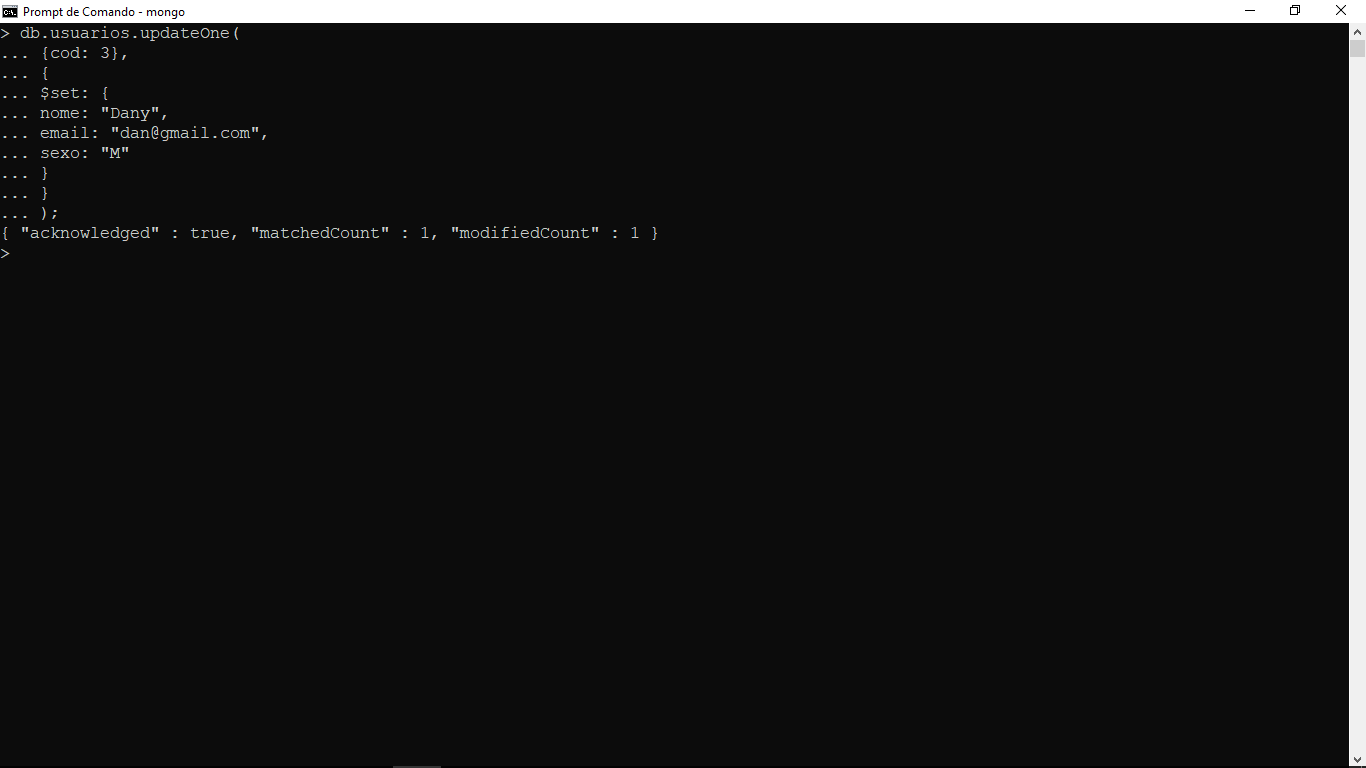
email: "dan@gmail.com",

sexo: "M"

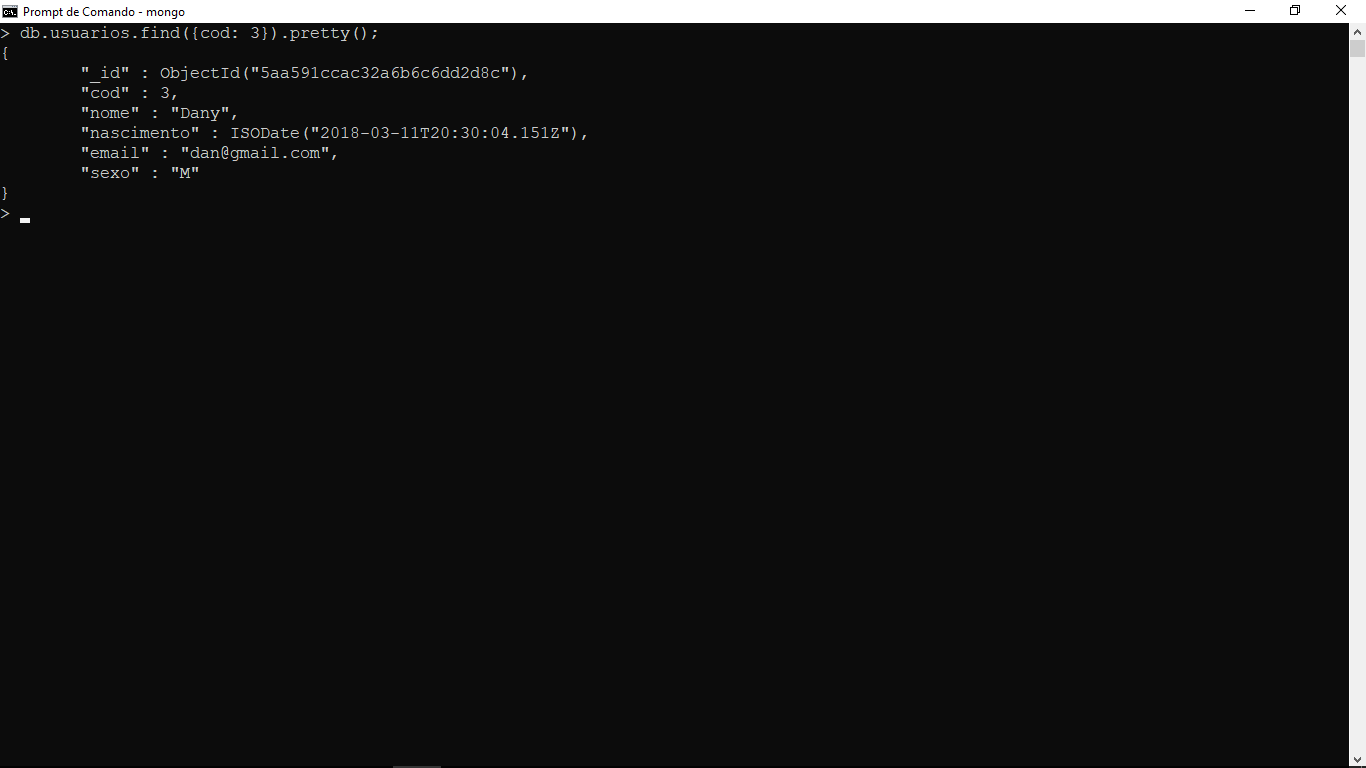
}

}

);



Visualizando alteração:



Nos *“update”* anteriores o comando só altera a primeira coleção que for encontrada de acordo com o parametro informado.

Agora vamos testar o *“updateMany()”*:

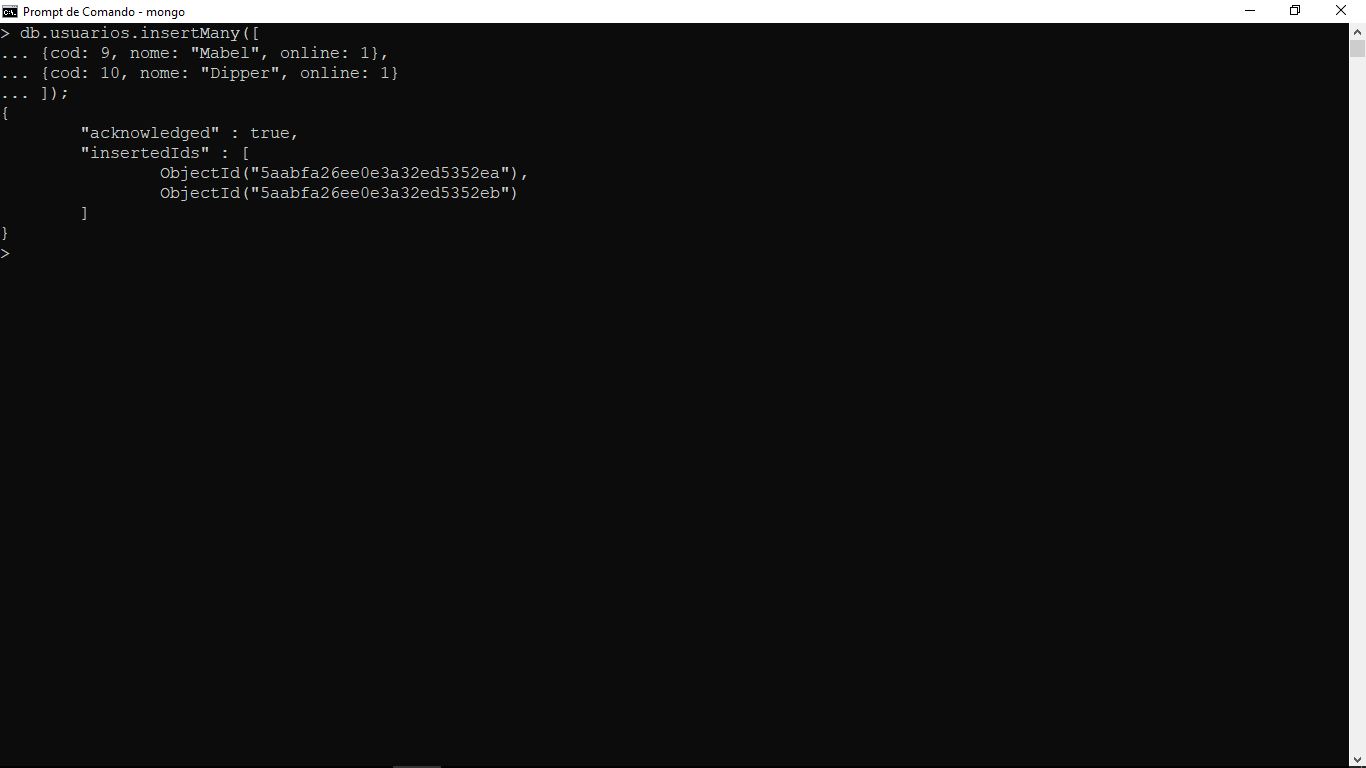
Insira dois usuários em que as chaves tenham valores idênticos:

* db.usuarios.insertMany([

{cod: 9, nome: "Mabel", online: 1},

{cod: 10, nome: "Dipper", online: 1}

]);



Agora use o *“updateMany()”* para modificar todas as coleções que for igual a chave e o valor passado como parametro:

* db.usuarios.updateMany(

{online: 1},

{

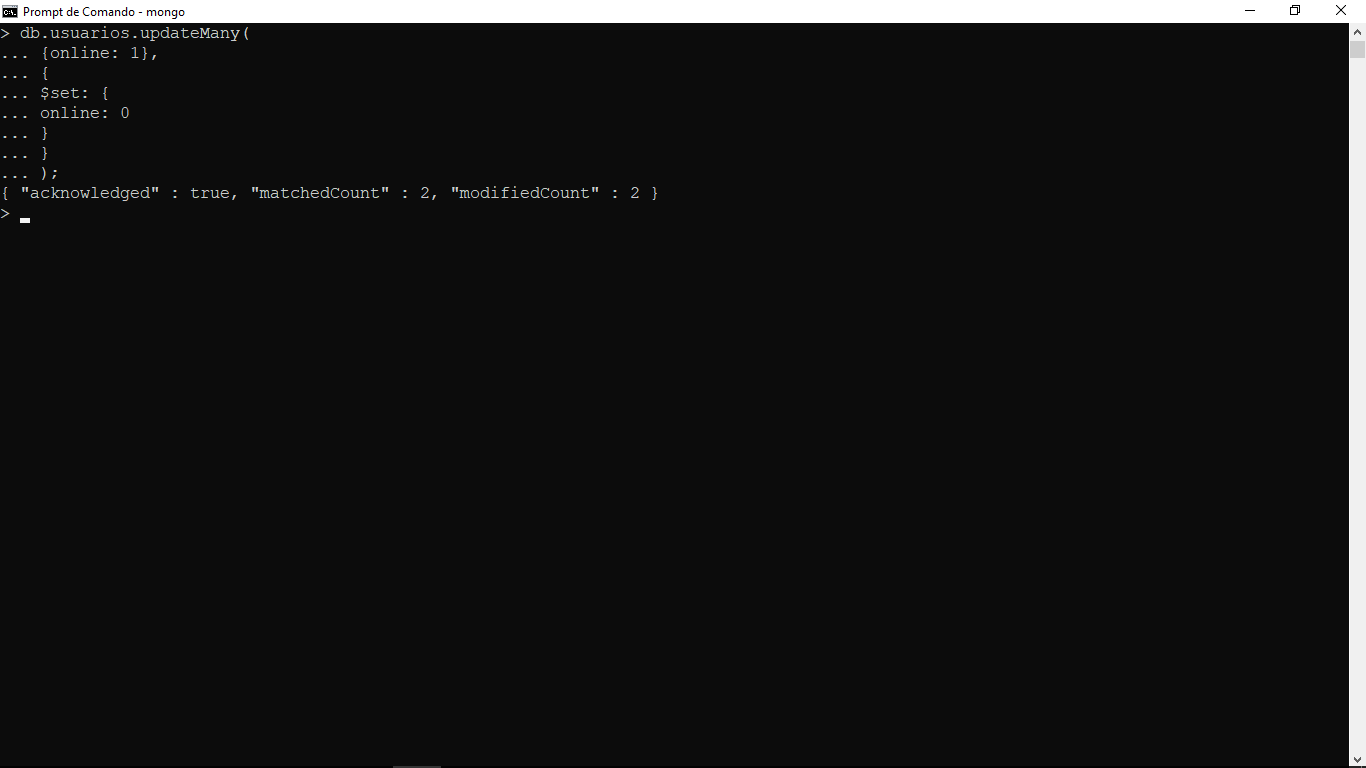
$set: {

online: 0

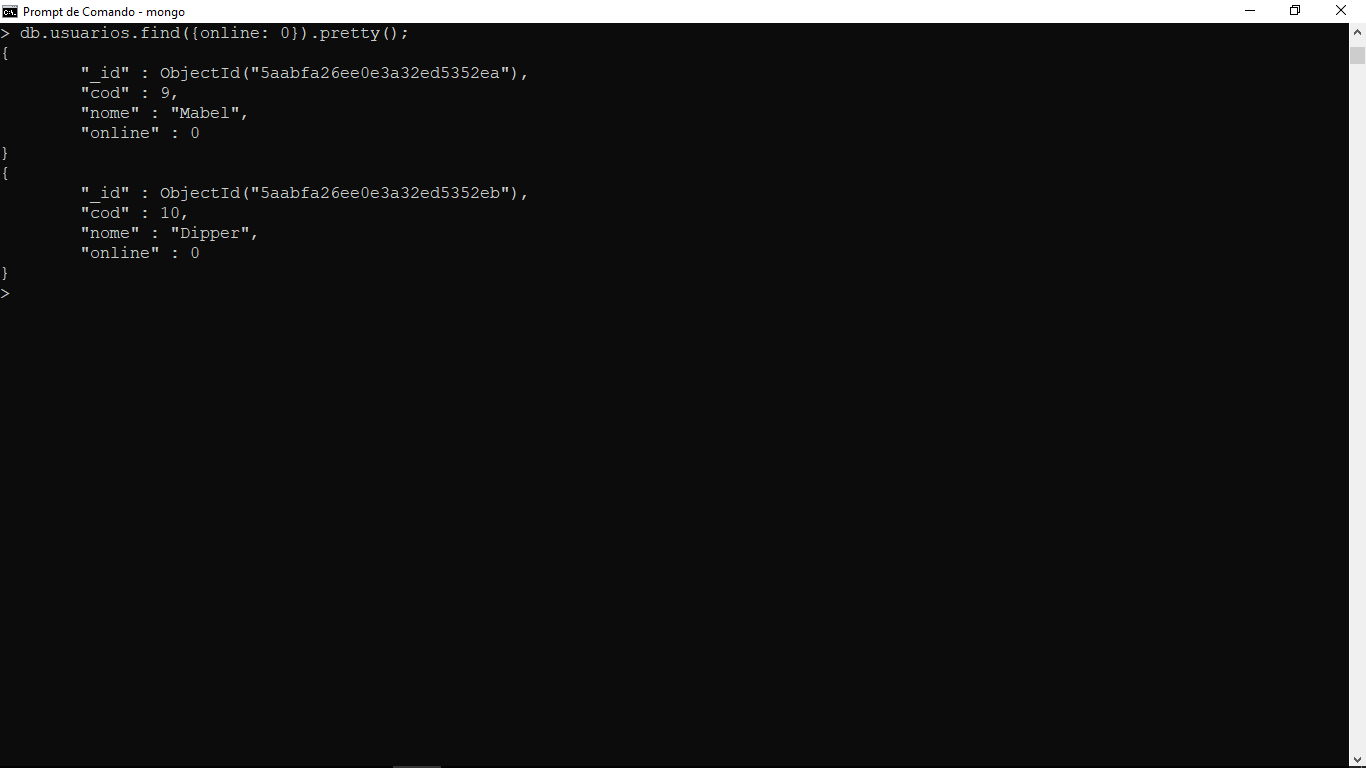
}

}

);



Visualizando alterações:



Se o parametro *“$set”* não for colocado o objeto é substituído.

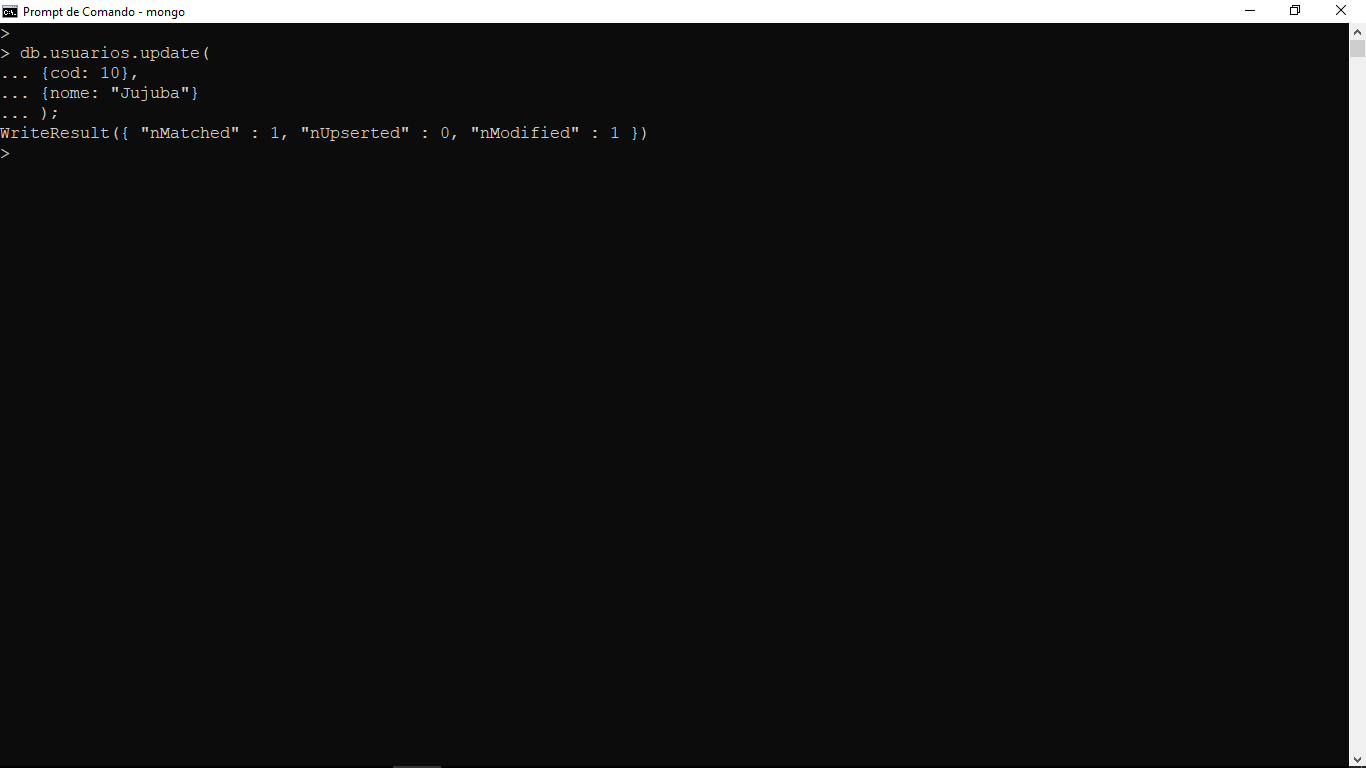
Exemplo:

* db.usuarios.update(

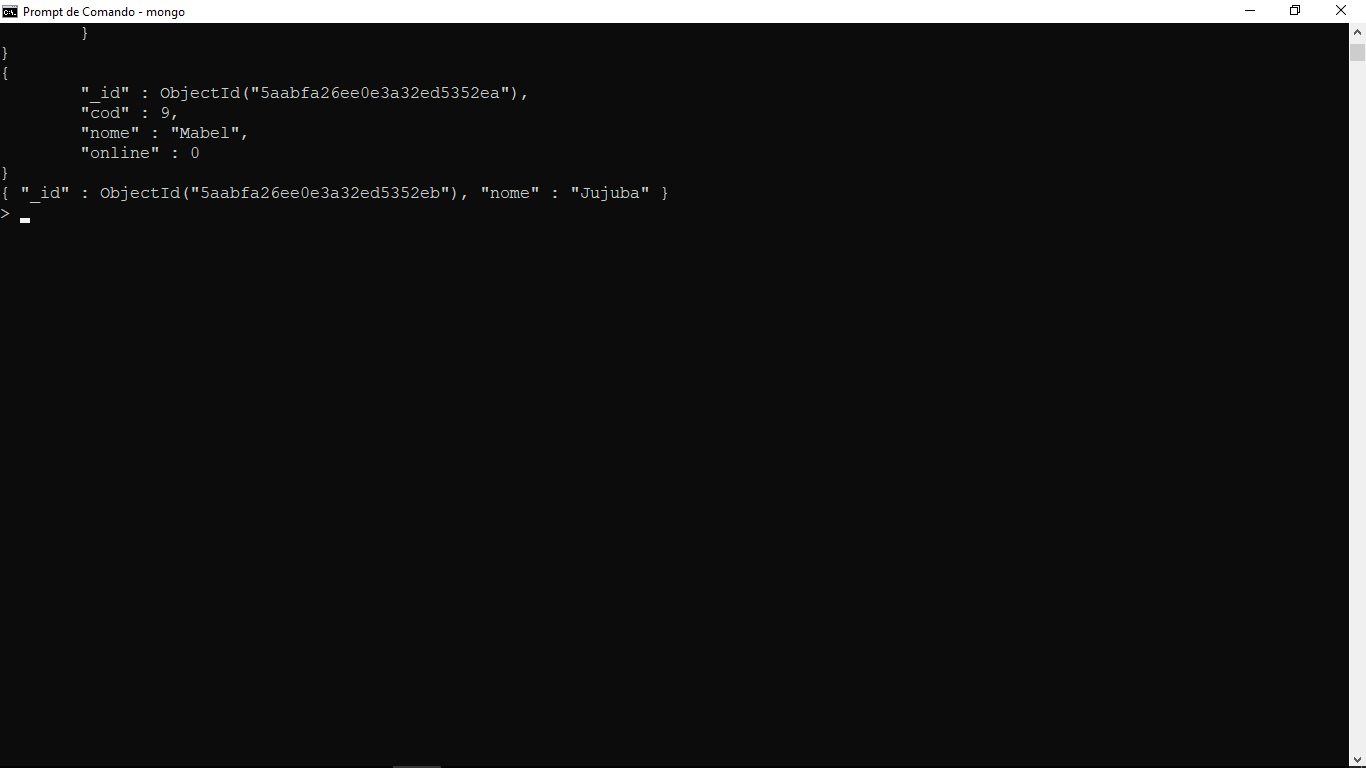
{cod: 10},

{nome: "Jujuba"}

);



Quando o parametro *“$set”* não é informado o objeto é substituído:



O documento com “cod: 10” foi substituído por outro documento que contem apenas o “ObjectId” e a chave “nome”.

## DELETE.

1. db.collection.remove({});

Remove todos os documentos.

2. db.collection.deleteOne({});

Remove o primeiro documento encontrado.

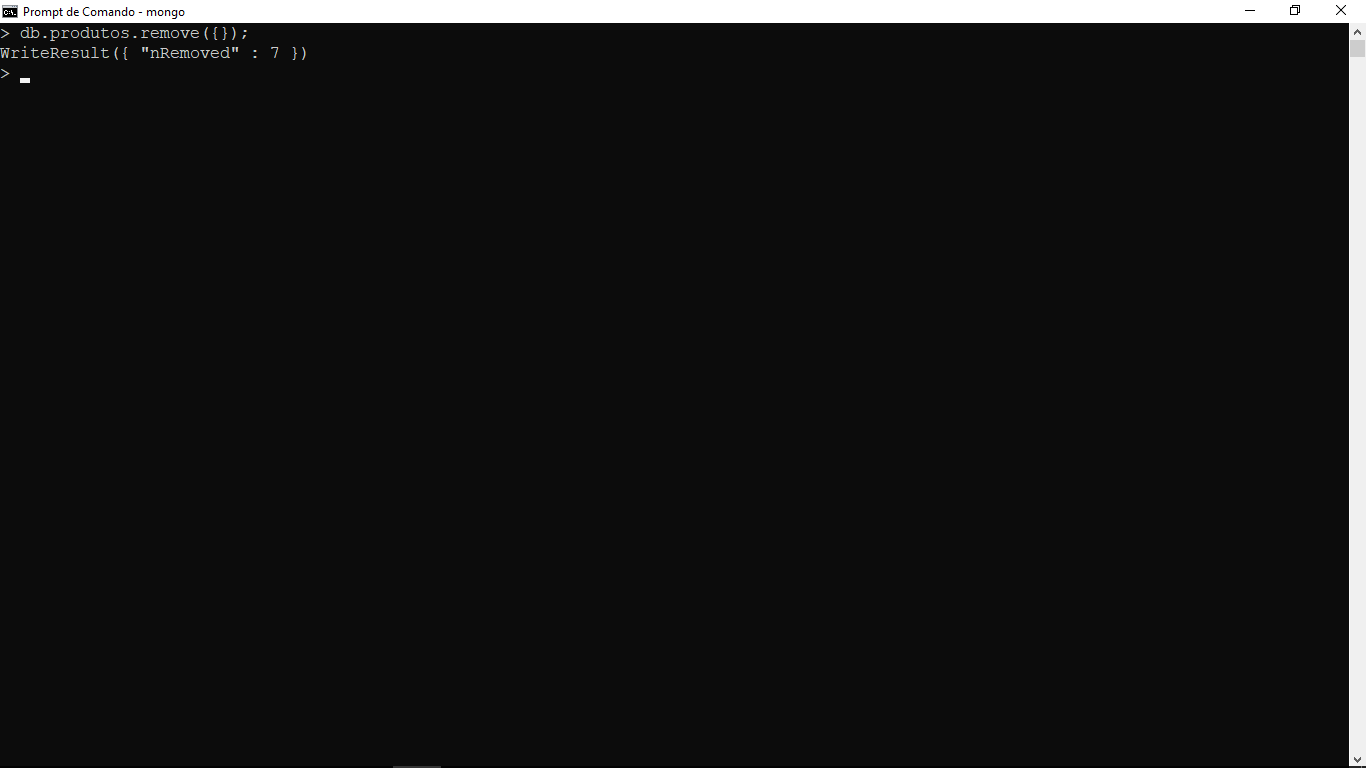
3. db.collection.deleteMany({});

Remove todos os documentos encontrados.

Usando o “remove({})”:

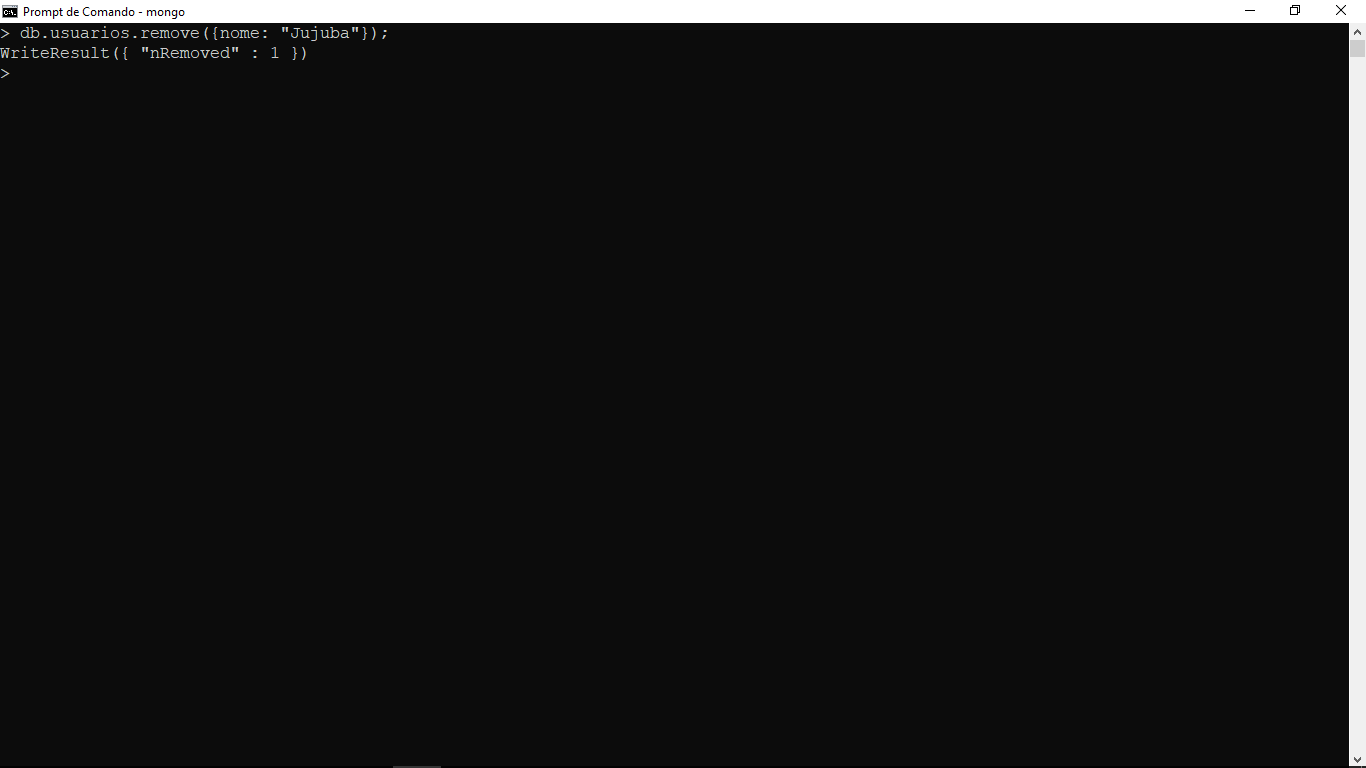
* db.produtos.remove({});

Este comando irá remover todos os documentos pois não foi informado nenhum parametro de busca.



Removendo apenas o documento especificado:

* db.usuarios.remove({nome: "Jujuba"});



## READ.

1. db.collection.find({});

Retorna todos os documentos da coleção.

2. db.collection.find().pretty();

Retorna todos os documentos em formato Legível(JSON).

3. db.collection.find({chave: valor}).pretty();

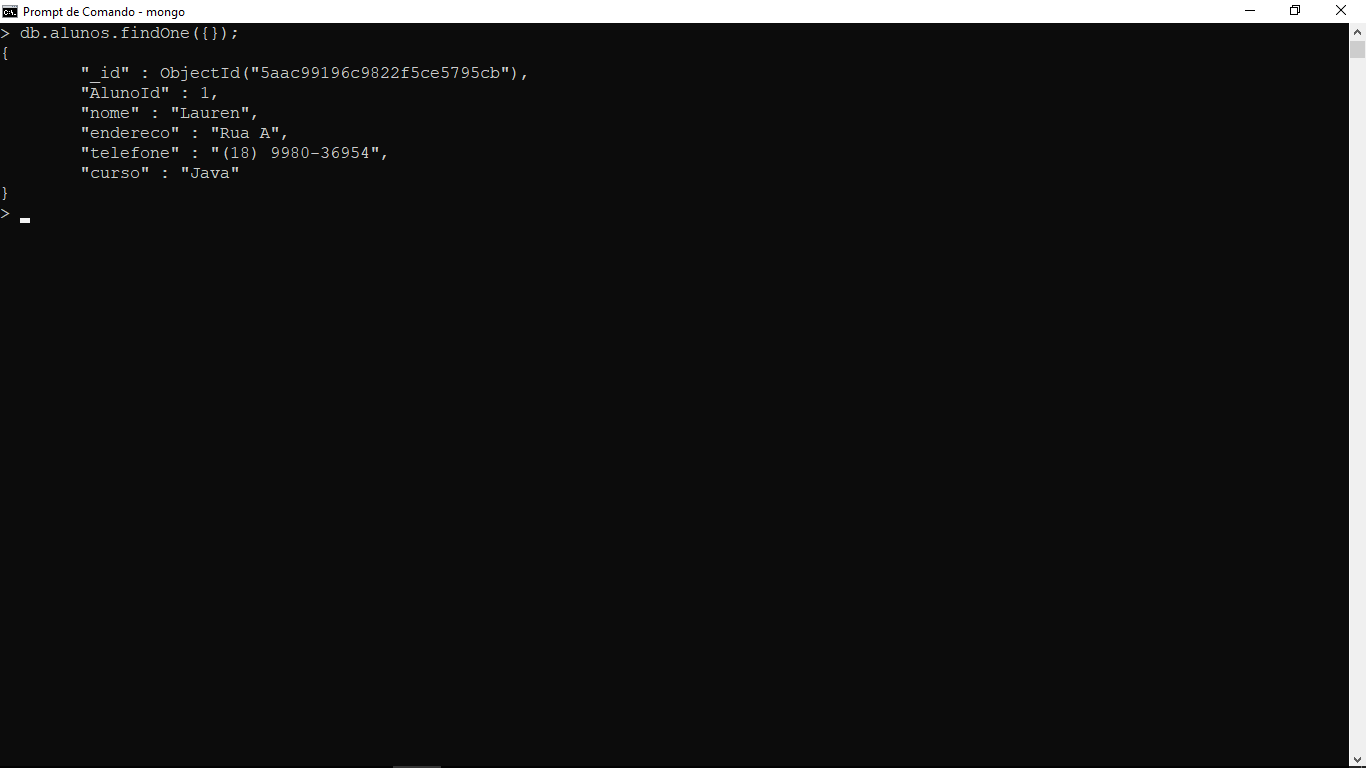
Retorna todos os documentos encontrados. Aceita mais de um parametro.

3. db.collection.findOne({});

Retorna o primeiro documento encontrado.

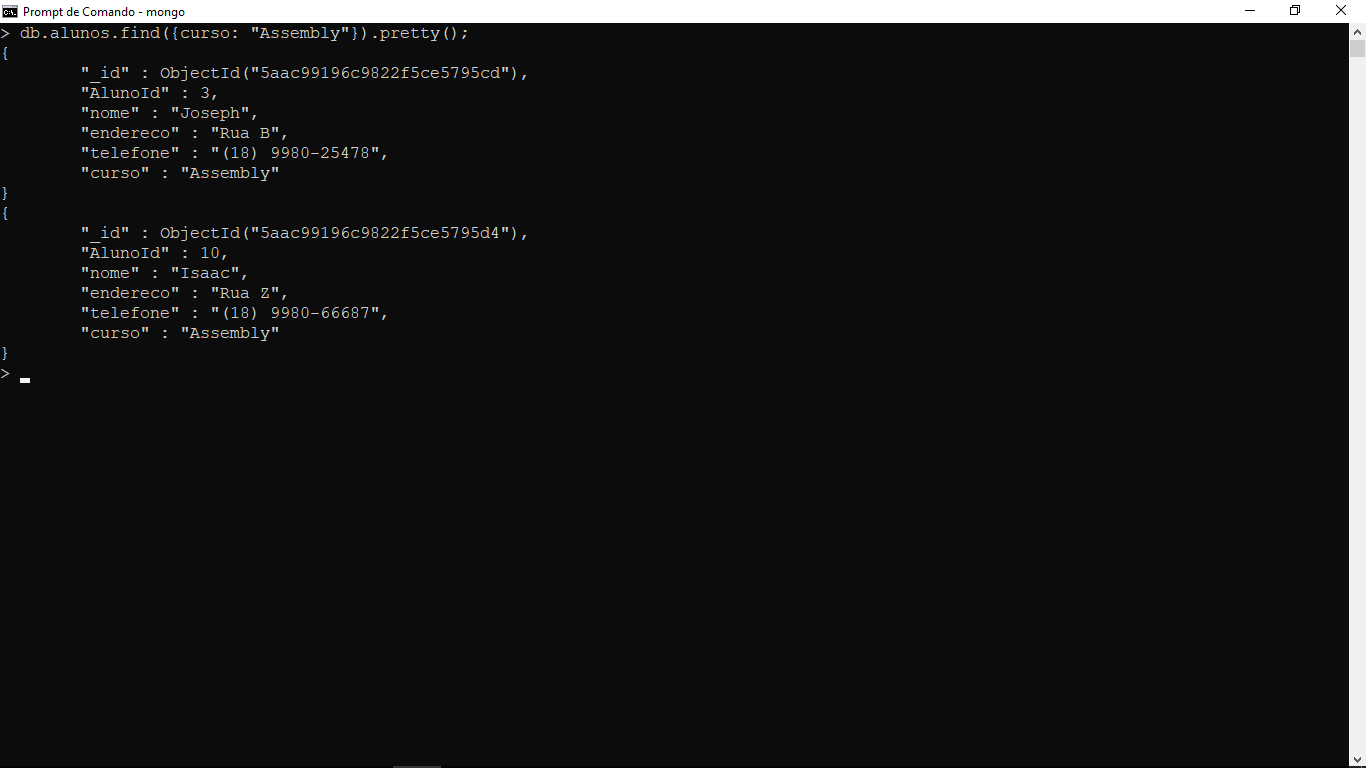
Usando o “findOne()”;

* db.alunos.findOne({});



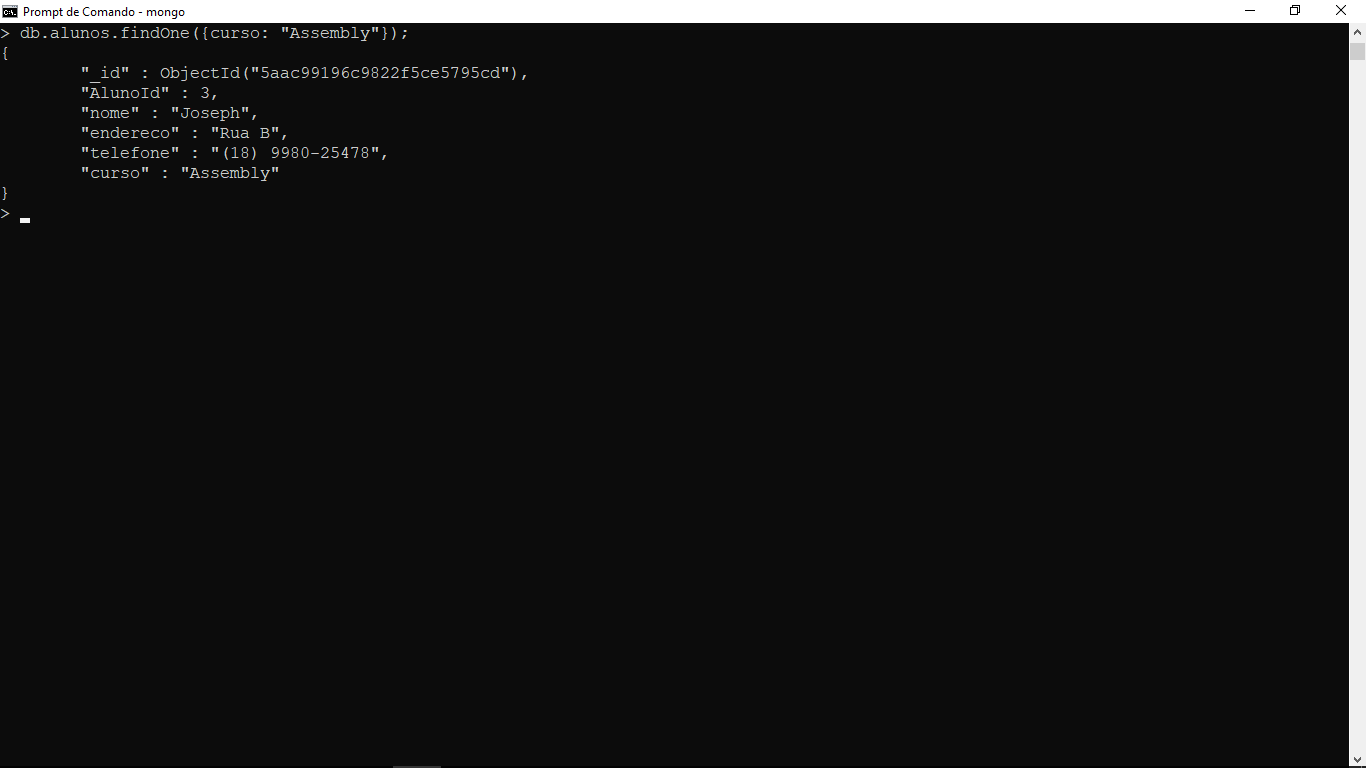
Usando “find({chave: valor})”.

* db.alunos.find({curso: “Assembly”}).pretty();



Usando “findOne({chave: valor})”.

* db.alunos.find({curso: “Assembly”}).pretty();



# Operadores Lógicos:

## $and:

* db.alunos.find({

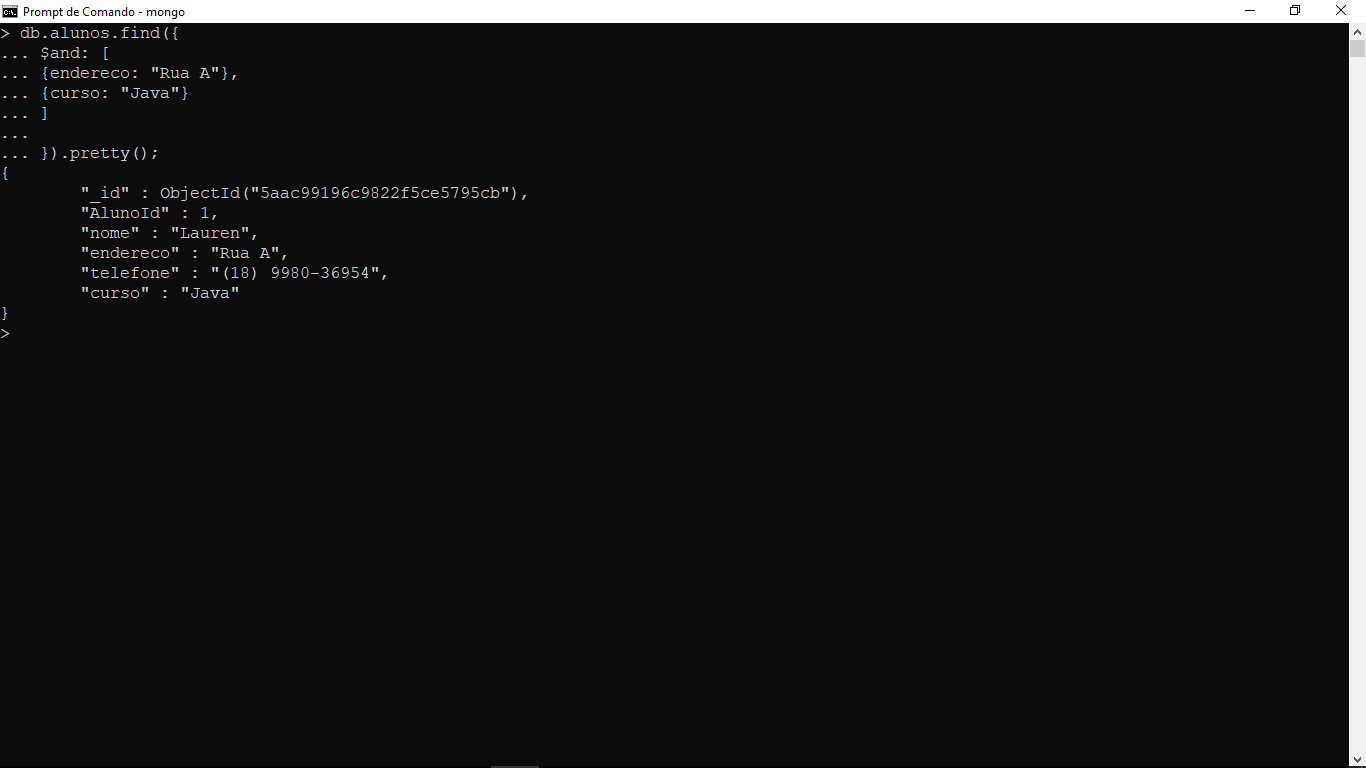
$and: [

{endereco: "Rua A"},

{curso: "Java"}

]

}).pretty();



## $or:

* db.alunos.find({

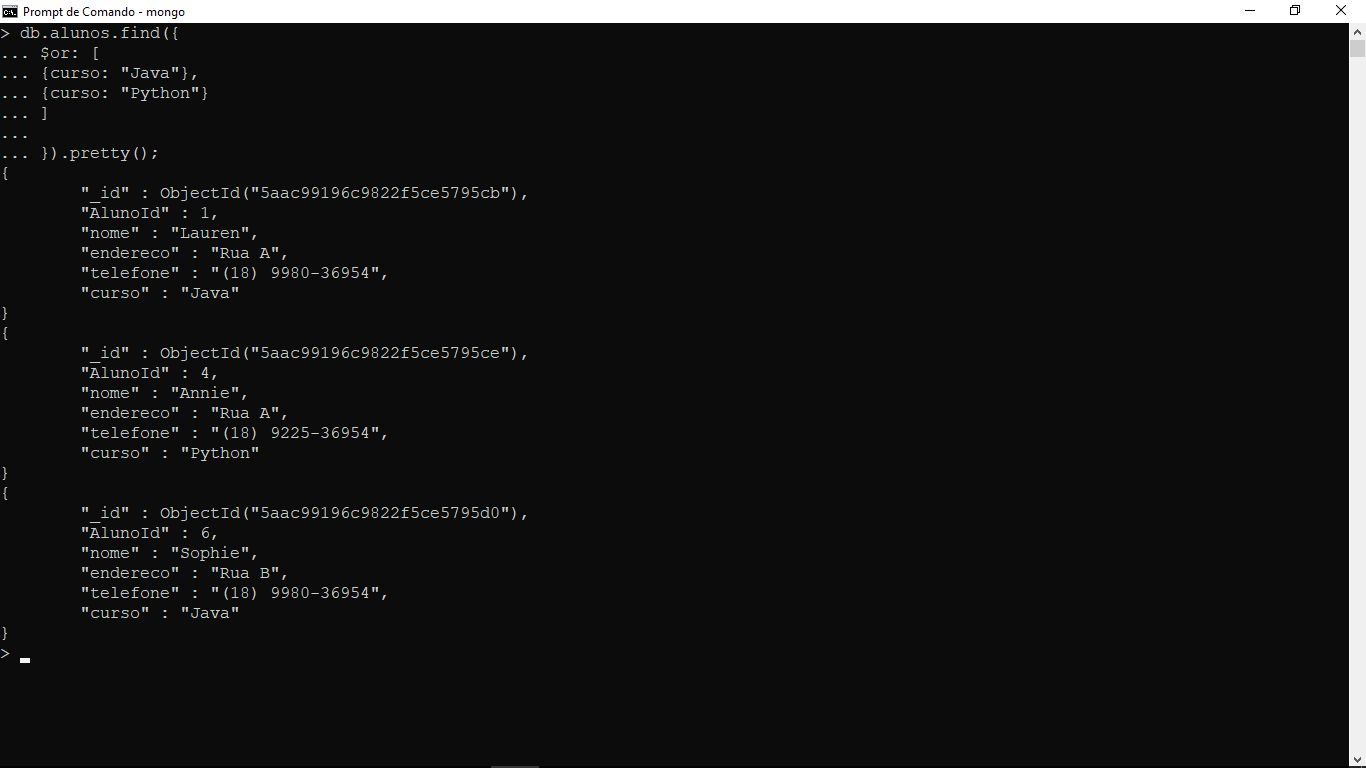
$or: [

{curso: "Java"},

{curso: "Python"}

]

}).pretty();



## $exists:

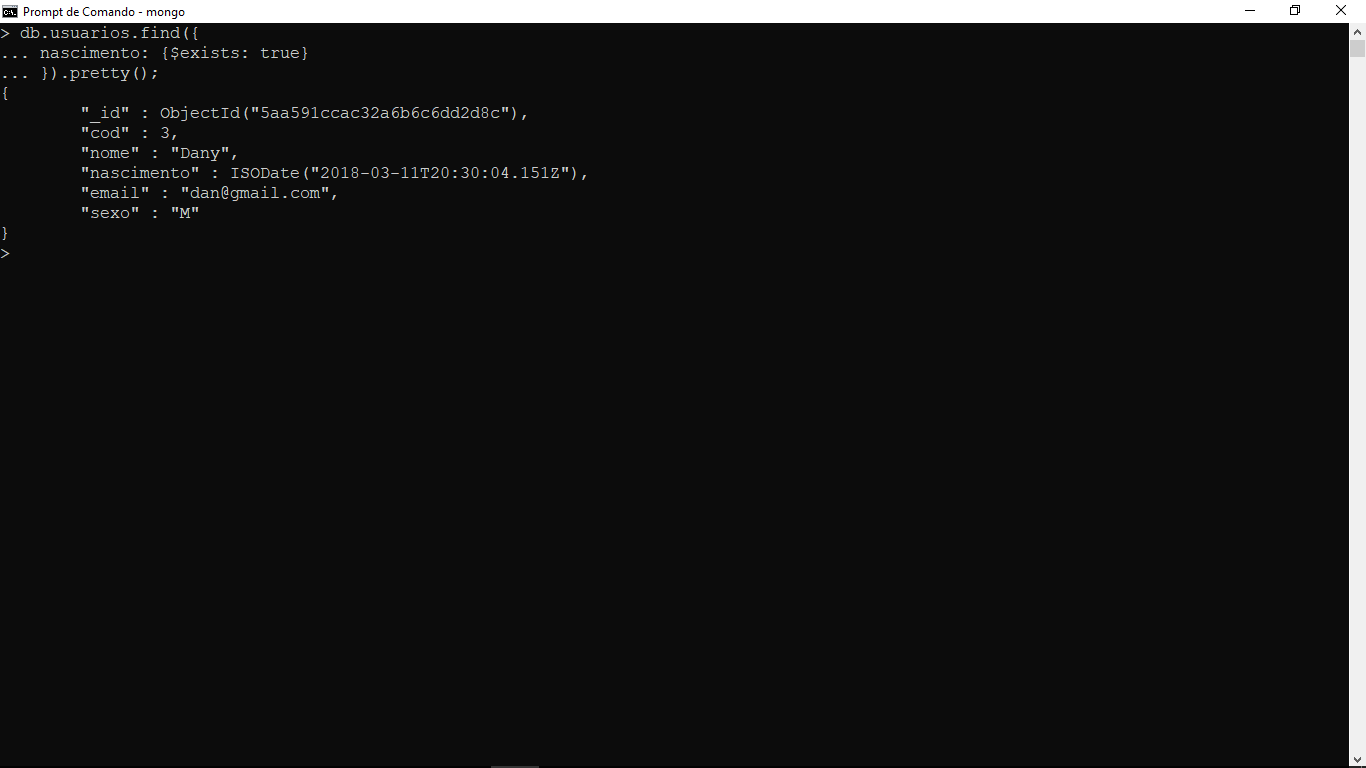
Este operador verifica se a chave/atributo existe na coleção.

* db.usuarios.find({

nascimento: {$exists: true}

}).pretty();

Retorna apenas os documentos ao qual o atributo nascimento exista.

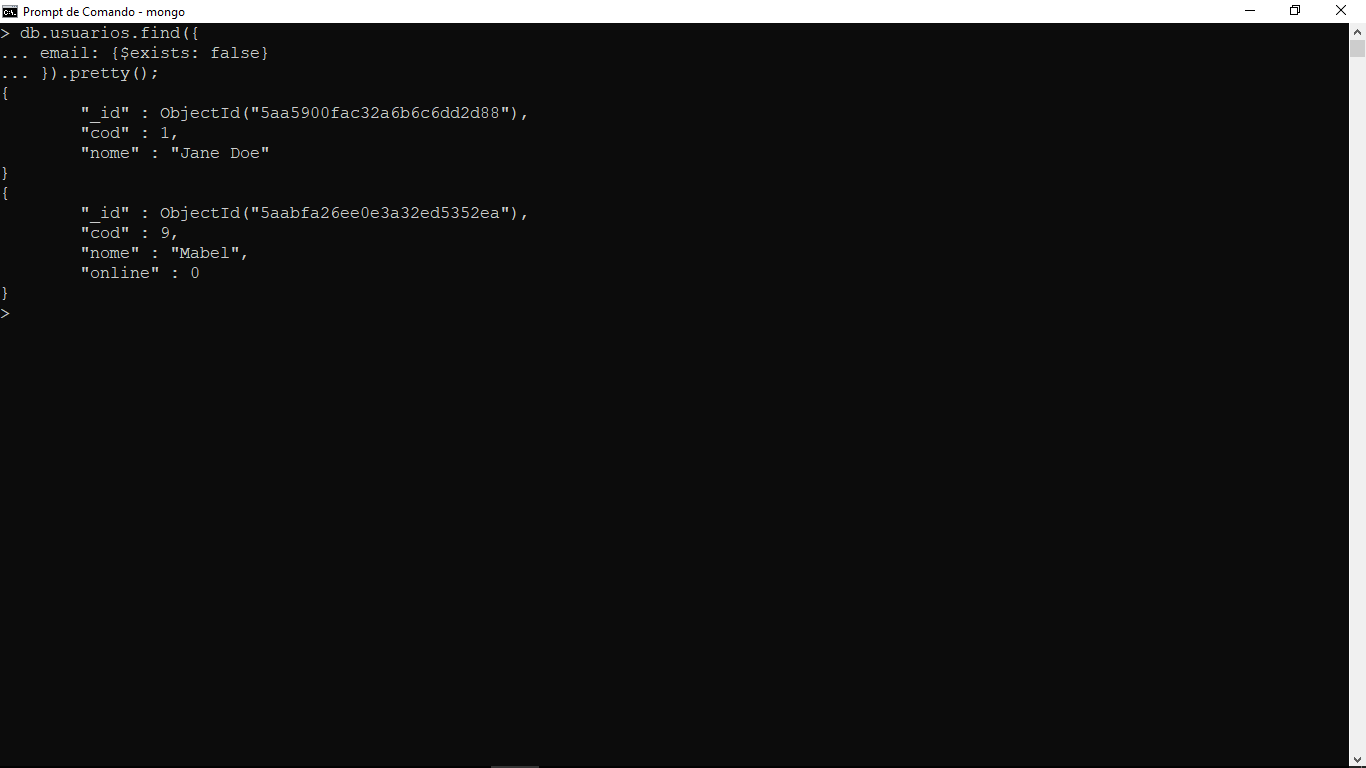


* db.usuarios.find({

email: {$exists: false}

}).pretty();

Retorna apenas os documentos ao qual o atributo email não exista.



## $not:

* db.alunos.find({

endereco: {

$not: {

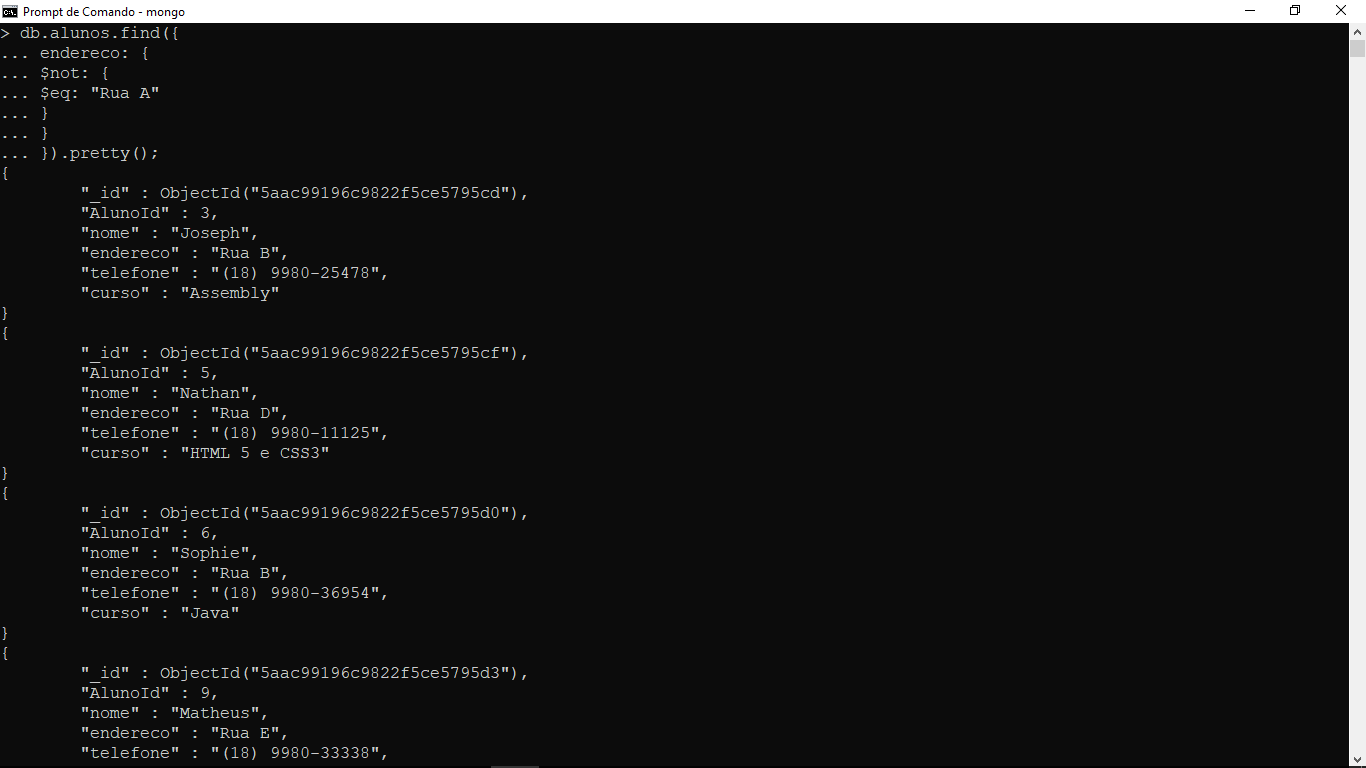
$eq: "Rua A"

}

}

}).pretty();

Retorna os documentos em que o endereco não seja “Rua A”.



## $nor:

* db.alunos.find({

$nor: [

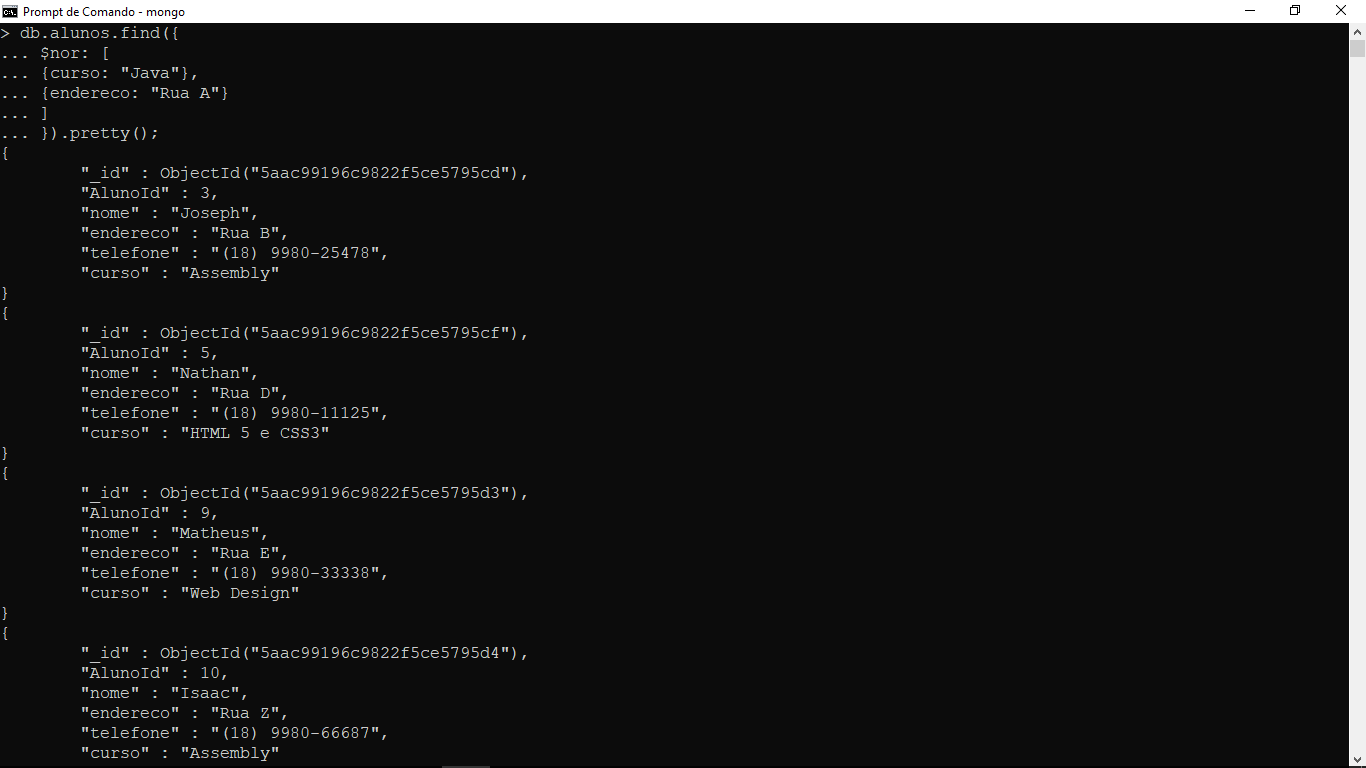
{curso: "Java"},

{endereco: "Rua A"}

]

}).pretty();

Retorna os documentos que não possuam um dos parâmetros passados.



# Operadores de Comparação:

## $eq:

* db.produtos.find({

quantidade: {

$not: {

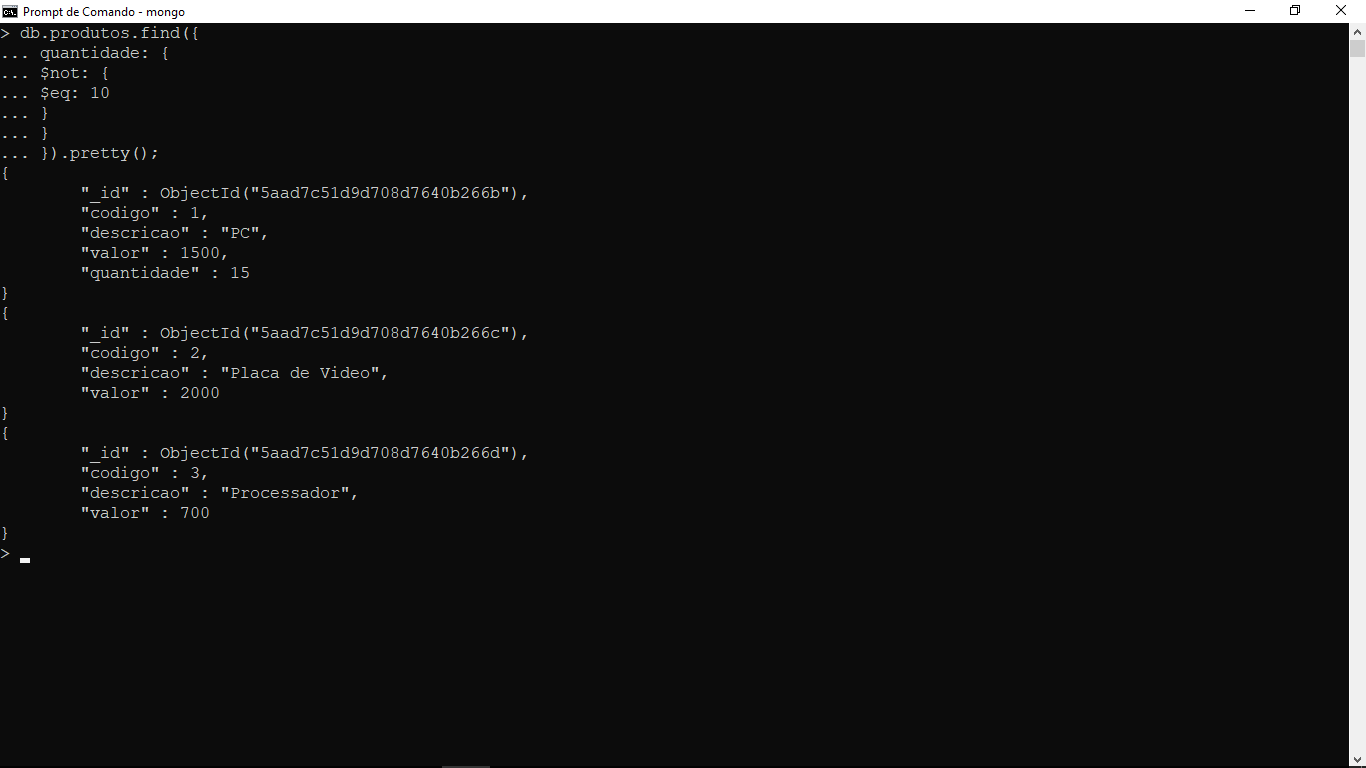
$eq: 10

}

}

}).pretty();

Este é o operador de igualdade geralmente é mais usado em conjunto com o operador $not.



## $ne:

* db.produtos.find({

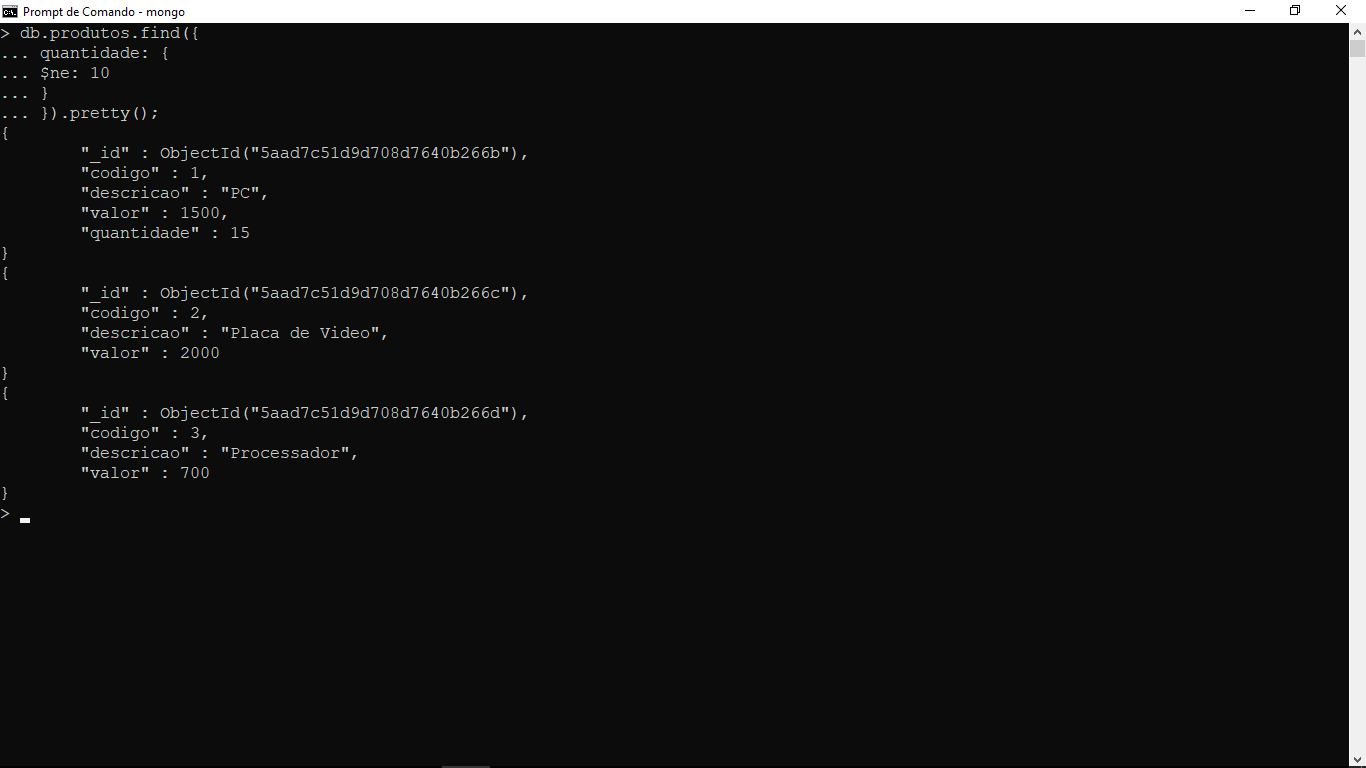
quantidade: {

$ne: 10

}

}).pretty();

Comando para negar igualdade, numa forma mais simplificada.



# Operadores de Maior e Menor que.

Operador Maior que:

## $gt:

* db.alunos.find({

AlunoId: {

$gt: 9

}

}).pretty();



Operador Menor que:

## $lt:

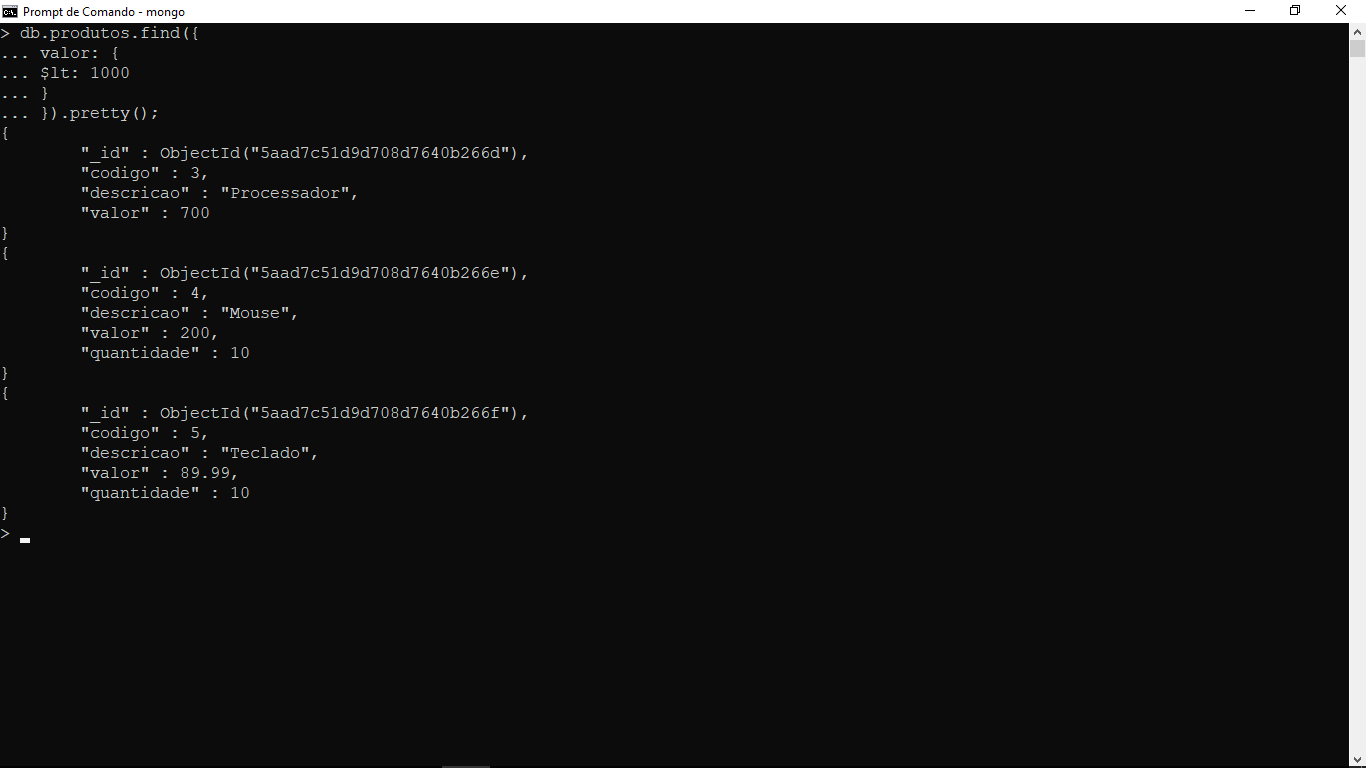
* db.produtos.find({

valor: {

$lt: 1000

}

}).pretty();



Operador Menor ou igual à:

## $lte:

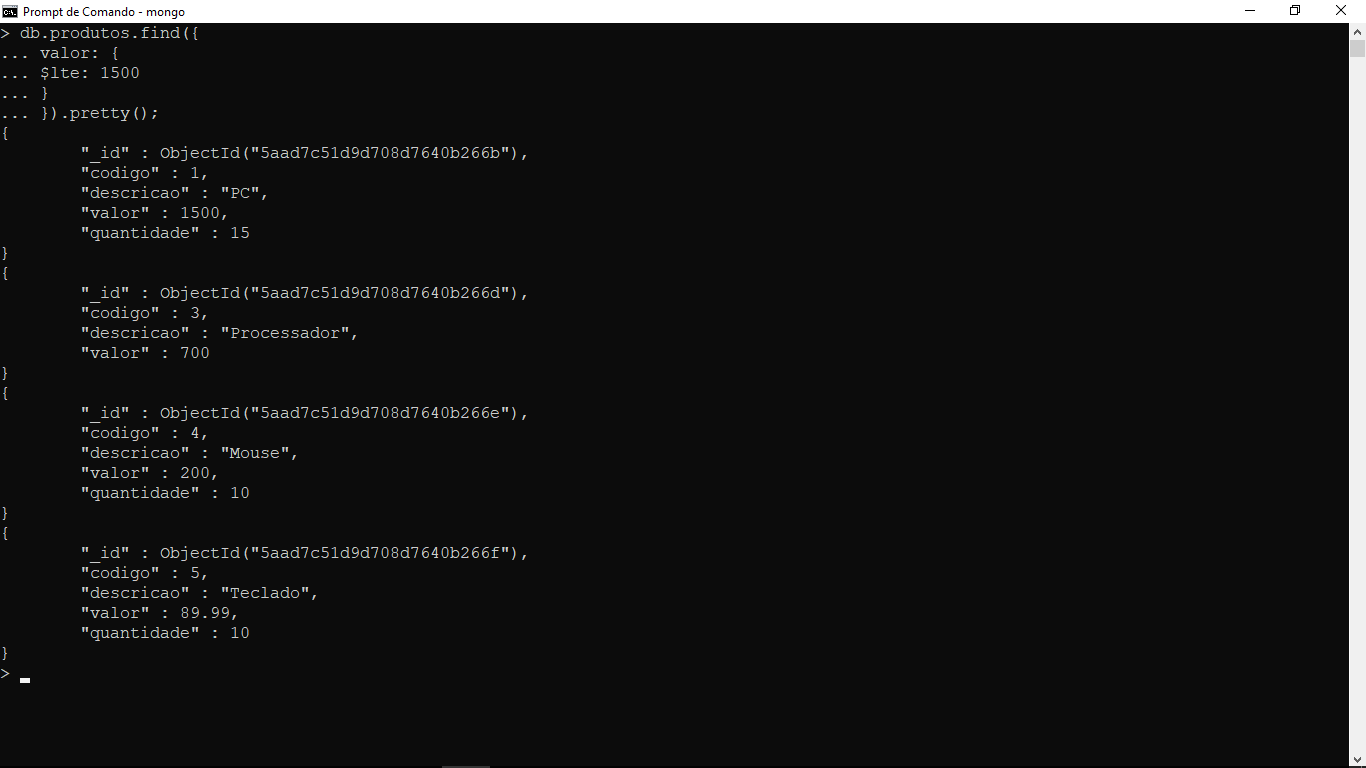
* db.produtos.find({

valor: {

$lte: 1500

}

}).pretty();



Operador Maior ou igual à:

## $gte:

* db.produtos.find({

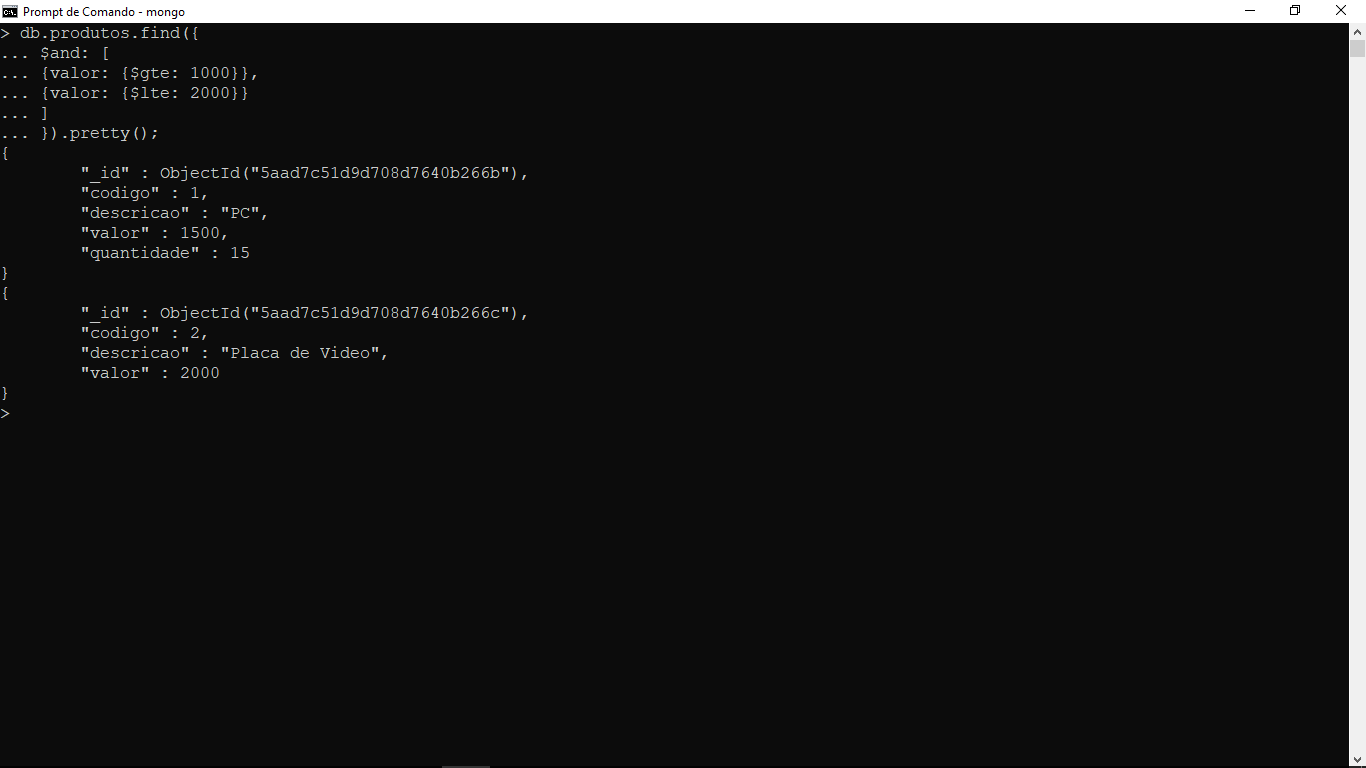
$and: [

{valor: {$gte: 1000}},

{valor: {$lte: 2000}}

]

}).pretty();



# **Projections.**

## Visualizar atributos específicos:

db.collection.find({}, {}).pretty;

A primeira chaves determina os valores a serem encontrados.

A segunda chaves determina as colunas que serão mostradas.

Exemplo:

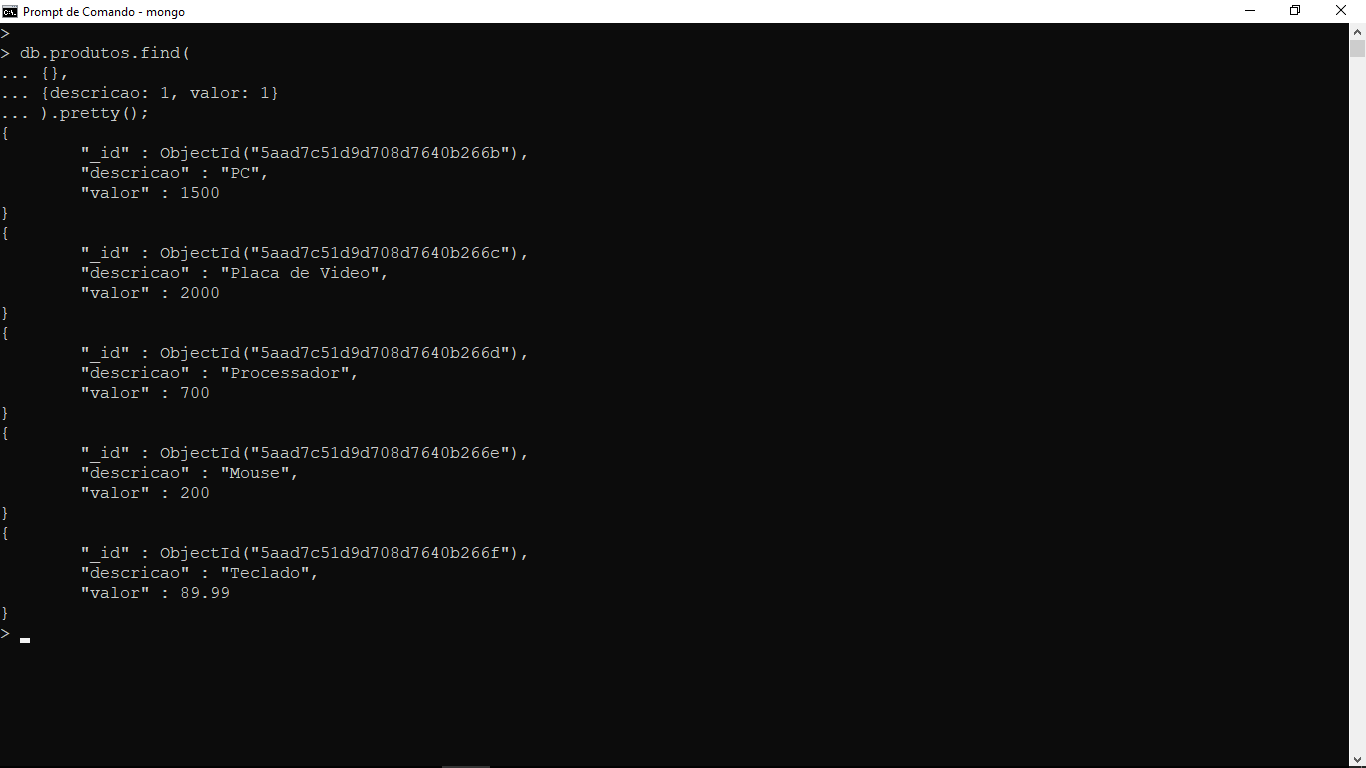
* db.produtos.find(

{},

{descricao: 1, valor: 1}

).pretty();

Apenas os atributos com valores 1 serão mostrados



## Como omitir colunas:

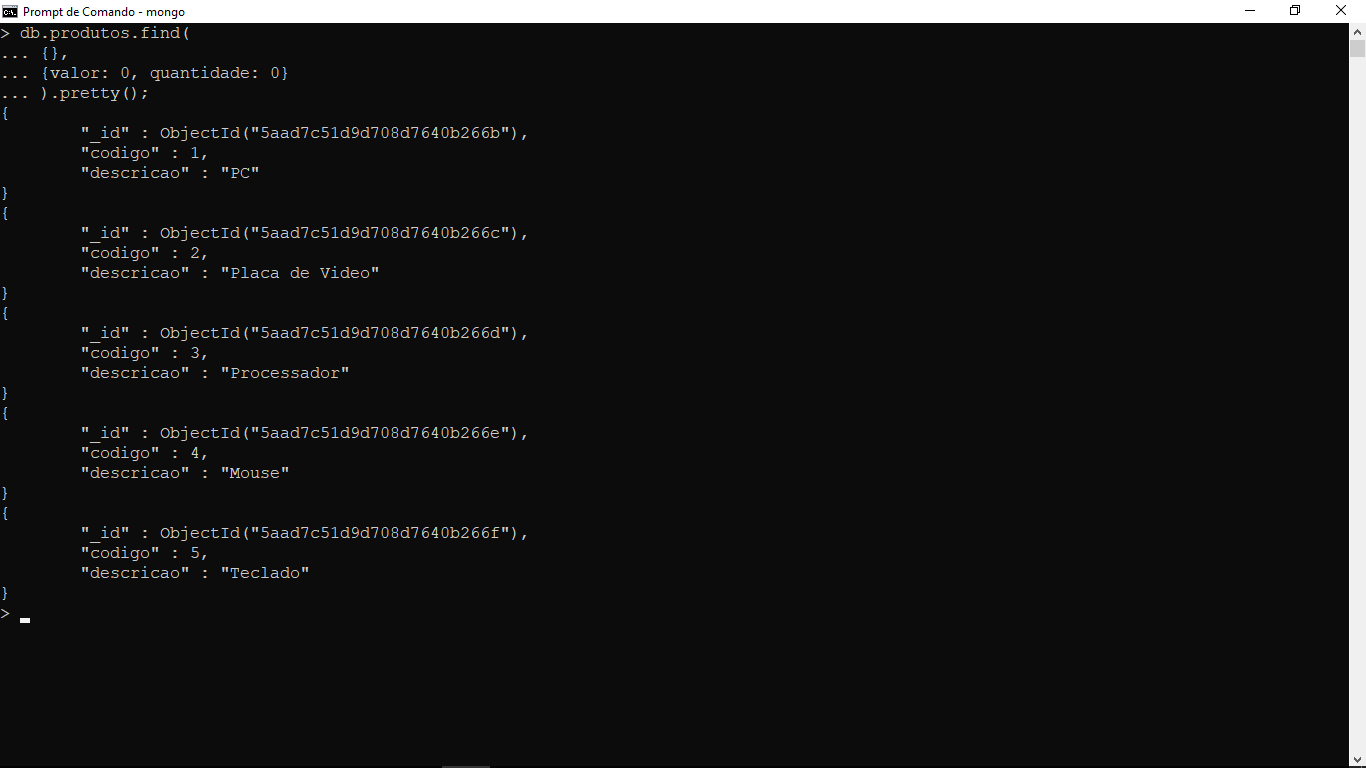
* db.produtos.find(

{},

{valor: 0, quantidade: 0}

).pretty();

Os atributos com valores 0 serão escondidos.



# Buscas Ordenadas:

SORT.

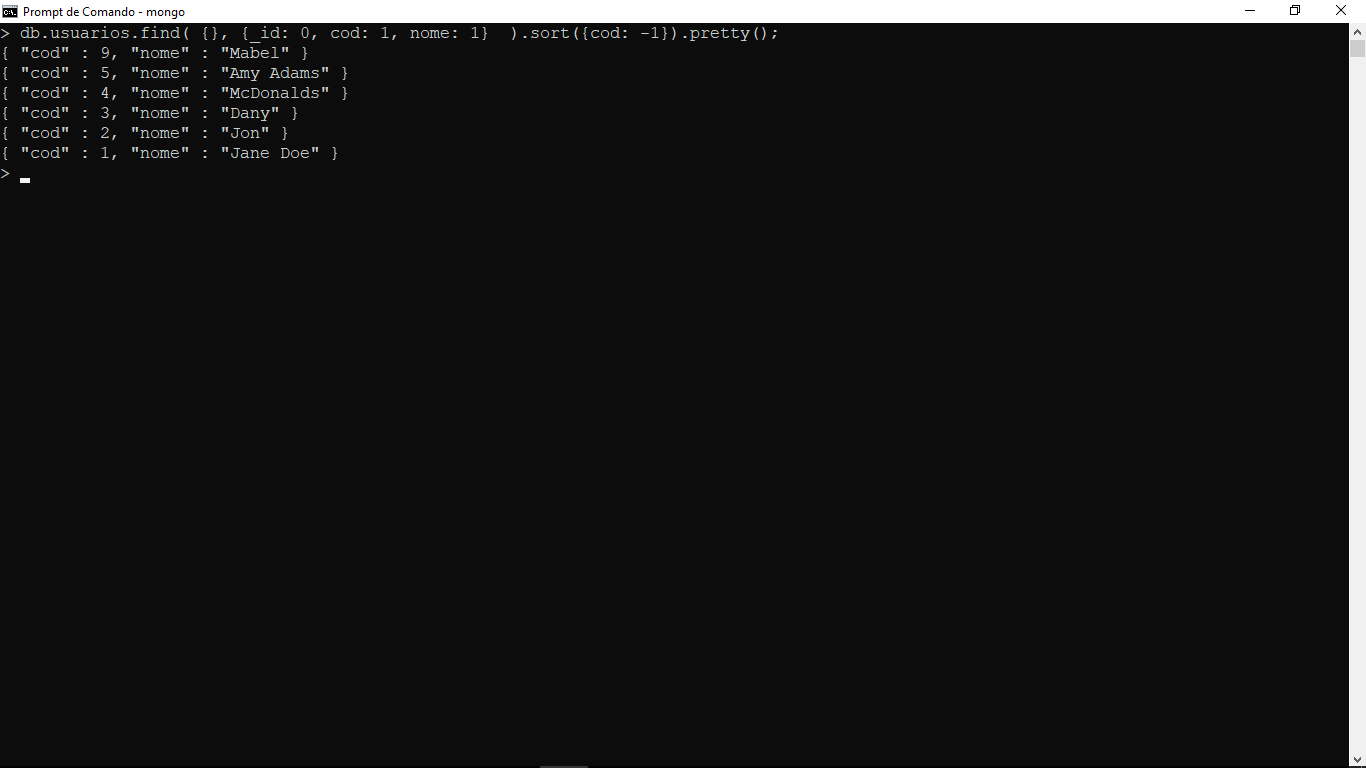
* db.usuarios.find(

{},

{\_id: 0, cod: 1, nome: 1}

).sort({cod: -1}).pretty();

O parametro “-1” indica que será ordenado em ordem decrescente.

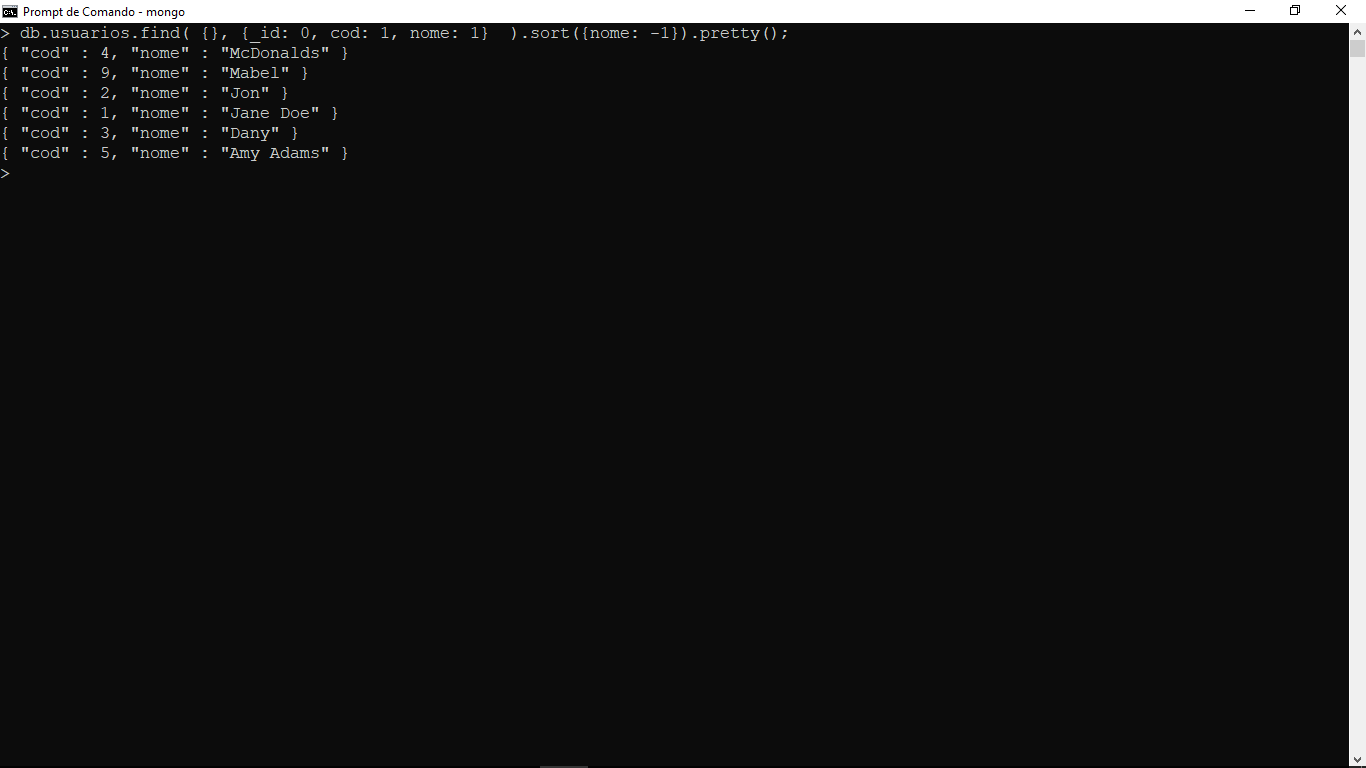


* db.usuarios.find(

{},

{\_id: 0, cod: 1, nome: 1}

).sort({nome: -1}).pretty();



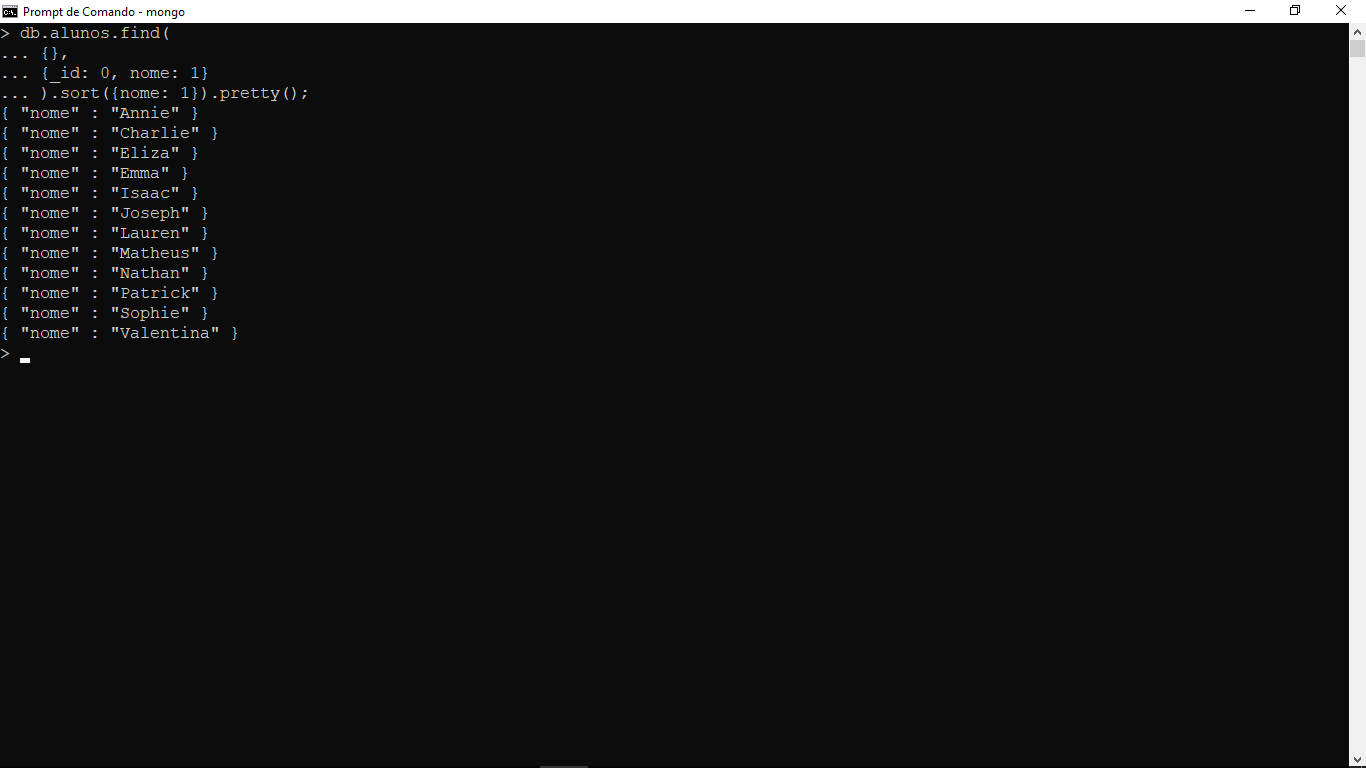
Ordenando em ordem Crescente:

* db.alunos.find(

{},

{\_id: 0, nome: 1}

).sort({nome: 1}).pretty();

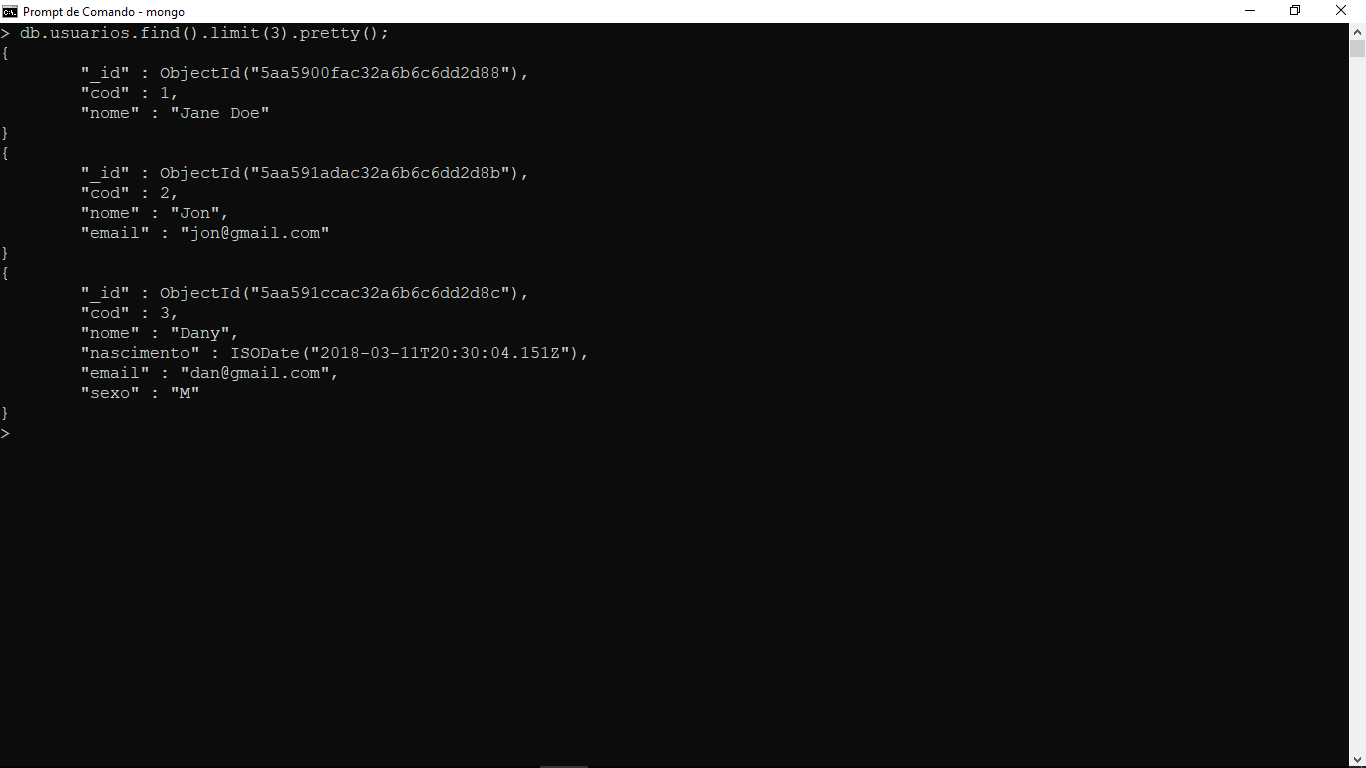


# **LIMIT.**

Limitando a quantidade de documentos que serão mostrados.

Exemplos:

* db.usuarios.find().limit(3).pretty();

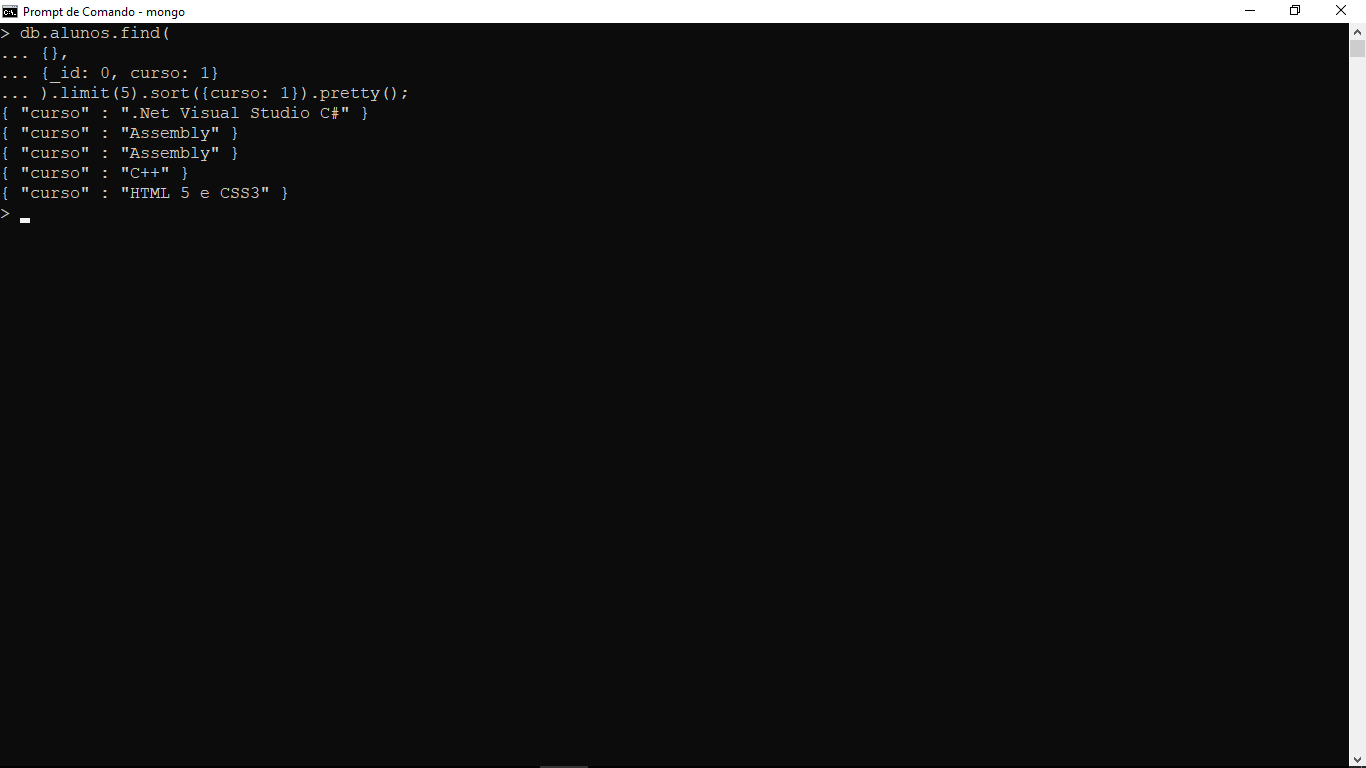


* db.alunos.find(

{},

{\_id: 0, curso: 1}

).limit(5).sort({curso: 1}).pretty();

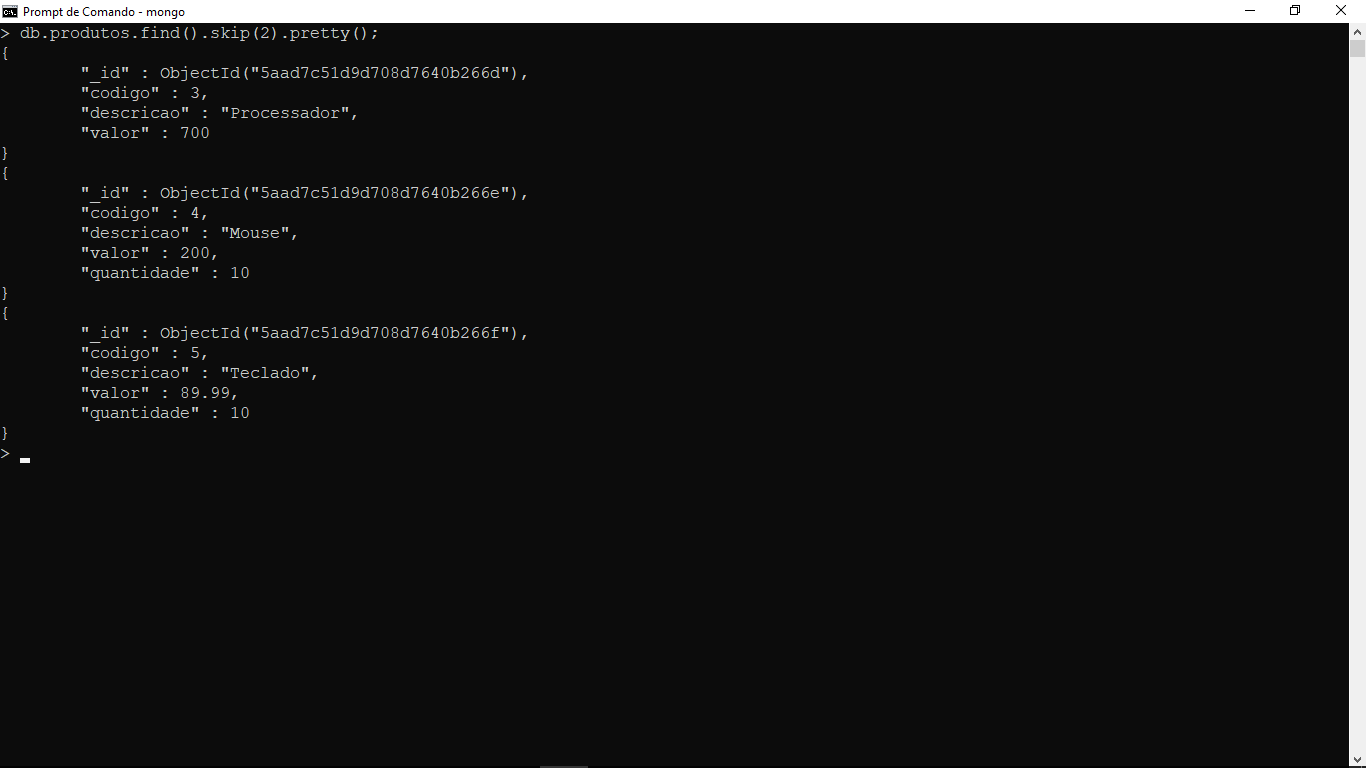


# **SKIP.**

Pula a quantidade de documentos que for passado como parametro.

Exemplos:

* db.produtos.find().skip(2).pretty();

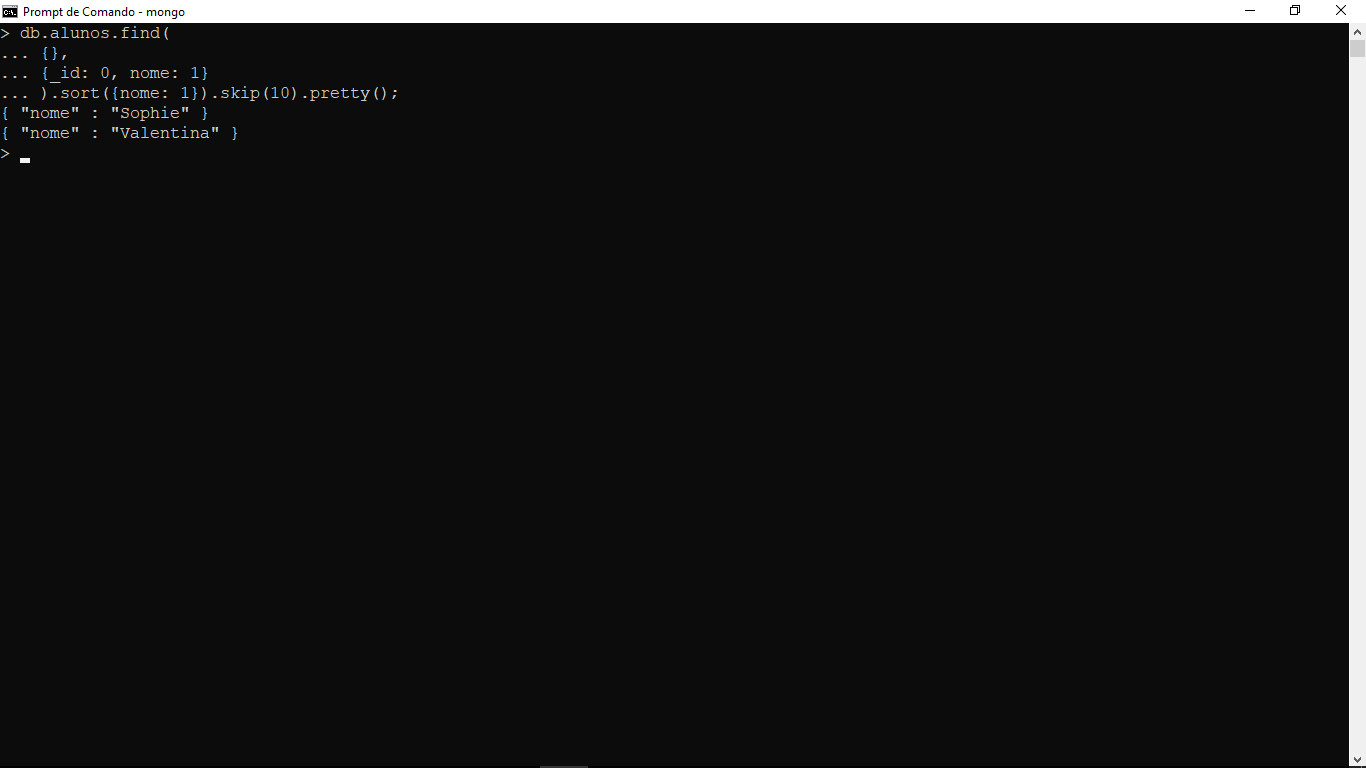


* db.alunos.find(

{},

{\_id: 0, nome: 1}

).sort({nome: 1}).skip(10).pretty();

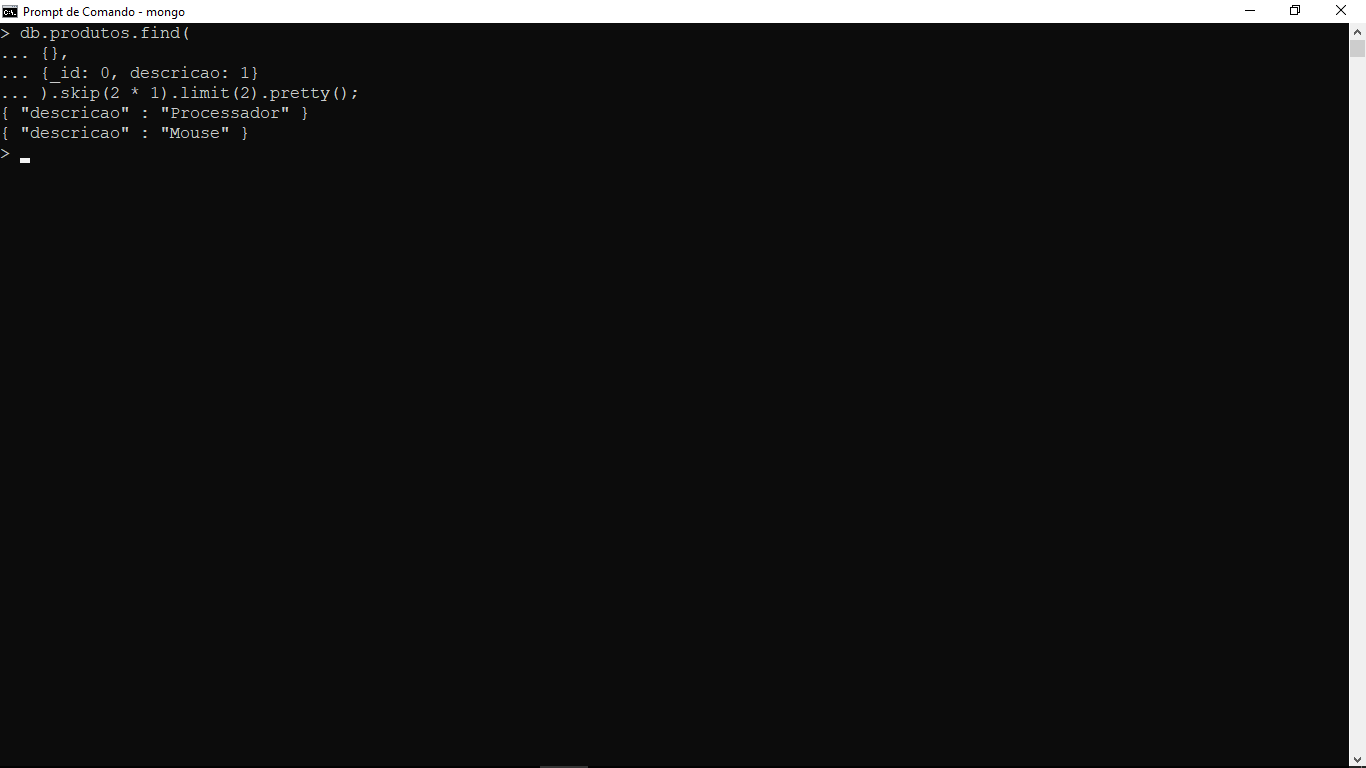


* db.produtos.find(

{},

{\_id: 0, descricao: 1}

).skip(2 \* 1).limit(2).pretty();

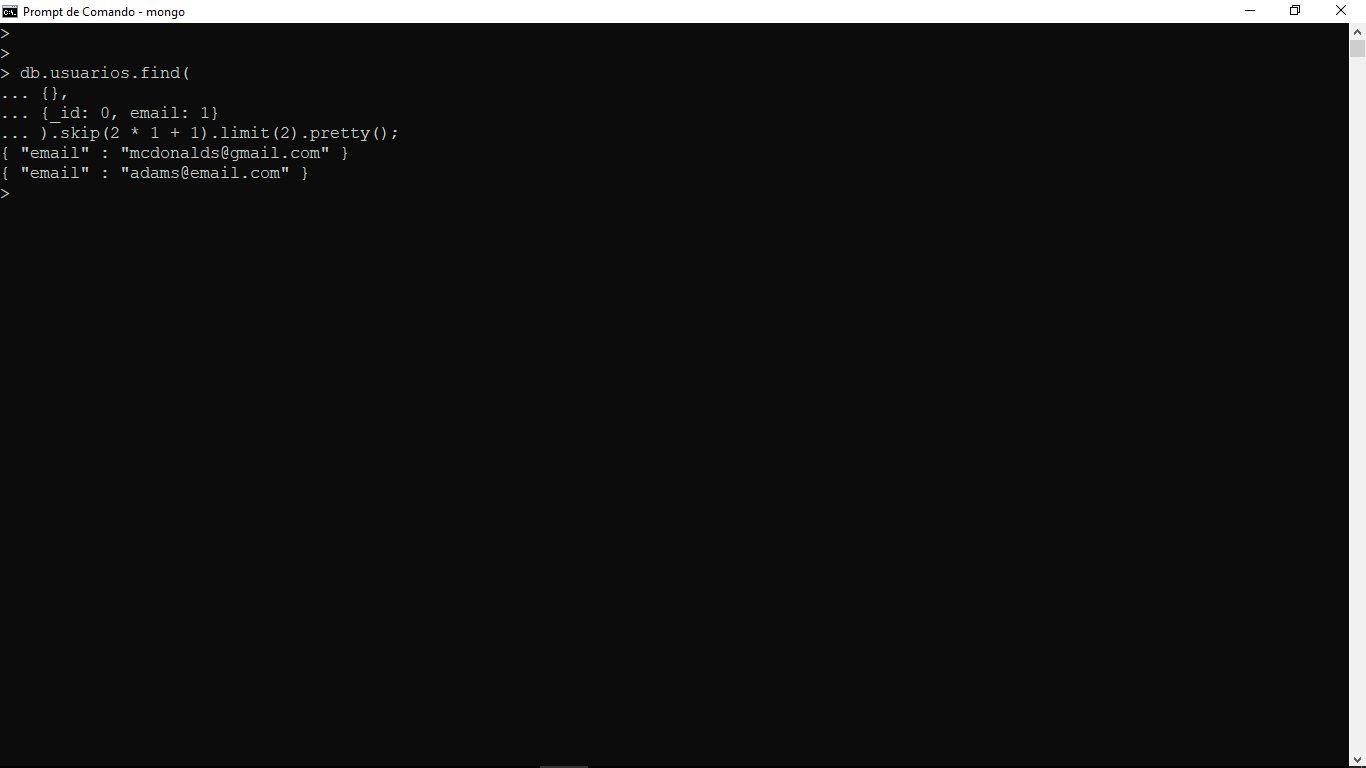


* db.usuarios.find(

{},

{\_id: 0, email: 1}

).skip(2 \* 1 + 1).limit(2).pretty();



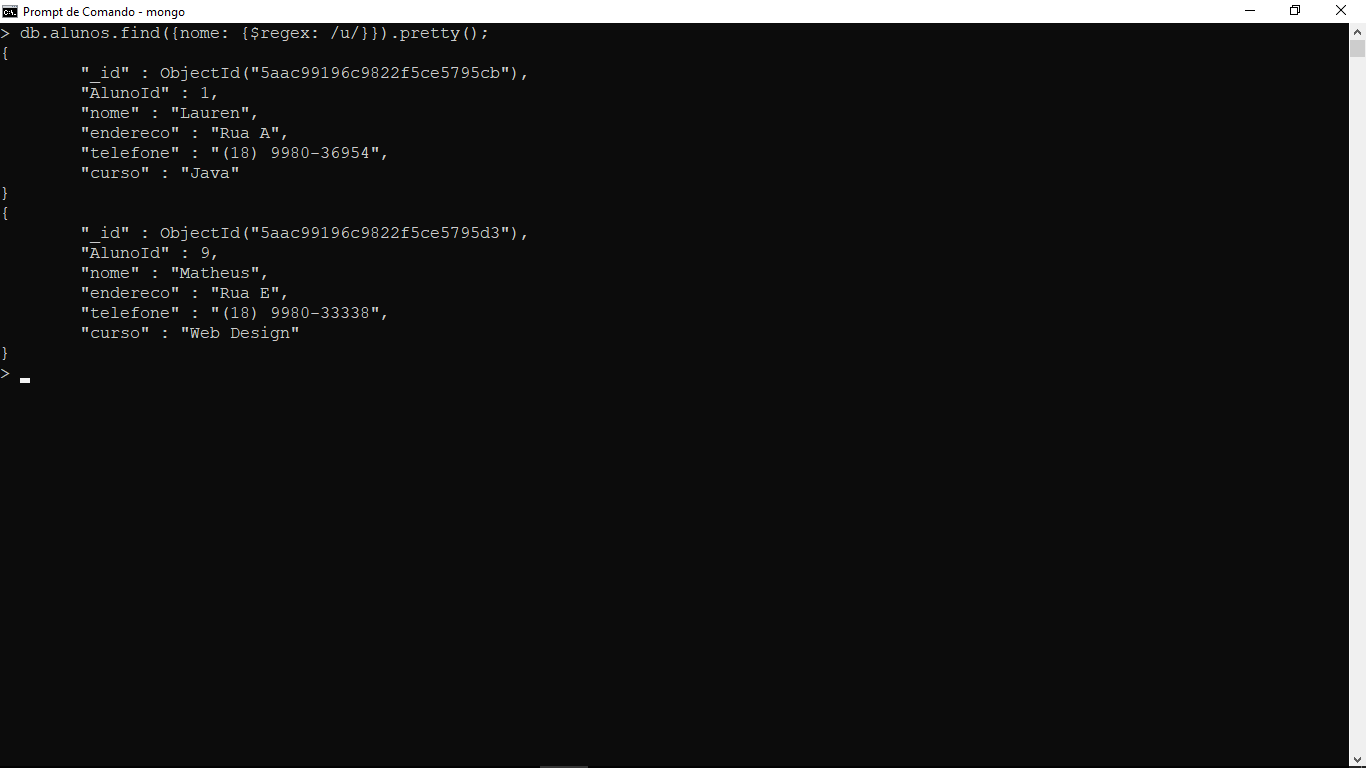
# Busca por palavras específicas:

## $regex:

* db.alunos.find({nome: {$regex: /u/}}).pretty();

“$regex” é semelhante ao “like” do MySQL.

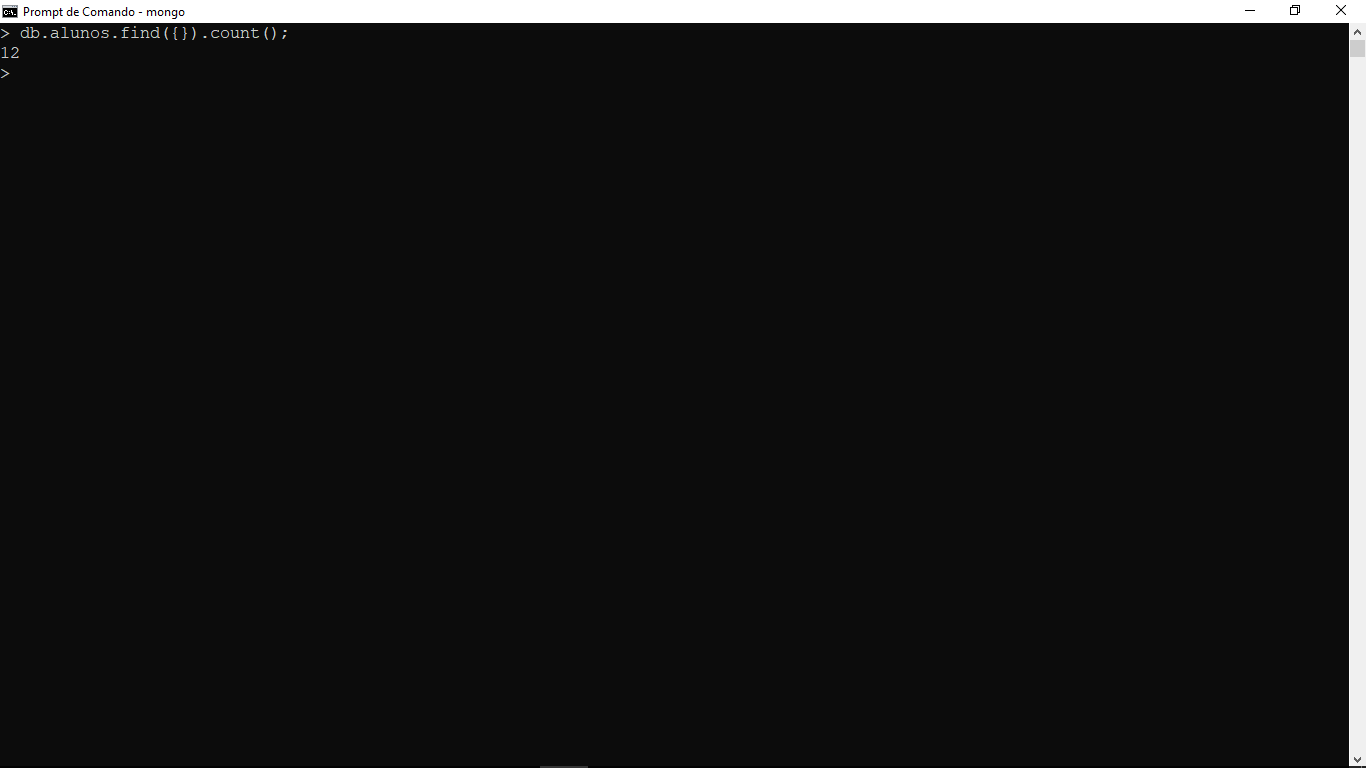
Neste exemplo busca apenas os nomes que possuam a letra “u”.



Contar documentos:

# **COUNT.**

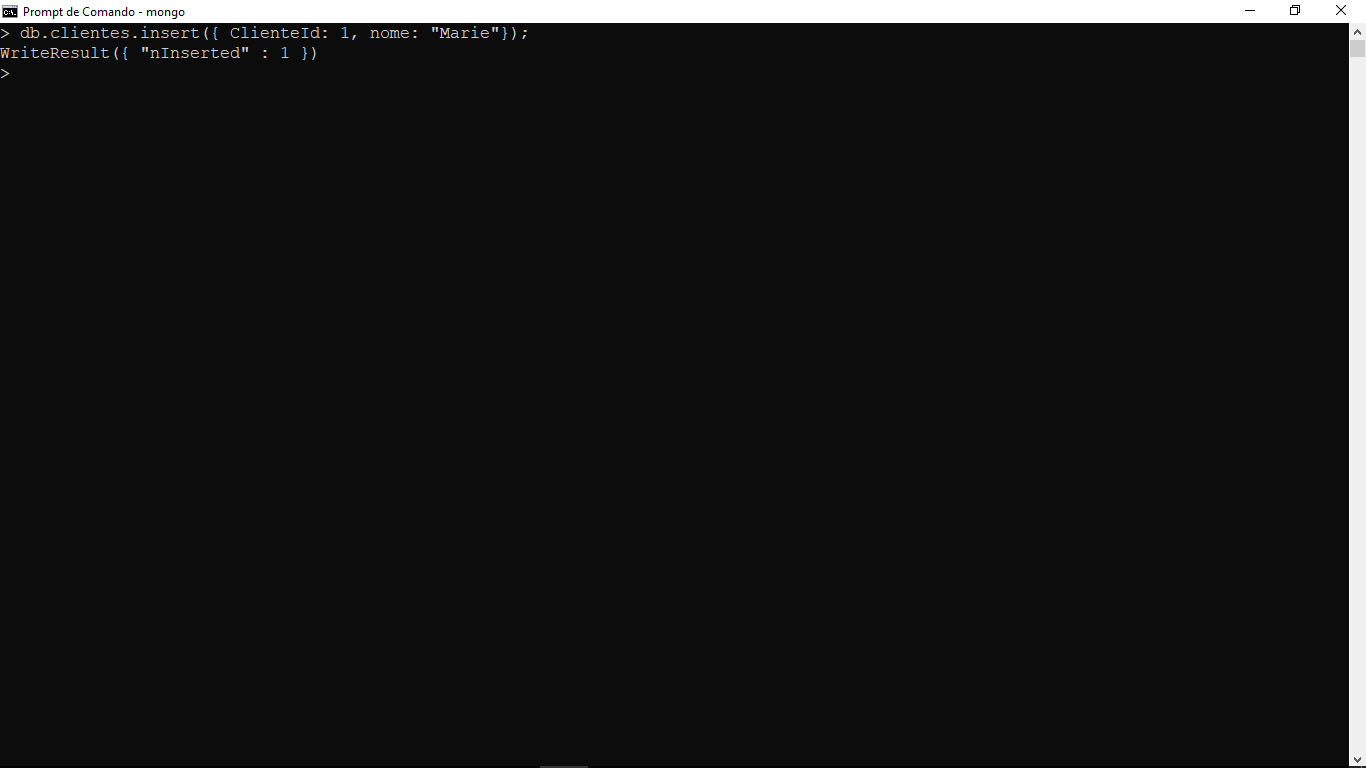
* db.alunos.find({}).count();



# **Associações.**

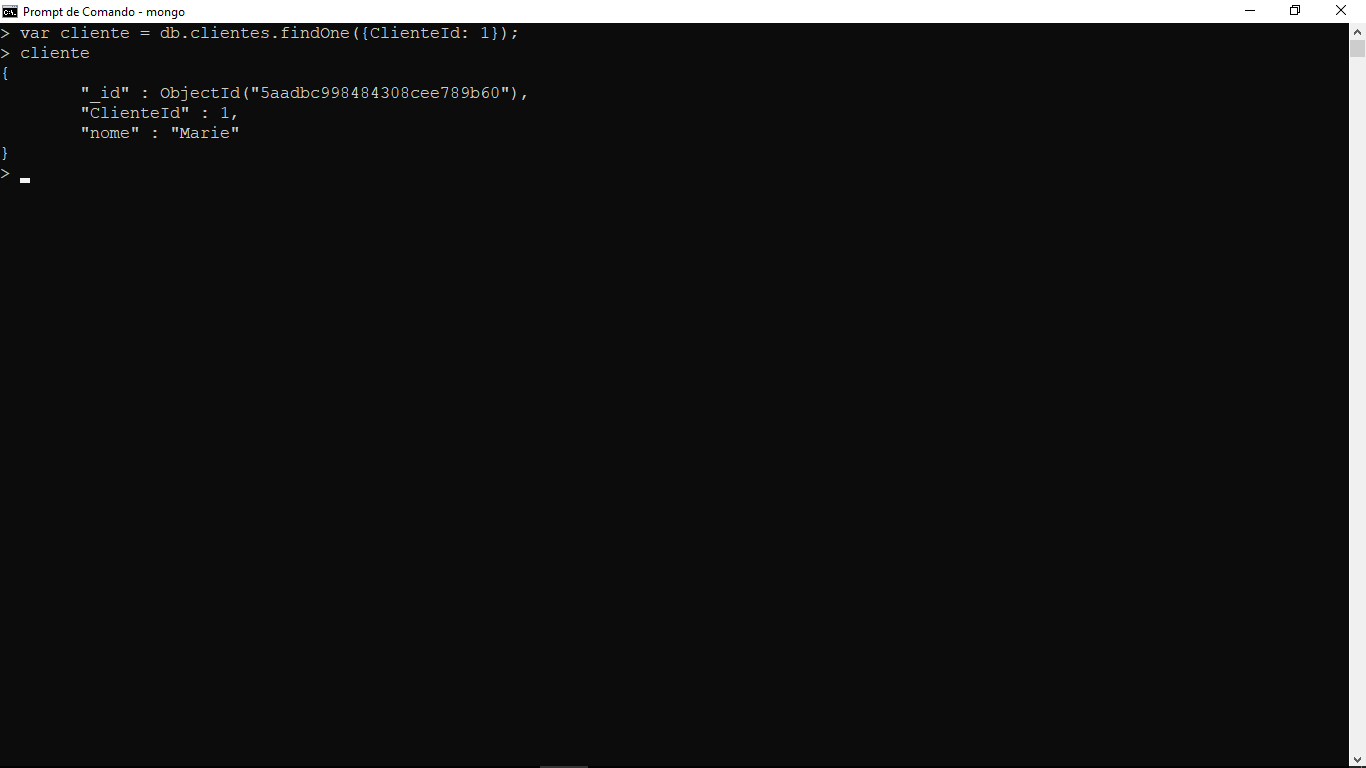
Criando uma coleção de clientes:

* db.clientes.insert({ ClienteId: 1, nome: "Marie"});



Salve este cliente numa variável:

* var cliente = db.clientes.findOne({ClienteId: 1});



Agora é necessário selecionar os produtos comprados e guardar numa variável.

Caso retorna mais de um objeto é necessário adicionar a função *“toArray()”* para salvar todos os dados na variável.

* var produtos = db.produtos.find(

{

$and: [

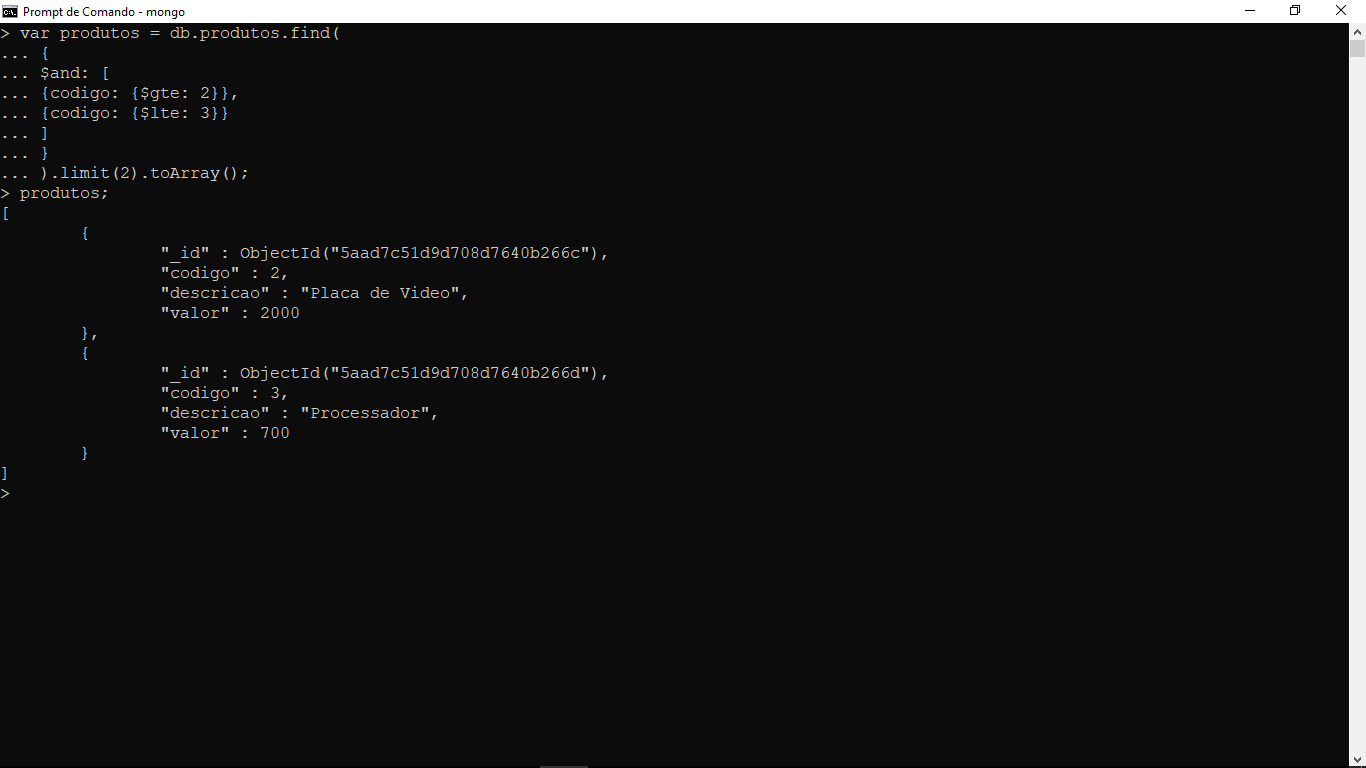
{codigo: {$gte: 2}},

{codigo: {$lte: 3}}

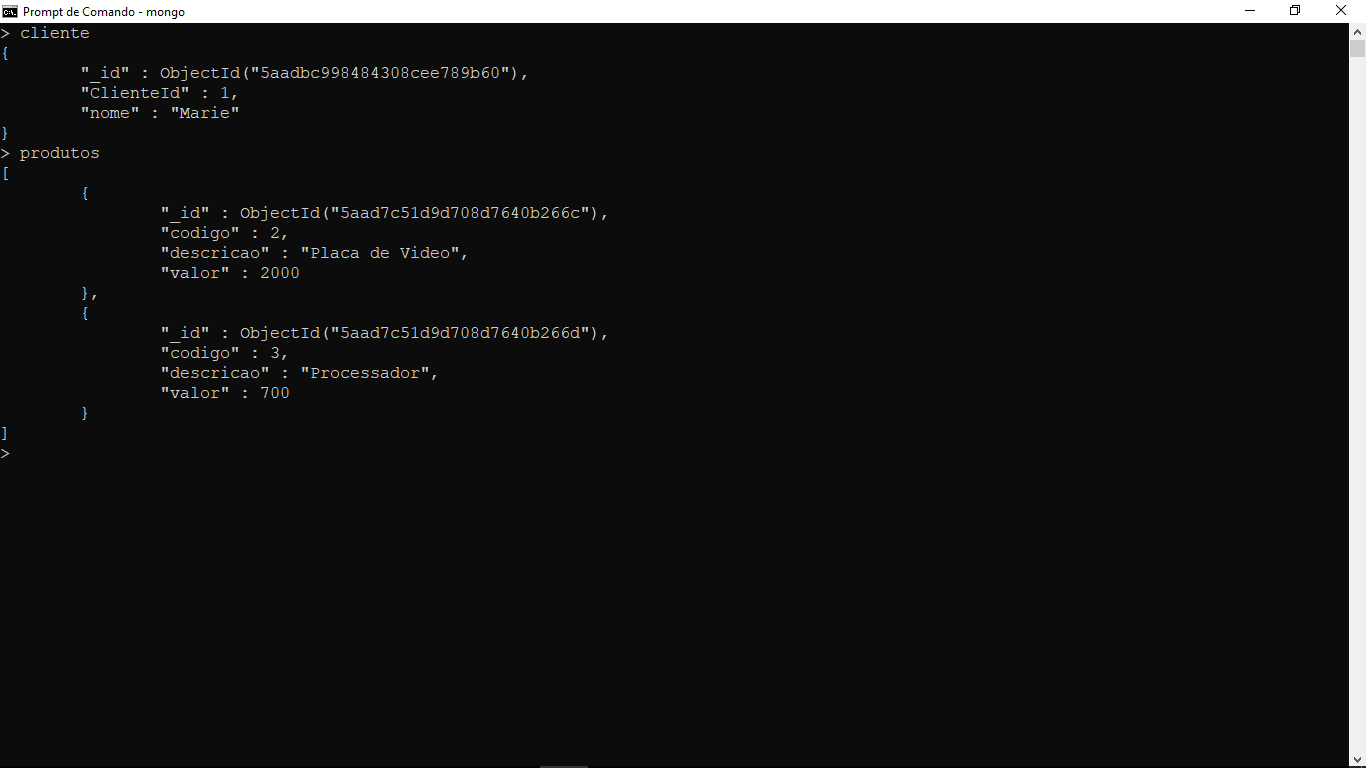
]

}

).limit(2).toArray();



Conferindo os dados das variáveis:



Agora é só inserir as variáveis na coleção de vendas:

* db.vendas.insert(

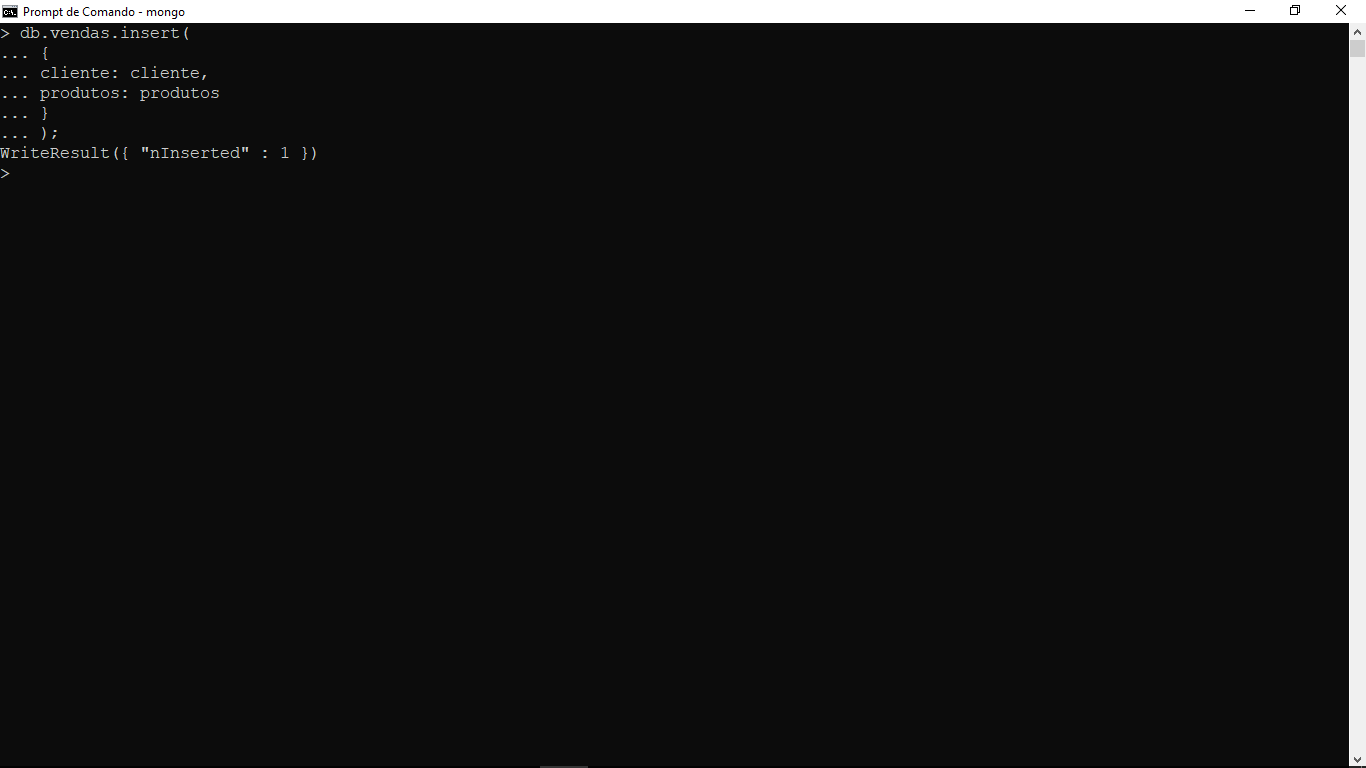
{

cliente: cliente,

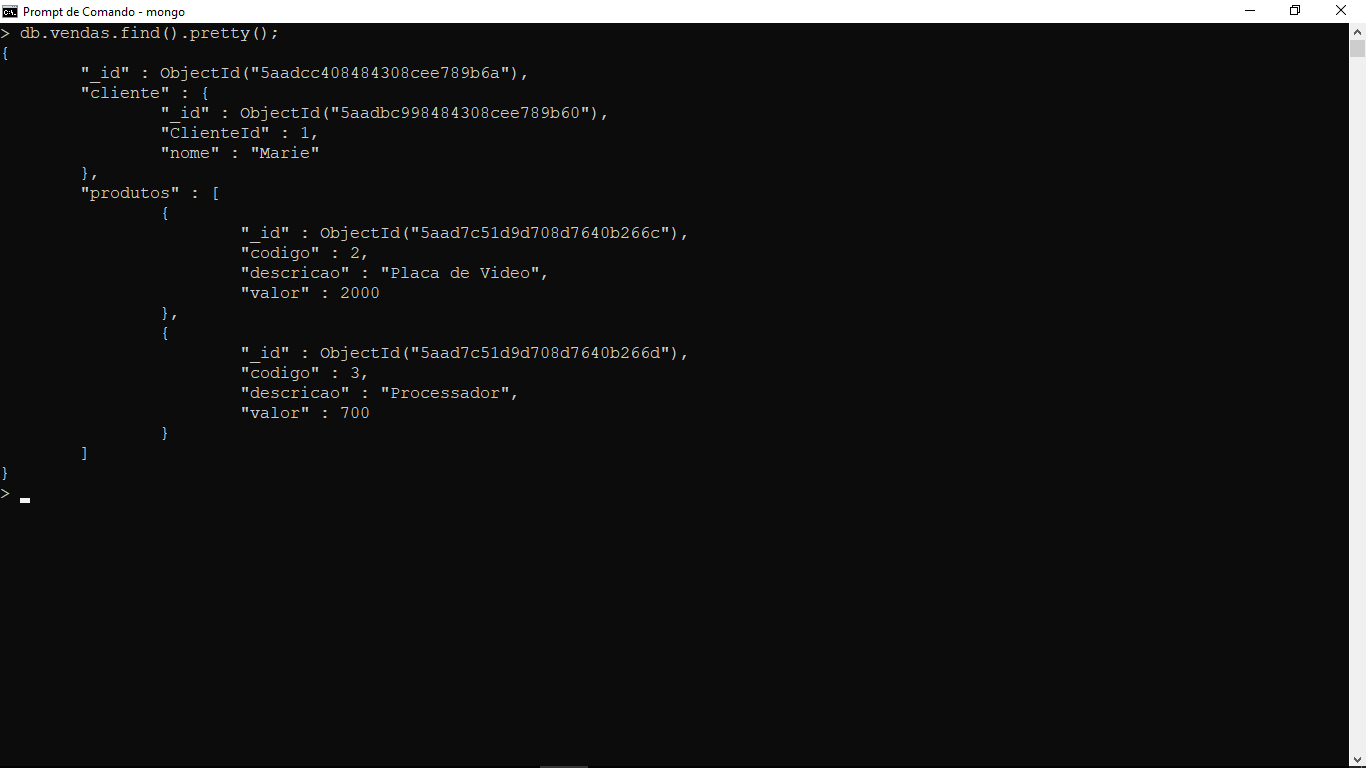
produtos: produtos

}

);



Visualizando os dados inseridos:



Venda realizada com sucesso!

# Fim.