# Banco de Dados NoSql (MongoDB)


## Introdução ao MongoDB

# Banco de Dados NoSql (MongoDB)

O **MongoDB** é um banco de dados de alta performance orientado a documentos. É poderoso, flexível e escalonável.

Ele não usa o conceito de tabelas, e sim, um conjunto de documentos em formato JSON. Com isso podemos modelar dados de forma mais natural (como serão utilizados em nossas aplicações), ao invés de criar várias ligações entre tabelas, o que lhe dá a característica de banco não-relacional. Isso significa que ao invés de ligar uma linha de uma tabela com a linha de outra tabela, podemos colocar nesta linha outro documento ou um *array*.

Não há *schemas* definidos. Vários documentos de uma mesma coleção podem ter formatos diferentes, diferente no mundo relacional, onde temos que ter os dados de uma mesma tabela seguindo um formato pré-definido.

## JSON e BSON

JSON (*JavaScript Object Notation*) é basicamente um objeto *JavaScript*, que possui uma chave (que define o nome do campo) seguida de seu respectivo valor.

Abaixo podemos ver uma representação de um objeto no formato JSON.

```
{
   "nome" : "João",
    "idade" : 15
}
```

Acima, temos um JSON onde o campo “nome” tem “João” como valor e o campo “idade” vale 15.

## JSON e BSON

BSON (Binary JSON) é uma representação binária do formato JSON. O MongoDB representa o JSON em binário, estendendo o seu modelo para oferecer mais tipos de dados além dos que fazem parte da especificação do JSON. Alguns tipos de dados que são disponibilizados pelo BSON são:

- string;
- integer;
- double;
- date;
- byte array;
- boolean;
- null;
- BSON object;
- BSON array;
- código JavaScript;
- MD5 Binary Data;
- expressão regular.

# Banco de Dados NoSql (MongoDB)

## Terminologias no MongoDB

**Base de dados:** O banco de dados é um contêiner físico para coleções.
**Coleção:** Coleção é um grupo de documentos do MongoDB. É o equivalente a uma tabela RDBMS.
**Documento:** Um documento é um conjunto de pares de valores-chave. Os documentos têm esquema dinâmico.

### Comparação com os nomes usados em SQL

| RDBMS | MongoDB |
|---|---|
| Banco de Dados | Banco de Dados |
| Tabela | Coleção |
| Linha | Documento |
| Coluna | Campo |
| Junção de tabela | Documentos incorporados |
| Chave Primária | Chave Primária (chave padrão _id fornecida pelo próprio mongodb) |
| mysqld/oracle | mongod |
| mysql/sqlplus | mongo |

# INSTALAÇÃO MONGO DB E FERRAMENTAS

https://www.mongodb.com/try/download/community

https://www.mongodb.com/try/download/shell

https://www.mongodb.com/try/download/compass

# Banco de Dados NoSql (MongoDB)

## Usando o mongodb (comandos)

- **bancos de dados (databases)**
  - show dbs - lista todos os bancos de dados, o alias desse comando é show databases;
  - use [nome-do-banco] - selecionar um banco de dados, ex.: use admin;
  - db - verifica qual o banco de dados em uso no momento;
  - use novoBanco - cria um banco de dados, mas só passa a existir efetivamente quando você cria uma collection e insere algum dado nela, se não o mesmo não estará disponível quando você listar os bancos, deixará de existir;
  - db.dropdatabase() - **apaga um banco de dados**, usar após selecionar use nome-do-banco que deseja;

# Banco de Dados NoSql (MongoDB)

**Usando o mongodb (comandos)**

- **collections (tabelas)**
  - show collections - mostra as collections;
  - createCollection() - cria uma collection, protótipo dela é createCollection("nomedatabela", opções),
    exemplo: db.createCollection("minhacolecao").
  - db.nome_da_colecao.find().pretty() - ler todos os dados de uma coleção,
    ex.: db.system.users.find().pretty() , ler todos os dados da coleção system.users, equivalente à select * from tabela. essa saída sairá formatada, se quiser os dados numa única linha, use sem o método .pretty(): db.system.users.find();
  - db.nome_da_colecao.insert() - insere dados numa coleção, ex.: db.minhacolecao.insert( { "_id" : 0, "site" : “siteexemplo", "url" : “siteexemplo.com.br", "content" : "sobre mongodb" } );
  - deleta uma coleção, ex.: db.minhacolecao.drop(),

# Banco de Dados NoSql (MongoDB)

**Criando uma collection(ela é criada automaticamente ao inserir os dados) e já inserindo dados:**

```
use bancoExemplo
db.createCollection("dados")
db.dados.insert({ "nome" : “Flavio”,”sobrenome”:”Mota”, "mail" : “flavio@professor.gov"})
```

**Verificando os dados inseridos**

```
db.dados.find().pretty()
{ "_id" : ObjectId("5f5adaa1a7febd9d7c7ff5b5"), "nome" : "Flavio", "sobrenome" : "Mota",
"email" : "flavio@professor.cova" }
```


# Operações de criação, atualização e deleção de itens

# Insert(): criando registros com o MongoDB

Para inserir dados utilizamos a função *insert*.

```
db.frutas.insert()
```

Como parâmetro da função, passamos o documento que queremos salvar:

```
db.frutas.insert({"nome": "laranja", "quantidade": 5})
```

O MongoDB automaticamente irá inserir uma chave *"_id"* no documento, caso não exista uma no documento a ser inserido.
Você ainda pode inserir uma série de registros ao mesmo tempo: basta você passar um *array* de documentos para a função:

```
db.frutas.insert([
   {"nome": "morango", "quantidade": 10},
   {"nome": "banana", "quantidade": 7},
   {"nome": "abacaxi", "quantidade": 3}
])
```

## Remove(): apagando registros com o MongoDB

Para remover elementos do MongoDB, utilizamos o comando “deleteOne, deleteMany”:

Como parâmetro, passamos um documento. Dentro dele, especificamos os critérios a serem seguidos para a remoção:

**db.frutas.deleteOne({"nome": "abacaxi"})**

O comando acima irá remover todos os documentos da coleção *frutas* que tenha como *“nome”* o valor *“abacaxi”*.

Exclusão por parâmetros:

**db.clientReact.deleteMany({"id":{$gt:25}})**

Apaga todos os cliente com o **id** maiores que 25

Para apagar todos os documentos usamos:

**db.frutas.drop()**

# Update(): atualizando registros com o MongoDB

O MongoDB permite modificar documentos em uma coleção. Existem várias formas de usar o **update**, e em versões mais recentes do MongoDB, a recomendação é usar o método **updateOne** ou **updateMany** com o operador **$set** para fazer atualizações específicas. Aqui estão alguns exemplos de como você pode usar o **update** no seu conjunto de documentos:

A função update recebe 2 parâmetros: O primeiro é a condição para o MongoDB achar o documento que precisa ser atualizado. O segundo é o novo documento, com as informações atualizadas:

```
db.suaColecao.updateOne(
  { "email": "amilda.juca@gmail.com" },
  { $set: { "name": "Novo Nome" } }
);  //atualiza um documento
```

```
db.suaColecao.updateMany(
  { "email": "amilda.juca@gmail.com" },
  { $set: { "name": "Novo Nome" } }
); //atualiza todos os documentos
```

```
db.frutas.updateOne({"nome": "banana"}, {$set : {"quantidade": 9} })
```

O comando acima irá localizar o documento cujo “nome” seja “banana” e irá atualizar o atributo “quantidade” com o valor 9, caso não exista será criado.

# Banco de Dados NoSql (MongoDB)

Se quisermos remover um campo, podemos utilizar o modificador ***$unset***. Ele é responsável por remover campos do documento:

```
db.frutas.updateOne({nome: "morango"},  { $unset : {"preco": 1}  }  )
```

Também há o modificador ***$inc***, utilizado para campos em que temos valores numéricos que sempre serão incrementados ou decrementados:

```
db.frutas.updateOne({nome: "morango"},  { $inc : {"quantidade": 1}  }  )
```

Temos o ***$push***, que adiciona um elemento ao final do *array*. Se o campo não existir, ele será criado.

```
db.frutas.updateOne({nome: "banana"}, {$push: {tipo: "nanica"}})
```

Se quisermos inserir vários itens na nossa lista, utilizamos o sub-operador ***$each***.

```
db.frutas.updateOne({nome: "banana"}, {$push: {tipo: {$each:  ["maca", "terra"] } } })
```

Se não quisermos itens duplicados, podemos utilizar o modificador ***$addToSet*** ao invés do ***$push***

Para remover elementos do array, utilizamos o modificador ***$pop***.

```
db.frutas.updateOne({nome: "banana"}, {$pop: {tipo: 1} } })
```

# Banco de Dados NoSql (MongoDB)

Você pode remover um elemento partindo de um critério. Para isso utilizamos o modificador ***$pull***.

```
db.frutas.updateOne({nome: "banana"}, {$pull: {tipos: "maca"} })
```

Para alterar um item de uma determinada posição, utilizamos a notação de “ponto”:

```
db.frutas.updateOne({nome: "banana"}, {$set: {"tipo.1": "Prata"} })
```

A função *update* apenas altera o documento que for encontrado de acordo com os parâmetros de busca. Se nada for encontrado, nada acontecerá. Caso você queira que um documento seja criado caso nada seja encontrado, você pode passar o valor *true* como 3º parâmetro para a função:

```
db.frutas.updateOne({nome: "morango"}, {$set: {quantidade: 100}}, true)
```

A função *update* apenas altera o primeiro documento encontrado. Para alterar todos os documentos que batam com a condição, passe “true” como 4º parâmetro:

```
db.frutas.updateOne({nome: "morango"}, {$set: {quantidade: 100}}, false, true)
```


# Operações de listagem e busca de itens

# Banco de Dados NoSql (MongoDB)

## Operações de listagem e busca de itens

| SQL | MongoDB |
|---|---|
| SELECT | find e findOne |
| INSERT | insertOne e insertMany |
| DELETE | remove |
| UPDATE | update e updateOne |
| TOP, LIMIT (dependendo do fornecedor) | limit |
| ORDER BY | sort |
| CREATE INDEX | createIndex |
| LIKE | find usando expressões regulares |
| CREATE TABLE | não precisa, a coleção é criada ao inserir o primeiro documento |

| | |
|---|---|
| CREATE DATABASE | não precisa, a database é criada quando a primeira coleção é criada |
| ALTER TABLE | existem diversas funções para alterar a estrutura dos seus documentos |
| COUNT | count |
| GROUP BY | aggregate |
| DISTINCT | distinct |

# Find(): buscando registros com o MongoDB

Para fazermos uma busca, nós utilizamos a função *find*.

**db.frutas.find()**

Para que o MongoDB exiba os JSONs de uma forma mais legível no console, utilizamos a função *pretty()* no final do comando:

**db.frutas.find().pretty()**

A função *find* espera como primeiro parâmetro um JSON que defina o que estamos buscando, um *critério* de busca:

**db.frutas.find({nome: "morango"}).pretty()**

Podemos colocar outros campos também:

**db.frutas.find({nome: "morango", quantidade: 10}).pretty()**

Quando fazemos uma consulta em um campo com dados do tipo *string*, também podemos usar expressões regulares.

**db.frutas.find({nome: /mor/}).pretty()**

## Limit(), skip() e sort(): buscando registros com o MongoDB

Para limitar o número de resultados podemos utilizar a função *limit*.

**db.frutas.find().limit(2).pretty()**

Se quisermos retornar só o primeiro documento, podemos usar a função *findOne* no lugar de *limit(1)*.

**db.frutas.findOne()**

Para ordenar os resultados, temos a função *sort*. Nela, passamos qual campo queremos organizar e se é na ordem crescente (1) ou decrescente (-1):

**db.frutas.find().sort({nome: 1}).pretty()**
**db.frutas.find().sort({quantidade: -1}).pretty()**

Para pular resultados, utilizamos a função *skip*.

**db.frutas.find().skip(2).pretty()**

Podemos juntar *limit* com *skip* para criar paginações, por exemplo!

**db.frutas.find().skip(1).limit(2).pretty()**

# Banco de Dados NoSql (MongoDB)

## Quais chaves retornar?

No método *find*, podemos passar como segundo parâmetro os campos que desejamos que sejam retornados:

```
db.frutas.find({}, {"nome": 1, "quantidade": 1}).pretty()
```

O campo *_id* já vem por padrão. Se não o quisermos, podemos colocar o campo *_id* com valor 0.

# Banco de Dados NoSql (MongoDB)

## Operadores de comparação

Para compararmos valores, utilizamos os seguintes operadores de comparação:

| Operador no MongoDB | Operador de comparação correspondente |
|---|---|
| $lt | < |
| $lte | <= |
| $gt | > |
| $gte | >= |
| $eq | = |
| $ne | != |

Por exemplo...

**db.frutas.find({quantidade: {$gt: 5, $lt: 10} }).pretty()**

Aqui, combinamos dois operadores para criar um intervalo de valores: queremos todas as frutas onde a quantidade seja maior que 5 e menor que 10. Esse tipo de consulta é muito utilizado quando estamos trabalhando com datas.

Um outro exemplo:

**db.frutas.find({nome: {$ne : 'morango' } }).pretty()**

Acima, buscamos todas as frutas cuja chave *nome* não seja "morango".

# Banco de Dados NoSql (MongoDB)

## Operadores lógicos ou condicionais

Nós também temos operadores lógicos ou condicionais no MongoDB:

| Operador condicional no MongoDB | Operador condicional correspondente |
|---|---|
| $or | OR (\|\|) |
| $and | AND (&&) |
| $not | NOT (!) |
| $nor | NOR |

Por exemplo:

**db.frutas.find( {$or:  [ {nome: "morango"}, {quantidade: 5} ]  } ).pretty()**

Acima, buscamos as frutas cujo nome igual a morango OU quantidade igual a 5.

## Acessando documentos internos

Para acessar campos de documentos internos, podemos utilizar a notação de ponto:

```
db.frutas.find( { "tipos.nome" : "prata" } ).pretty()
```

E se precisarmos verificar mais de um campo dos documentos do array “tipos”? Teríamos que escrever “tipos.nome” e “tipos.quantidade”?

```
db.frutas.find(  {"tipos.nome": "prata", "tipos.quantidade": 2}  ).pretty()
```

O correto seria utilizar o $elemMatch! A consulta ficaria assim:

```
db.frutas.find(  {"tipos":  {$elemMatch: {"nome": "prata", "quantidade": 2}  } }  ).pretty()
```

Assim não obrigamos o MongoDB entrar em “tipos” 2 vezes. Ele entra 1 vez e já faz as 2 verificações.

## Lidando com Arrays

No exemplo anterior, buscamos quando o campo de um documento interno tinha determinado valor. Mas não indicamos qual documento interno, ele podia pegar de qualquer posição do *array tipos*.

Se precisarmos indicar uma posição específica, podemos utilizar a notação de ponto para indicar uma determinada posição no *array*:

```
db.frutas.find( {"tipo.0.nome" : "nanica" } ).pretty()
```

Se quisermos um documento que contenha todos nomes especificados tenha, utilizamos o operador ***$all***.

```
db.frutas.find( { "tipo.nome": {$all : ["prata", "nanica"]  }  } ).pretty()
```

A ordem não é importante, mas TODOS os valores passados no *array* têm de estar presentes. É como se fosse um operador condicional AND. Se utilizássemos o *$in* no lugar de *$all*, estaríamos fazendo uma condição OR, e o documento seria retornado mesmo que não tivesse um valor que pedimos.

```
db.frutas.find( { "tipo.nome": {$all : ["prata", "nanica", 53]  }  } ).pretty() /*não retorna nada */
db.frutas.find( { "tipo.nome": {$in : ["prata", "nanica", 53]  }  } ).pretty()
```

## Lidando com arrays

Também podemos buscar documentos pelo tamanho de seu array com ***$size***.

```
db.frutas.find({ "tipo": {$size : 2}  }).pretty()
```

Também podemos limitar a quantidade de elementos de um array com o ***$slice***.

```
db.frutas.find({}, {"tipo": {$slice: 1}}).pretty()
```

# Banco de Dados NoSql (MongoDB)

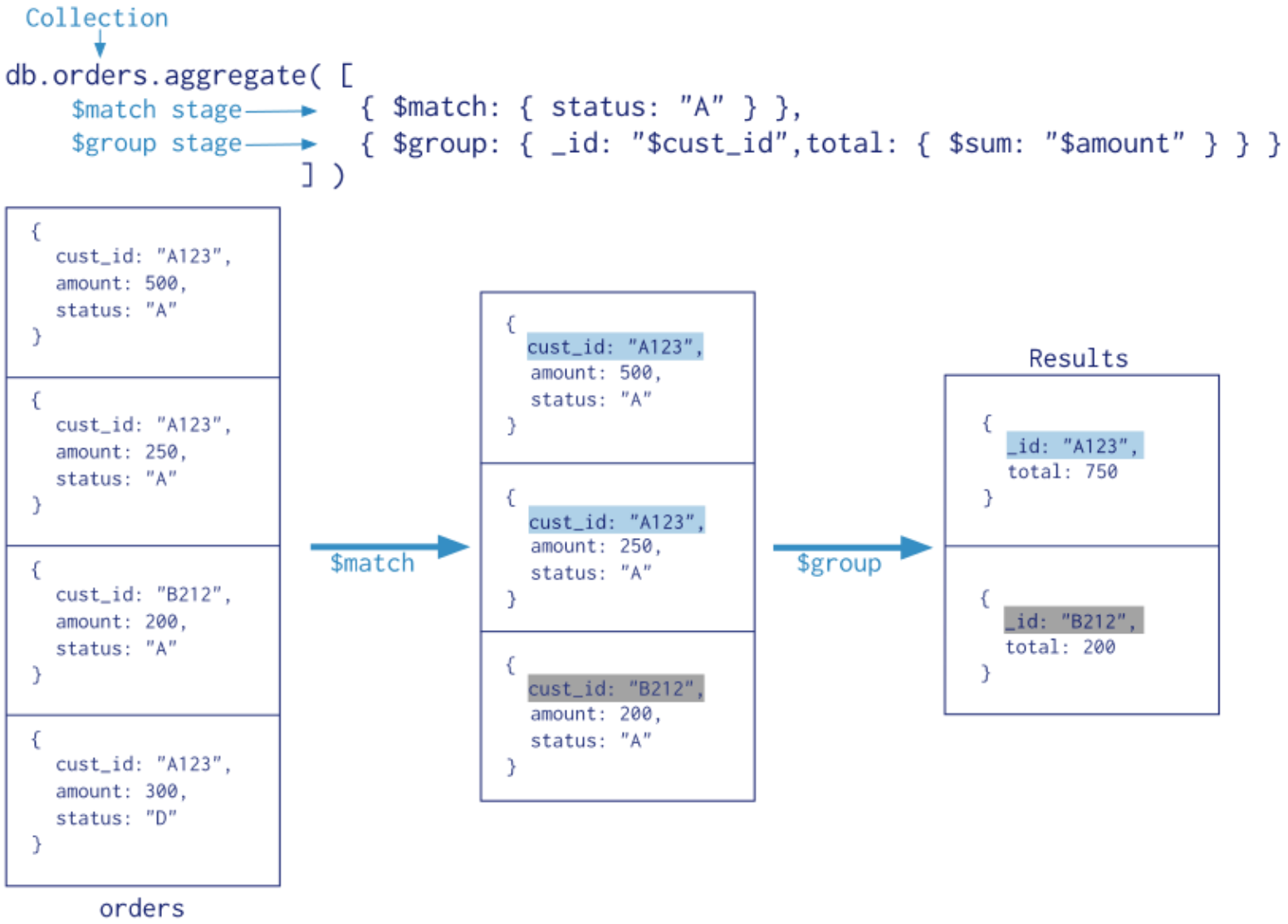
## Agregação Básica no MongoDB

| Expressão | Descrição | Exemplo |
|---|---|---|
| $sum | Soma o valor definido a partir de todos os documentos da coleção. | db.mycol.aggregate([{$group:{_id : "$by_user", num_tutorial : {$sum : "$likes"}}}]) |
| $avg | Calcula a média de todos os valores dados de todos os documentos da coleção. | db.mycol.aggregate([{$group:{_id : "$by_user", num_tutorial : {$avg : "$likes"}}}]) |
| $min | Obtém o mínimo dos valores correspondentes de todos os documentos da coleção. | db.mycol.aggregate([{$group:{_id : "$by_user", num_tutorial : {$min : "$likes"}}}]) |
| $max | Obtém o máximo dos valores correspondentes de todos os documentos da coleção. | db.mycol.aggregate([{$group:{_id : "$by_user", num_tutorial : {$max : "$likes"}}}]) |
| $push | Insere o valor para um array no documento resultante. | db.mycol.aggregate([{$group:{_id : "$by_user", url : {$push: "$url"}}}]) |
| $addToSet | Insere o valor para um array no documento resultante mas não faz criações duplicadas. | db.mycol.aggregate([{$group:{_id : "$by_user", url : {$addToSet : "$url"}}}]) |
| $first | Obtém o primeiro documento vindo de conjunto de documentos da agregação. Tipicamente isto só faz sentido quando vem junto de aplicação prévia de alguma ordenação "$sort". | db.mycol.aggregate([{$group:{_id : "$by_user", first_url : {$first : "$url"}}}]) |
| $last | Obtém o último documento vindo de conjunto de documentos da agregação. Tipicamente isto só faz sentido quando vem junto de aplicação prévia de alguma ordenação "$sort". | db.mycol.aggregate([{$group:{_id : "$by_user", last_url : {$last : "$url"}}}]) |

db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)

# Banco de Dados NoSql (MongoDB)

**Agregação Básica no MongoDB**



# Banco de Dados NoSql (MongoDB)

## Agregação Básica no MongoDB

**Listar todos os contratos para um cliente específico:**

```
db.Clientes.aggregate([
   { $match: { Cliente: "Souza Santos" } }
])
```

**Calcular o valor total de todos os contratos:**

```
db.Clientes.aggregate([
   { $group: { _id: null, totalValorContrato: { $sum: "$ValorContratoAnual" } } }
])
```

**Encontrar o contrato com o maior valor:**

```
db.Clientes.aggregate([
   { $sort: { ValorContratoAnual: -1 } },
   { $limit: 1 }
])
```

# Banco de Dados NoSql (MongoDB)

## Agregação Básica no MongoDB

**Agrupar contratos por nível de importância e calcular a média de valores para cada grupo:**

```
db.Clientes.aggregate([
    { $group: { _id: "$NivelImportancia", mediaValor: { $avg: "$ValorContratoAnual" } } }
])
```

**Listar todos os contratos com um valor anual acima de um determinado limite:**

```
db.Clientes.aggregate([
    { $match: { ValorContratoAnual: { $gt: 300000 } } }
])
```

**Calcular a média de serviços contratados por nível de importância:**

```
db.Clientes.aggregate([
    { $group: { _id: "$NivelImportancia", mediaQtdServicos: { $avg: "$QtdServicos" } } }
])
```

# Banco de Dados NoSql (MongoDB)

## Agregação Básica no MongoDB

**Calcular a média de serviços contratados por nível de importância, apenas para contratos com mais de 20 serviços:**

```
db.Clientes.aggregate([
   { $match: { QtdServicos: { $gt: 20 } } },
   { $group: { _id: "$NivelImportancia", mediaQtdServicos: { $avg: "$QtdServicos" } } }
])
```

**Encontrar os cinco contratos com o maior valor anual, mostrando apenas as informações essenciais:**

```
db.clientes.aggregate([
   { $sort: { ValorContratoAnual: -1 } },
   { $limit: 5 },
   { $project: { Cliente: 1, ValorContratoAnual: 1, DataInicioContrato: 1 } }
])
```

# Banco de Dados NoSql (MongoDB)

## Atividade em GRUPO – Usando o MongoDB

Crie um banco chamado **Escola** e três **collections: Aluno, Professor, Curso:**

- **Professor**: Nome, RG, CPF, Formação
- **Aluno**: Nome, RM, Telefone
- **Curso**: NomeDoCurso, Ano, Semestre, CargaHoraria

- Insira 5 documentos para cada collection
- Faça 2 consultas diferentes para cada collection
- Faça 2 atualizações diferentes para cada collection
- Delete 1 documento de cada collection
- Coloque o print do passo a passo em um arquivo .doc(Word) e envie no privado