INSTITUTO FEDERAL
Paraíba
Campus João Pessoa

Aula
13

Banco de Dados II

Atualizada em 16/12/2019

C.R.U.D. no MongoDB

Professor:

Dr. Alex Sandro da Cunha Rêgo

alex@ifpb.edu.br

Manipulação de Dados

Inserção de Documentos

- Método **insert()**

```
db.<coleção>.insert( document )
```

  - Se o parâmetro **_id** não for especificado no documento, MongoDB atribui um **ObjectId** único automaticamente
  - Para inserir múltiplos documentos em uma única instrução **insert()**, passe um array como argumento

```
db.<coleção>.insert([{...},{...},...])
```

```
> db.players.insert(
[ {nome: 'Alex', score: 94.5},
  {nome: 'Nathan', score: 96.7}
])
```

Inserção de Documentos

- Método **insert()**: **Orientações**
  - O nome da chave não pode começar por **$**
  - O caracter **.** não deve aparecer em nenhum local no nome da chave
  - O identificador **_id** é reservado para uso como ID de um documento
  - O caracter **-** (hífen) pode ser usado para formar a chave, porém, o nome da chave deve ser obrigatoriamente envolvido por **“”**
  - As chaves são **case-sensitive**

Inserção de Documentos

- Método **insert()**

```
INSERT INTO book (isbn, title, author,year)
VALUES (987989976576,'Python Avançado',
    'Igor Putsk', 2018)
```

```
db.book.insert( {
    ISBN: 987989976576,
    title: 'Python Avançado',
    author: 'Igor Putsk',
    year: 2018
})
```

```
WriteResult({ "nInserted" : 1 })
```

Atualização de Documentos

- Método **update()**

```
db.<coleção>.update({doc1},{doc2},{options})
```

doc1: documento com a relação de chaves/valores que determinam o critério de seleção dos documentos da coleção

doc2: documento com as chaves/valores a serem atualizados

- O **update** pode afetar mais de um documento

- Método **updateOne()**
  - Método que realiza um único update

```
db.<coleção>.updateOne({doc1},{doc2})
```

  - ✓ Mesmo que o critério **{doc1}** afete mais de um documento

Atualização de Documentos

- Exemplo1
  - Atualizando o documento inteiro (substituição)

```
> db.players.update(
  {nome: 'Alex Sandro'},
  {nome: 'Alex Sandro', score: 98.6}
  )
```

documento de filtro

documento de atualização

  - Atualizando parte(s) do documento

$set: operador de atualização, que modifica apenas os campos informados do objeto

```
> db.players.updateOne(
  {nome: 'Alex Sandro'},{$set: {score: 99.1}})
```

Se a chave alvo do operador **$set** não existir, a mesma será adicionada juntamente com o valor definido na instrução

Atualização de Documentos

- Parâmetros **{options}**
  - **upsert**: true/**false** – atualiza o documento se existir e criá-lo caso não exista
  - **multi**: true/**false** – especifica se todos os documentos correspondentes devem ser atualizados ou somente o primeiro (ação default)
  - **upsert e multi** podem ser usados ao mesmo tempo , desde que o operador **$set** seja usado

```
> db.players.update(
  {nome: 'Alex Sandro'},
  {$set: { score: 98.6} },
  {multi:true}
  )
```

Atualização de Documentos

- Outros operadores **de atualização**
  - **$inc**: incrementa um valor em particular de uma quantidade especificada

```
> db.books.update(
  {nome: 'MongoDB para iniciantes'},
  {$inc: { read: 4} )
>db.books.find()
```

    - ✓ Considere que a chave **read** possuía valor **8** antes da operação. Agora, passa a ter valor **12**
  - **$unset**: remove um chave/valor do documento

```
> db.books.update(
  {nome: 'MongoDB para iniciantes'},
  {$unset: { publisher: ''} )
>db.books.find()
```

Atualização de Documentos

- Outros operadores **de atualização**
  - **$push:** realiza um append de um valor a um campo **array** especificado
    - ✓ Se o campo estiver ausente no documento a ser atualizado, o operador **$push** adiciona-o como um novo campo e inclui o valor

```
> db.books.update(
  {nome: 'MongoDB para iniciantes'},
  {$push: { autor: 'John Cash'} )
>db.books.find()
```

Atualização de Documentos

- Outros operadores **de atualização**
  - **$pop**: remove um único elemento de um array (o primeiro ou o último). Use **1** para o último elemento e **-1** para o primeiro elemento

```
> db.books.update(
  {nome: 'MongoDB para iniciantes'},
  {$pop: { autor: 1} )
>db.books.find()
```

Atualização de Documentos

- Outros operadores **de atualização**
  - **$pull**: permite remover todas as ocorrências de um valor existente em um array

```
> db.books.update(
  {nome: 'MongoDB para iniciantes'},
  {$pull: { keyword: 'RDBMS'} )
>db.books.find()
```

  - **$pullAll**: utilizado para remover múltiplos valores em um array

```
> db.books.update(
  {nome: 'MongoDB para iniciantes'},
  {$pullAll: { keyword: ['OODBMS', 'RDBMS']}} )
>db.books.find()
```

Atualização de Documentos

- Outros operadores **de atualização**
  - **$pull**: em um array de documentos

```
{
   _id: 1,
   pedido: [
      { prod: "A", quant: 3 },
      { prod: "B", quant: 8, obs: "renewed" }
   ]
}
```

```
> db.books.update(
    {_id: 1},
    {$pull: { pedido: { prod: "A"} }}
  )
```

Atualização de Documentos

- Outros operadores **de atualização**
  - IMPORTANTE: se o update não tem condição de match ({ }), os operadores **`$pull`** e **`$pullAll`** não surtirão efeito se a opção **`multi:true`** não for acrescentada

```
> db.books.update(
  {},
  {$pull: { keyword: 'RDBMS'} )
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
```

```
> db.books.update(
  {},
  {$pull: { keyword: 'RDBMS'}, {multi:true} )
```

```
WriteResult({ "nMatched" : 7, "nUpserted" : 0, "nModified" : 1 })
```

Atualização de Documentos

- Método **update()**

```
UPDATE book
SET preço = 54.38
WHERE ISBN = '987989976576'
```

```
db.book.update(
    { ISBN: '987989976576' } ,
    { $set : { preço: 54.38} }
)
```

Documento com chave de busca

Campo para alteração

Consulte e saiba mais sobre os operadores de atualização **$rename**, **$setOnInsert**

Remoção de Documentos

- Método **remove()**

```
db.<coleção>.remove({critério})
```

  - **critério**: documento de filtro, contendo os campos que definem o critério de remoção
  - Pode remover **0**, **1** ou **mais** documentos

- Exemplo
  - Removendo um document com **_id** específico

```
> db.players.remove( {_id: ObjectId('56a12d3b12c')})
```

  - Apagando todos os documentos de uma coleção

```
> db.players.remove( { } )
```

Remoção de Documentos

- Método **deleteOne()**

```
db.<coleção>.deleteOne({criterio})
```

  - Remove **um único documento**, mesmo que mais de um documento atenda ao critério

- Método **deleteMany()**
  - Remove todos os documentos que atendem ao critério (o mesmo que **remove()**)

```
db.<coleção>.deleteMany({criterio})
```

- **Importante:**
  - Ao remover um documento, lembre-se de que qualquer referência a esse documento permanecerá no banco de dados

Remoção de Documentos

- Método **remove()**

```
DELETE FROM book
WHERE ISBN = '987989976576'
```

```
db.book.remove(
   { ISBN: '987989976576' }
)
```

Queries de Dados

- O método **find()**
  - Exibe os documentos inseridos na coleção, de acordo com a expressão desejada
  - O método **pretty()** exibe um resultado "mais limpo" da consulta
  - Há uma variedade de opções, operadores, expressões, filtros ou outros recursos disponíveis para enriquecer a consulta
- Sintaxe:

```
db.<coleção>.find()
db.<coleção>.findOne()
db.<coleção>.find().pretty()
```

  - Uma chamada a **find(),** sem argumentos, exibe todos os documentos existentes na coleção

Queries de Dados

- **Exemplos**
  - Na coleção **players,** buscar apenas os documentos com nome 'Nathan' e score = 96.7

```
> db.players.find(
  {nome: 'Nathan', score: 96.7}
)
```

  - A chave do documento especificada na busca é **case-sensitive**
- Limitando os campos a serem **exibidos**
  - Acrescentar um parâmetro adicional à query com o **nome da chave** que quer retornar , seguido de **1**

```
> db.players.find( {nome: 'Nathan'},
                   {nome:1, esporte: 1} )
```

Queries de Dados

- **Pesquisando em arrays ou documentos embutidos**
  - O uso do **.** depois do nome da chave diz ao **find()** que procure informações embutidas em seus documentos

```
> doc = {cd: 'Nirvana',tracklist : [{track:1,title:
'In bloom'},{track:2,title:'smells like teen
spirit'}]}
> db.media.insert(doc)
> db.media.find( {tracklist.title: 'In bloom'} )
```

  - Se desejar buscar em um array, basta definir a chave e o valor desejado

```
> doc = {...,assunto:['Python', 'Java', 'C', 'C#']}
> db.books.find( { assunto: 'Python'})
```

Queries de Dados

- **Pesquisando em documentos embutidos**
  - Se uma chave possui caracteres especiais ('-') que não são reservados, envolva a chave com " ou ""

```
{
        "_id" : ObjectId("593de852c70505037646242c"),
        "NUMERO-IDENTIFICADOR" : "0001925851714021",
        "CURRICULO-VITAE" : {
                "@SISTEMA-ORIGEM-XML" : "LATTES_OFFLINE",
                "@DATA-ATUALIZACAO" : ISODate("2011-11-04T00:00:00Z"),
                "@HORA-ATUALIZACAO" : "214129",
                "@NUMERO-IDENTIFICADOR" : "0001925851714021",
                "DADOS-GERAIS" : {
                        "@NOME-COMPLETO" : "Emerson Costa Cruz",
                        "@NOME-EM-CITACOES-BIBLIOGRAFICAS" : "CRUZ, E. C.",
                        "@NACIONALIDADE" : "B",
                        "@PAIS-DE-NASCIMENTO" : "Brasil",
                        "@UF-NASCIMENTO" : "PB",
                        "@CIDADE-NASCIMENTO" : "Campina Grande",
                        "@PERMISSAO-DE-DIVULGACAO" : "NAO",
```

```
> db.players.find( {},
  {'CURRICULO-VITAE.DADOS-GERAIS.@NOME-COMPLETO':1,
    _id:0} )
```

Queries de Dados

- Operadores **Comparativos**
  - **`$gt`**: (greater than) **>**
  - **`$gte`**: (greather than equal) **≥**
  - **`$lt`**: (less than) **<**
  - **`$lte`**: (less than equal) **≤**
  - **`$ne`**: (not equal) **≠**
  - **`$in`**: (contido em uma lista de valores)
  - **`$nin`**: (não está contido em uma lista de valores)

Os operadores condicionais não estão limitados para uso apenas pelo **find()**. Podem ser utilizados nos métodos **update()** ou **delete()**

Queries de Dados

- Operadores **comparativos: Exemplos**
  - Exibir todos os livros lançados depois do ano de 2010 (exceto o campo preço)

```
> db.books.find({ ano: {$gt: 2010}}, {preço:0})
```

  - Exibir todos os livros de Python lançados entre os anos de 2010 e 2015

```
> db.books.find({ assunto: 'Python',
                  ano: {$gte: 2010, $lte: 2015 }})
```

  - Exibir todos os livros que não foram publicados pela editora 'Pearson'

```
> db.books.find({ editora: {$ne: 'Pearson'}})
```

Queries de Dados

- Operadores **comparativos: Exemplos**
  - Um exemplo com o **método update():**
    - ✓ atualizar o campo status de todos os livros para '**vendido**', exceto o document de id = '5c7u8a'

```
> db.books.update(
   { _id: {$ne: ObjectId('5c7u8a')}},
   {$set: {status: 'vendido'}},
   {multi: True}
)
```

Queries de Dados

- Operadores **$in** e **$nin**
  - Verifica se o valor de um campo está (**$in**) ou não está (**$nin**) contido em uma lista de valores
    - ✓ Equivalente à cláusula **IN e NOT IN** no SQL
  - Exemplo: remover todos os documentos em que o campo nome possua o valor 'Jordan', 'Bird' ou 'Shaquile'

```
> db.players.deleteOne(
    { nome: {$in: ['Jordan','Bird', 'Shaquile']}}
  )
```

Queries de Dados

- Operadores **lógicos**
  - **$or**: realiza o papel do OU lógico (ou uma opção é atendida ou a outra)
  - **$nor**: nem uma condição nem a outra deve ser correspondida
  - **$and**: realiza o papel do AND lógico
  - **$not**: negação (oposição da expressão)
  - **Exemplo**: Selecionar todos os livros de Python com ISBN igual a '2456000' ou ano de lançamento igual a 2019

```
> db.books.find({assunto: 'Python', $or: [{isbn:
'2456000'}, {ano: 2019}]})
```

Queries de Dados

- Operadores **lógicos**
  - **Exemplo**: Selecionar todos os livros da coleção **books** em que o preço é maior do que 100.00 e o ano de publicação é 2015

```
> db.books.find({ $and: [
    {preço: {$gt: 100.00}},
    {ano: 2015}
   ]}
  )
```

Queries de Dados

- Operadores de **Query em arrays**
  - **$all:** similar ao **$in**, porém todos os valores do array devem attender à correspondência
  - Exemplo:

```
doc= { frutas:['pera','maçã','uva'] }
> db.coleção.find({ frutas: {$all: ['pera','uva']}})
  (Sim) há correspondência
> db.coleção.find({ frutas: {$all:
['pera','uva','acerola']}})
  (Não) não haverá correspondência
```

Operadores em arrays

- Operadores de Query em arrays
  - $elemMatch: usado quando se deseja especificar mais de um critério de correspondência aos elementos de um array

```
{
  filme: 'Homens de Honra',
  detalhes: [
  {ator:'Robert De Niro’, personagem: 'Billy Sunday'},
  {ator:'Cuba Gooding Jr', personagem: 'Carl'} ]
}
{
  filme: 'Mãos talentosas',
  detalhes: [
  {ator:'Loren Bass', personagem: 'Yale Professor'},
  {ator:'Cuba Gooding Jr', personagem: 'Ben Carson'} ]
}
```

Operadores em arrays

- Operadores de **Query em arrays**
  - **Exemplo:** Selecionar o filme estrelado por 'Cuba Gooding Jr' em que interpreta o médido 'Ben Carson'

```
db.filmes.find(
  {
    detalhes:{$elemMatch:{ator:'Cuba Gooding Jr',
                          personagem: 'Ben Carson'}}
  }
)
```

Operadores em arrays

- Operadores de **Query em arrays**
  - **`$size`**: consultar o número de elementos de um array

```
> db.players.find(
      { jogos: {$size: 3} })
# seleciona docs em que o array jogos tem 3 itens
```

Operadores em arrays

- Operadores de **Projeção em Arrays**
  - **$slice**: operador que 'fatia' uma faixa de valores de um array retonada por uma consulta

```
db.<coleção>.find(
      {field:value} ,
      {array: {$slice: count | [skip,limit]}}
)
```

  - ✓ **count**: nº de itens a ser recuperado no array
  - ✓ **skip**: nº de itens a ser avançado do início do array
  - ✓ **limit**: nº de itens a ser recuperado a partir de **skip**

{**$slice**: **n** } retorna os **n** primeiros elementos de um array
{**$slice**: -**n** } retorna os **n** últimos elementos de um array

Para o **skip** e o **limit**, os valores são indicados como se acessa elementos de um array (1º elemento: **0**, 2º elemento: **1**, ...)

Operadores em arrays

- Operadores de **Projeção em Arrays**
  - **Exemplo**: exibir o documento com **idAluno** igual a 'TSI001' e as 3 primeiras médias de disciplinas

```
{
  idAluno: 'TSI001',
  cre: [ 9.5, 8.6, 10.0, 5.8, 9.8, 8.9, 7.7, 9.3]
}
db.alunos.find({idAluno: 'TSI001'},{cre:{$slice:3}})
```

```
{
  "_id" : ObjectId("5c824876395ad775e1dc4c69"),
  "idAluno" : "TSI001",
  "cre" : [     9.5,
                8.6,
                10.0  ]
}
```

Operadores em arrays

- Operadores de **Projeção em arrays**
  - **Exemplo**: exibir a nota da 4ª e 5ª disciplina de todos os alunos

```
{
  idAluno: 'TSI001',
  cre: [ 9.5, 8.6, 10.0, 5.8, 9.8, 8.9, 7.7, 9.3]
}
db.alunos.find({},{cre:{$slice:[3,2]}})
```

Queries de Dados

- Operadores de **consulta de elementos**
  - **`$exists`**: permite retornar um documento específico se um campo definido estiver ausente ou presente em sua estrutura
    - ✓ Exemplo: selecionar todos os documentos da coleção **books** que possuam o campo **editora**

```
> db.books.find({ editora: {$exists: true}})
```

  - **`$type`**: seleciona os documentos em que o valor do campo corresponde a uma instância de um tipo numérico BSON especificado

```
> db.books.find({ title: {$type: 2}})        string
> db.books.find({ preço: {$type: 1}})        double
```

Queries de Dados

- Operadores de **consulta de elementos**
  - **`$type`**: tabela de tipos

Tipo	Valor
Double	1
String	2
Object	3
Array	4
Binary Data	5
Object Id	7
Boolean	8
Date	9
Null	10
Regular Expression	11
Timestamp	17

Queries de Dados

- Função **sort()**
  - Ordena o resultado retornado por uma query, de forma ascendente (1) ou descendente (-1)
  - Análoga à clausula **ORDER BY** em **SQL**

```
> db.players.find().sort( {nome: 1} )
```

  - O uso de **distinct()** em conjunto com o **sort()** só funciona com o resultado em **ordem ascendente**.

```
> db.players.distinct('nome').sort( {nome: -1} )
```

- Função **limit()**
  - Especifica o número máximo de resultados. O valor **0** retorna os resultados

```
> db.players.find( {esporte: 'basquete'}).limit(10)
```

Queries de Dados

- Função **skip()**
  - Ignora os **n** primeiros documentos do resultado de um find()
  - Útil quando se deseja implementar 'paginação'

```
> db.players.find().skip( 20 )
```

- Combinação de **comandos**
  - As funções **sort()**, **limit()** e **skip()** podem ser usadas em conjunto

```
>
db.players.find().sort({score:1}).skip(20).limit(10)
```

    - ✓ Avança 20 documentos (**skip**), e exibe os 10 seguintes (**limit**)

Queries de Dados

- Função **count()**
  - Retorna a quantidade de documentos armazenados em uma coleção

```
> db.players.count()
```

  - Retorna a quantidade de documentos selecionados a partir de um filtro

```
> db.players.find(
      {esporte: 'tênis',idade: 25}).count()
```

Queries de Dados

- Função **distinct()**
  - Retorna valores únicos correspondentes a uma chave (simples, documento embutido ou array)

```
> db.players.distinct( 'campo', query )
```

filtro

  - Considere os documentos inseridos em **teste**:

```
d1={title: 'abc', ano: 2018, país:['br']}
d2={title: 'abc', ano: 2016, país:['br']}
d3={title: 'xyz', ano: 2019, país:['usa','arg']}
> db.teste.distinct('title')
> db.teste.find({},{title:1,_id:0})
```

  - Retornar os distintos valores do campo **país**, para os documentos em que o **title** é 'abc'

```
> db.players.distinct( 'país', {title: 'abc'} )
```

Seleção de Documentos

- Método **find()**

```
SELECT title FROM book
WHERE price > 105.28
```

```
db.book.find(
    { price: {$gt : 105.28 } },
    { _id: 0, title: 1}
)
```

critério

Omite a exibição do _id (0)

Ao invés de todos os campos, só title será exibido (1)

Funções de Agregação

- Função **group()**
  - Retorna um array de itens agrupados por uma chave específica
    - ✓ Ideal quando estiver procurando uma solução para o problema de 'tag cloud'
  - **Sintaxe**

```
> db.players.group( {key, initial, reduce [,cond]
[,keyf] [,finalize] })
```

    - ✓ **key**: especifica **qual(is) campo(s)** você deseja agrupar
    - ✓ **initial**: define um valor de inicialização do documento de resultado de agregação
    - ✓ **reduce**: função de agregação que agrupa todos os itens semelhantes (pode retornar um **sum** ou **count**). Recebe dois argumentos: o documento atual e o documento de resultado de agregação do grupo
    - ✓ **cond**: critério de seleção

Funções de Agregação

- Função **group()**
  - **Exemplo**

Assumir uma coleção 'pedidos' com documentos seguindo a descrição abaixo

```
doc ={
  _id: ObjectId("5085a95c8fada716c89d0021"),
  dtPedido: ISODate("2017-07-01T04:00:00Z"),
  dtEmbarque: ISODate("2017-07-02T04:00:00Z"),
  item: { descricao: "abc123",
          preço: 1.99,
          unidade: "pcs",
          quant: 25 }
}
```

Funções de Agregação

- Função **group()**
  - Agrupar por **data do pedido** e **descrição**, para os documentos cuja data do pedido for maior do 01/01/2016
    - ✓ **Obervar que não há função de agregação**

```
> db.pedidos.group(  {
      key: {dtPedido:1, item.descricao: 1},
      cond: {dtPedido : {$gt: new Date('01/01/2016')}},
      initial: {},
      reduce: function (curr, result) { }
   } )
```

```
SELECT dtPedido, itemDescricao FROM pedidos
WHERE dtPedido::date > '01/01/2016'
GROUP BY dtPedido, itemDescricao
```

Funções de Agregação

- Função **group()**
  - Agrupar por **data do pedido** e **descrição**, para os documentos cuja data do pedido for maior do 01/01/2016, e apresente a soma do campo **quant** para cada grupo

```
> db.pedidos.group(
    {
        key: {dtPedido:1, item.descricao: 1},
        cond: {dtPedido : {$gt: new Date('01/01/2016')}},
        initial: {total: 0},
        reduce: function (curr, result) {
            result.total += curr.item.quant
        }
    }
)
```

Funções de Agregação

- Função **group()**
  - …apresente a **soma**, **count** e **average** do campo **quant** para cada grupo

Vamos observar o link abaixo:

https://docs.mongodb.com/manual/reference/method/db.collection.group/

Método de Agregação

- **Definição**
  - Um **conjunto de funções** que permite ao usuário manipular os dados retornados em uma query
  - As tranformações de dados são realizadas utilizando **operações de agregação**
- Operações/funções
  - **`$match`**: define o critério de filtragem de documentos
  - **`$group`**: agrupa documentos relacionados e executa transformações/operações sob os dados
  - **`$sum`**: somatório de um conjunto de dados
  - **`$avg`**: média aritmética de um conjunto de dados
  - **`$min/$max`**: menor/maior valor de uma série de dados

Método de Agregação

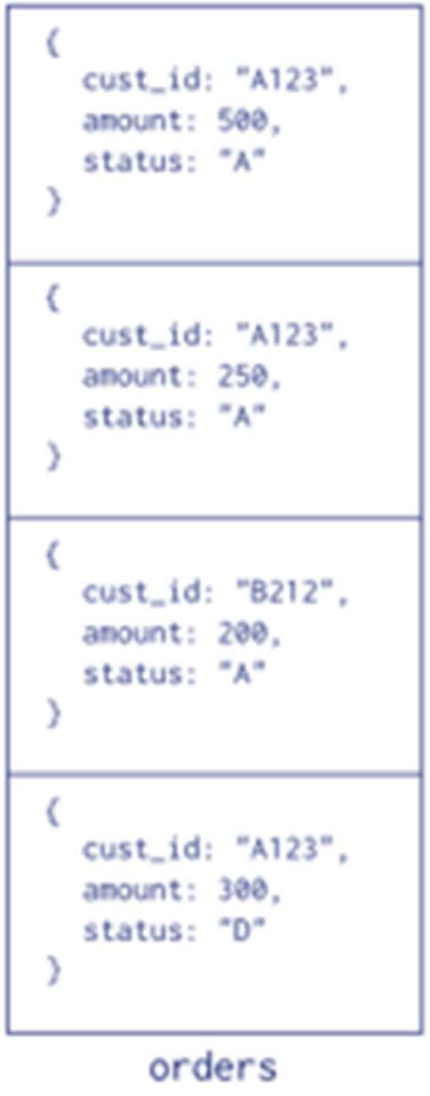
- **Sintaxe**

```
> db.coleção.aggregate( [
$unwind stage   {$unwind: '$campo_tipo_array' }},
$project stage  {$project: { <especificações> }},
$match stage    {$match: { campo: <condição>  }},
$group stage    {$group: { _id: '$campo',
                           result: {$sum:'$alvo'}}},
$sort stage     {$sort: { campo: <-1 ou 1>}},
$limit stage    {$limit: valor_inteiro_positivo}
   ] )
```

  - O resultado será sempre um array
  - Os estágios são arranjados conforme necessidade, de maneira a pensar que um estágio corrente considera o que o estágio anterior produziu

Método de Agregação

- **Exemplo 1**

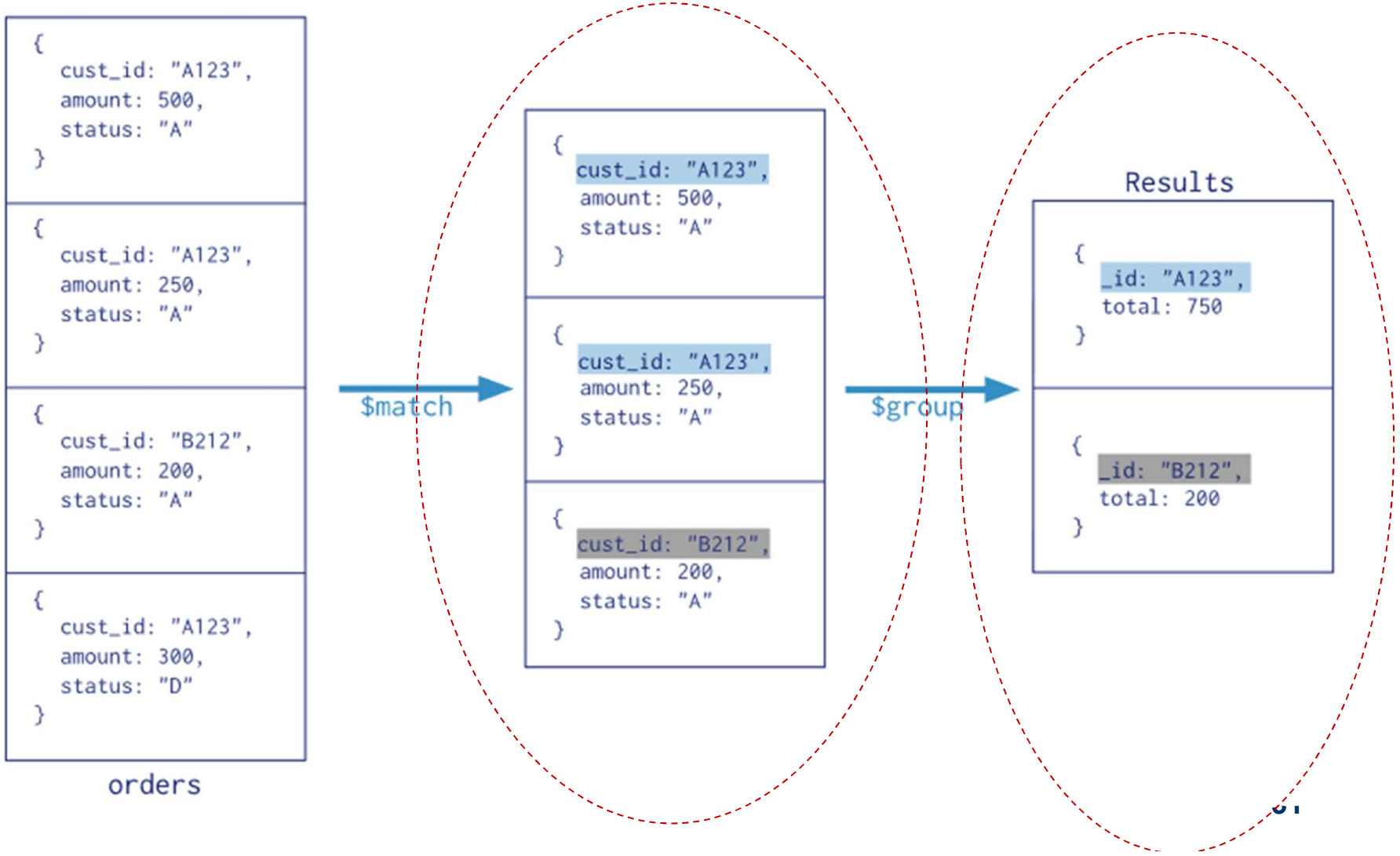


_id (out) : obrigatório

```
> db.orders.aggregate( [
  {$match: {status: 'A'}},
  {$group: { _id: '$cust_id',
             total:{$sum:'$amount'}}},
  {$sort: {total: 1}}
   ])
```

Método de Agregação

- **Exemplo 1**



Funções de Agregação

- **Exemplo 2**

posts

```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

Funções de Agregação

- **Exemplo 2**
  - Exibir um lista com a informação sobre quantos tutoriais foram escritos por cada usuário

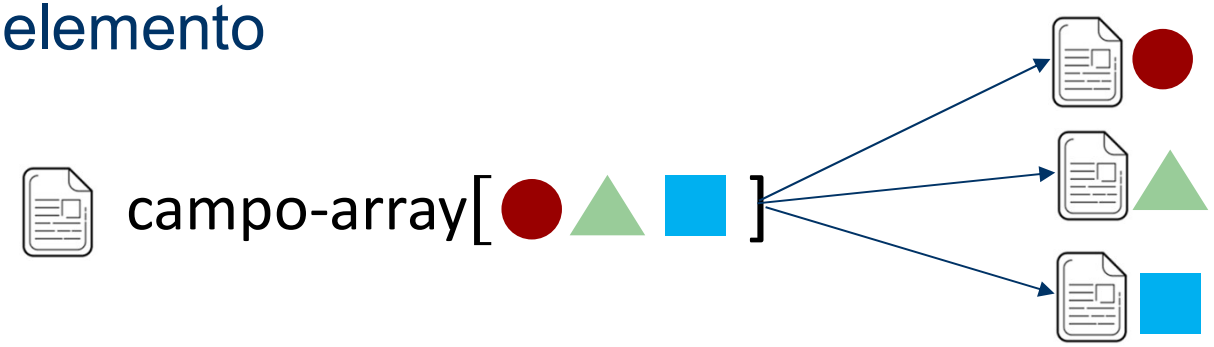
```
> db.posts.aggregate([
      { $group: { _id: '$by_user', num_tutorial:
                                          {$sum:1}}}
   ])
```

O **$sum** com valor 1 atua como um count()

```
{
 "result" : [
  { "_id" : "tutorials point", "num_tutorial" : 2 },
  { "_id" : "Neo4j", "num_tutorial" : 1 }
  ],
 "ok" : 1
}
```

Função de Agregação

- **$unwind**
  - Desconstrói um campo tipo 'array' do documento de entrada para produzir um documento para cada elemento



    - ✓ Cada documento de saída é o documento de entrada com o valor do campo array substituído pelo elemento
  - Sintaxe

```
> db.coleção.aggregate(
   { { $unwind:  "$<campo-array>" }
   })
```

Função de Agregação

- **$unwind:** exemplo

```
{ "_id" : 4,
  title: "Flex 3 in action",
  authors: ["Tariq Ahmed", "Faisal Abid" ]
}
```

```
db.books.aggregate( [
  { $project: {'titulo':'$title'}
] )
```

```
[ {
   "_id" : 4,
   title: "Flex 3 in action",
  }
]
```

Função de Projeção

- **$project**
  - função que projeta um documento com apenas os campos específicos para o próximo estágio da instrução **aggregate**.
    - ✓ Os campos podem ser **aqueles existentes** no documento ou **recém-calculados**
  - **Sintaxe**

```
> db.coleção.aggregate(
   { $project: { <especificação> }
   })
```

<**campo**> : <1 ou True> especifica a inclusão de um campo

_id: <0 ou False> especifica a supressão do _id

<**campo**>: <**expressão**> adiciona um novo campo ou "reseta" o valor de um campo existente

Função de Projeção

- **$project:** exemplo

```
{ "_id" : 1,
  titulo: "abc123", isbn: "0001122223334",
  autor: { last: "zzz", first: "aaa" },
  copias: 5
}
```

```
db.books.aggregate( [ { $project: {
    titulo: 1,
    isbn: {
       prefixo: { $substr: [ "$isbn", 0, 3 ] },
       codEditora: { $substr: [ "$isbn", 5, 4 ] }
    },
    copiasVendidas: "$copies" }
  } ] )
```

Função de Projeção

- **$project:** exemplo
  - Resultado

```
[
{
   "_id" : 1,
   "titulo" : "abc123",
   "isbn" : {
      "prefixo" : "000",
      “codEditora" : "2222"
   },
   "copiasVendidas" : 5
}
]
```

Função de Projeção

- **$project**
  - É possível renomear o nome original de um campo na saída do operador **$project**

```
{ "_id" : 4,
  title: "Flex 3 in action",
  authors: ["Tariq Ahmed", "Faisal Abid" ]
}
```

```
> db.coleção.aggregate(
   { $project: { <especificação> }
   })
```

Funções de Agregação

- **$group:** exemplos
  - Exibir quantos **likes** foram recebidos nas postagens de cada usuário

```
> db.posts.aggregate([
     { $group: { _id: '$by_user',
                 num_likes: {$sum: '$likes'}}}
   ])
```

  - Exibir a média **likes** recebidos por cada usuário

```
> db.posts.aggregate([
     { $group: { _id: '$by_user',
                 avg_likes: {$avg: '$likes'}}}
   ])
```

Funções de Agregação

- **$group, $max, $min:** exemplos
  - Exibir o menor e o maior número de **likes** recebidos por cada usuário em suas postagens

```
> db.posts.aggregate([
     { $group: { _id: '$by_user',
                 min_likes: {$min: '$likes'},
                 max_likes: {$max: '$likes'}
               }
     }
   ])
```

Funções de Agregação

- **$group, $match, $sum:** exemplo
  - Exibir o total de vendas realizado no mês de janeiro de 2018

```
{ _id: 1,
  productId: 1,
  customerId: 1,
  amount: 20.00,
  transactionDate: ISODate('2018-01-093T15:25:56.314Z')
}
{ _id: 2,
  productId: 12,
  customerId: 6,
  amount: 35.00,
  transactionDate: ISODate('2018-02-23T18:30:51.217Z')
}
```

Modelo de documento

Funções de Agregação

- **$group, $match, $sum:** exemplo

```
> db.transactions.aggregate([
  { $match: {
      transactionDate: {
        $gte: ISODate('2018-01-01T00:00:00.000Z'},
        $lte: ISODate('2018-02-01T00:00:00.000Z'}
      } }
   },
   { $group: { _id: null,
               total: {$sum: '$amount'} },
   }
])
```

Veja mais exemplos complexos em:
https://www.compose.com/articles/aggregations-in-mongodb-by-example/

Pipeline (aggregation)

- **$out**
  - Direciona o resultado retornado pelo pipeline **aggregate** para uma coleção específica
    - ✓ Útil quando desejamos criar uma coleção "limpa" a partir de uma coleção com estrutura mais complexa
  - Exemplo e Sintaxe

```
{ "_id" : 4, title: "Flex 3 in action", copies: 2 }
{ "_id" : 6, title: "MongoDB", copies: 1 }
{ "_id" : 8, title: "Python 3.0", copies: 5 }
```

```
> db.coleção.aggregate(
   { $project: { title: 1  } },
   { $out : "book_list"}
   })
```

Funções de Agregação

- **Aggregation Pipeline Stages**
  - Não deixe de explorar a documentação do MongoDB em busca dos demais estágios de uma **agregação.** Eles enriquecerão seu conhecimento na elaboração de consultas mais ponderosas

    $addField, $count, $lookup, $skip, $merge, $redact …
- **Aggregation Pipeline Operators**

  $dateFromString, $dateToString, $dayOfWeek, $cond, $convert, $concat, $hour, $minute, $split, $substr, $toLower, $toUpper, $month, $year, …

Consulte a documentação do MongoDB:
https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/
https://docs.mongodb.com/manual/reference/operator/aggregation/

Cursor no MongoDB

- **db.collection.find()**
  - Método usado para buscar documentos em uma coleção.
  - O resultado é um *ponteiro para o subconjunto de documentos retornados*
- **Cursor**
  - Uma coleção de documentos do MongoDB retornada após a execução do método **find()**
    - ✓ Os elementos retornados são indexados de `0..n-1`
  - Os documentos retornados podem ser **iterados** invidualmente
    - ✓ Quando atribuído a uma **variável** utilizando `var` keyword

Cursor no MongoDB

- **Cursor**

```
> var meuCursor = db.books.find({status:'PUBLISH'})
> meuCursor
```

  - O método **next()** em conjunto com **hasNext()** podem ser utilizados para acessar os documentos

```
> while (meuCursor.hasNext()) {
      print(tojson(meuCursor.next()));
      // printjson(meuCursor.next())
  }
```

  - Acesso indexado

```
> printjson(meuCursor[2])
```

  - O MongoDB dispõe de uma série de métodos para manipular o cursor (https://docs.mongodb.com/manual/reference/method/js-cursor/)

Cursor no MongoDB

- O método **forEach()**
  - Método que itera sob um **cursor** para aplicar uma **função javascript** em cada documento do cursor
  - Sintaxe

```
db.books.find().forEach(
    function(arg) { <corpo js da função> }
)
```

  - Exemplo

```
db.books.find({status:'PUBLISH'}).forEach(
    function(arg) {
      print('ISBN:' + arg.isbn + '/' + arg.title); }
)
```

```
ISBN:1933988746/Flex 3 in Action
ISBN:1935182420/Flex 4 in Action
ISBN:1933988312/Collective Intelligence in Action
ISBN:1617290084/Specification by Example
ISBN:1933988320/Zend Framework in Action
```

Substitui o uso de uma estrutura **loop**

Cursor no MongoDB

- O método **forEach()**
  - Exemplo:

```
{ _id : 1, title: "Flex 3 in action", price: 75.00 },
{ _id : 2, title: "MongoDB", price: 110.00 },
{ _id : 3, title: "Python 3.0", price: 98.00 }
```

  - Reajustar o preço dos livros com _id impar em 10%:

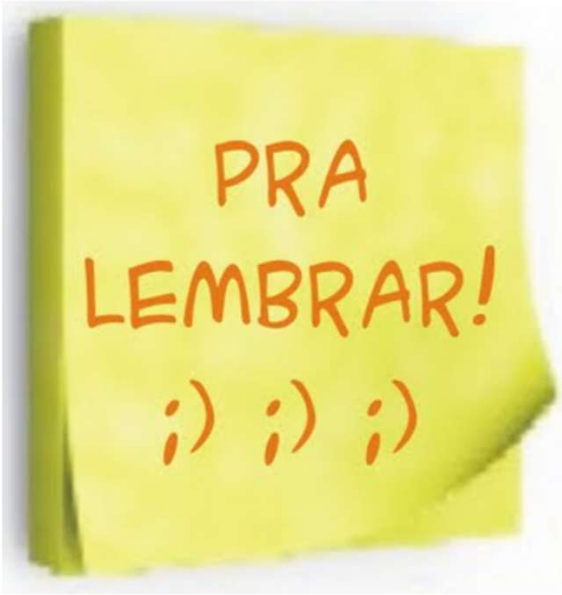
```
db.books.find({_id:{$mod:[2,1]}}).forEach(
   function(doc) {
      var updatePrice = doc.price * 1.10;
      db.books.update({_id: doc._id},
                      {$set:{price:updatePrice}});
   }
)
```

Incluir vs. Referenciar informações

- **Desnormalização de documento**

  - Não é necessário criar um documento "publisher" e ligar com uma chave (normalização). A informação do "publisher" pode ser acrescentado ao próprio documento

Incluir vs. Referenciar informações



A regra geral ao usar MongoDB é incluir os dados no próprio documento sempre que for possível

- Essa abordagem é muito mais eficiente e quase sempre viável

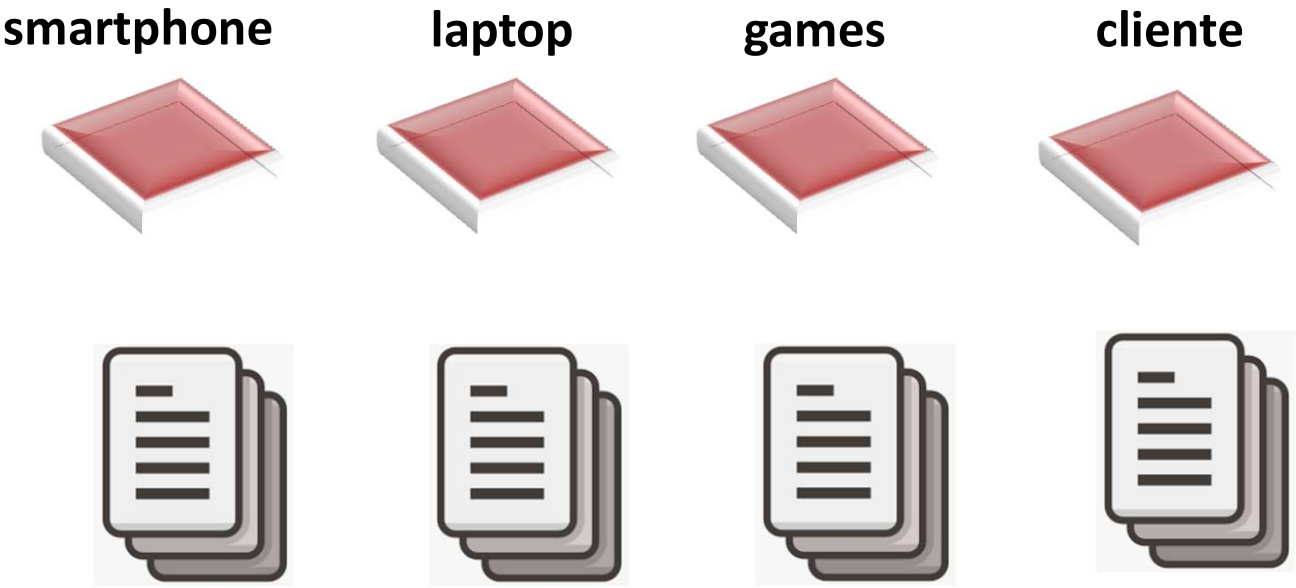
Incluir vs. Referenciar informações

- Pode-se optar por referenciar informações em outro documento
  - **Manualmente**: o usuário do mongodb armazena manualmente a referência do id de um documento dentro de outro documento
  - **DBRefs**: para o caso em que um documento contém referência de diferentes coleções

Incluir vs. Referenciar informações

- **Cenário**

Quais produtos **já foram registrados** pelos clientes de uma empresa de tecnologia?



Incluir vs. Referenciar informações

- **Referenciamento Manual**
  - Seleciona o ObjectId() do documento que será referenciado

```
> use registro
> original_id = ObjectId()
> db.smartphone.insert(
  {
    _id: original_id,
    modelo: 'iPhone 7 Plus',
    fabricante: 'Apple',
    dataFabricação: '2018-11-11',
    serial: 1365986521
  }
)
```

Incluir vs. Referenciar informações

- **Referenciamento Manual**
  - O _id do documento é incluído dentro de outro documento

```
> db.cliente.insert({
    nome : 'Alex Sandro',
    email : 'alex@ifpb.edu.br',
    cidade : 'João Pessoa',
    smartphone_id : original_id
})
>var user = db.cliente.findOne({nome:"Nathan"})
>var device = user.smartphone_id
>db.smartphone.find({_id: device})
```

Quando uma query retornar o documento **cliente** desejado, faz-se uma segunda query para o documento referenciado pelo campo **smartphone_id**

Incluir vs. Referenciar informações

- **Refernciamento DBRef()**
  - **`$ref`**: especifica a coleção do documento referenciado
  - **`$id`**: especifica o campo **`_id`** do documento referenciado
  - **`$db`**: campo opcional, que contém o nome do database em que se o documento se encontra
- **Onde inserir tais campos**?
  - No documento em que se deseja definir a referência para um documento externo (coleção **cliente**)
  - Os campos DBRef serão elementos do campo que representa a referência externa

Incluir vs. Referenciar informações

- **Como implementar?**

```
> use registro
> doc = {
    modelo: 'Moto G5 Plus',
    fabricante: 'Motorola',
    dataFabricação: '2017-06-01',
    serial: 125489652
}
> db.smartphone.insert(doc)
```

Os documentos da coleção **smartphone** serão referenciados por documentos na coleção **cliente.**
Logo, nada de novo acontece

Incluir vs. Referenciar informações

- **Como implementar?**

```
> cliDoc = {
    nome : 'Alex Sandro',
    email : 'alex@ifpb.edu.br',
    cidade : 'João Pessoa',
    smartphone: {
      $ref: 'smartphone',
      $id: ObjectId("5c798db6788d1fc6be4714b1"),
      $db: 'ifpb'
    }
}
> db.cliente.insert(cliDoc)
```

chave DBRef

campos DBRef

ObjectID do documento na coleção **smartphone**

Incluir vs. Referenciar informações

- **Como consultar?**
  - Recuperando o documento referente ao **cliente**

```
> var user = db.cliente.findOne(
  { nome:"Alex Sandro"}
  )
```

  - Obtendo as informações constituintes do campo DBRef

```
> var dbRef = user.smartphone
```

  - Pesquisando dinamicamente na coleção especificada pelo parâmetro **$ref** por um documento especificado pelo parâmetro **$id** do DBRef

```
> db[dbRef.$ref].findOne( dbRef.$id) )
```

Referências Bibliográficas

- MongoDB Tutorial (Quackit). https://www.quackit.com/mongodb/tutorial/
- MongoDB Tutorial (Tutorialspoint): https://www.tutorialspoint.com/mongodb/index.htm
- Tutorial MongoDB para Iniciantes em No-SQL: https://www.luiztools.com.br/post/tutorial-mongodb-para-iniciantes-em-nosql-2/
- Hows, D.; Membrey, P., Plugge,E. **Introdução ao MongoDB**. São Paulo: Novatec, 2015.