

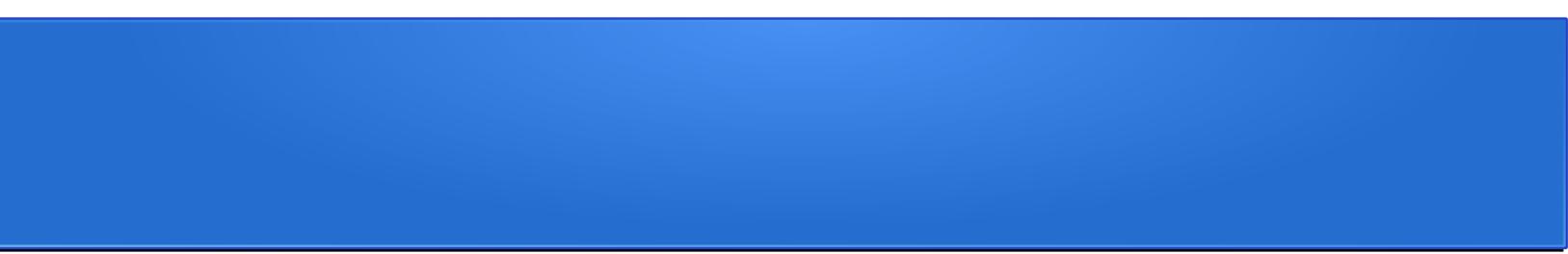
Aula 17

Introdução ao MongoDB (Parte 2) Operações de Modificação de Dados e Consultas com Agrupamentos e Agregações

11 de maio de 2018

Profa. Kelly Rosa Braghetto

Slides baseados no material confeccionado por
Elaine Naomi Watanabe (elainew@ime.usp.br),
mestre pelo DCC-IME-USP



Operações de Modificação de Dados

Remoção de documentos

- Forma geral do comando de remoção:

```
db.<nome_da_coleção>.remove(  
    { <critério_selecao1>:<valor1>,  
      <critério_selecao2>:<valor2>,  
      ...  
    }  
);
```

- Todas as tuplas que satisfizerem os critérios de remoção são removidas

Remoção de documentos - Exemplos

- Ex1: Remove todas as disciplinas com prefixo 'MAC' ou 'MAT'

```
db.disciplinas.remove(  
    { $or:[{"codigo":/^MAC/}, {"codigo":/^MAT/}] }  
);
```

- Ex2: Remove todas as disciplinas

```
db.disciplinas.remove();
```

Ou

```
db.disciplinas.remove({});
```

Alteração de documentos

- Forma geral do comando de alteração:

```
db.<nome_coleção>.update(  
    { <critério_selecao1>:<valor1>,  
      <critério_selecao2>:<valor2>,    ...  
    },  
    { <atributo_a_atualizar1>:<valor1>,  
      <atributo_a_atualizar2>:<valor2>,    ...  
    },  
    {  
      upsert: <boolean>,  
      multi: <boolean>,  
      writeConcern: <document>  
    }  
);
```

Alteração de documentos

- **Parâmetro multi:** quando `multi: true`, todos os documentos que satisfazem a condição de busca são alterados
 - Por padrão, uma operação de atualização modifica apenas o primeiro documento que ela encontrar que satisfaça a condição de busca, portanto **o valor default de multi é false.**
- **Parâmetro upsert:** quando `upsert: true`, indica que o documento será adicionado à coleção caso nenhum dos documentos existentes satisfaça à condição de busca
 - **O valor default de upsert é false**

Alteração de documentos

- O parâmetro **writeConcern** descreve o nível de garantia que o MongoDB deve prover para que a operação de alteração seja considerada "executada com sucesso".
- Esse parâmetro define, por exemplo, o número de réplicas que devem responder para o servidor mestre afirmando que receberam a atualização de um documento.
- Ex:

```
writeConcern: {w: 'majority'}
```

ou

```
writeConcern: { w: 2, wtimeout: 5000 }
```

Alteração de documentos - Exemplo

- Como dados de teste, vamos inserir duas disciplinas:

```
db.disciplinas.insert(  
  {"codigo":"MAC0439", "nome":"Laboratório de Bancos de  
  Dados"}  
);
```

```
db.disciplinas.insert(  
  {"codigo":"MAC0426", "nome":"Introdução aos Sistemas de  
  Bancos de Dados"}  
);
```

Alteração de documentos - Exemplo de uso da função update()

- Agora, vamos tentar alterar a sigla da disciplina MAC0439:

```
db.disciplinas.update(  
  {"codigo":"MAC0439"}, {"codigo":"MAC0439-B"}  
);
```

- O comando acima não altera somente o atributo codigo, ele sobrescreve o documento todo. Ou seja, o documento que antes era

```
{_id:ObjectId("57ea7e8bd3d26dd28316393f"), "codigo":"MAC0439",  
"nome":"Laboratório de Bancos de Dados"}
```

passa a ser apenas:

```
{_id: ObjectId("57ea7e8bd3d26dd28316393f"), "codigo":"MAC0439-B"}
```

- **O atributo `_id` nunca é sobrescrito ou tem seu valor modificado!**

Alteração de documentos - Exemplo de uso do \$set e do multi

- Com o operador **\$set**, somente os atributos desejados serão afetados pela operação de modificação
- Ex: o comando abaixo altera (ou inclui, caso ele ainda não exista) o atributo `numero_creditos` em todas as disciplinas que têm MAC no prefixo do código:

```
db.disciplinas.update(  
  {"codigo":/^MAC/},  
  {$set:{"numero_creditos":4}},  
  {multi:true}  
);
```

Alteração de documentos - Observação sobre o multi

- No MongoDB 3.2, foram introduzidas duas novas funções para evitar a necessidade de uso do multi:
 - `updateOne({...},{...})` = `update({...},{...},{multi:false})`
 - `updateMany({...},{...})` = `update({...},{...},{multi:true})`

- Exemplo:

```
db.disciplinas.updateMany(  
  {"codigo":/^MAC/},  
  {$set:{"numero_creditos":4}}  
);
```

equivale a:

```
db.disciplinas.update(  
  {"codigo":/^MAC/},  
  {$set:{"numero_creditos":4}},  
  {multi:true}  
);
```

Alteração de documentos - Exemplo

- Uma outra forma de atualizar parcialmente um documento é armazená-lo em uma variável, atualizá-lo e depois gravar o conteúdo da variável no BD, como ilustrado no exemplo abaixo:

```
var disc =  
    db.disciplinas.findOne({"codigo":"MAC0439"});  
disc.codigo = "MAC0439-B"; // muda o código  
disc.numero_creditos = 4; // adiciona um novo campo  
db.disciplinas.updateOne({_id:disc._id},disc);
```

Alteração de documentos - Exemplo

- O código a seguir cria ou modifica o atributo `numero_creditos` de todas as disciplinas que tem MAC como prefixo do código:

```
var cursor = db.disciplinas.find({"codigo":/^MAC/});
var disc;
while (cursor.hasNext()) {
    disc = cursor.next();
    disc.numero_creditos = 4; // adiciona um novo campo
    db.disciplinas.updateOne({_id:disc._id},disc);
}
```

Alteração de documentos - Exemplo de uso do upsert

- Tenta alterar o nome da disciplina MAC0110, mas se ela não existir, inclui no BD:

```
db.disciplinas.updateOne(  
  {"codigo": "MAC0110"},  
  {"codigo": "MAC0110", "nome": "Introdução à  
  Computação"},  
  {upsert: true}  
);
```

Alteração de documentos - A função `save()`

- A função **save** pode ser usada em substituição ao **update + upsert: true**
 - Se o documento a ser gravado não existe, ele é inserido no BD

```
db.disciplinas.save(  
  {"codigo": "MAC0110"},  
  {"codigo": "MAC0110", "nome": "Introdução à  
  Computação"}  
);
```

Alterações de documentos – Outros usos da função update()

Operadores que podem ser usados com a update() para atributos simples:

- **\$set**: Define um valor para um ou mais atributos (sem sobrescrever a estrutura anterior do documento)
- **\$unset**: Remove um ou mais atributos de um documento
- **\$inc**: Incrementa o valor de um ou mais atributos
- **\$rename**: Renomeia um ou mais atributos

...

<https://docs.mongodb.com/manual/reference/operator/update/>

Alterações de documentos - Exemplos do uso de operadores

- Incrementa o numero de créditos de MAC0439 em 2 unidades:

```
db.disciplinas.updateOne({"codigo":"MAC0439"},  
                          {$inc:{"numero_creditos":2}});
```
- Remove os atributos numero_creditos e nome de todas as disciplinas com prefixo MAC no código:

```
db.disciplinas.updateMany({"codigo":/^MAC/},  
                          {$unset:{"numero_creditos":"","nome":""}});
```
- Renomeia o atributo codigo para todas as disciplinas que o possuem:

```
db.disciplinas.updateMany({},  
                          {$rename:{"codigo":"cod_disciplina"}});
```

Alterações de documentos – Outros usos da função update()

Operadores que podem ser usados com a update() para atributos do tipo vetor:

- **\$addToSet** : Adic. elementos a um vetor se não existirem ainda
- **\$pop**: Remove o primeiro (-1) ou o último item (1) de um vetor
- **\$pull**: Remove de um vetor todos os valores que satisfazem um dado critério de busca.
- **\$pullAll**: Remove de um vetor todas as ocorrências de valores especificados
- **\$push**: Adiciona um elemento a um vetor (mesmo se ele já existir)

...

<https://docs.mongodb.com/manual/reference/operator/update/>

Alterações de documentos – Exemplos do uso de operadores

```
{"nusp": 12345,
```

```
  "notas": {"MAC0439": [10, 8, 10, 9], "MAC0426": [6, 9, 10, 7]}}
```

- Adiciona as notas 6 e 4 nas notas do MAC0439 do aluno com NUSP = 12345:

```
db.alunos.updateOne({"nusp": 12345},  
  {$push: {"notas.MAC0439": {$each: [6, 4]}}});
```

- Remove as notas de MAC439 menores do que 5 de todos os alunos:

```
db.alunos.updateMany({}, {$pull: {"notas.MAC0439": {$lt: 5}}});
```

- Remove todas as notas 9 e 10 de MAC0439 de todos os alunos:

```
db.alunos.updateMany({},  
  {$pullAll: {"notas.MAC0439": [9, 10]}});
```

Alterações de documentos – Exemplos do uso de operadores

```
{"nusp": 12345,
```

```
"notas": {"MAC0439": [10, 8, 10, 9], "MAC0426": [6, 9, 10, 7]}}
```

- Remove a primeira nota de MAC0426 do aluno com NUSP = 12345:

```
db.alunos.updateOne({"nusp": 12345},  
                    {"$pop": {"notas.MAC0426": -1}});
```

- Remove a última nota de MAC0426 do aluno com NUSP = 12345:

```
db.alunos.updateOne({"nusp": 12345},  
                    {"$pop": {"notas.MAC0426": 1}});
```

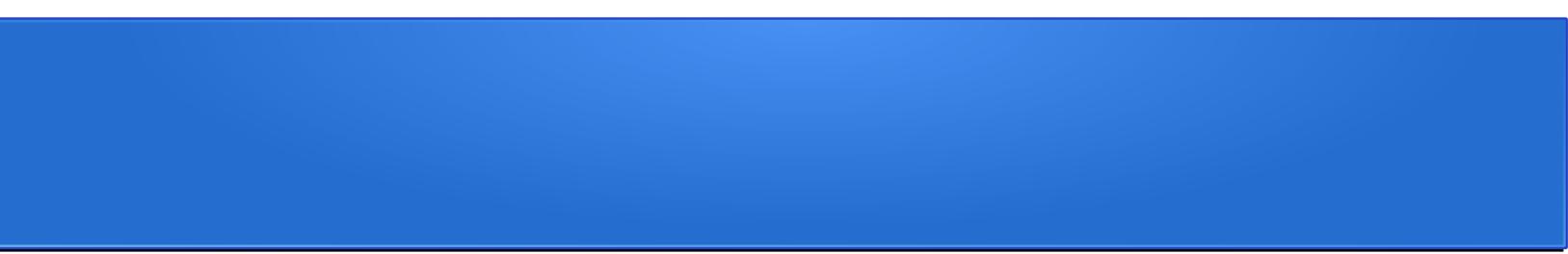
Remoção de coleções e BDs

- Para apagar a coleção <nome_coleção>:

```
db.<nome_colecao>.drop();
```

- Para apagar o banco de dados atualmente em uso:

```
db.dropDatabase();
```



Consultas com Agrupamentos e Agregações

Aggregation Framework

- O Aggregation Framework do MongoDB permite definir *pipelines* de processamento dos dados de uma coleção
 - Realiza o processamento de atributos existentes nos documentos de uma coleção, agregando-os
- Exemplos:
 - soma de todos os valores dos itens de um pedido
 - média das notas por aluno de uma disciplina
- Essas operações se assemelham às de GROUP BY, HAVING, COUNT, AVG, SUM, MAX e MIN da linguagem SQL

<https://docs.mongodb.com/manual/aggregation/#aggregation-framework>

Aggregation Framework – Agregação simples

- Forma geral de agregações simples (sem agrupamento):

```
db.collection.aggregate(  
  { $group :  
    { _id : null,  
      <nome_resultado_agregação1>:{  
        <função_agregação1>:"$<atributo_a_agregar>"},  
      <nome_resultado_agregação2>:{  
        <função_agregação2>:"$<atributo_a_agregar>"},  
      ...  
    }  
  }  
);
```

Aggregation Framework – Banco de dados para os exemplos

- Para os exemplos de agregação/agrupamento, usaremos a coleção de documentos de “zipcodes” (contendo informações sobre CEPs dos EUA). Para criá-la, execute o arquivo “bd_zipcodes_parcial.js”:
- Estrutura de um documento da coleção “zipcodes”:

```
{  
  "_id": "10280",  
  "city": "NEW YORK",  
  "state": "NY",  
  "pop": 5574,  
  "loc": [  
    -74.016323,  
    40.710537  
  ]  
}
```

Aggregation Framework – Exemplo de agregação simples

- Calcula a soma do atributo `pop` (ou seja, calcula a população total) e nomeia o resultado de `totalPop`:

```
db.zipcodes.aggregate(  
  [  
    { $group: { _id: null,  
                totalPop: { $sum: "$pop" }  
              }  
    }  
  ]  
);
```

Aggregation Framework – Agregação com agrupamento

- Forma geral de agregações com agrupamento:

```
db.collection.aggregate(  
  { $group :  
    { _id : <atributo(s)_agrupador(es)>,  
      <nome_resultado_agregação1>:{  
        <função_agregação1>:"$<atributo_a_agregar>"},  
      <nome_resultado_agregação2>:{  
        <função_agregação2>:"$<atributo_a_agregar>"},  
      ...  
    }  
  }  
);
```

Aggregation Framework – Exemplo de agrupamento

- Para cada estado, calcula a população total e a população média:

```
db.zipcodes.aggregate(  
  [  
    { $group: { _id: "$state",  
                stateTotalPop: { $sum: "$pop" },  
                stateAvgPop:    { $avg: "$pop" }  
            }  
    }  
  ]  
);
```

Aggregation Framework – Exemplo de filtro de grupo

- Para cada estado, calcula a população total e a população média, mas filtra o resultado (com o operador **\$match**), mostrando apenas os dados dos estados que possuem população total superior a 5 milhões de habitantes:

```
db.zipcodes.aggregate(  
  [  
    { $group: { _id: "$state",  
                stateTotalPop: { $sum: "$pop" },  
                stateAvgPop:    { $avg: "$pop" }  
            }  
    },  
    { $match: { stateTotalPop: { $gte: 5000000 } } }  
  ]  
);
```

Aggregation Framework – Exemplo de agrupamento envolvendo mais de um atributo

- Calcule a população total de cada cidade.

Problemas: cada cidade pode conter mais de um zip-code e podem existir duas cidades com o mesmo nome em estados diferentes.

Solução: agrupar por estado e cidade!

```
db.zipcodes.aggregate(  
  [  
    { $group: { _id: { gState: "$state", gCity: "$city" },  
                cityPop: { $sum: "$pop" } }  
    }  
  ]  
);
```

Aggregation Framework – Exemplo de *pipeline*

- Para cada estado, calcule a população média das cidades do estado.
Problema: cada cidade pode ter mais de um zip-code.

Solução: pipeline de agregações/agrupamentos!

```
db.zipcodes.aggregate(  
  [  
    { $group: { _id: { gState: "$state", gCity: "$city" },  
               cityPop: { $sum: "$pop" } }  
    },  
    { $group: { _id: "$_id.gState",  
               avgCityPop: { $avg: "$cityPop" } }  
    }  
  ]  
);
```

Aggregation Framework – Exemplo de *pipeline*

Esse bloco agrupa os zipcodes por estado e cidade e calcula a população total de cada grupo. Resultado do bloco: conjunto de triplas (gState,gCity,cityPop).

```
db.zipcodes.aggregate(  
  [  
    { $group: { _id: { gState: "$state", gCity: "$city" },  
                cityPop: { $sum: "$pop" } }  
    },  
    { $group: { _id: "$_id.gState",  
                avgCityPop: { $avg: "$cityPop" } }  
    }  
  ]  
);
```

Esse bloco agrupa o resultado do bloco anterior por estado e calcula a média da população em cada grupo. Resultado do bloco: conjunto de duplas (gState,avgCityPop).

Aggregation Framework – Funções de agregação existentes

- Iguais da SQL: **\$sum**, **\$avg**, **\$min**, **\$max**
- **\$push**: devolve um vetor com todos os valores que apareceram no grupo para o atributo escolhido
- **\$addToSet**: devolve um vetor com todos os valores que apareceram no grupo para o atributo escolhido, mas sem repetições de valores
- **\$first**: retorna o valor de um atributo do primeiro documento de um conjunto ordenado de documentos
- **\$last**: retorna o valor de um atributo do último documento de um conjunto ordenado de documentos

Aggregation Framework – Exemplo de uso de outras funções de agregação

- Para cada estado, obtém um vetor com todas as cidades do estado com repetições e outro vetor com as cidades sem repetições:

```
db.zipcodes.aggregate(  
  [  
    { $group: { _id: "$state",  
                bagOfCities: { $push: "$city" },  
                setOfCities: { $addToSet: "$city" }  
            }  
    }  
  ]  
);
```

Aggregation Framework – Outro exemplo de *pipeline*

```
db.zipcodes.aggregate( [  
  { $group:  
    { _id: { gState: "$state", gCity: "$city" },  
      cityPop: { $sum: "$pop" } } },  
  { $sort: { cityPop: 1 } },  
  { $group:  
    { _id : "$_id.gState",  
      biggestCity: { $last: "$_id.gCity" }, biggestPop: { $last: "$cityPop" },  
      smallestCity: { $first: "$_id.gCity" }, smallestPop: { $first: "$cityPop" } } },  
  { $project:  
    { _id: 0,  
      state: "$_id",  
      biggestCity: { name: "$biggestCity", pop: "$biggestPop" },  
      smallestCity: { name: "$smallestCity", pop: "$smallestPop" } } }  
] );
```

Para cada estado, obtém a a cidade com a maior e a cidade com a menor população, junto com as suas respectivas populações.

Aggregation Framework – Explicação do *pipeline*

Obtém a população de cada cidade.

```
db.zipcodes.aggregate( [
```

```
  { $group:
    { _id: { gState: "$state", gCity: "$city" },
      cityPop: { $sum: "$pop" } } },
```

Ordena a resposta do bloco anterior em ordem cresc. de população.

```
  { $sort: { cityPop: 1 } } ]
```

A partir da resposta do bloco anterior, obtém a maior e a menor população e suas respectivas cidades.

```
  { $group:
    { _id : "$_id.gState",
      biggestCity: { $last: "$_id.gCity" }, biggestPop: { $last: "$cityPop" },
      smallestCity: { $first: "$_id.gCity" }, smallestPop: { $first: "$cityPop" } } },
```

```
  { $project:
    { _id: 0,
      state: "$_id",
      biggestCity: { name: "$biggestCity", pop: "$biggestPop" },
      smallestCity: { name: "$smallestCity", pop: "$smallestPop" } } } ] );
```

Formata a resposta do bloco anterior para ser exibida como resultado final.

```
] );
```

Aggregation Frame – Outro exemplo de *pipeline*

- Dados para teste (usuários e suas preferências de esportes):

```
db.pessoas.insertMany([
  { _id : "jane", gosta : ["golf", "vôlei"] },
  { _id : "joe",  gosta : ["tênis", "golf", "natação"]},
  { _id : "anne", gosta : ["vôlei", "basquete", "futebol", "golf"]},
  { _id : "peter", gosta : ["futebol", "vôlei"]}
]);
```

Aggregation Framework – Outro exemplo de *pipeline*

- Obtém os dois esportes preferidos (ou seja, os “mais gostados”) das pessoas cadastradas no BD:

```
db.pessoas.aggregate(  
  [  
    { $unwind : "$gosta" },  
    { $group : { _id : "$gosta" , contGosta : { $sum : 1 } } },  
    { $sort : { contGosta : -1 } },  
    { $limit : 2 }  
  ]  
);
```

Aggregation Framework – Explicação do *pipeline*

- Resultado da operação **\$unwind** no slide anterior:

```
{ "_id" : "jane", "gosta" : "golf" }  
{ "_id" : "jane", "gosta" : "vôlei" }  
{ "_id" : "joe", "gosta" : "tênis" }  
{ "_id" : "joe", "gosta" : "golf" }  
{ "_id" : "joe", "gosta" : "natação" }  
{ "_id" : "anne", "gosta" : "vôlei" }  
{ "_id" : "anne", "gosta" : "basquete" }  
{ "_id" : "anne", "gosta" : "futebol" }  
{ "_id" : "anne", "gosta" : "golf" }  
{ "_id" : "peter", "gosta" : "futebol" }  
{ "_id" : "peter", "gosta" : "vôlei" }
```

Referências Bibliográficas

- Documentação do MongoDB
 - <https://docs.mongodb.com/>
- Tutoriais oficiais:
 - <https://docs.mongodb.com/manual/>
 - <https://docs.mongodb.com/getting-started/shell/>
- Tabela de mapeamento de SQL para MongoDB
 - http://s3.amazonaws.com/info-mongodb-com/sql_to_mongo.pdf
- Livros:
 - “MongoDB: The Definitive Guide, 2nd Edition – Powerful and Scalable Data Storage“, de Kristina Chodorow, Editora: O'Reilly Media
 - “MongoDB: Construa novas aplicações com novas tecnologias“, de Fernando Boaglio, Editora: Casa do Código