Introdução

O Aggregation Framework é um recurso muito interessante do MongoDB . Totalmente nativo e escrito em C++ , ele oferece um conjunto de ferramentas para realizar operações muito mais complexas do que as vistas até aqui.
Basicamente, as operações de agregação processam dados e retornam resultados calculados. Essas operações podem agrupar valores de múltiplos documentos ou coleções , executar uma variedade de operações nesses dados agrupados e, por fim, retornar um único resultado. O MongoDB fornece três caminhos para executar operações de agregação: aggregation pipeline , map-reduce function e single purpose aggregation methods . Neste módulo, você se aprofundará no aggregation pipeline, o método mais utilizado e recomendado pela MongoDB .

Aggregation Pipeline

O Aggregation Framework foi modelado sob o conceito de processamento de dados por meio de *pipelines* , ou seja, um "funil" . Um *pipeline* contém múltiplos estágios. Os documentos entram nesse "funil" e vão se transformando à medida que vão passando por esses estágios até chegarem ao estágio final, com um resultado "agregado". Veja um exemplo desses estágios no vídeo abaixo.
No exemplo do vídeo, temos a seguinte operação:

Copiar

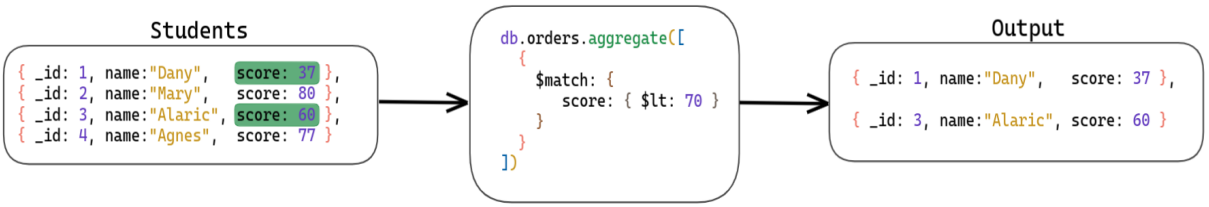
```
db.orders.aggregate([
  { $match: { status: "A" } },
  { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
]);
```

Essa operação possui dois estágios:
Primeiro Estágio : O estágio `$match` filtra os documentos pelo campo `status` , e passam para o próximo estágio somente os documentos que têm `status` igual a `"A"` .
Segundo Estágio : O estágio `$group` agrupa os documentos pelo campo `cust_id` para calcular a soma dos valores do campo `amount` para cada `cust_id` único.
Note que a sintaxe é como a de uma *query* em MQL (*MongoDB Query Language*). O que aparece de novo é justamente o método `aggregate` . Esse método recebe como primeiro parâmetro um *array* de documentos, que nada mais são do que os estágios do *pipeline* . Você pode ter quantos estágios forem necessários dentro do mesmo `aggregate` .
Um estágio do *pipeline* , por mais básico que seja, já consegue fornecer filtros que atuam como *queries* e podem realizar transformações de documentos que modificam a forma de saída do documento no estágio.
Outras operações do *pipeline* fornecem ferramentas para agrupamento e ordenação de documentos por campos específicos, bem como ferramentas para agregar

conteúdos de *arrays* (incluindo *arrays* de documentos). Os estágios do *pipeline* podem utilizar operadores para tarefas que calculam médias ou concatenam *strings* , por exemplo.
Para melhorar a performance durante os estágios, o aggregation pipeline pode utilizar índices e tem também uma fase interna de otimização.
Agora que você já tem uma boa ideia do que é o aggregation pipeline , vamos explorar alguns de seus estágios para ver todo o seu poder! 😉

Operador $match

O estágio representado pelo operador $match filtra os documentos da mesma maneira que os filtros no método find({ $match }) .



É recomendado sempre priorizar o $match o mais "cedo" possível no pipeline . Isso melhora muito a performance, uma vez que o $match limita o número de documentos passados para o próximo estágio. E se o $match for utilizado bem no começo do *pipeline* , a *query* tem a vantagem da utilização de índices.
Veja esses documentos na coleção articles :

Copiar

```
db.articles.insertMany([
{ _id: ObjectId("512bc95fe835e68f199c8686"), author: "dave", score: 80, views: 100 },
{ _id: ObjectId("512bc962e835e68f199c8687"), author: "dave", score: 85, views: 521 },
{ _id: ObjectId("55f5a192d4bede9ac365b257"), author: "ahn", score: 60, views: 1000 },
{ _id: ObjectId("55f5a192d4bede9ac365b258"), author: "li", score: 55, views: 5000 },
{ _id: ObjectId("55f5a1d3d4bede9ac365b259"), author: "annT", score: 60, views: 50 },
{ _id: ObjectId("55f5a1d3d4bede9ac365b25a"), author: "li", score: 94, views: 999 },
{ _id: ObjectId("55f5a1d3d4bede9ac365b25b"), author: "ty", score: 95, views: 1000 }
])
```

Exemplo 1: Igualdade simples

Vamos fazer uma operação utilizando o operador $match com igualdade simples:

Copiar

```
db.articles.aggregate(
  [{ $match : { author : "dave" } }]
);
```

A operação citada seleciona todos os documentos em que o campo `author` seja igual a `dave` . Note que a sintaxe do filtro é exatamente igual à utilizada como filtro no método `find()` . A agregação retornará os seguintes documentos:

Copiar

```
{ _id: ObjectId("512bc95fe835e68f199c8686"), author: "dave", score: 80, views: 100
}
{ _id: ObjectId("512bc962e835e68f199c8687"), author: "dave", score: 85, views: 521
}
```

Exemplo 2: Igualdade complexa

É possível, dentro do `match` , utilizar operadores como `or` , `and` , `in` etc.

Copiar

```
db.articles.aggregate(
  [
    {
      $match: {
        $or: [
          { score: { $gt: 70, $lt: 90 } },
          { views: { $gte: 1000 } }
        ]
      }
    }
  ]
);
```

Nessa operação de agregação, o primeiro e único estágio seleciona todos os documentos da coleção `articles` em que o `score` seja maior que `70` e menor que `90` , ou o campo `views` seja maior ou igual a `1000` :

Copiar

```
{ "_id" : ObjectId("512bc95fe835e68f199c8686"), "author" : "dave", "score" : 80,
"views" : 100 }
{ "_id" : ObjectId("512bc962e835e68f199c8687"), "author" : "dave", "score" : 85,
"views" : 521 }
{ "_id" : ObjectId("55f5a192d4bede9ac365b257"), "author" : "ahn", "score" : 60,
"views" : 1000 }
{ "_id" : ObjectId("55f5a192d4bede9ac365b258"), "author" : "li", "score" : 55, "views"
: 5000 }
```

```
{ "_id" : ObjectId("55f5a1d3d4bede9ac365b25b"), "author" : "ty", "score" : 95, "views" : 1000 }
```

Operador $limit

O operador `$limit` limita o número de documentos que será passado para o próximo estágio do pipeline. Ele sempre recebe um valor do tipo inteiro e positivo.
Limitar o número de documentos numa operação de agregação na coleção `articles` é bem simples:

Copiar

```
db.articles.aggregate(
  [
    { $limit : 5 }
  ]
);
```

Essa operação retorna apenas `5` documentos.

Para Fixar

Antes de começar, crie um banco de dados chamado `agg_example` e rode a `query` abaixo para os exercícios.

Copiar

```
use agg_example;
db.transactions.insertMany([
  { value: 5900, from: "Dave America", to: "Ned Flanders", bank: 'International' },
  { value: 1000, from: "Mark Zuck", to: "Edna Krabappel", bank: 'FloridaBank' },
  { value: 209, from: "Lisa Simpson", to: "Dave America", bank: 'bankOfAmerica' },
  { value: 10800, from: "Arnold Schuz", to: "Mark Zuck", bank: 'JPMorgan' },
  { value: 850, from: "Barney Gumble", to: "Lisa Simpson", bank: 'Citigroup' },
  { value: 76000, from: "Ned Flanders", to: "Edna Krabappel", bank: 'JPMorgan' },
  { value: 1280, from: "Dave America", to: "Homer Simpson", bank: 'Citigroup' },
  { value: 7000, from: "Arnold Schuz", to: "Ned Flanders", bank: 'International' },
  { value: 59020, from: "Homer Simpson", to: "Lisa Simpson", bank: 'International' },
  { value: 100, from: "Mark Zuck", to: "Barney Gumble", bank: 'FloridaBank' },
]);
```

Utilizando o banco de dados `agg_example` , faça os seguintes exercícios:

1. Selecione todas as transações feitas pelo cliente chamado "Dave America".
2. Selecione todas as transações com o valor entre 700 e 6000, ou que sejam recebidas pela cliente "Lisa Simpson".
3. Selecione três transações com o valor acima de 1000.

Operador $project

O operador $project tem como uma de suas funções passar adiante no pipeline apenas alguns campos dos documentos vindos do estágio anterior, fazendo isso por meio de uma "projeção", como no método find({}, { $project }) . Mas aqui temos uma diferença: esses campos podem ser novos, sendo resultado de um cálculo ou de uma concatenação.
Assim como numa projeção comum, o único campo que precisa ser negado explicitamente é o _id .
Se você especificar um campo que não existe, o $project simplesmente ignorará esse campo, sem afetar em nada a projeção.
Veja alguns exemplos, considerando este documento da coleção books .

Copiar

```
db.books.insertOne(
  {
    _id: 1,
    title: "A Fundação",
    isbn: "0001122223334",
    author: { last: "Asimov", first: "Isaac" },
    copies: 5
  }
)
```

Exemplo 1: Incluindo campos específicos

Para incluir apenas os campos _id , title e author no documento de saída, utilize o operador $project da seguinte maneira:

Copiar

```
db.books.aggregate(
  [
    {
      $project : {
        title : 1,
        author : 1
      }
    }
  ]
);
```

Exemplo 2: Excluindo o campo _id

Como você já viu, o campo _id é padrão e é o único que necessita de uma negação explícita para que não seja incluído no documento de saída:

Copiar

```
db.books.aggregate([
  {
    $project : {
      _id: 0,
      title : 1,
      author : 1
    }
  }
]);
```

Exemplo 3: Excluindo outros campos

Quando você nega um campo específico, todos os outros serão incluídos no documento de saída. O exemplo abaixo exclui do documento de saída apenas o campo `copies` :

Copiar

```
db.books.aggregate([
  {
    $project : {
      copies: 0
    }
  }
]);
```

Exemplo 4: Excluindo campos em subdocumentos

Para documentos *embedados* , seguimos os mesmos conceitos de *dot notation* :

Copiar

```
db.books.aggregate([
  {
    $project : {
      "author.first": 0,
      copies: 0
    }
  }
]);
```

Para inclusão de campos *embedados* , utilize a mesma lógica, apenas substituindo o `0` por `1` .

Exemplo 5: Incluindo campos calculados

Podemos usar uma string iniciada com o caractere $ para indicar que queremos projetar um campo, assim: "$nomeDoCampo".
A operação a seguir adiciona os novos campos isbn , lastname e copiesSold :

Copiar

```
db.books.aggregate([
  {
    $project: {
      title: 1,
      isbn: {
        prefix: { $substr: ["$isbn", 0, 3] },
        group: { $substr: ["$isbn", 3, 2] },
        publisher: { $substr: ["$isbn", 5, 4] },
        title: { $substr: ["$isbn", 9, 3] },
        checkDigit: { $substr: ["$isbn", 12, 1] }
      },
      lastName: "$author.last",
      copiesSold: "$copies"
    }
  }
]);
```

Depois disso, o documento terá o seguinte formato:

Copiar

```
{
  "_id" : 1,
  "title" : "A Fundação",
  "isbn" : {
    "prefix" : "000",
    "group" : "11",
    "publisher" : "2222",
    "title" : "333",
    "checkDigit" : "4"
  },
  "lastName" : "Asimov",
  "copiesSold" : 5
}
```

Lembre-se: esses novos campos são apenas adicionados para a visualização final, não serão salvos no banco.

Operador $project

O operador $project tem como uma de suas funções passar adiante no pipeline apenas alguns campos dos documentos vindos do estágio anterior, fazendo isso por

meio de uma "projeção", como no método `find({}, { $project })` . Mas aqui temos uma diferença: esses campos podem ser novos, sendo resultado de um cálculo ou de uma concatenação.
Assim como numa projeção comum, o único campo que precisa ser negado explicitamente é o `_id` .
Se você especificar um campo que não existe, o `$project` simplesmente ignorará esse campo, sem afetar em nada a projeção.
Veja alguns exemplos, considerando este documento da coleção `books` .

Copiar

```
db.books.insertOne(
  {
    _id: 1,
    title: "A Fundação",
    isbn: "0001122223334",
    author: { last: "Asimov", first: "Isaac" },
    copies: 5
  }
)
```

Exemplo 1: Incluindo campos específicos

Para incluir apenas os campos `_id` , `title` e `author` no documento de saída, utilize o operador `$project` da seguinte maneira:

Copiar

```
db.books.aggregate(
  [
    {
      $project : {
        title : 1,
        author : 1
      }
    }
  ]
);
```

Exemplo 2: Excluindo o campo `_id`

Como você já viu, o campo `_id` é padrão e é o único que necessita de uma negação explícita para que não seja incluído no documento de saída:

Copiar

```
db.books.aggregate([
  {
    $project : {
```

```
    _id: 0,
    title : 1,
    author : 1
  }
 }
]);
```

Exemplo 3: Excluindo outros campos

Quando você nega um campo específico, todos os outros serão incluídos no documento de saída. O exemplo abaixo exclui do documento de saída apenas o campo `copies` :

Copiar

```
db.books.aggregate([
 {
  $project : {
    copies: 0
  }
 }
]);
```

Exemplo 4: Excluindo campos em subdocumentos

Para documentos *embedados* , seguimos os mesmos conceitos de *dot notation* :

Copiar

```
db.books.aggregate([
 {
  $project : {
    "author.first": 0,
    copies: 0
  }
 }
]);
```

Para inclusão de campos *embedados* , utilize a mesma lógica, apenas substituindo o `0` por `1` .

Exemplo 5: Incluindo campos calculados

Podemos usar uma `string` iniciada com o caractere `$` para indicar que queremos projetar um campo, assim: "$nomeDoCampo".
A operação a seguir adiciona os novos campos `isbn` , `lastname` e `copiesSold` :

Copiar

```
db.books.aggregate([
```

```
  {
    $project: {
      title: 1,
      isbn: {
        prefix: { $substr: ["$isbn", 0, 3] },
        group: { $substr: ["$isbn", 3, 2] },
        publisher: { $substr: ["$isbn", 5, 4] },
        title: { $substr: ["$isbn", 9, 3] },
        checkDigit: { $substr: ["$isbn", 12, 1] }
      },
      lastName: "$author.last",
      copiesSold: "$copies"
    }
  }
]);
```

Depois disso, o documento terá o seguinte formato:

Copiar

```
{
  "_id" : 1,
  "title" : "A Fundação",
  "isbn" : {
    "prefix" : "000",
    "group" : "11",
    "publisher" : "2222",
    "title" : "333",
    "checkDigit" : "4"
  },
  "lastName" : "Asimov",
  "copiesSold" : 5
}
```

Lembre-se: esses novos campos são apenas adicionados para a visualização final, não serão salvos no banco.

Operador $unwind

O operador $unwind "desconstrói" um campo *array* do documento de entrada e gera como saída um documento para cada elemento do *array* . Cada documento de saída é o documento de entrada com o valor do campo *array* substituído por um elemento do *array* .
Na prática fica mais fácil de entender. Insira o seguinte documento na coleção inventory :

Copiar

```
db.inventory.insertOne({ _id: 7, item: "ABC1", sizes: ["S", "M", "L"] });
```

E agora, utilizando o $unwind como um estágio do pipeline :

Copiar
```
db.inventory.aggregate([{ $unwind : "$sizes" }]);
```

O retorno é o seguinte:

Copiar
```
{ "_id" : 7, "item" : "ABC1", "sizes" : "S" }
{ "_id" : 7, "item" : "ABC1", "sizes" : "M" }
{ "_id" : 7, "item" : "ABC1", "sizes" : "L" }
```

Note que temos a "expansão" do *array* sizes , e a saída são três documentos com os valores _id e item preservados.
Você verá mais exemplos com o operador $unwind quando "juntarmos" tudo em vários estágios!

Operador $lookup

O operador $lookup foi introduzido na versão 3.2 do MongoDB e vem evoluindo desde então. Com ele, é possível juntar documentos de outra coleção (join). Como resultado dessa junção, um elemento do tipo *array* é adicionado a cada documento da coleção de entrada, contendo os documentos que deram "match" na coleção com a qual se faz o "join".
Existem quatro parâmetros básicos para montar um $lookup :

- from : uma coleção no mesmo database para executar o join ;
- localField : o campo da coleção de onde a operação de agregação está sendo executada. Será comparado por igualdade com o campo especificado no parâmetro foreingField ;
- foreingField : o campo da coleção especificada no parâmetro from que será comparado com o campo localField por igualdade simples;
- as : o nome do novo *array* que será adicionado.

Join com igualdade simples

Considere os seguintes documentos nas coleções orders e inventory :

Copiar
```
// orders
db.orders.insertMany([
{ _id: 1, item: "almonds", price: 12, quantity: 2 },
{ _id: 2, item: "pecans", price: 20, quantity: 1 },
{ _id: 3 }
])
```

Copiar

```
// inventory
db.inventory.insertMany([
{ _id: 1, sku: "almonds", description: "product 1", instock: 120 },
{ _id: 2, sku: "bread", description: "product 2", instock: 80 },
{ _id: 3, sku: "cashews", description: "product 3", instock: 60 },
{ _id: 4, sku: "pecans", description: "product 4", instock: 70 },
{ _id: 5, sku: null, description: "Incomplete" },
{ _id: 6 }
])
```

Imagine que você queria retornar em uma única *query* os documentos correspondentes das duas coleções mencionadas. A primeira coisa é encontrar um campo em comum entre elas. Nesse caso, seriam os campos item (coleção orders) e sku (coleção inventory). Quando cruzados na operação a seguir, um novo campo, chamado inventory_docs , será adicionado ao resultado final:

Copiar

```
db.orders.aggregate([
  {
    $lookup: {
      from: "inventory",
      localField: "item",
      foreignField: "sku",
      as: "inventory_docs"
    }
  }
]);
```

Como resultado do *pipeline* , os documentos abaixo serão retornados:

Copiar

```
{
  "_id" : 1,
  "item" : "almonds",
  "price" : 12,
  "quantity" : 2,
  "inventory_docs" : [
    {
      "_id" : 1,
      "sku" : "almonds",
      "description" : "product 1",
      "instock" : 120
    }
  ]
}
{
```

```
  "_id" : 2,
  "item" : "pecans",
  "price" : 20,
  "quantity" : 1,
  "inventory_docs" : [
    {
      "_id" : 4,
      "sku" : "pecans",
      "description" : "product 4",
      "instock" : 70
    }
  ]
}
{
  "_id" : 3,
  "inventory_docs" : [
    {
      "_id" : 5,
      "sku" : null,
      "description" : "Incomplete"
    },
    {
      "_id" : 6
    }
  ]
}
```

Embora não seja possível realizar uma operação idêntica, uma *query* equivalente em SQL seria algo do tipo:

Copiar

```
SELECT * inventory_docs
FROM orders
WHERE inventory_docs IN (
  SELECT *
  FROM inventory
  WHERE sku = orders.item
);
```

Para Fixar

Utilizando o banco de dados agg_example , adicione a seguinte collection e faça os exercícios:

Copiar

```
use agg_example;
db.clients.insertMany([
  { name: "Dave America", State: "Florida" },
  { name: "Ned Flanders", State: "Alasca" },
  { name: "Mark Zuck", State: "Texas" },
  { name: "Edna Krabappel", State: "Montana" },
  { name: "Arnold Schuz", State: "California" },
  { name: "Lisa Simpson", State: "Florida" },
  { name: "Barney Gumble", State: "Texas" },
  { name: "Homer Simpson", State: "Florida" },
]);
```

1. Selecione todos os clientes com as suas respectivas transações feitas;
2. Selecione quatro clientes com as suas respectivas transações recebidas;
3. Selecione todos os cliente do estado da "Florida" e suas respectivas transações recebidas.

Agora, a prática

Para esta etapa, utilizaremos um *dataset* que contém três coleções: clientes , produtos e vendas . Utilize os comandos abaixo para importar essas coleções para o banco erp :

1. Faça o download dos arquivos json , clicando com o botão direito e escolhando a opção "Salvar como":

- clientes
- produtos
- vendas

2. Faça a importação para sua instância do MongoDB:

Copiar

```
mongoimport --db erp <caminho_do_arquivo_clientes.json>
mongoimport --db erp <caminho_do_arquivo_produtos.json>
mongoimport --db erp <caminho_do_arquivo_vendas.json>
```

3. Conecte-se à sua instância e confira o número de documentos em cada coleção:

Copiar

```
use erp;
db.clientes.count(); // 499
db.produtos.count(); // 499
db.vendas.count(); // 4900
```

Com o dataset importado, é hora de colocar a mão na massa!

O MongoDb possui diversas ferramentas, como, por exemplo, mongo , mongosh , Compass e outras ferramentas de terceiros. Você pode utilizar o que achar melhor para executar as *queries* , o importante é realizá-las.

Exercício 1: Utilizando o estágio $match , escreva uma agregação para retornar somente os clientes do sexo "MASCULINO" .

Exercício 2: Utilizando o estágio $match , escreva uma agregação para retornar somente os clientes do sexo "FEMININO" e com data de nascimento entre os anos de 1995 e 2005 .

Exercício 3: Utilizando o estágio $match , escreva uma agregação para retornar somente os clientes do sexo "FEMININO" e com data de nascimento entre os anos de 1995 e 2005 , limitando a quantidade de documentos retornados em 5 .

Exercício 4: Conte quantos clientes do estado SC existem na coleção. Retorne um documento em que o campo _id contenha a UF e outro campo com o total.

Exercício 5: Agrupe os clientes por sexo . Retorne o total de clientes de cada sexo no campo total .

Exercício 6: Agrupe os clientes por sexo e uf . Retorne o total de clientes de cada sexo no campo total .

Exercício 7 : Utilizando a mesma agregação do exercício anterior, adicione um estágio de projeção para modificar os documentos de saída, de forma que se pareçam com o documento a seguir (não se importe com a ordem dos campos):

Copiar

```
{
  "estado": "SP",
  "sexo": "MASCULINO",
  "total": 100
}
```

Exercício 8 : Descubra quais são os 5 clientes que gastaram o maior valor.

Exercício 9 : Descubra quais são os 10 clientes que gastaram o maior valor no ano de 2019 .

Exercício 10 : Descubra quantos clientes compraram mais de 5 vezes. Retorne um documento que contenha somente o campo clientes com o total de clientes.

Dica: O operador $count pode simplificar sua *query* .

Exercício 11 : Descubra quantos clientes compraram menos de três vezes entre os meses de Janeiro de 2020 e Março de 2020 .

Exercício 12 : Descubra quais as três uf s que mais compraram no ano de 2020 . Retorne os documentos no seguinte formato:

Copiar

```
{
  "totalVendas": 10,
  "uf": "SP"
}
```

Exercício 13 : Encontre qual foi o total de vendas e a média de vendas de cada `uf` no ano de `2019` . Ordene os resultados pelo nome da `uf` . Retorne os documentos no seguinte formato:

Copiar

```
{
  "_id": "MG",
  "mediaVendas": 9407.129225352113,
  "totalVendas": 142
}
```
