Você será capaz de:

- Executar operações de soma e subtração de valores em um *pipeline* ;
- Trabalhar com datas em *pipelines* , adicionando ou subtraindo tempo;
- Executar operações de multiplicação e divisão em *pipelines* , utilizando valores fixos ou variáveis;
- Adicionar novos campos aos documentos durante um *pipeline* .

Por que isso é importante?

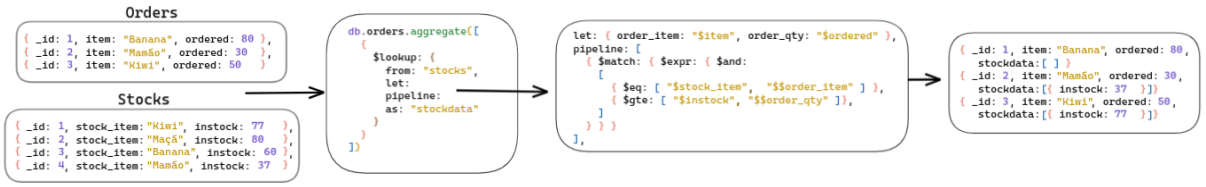
As operações de agregação são parte essencial no dia a dia de uma aplicação. Saber como utilizar bem os operadores para formatar os dados e tirar leituras e *insights* deles deve fazer parte do seu conhecimento. Operações aritméticas também são importantes para não sobrecarregar a aplicação com uma carga desnecessária de dados para processamento. Com o `aggregation pipeline` , é possível utilizar a camada de banco de dados para realizar esse processamento!

Aplicando condições ao Join com $lookup

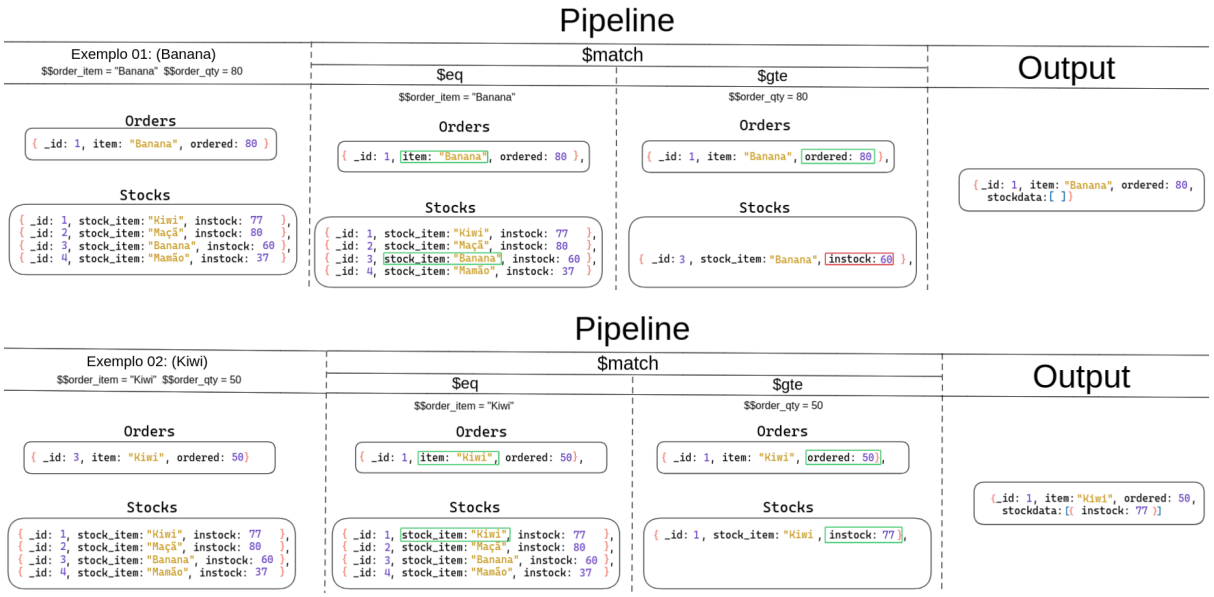
Aprendemos a base do operador `$lookup` no último conteúdo, mas você pode incrementar ainda mais adicionando a ele expressões mais elaboradas e aplicando vários operadores que você já conhece. Também pode referenciar campos dos documentos de entrada para serem utilizados nas condições e até mesmo montar um *pipeline* dentro dele. Para isso, existem mais dois parâmetros, ambos opcionais:

- `let` : define as variáveis que serão utilizadas no estágio *pipeline* dentro do `$lookup` . É necessário porque o estágio *pipeline* não consegue acessar diretamente os campos dos documentos de entrada, então esses campos precisam ser definidos previamente e transformados em variáveis;
- `pipeline` : define as condições ou o *pipeline* que será executado na coleção de junção. Se você quiser todos os documentos da coleção de junção, é só especificá-lo como vazio (`[]`).

Vamos observar na imagem a seguir o fluxo completo do lookup com `let` e `pipeline` :



Observamos as *collections* `orders` e `stocks` neste fluxo e aplicamos o operador `lookup` na *collection* `orders` com let e pipeline. O nosso `from` e `as` , continuam com as mesmas funcionalidades explicadas no dia anterior, ou seja, o `from` indica qual *collection* desejamos realizar a integração e o `as` indica a chave de destino que o resultado da nossa pipeline era ser arquivada.
Agora vamos olhar com mais cuidado as funcionalidades do `let` e `pipeline` nas imagens a baixo:



Nesses dois exemplos anteriores, foi destacado o processo ao aplicarmos o `$match` em conjunto com os demais operadores `$eq` e `$gte` .
O `$eq` é responsável por verificar quais itens tem os mesmos valores para o que declaramos em `$$ordem_item` e `$stock_item` que é o nome da nossa chave proveniente da nossa *collection* `stocks` . Passando por este filtro o próximo passo será aplicação do operador `$gte` para os documentos que foram alinhados. Nos exemplos anteriores simulamos dois itens passando por este processo e nosso exemplo 1 chegou ao fim sem valor em nossa chave `stock_data` , pois o valor da ordem foi superior ao que tinhamos em estoque.
Agora vamos colocar mão no código e entender ainda mais na prática. Considere os seguintes documentos na coleção `orders` :

Copiar

```
use example_db;
db.orders.insertMany([
```

```
  { _id: 1, item: "almonds", price: 12, ordered: 2 },
  { _id: 2, item: "pecans", price: 20, ordered: 1 },
  { _id: 3, item: "cookies", price: 10, ordered: 60 }
]);
```

E os seguintes documentos na coleção `warehouses` :

Copiar

```
use example_db;
db.warehouses.insertMany([
  { _id: 1, stock_item: "almonds", warehouse: "A", instock: 120 },
  { _id: 2, stock_item: "pecans", warehouse: "A", instock: 80 },
  { _id: 3, stock_item: "almonds", warehouse: "B", instock: 60 },
  { _id: 4, stock_item: "cookies", warehouse: "B", instock: 40 },
  { _id: 5, stock_item: "cookies", warehouse: "A", instock: 80 }
]);
```

A operação a seguir junta todos os documentos da coleção `orders` com a coleção `warehouse` através do campo `item` se a quantidade em estoque (`instock`) for suficiente para cobrir a quantidade vendida (`ordered`). Os documentos que dão *match* são colocados no campo `stockdata` .

Copiar

```
db.orders.aggregate([
  {
    $lookup: {
      from: "warehouses",
      let: { order_item: "$item", order_qty: "$ordered" },
      pipeline: [
        {
          $match: {
            $expr: {
              $and: [
                { $eq: [ "$stock_item",  "$$order_item" ] },
                { $gte: [ "$instock", "$$order_qty" ] }
              ]
            }
          }
        },
        { $project: { stock_item: 0, _id: 0 } }
      ],
      as: "stockdata"
    }
  }
]);
```

Note que, dentro do estágio *pipeline* , temos um operador `$match` que utiliza uma expressão (`$expr`). Esta, por sua vez, utiliza o operador `$and` . Dentro do `$and` , são utilizados operadores de igualdade (`$eq`) e de comparação (`$gte`). O símbolo `$` é utilizado para se referir aos campos da coleção `warehouse` (a coleção de junção), enquanto `$$` se refere às variáveis definidas no estágio `let` (os campos da coleção `orders`). Os campos `_id` e `stock_item` da coleção de join (`warehouse`) são excluídos com o uso do operador `$project` .
Como resultado, os documentos abaixo serão retornados:

Copiar

```
{
  "_id" : 1,
  "item" : "almonds",
  "price" : 12,
  "ordered" : 2,
  "stockdata" : [
     {
        "warehouse" : "A",
        "instock" : 120
     },
     {
        "warehouse" : "B",
        "instock" : 60
     }
  ]
}
{
  "_id" : 2,
  "item" : "pecans",
  "price" : 20,
  "ordered" : 1,
  "stockdata" : [
     {
        "warehouse" : "A",
        "instock" : 80
     }
  ]
}
{
  "_id" : 3,
  "item" : "cookies",
  "price" : 10,
```

```
  "ordered" : 60,
  "stockdata" : [
     {
        "warehouse" : "A",
        "instock" : 80
     }
  ]
}
```

Em um novo paralelo com a linguagem `SQL` , teríamos algo como o seguinte:

Copiar

```
SELECT * stockdata
FROM orders
WHERE stockdata IN (
  SELECT warehouse, instock
  FROM warehouses
  WHERE stock_item = orders.item
  AND instock >= orders.ordered
);
```

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" },
]);

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' },
]);
```

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

Antes de avançarmos no conteúdo, crie um banco de dados chamado `storage` e rode a `query` abaixo. Ele será necessário para os próximos exercícios de fixação.

Copiar

```
db.products.insertMany([
  { "name": "Ball", "purchase_price": 7.6, "taxes": 1.9,
"sale_price": 12.5, "quantity": 5 },
  { "name": "Baseball bat", "purchase_price": 18.5, "taxes": 5.3,
"sale_price": 39.9, "quantity": 12 },
  { "name": "Sneakers", "purchase_price": 10.4, "taxes": 1.50,
"sale_price": 14.9, "quantity": 3 },
  { "name": "Gloves", "purchase_price": 2.85, "taxes": 0.90,
"sale_price": 5.70, "quantity": 34 },
  { "name": "Jacket", "purchase_price": 28.9, "taxes": 10.80,
"sale_price": 59.9, "quantity": 20 },
  { "name": "Mousepad", "purchase_price": 16.6, "taxes": 3.40,
"sale_price": 29.9, "quantity": 8 },
```

```
  { "name": "Monitor", "purchase_price": 119.9, "taxes": 39.20,
"sale_price": 240.6, "quantity": 11 },
]);
```

Expressão $add

Com a expressão `$add` , é possível somar valores numéricos ou datas. Se um dos argumentos for do tipo `date` , o outro argumento será tratado como milissegundos e adicionado à data.
Considere os seguintes documentos na coleção `sales` :

Copiar

```
{ _id: 1, item: "abc", price: 10, fee: 2, date:
ISODate("2014-03-01T08:00:00Z") },
{ _id: 2, item: "jkl", price: 20, fee: 1, date:
ISODate("2014-03-01T09:00:00Z") },
{ _id: 3, item: "xyz", price: 5,  fee: 0, date:
ISODate("2014-03-15T09:00:00Z") }
```

Utilizando a expressão `$add` no estágio `$project` , você pode criar um novo campo com o valor total somando os campos `price` e `fee` :

Copiar

```
db.sales.aggregate([
  { $project: { item: 1, total: { $add: ["$price", "$fee"] } } }
]);
```

A operação retorna o seguinte resultado:

Copiar

```
{ "_id" : 1, "item" : "abc", "total" : 12 }
{ "_id" : 2, "item" : "jkl", "total" : 21 }
{ "_id" : 3, "item" : "xyz", "total" : 5 }
```

Para valores do tipo `date` , um dos argumentos sempre será tratado como milissegundos. Imagine que você queira adicionar `3` dias ao valor do campo `date` . Você consegue fazer de duas maneiras. A primeira é passar em um dos argumentos o número equivalente a 3 dias em milissegundos (`2,592e+8`). A segunda é criar uma expressão que devolva esse número:

Copiar

```
db.sales.aggregate([
  { $project: { item: 1, billing_date: { $add: ["$date", 2.592e+8] }
} }
]);
```

Copiar

```
db.sales.aggregate([
  { $project: { item: 1, billing_date: { $add: ["$date", 3 * 24 * 60 * 60 * 60000] } } }
]);
```

Note que as duas operações retornam o mesmo valor para o novo campo `billing_date` .

Para Fixar

Utilizando o banco de dados `storage` , faça o seguinte exercício:

1. Calcule qual o custo total de cada produto, considerando o preço de compra e os impostos.

Expressão $subtract

Com a expressão `subtract` , podemos subtrair dois valores numéricos para retornar a diferença entre eles, ou duas datas para retornar a diferença entre elas em milissegundos. O segundo argumento sempre será subtraído do primeiro .
Considere os seguintes documentos na coleção `sales` :

Copiar

```
{
  _id: 1,
  item: "abc",
  price: 10,
  fee: 2,
  discount: 5,
  date: ISODate("2014-03-01T08:00:00Z")
},
{
  _id: 2,
  item: "jkl",
  price: 20,
  fee: 1,
  discount: 2,
  date: ISODate("2014-03-01T09:00:00Z")
}
```

Em uma única operação no estágio `$project` , podemos montar uma expressão um pouco mais complexa, utilizando `$add` para calcular o total e o `$subtract` para aplicar um desconto no subtotal:

Copiar

```
db.sales.aggregate([
  {
    $project: {
      item: 1,
      total: {
        $subtract: [
          { $add: ["$price", "$fee"] },
          "$discount"
        ]
      }
    }
  }
]);
```

Observe que um dos argumentos do `$subtract` é o resultado de uma expressão (`$add`) que soma dois campos da coleção (`price` e `fee`). O segundo argumento (valor a ser subtraído) recebe o campo `$discount` . Os seguintes documentos serão retornados:

Copiar

```
{ "_id" : 1, "item" : "abc", "total" : 7 }
{ "_id" : 2, "item" : "jkl", "total" : 19 }
```

É possível subtrair duas datas também. A operação a seguir utiliza a expressão `$subtract` para subtrair o valor do campo `date` da data corrente, utilizando a variável de sistema `NOW` (disponível a partir da versão 4.2 do MongoDB) e retorna a diferença em milissegundos:

Copiar

```
db.sales.aggregate([
  {
    $project: {
      item: 1,
      dateDifference: {
        $subtract: ["$$NOW", "$date"]
      }
    }
  }
]);
```

Alternativamente, você pode utilizar a função `Date()` para obter a data corrente:

Copiar

```
db.sales.aggregate([
  {
```

```
    $project: {
      item: 1,
      dateDifference: {
        $subtract: [new Date(), "$date"]
      }
    }
  }
]);
```

Você também pode utilizar milissegundos como argumento da subtração. A operação seguinte subtrai `5` minutos do campo `date` :

Copiar

```
db.sales.aggregate([
  {
    $project: {
      item: 1,
      dateDifference: {
        $subtract: ["$date", 5 * 60 * 1000]
      }
    }
  }
]);
```

Para Fixar

Utilizando o banco de dados `storage` , faça o seguinte exercício:

1. Calcule qual o lucro total de cada produto, considerando o preço de compra, os impostos e seu valor de venda.

Expressão $ceil

A expressão `$ceil` basicamente arredonda o número especificado para "cima". Ela executa a função matemática `teto` que converte um número `x` no número inteiro mais próximo, que seja maior ou igual a `x` .

Considere os seguintes documentos na coleção `samples` :

Copiar

```
{ _id: 1, value: 9.25 },

{ _id: 2, value: 8.73 },
```

```
{ _id: 3, value: 4.32 },
{ _id: 4, value: -5.34 }
```

A operação a seguir utiliza a expressão `$ceil` no estágio `$project` para retornar um novo campo chamado `ceilingValue` :

Copiar

```
db.samples.aggregate([
  { $project: { value: 1, ceilingValue: { $ceil: "$value" } } }
]);
```

O valor original também é retornado:

Copiar

```
{ "_id" : 1, "value" : 9.25, "ceilingValue" : 10 }
{ "_id" : 2, "value" : 8.73, "ceilingValue" : 9 }
{ "_id" : 3, "value" : 4.32, "ceilingValue" : 5 }
{ "_id" : 4, "value" : -5.34, "ceilingValue" : -5 }
```

Expressão $floor

Já a expressão `$floor` retorna o maior número inteiro menor ou igual ao número especificado, ou seja, faz um arredondamento para baixo. Considere os mesmos documentos da coleção `sample` utilizados no exemplo anterior. Se você aplicar a expressão `$floor` no estágio `$project` :

Copiar

```
db.samples.aggregate([
  { $project: { value: 1, floorValue: { $floor: "$value" } } }
]);
```

Terá o retorno do valor original e o calculado:

Copiar

```
{ "_id" : 1, "value" : 9.25, "floorValue" : 9 }
{ "_id" : 2, "value" : 8.73, "floorValue" : 8 }
{ "_id" : 3, "value" : 4.32, "floorValue" : 4 }
{ "_id" : 4, "value" : -5.34, "floorValue" : -6 }
```

Funções de arredondamento podem ser úteis em vários casos de cálculos na camada de banco de dados.

Expressão $round

A expressão `$round` retorna o número inteiro mais próximo do valor atual e também permite definir a quantidade de casas decimais que você quer manter ao arredondar.
Considere os mesmos documentos da coleção `sample` utilizados no exemplo anterior. Se você aplicar a expressão `$round` no estágio `$project` :

Copiar
```
db.samples.aggregate([
  { $project: { value: 1, roundedValue: { $round: ["$value"] } } }
]);
```

Terá o retorno do valor original e o calculado:

Copiar
```
{ "_id" : 1, "value" : 9.25, "roundedValue" : 9 }
{ "_id" : 2, "value" : 8.73, "roundedValue" : 9 }
{ "_id" : 3, "value" : 4.32, "roundedValue" : 4 }
{ "_id" : 4, "value" : -5.34, "roundedValue" : -5 }
```

Observe que para todos os valores, o `$round` arredondou os valores para o mais próximo, podendo ser maior ou menor. O que interessa aqui é qual o inteiro mais próximo, independente se for maior ou menor que o valor anterior. Essa é uma das diferenças do `$round` para o `$ceil` e para o `$floor` .
Outra diferença é que para o `$round` nós passamos uma array como argumento, em vez de um valor *plano* , isso acontece, para caso, passemos um segundo parâmetro ele vai arredondar mantendo a quantidade de casas decimais que for definida. Vamos ver um exemplo.

Copiar
```
db.samples.aggregate([
  { $project: { value: 1, roundedValue: { $round: ["$value", 1] } }
}
]);
```

Terá o retorno do valor original e o calculado:

Copiar
```
{ "_id" : 1, "value" : 9.25, "roundedValue" : 9.2 }
{ "_id" : 2, "value" : 8.73, "roundedValue" : 8.7 }
{ "_id" : 3, "value" : 4.32, "roundedValue" : 4.3 }
```

```
{ "_id" : 4, "value" : -5.34, "roundedValue" : -5.3 }
```

Perceba que ele arrendodou para o valor em *float* mais próximo do valor atual, considerando que vai precisar manter uma casa decinmal. É assim que funciona o `$round` .
Funções de arredondamento podem ser úteis em vários casos de cálculos na camada de banco de dados.

Para Fixar

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

1. Retorne o menor número inteiro relativo ao preço de venda de cada produto;
2. Retorne o maior número inteiro relativo ao lucro total sobre cada produto. *Nota: Desconsiderar taxas (taxes)*

Expressão $abs

A expressão `$abs` retorna o valor absoluto de um número.
Essa expressão é muito útil para encontrar a diferença entre dois valores.
Veja um exemplo considerando os documentos da coleção `ratings` :

Copiar
```
{ _id: 1, start: 5, end: 8 },
{ _id: 2, start: 4, end: 4 },
{ _id: 3, start: 9, end: 7 },
{ _id: 4, start: 6, end: 7 }
```

Aplicando a expressão `$abs` combinada com a expressão `$subtract` no estágio `$project` , podemos retornar a diferença entre os valores dos campos `start` e `end` :

Copiar
```
db.ratings.aggregate([
  {
    $project: {
      delta: {
        $abs: { $subtract: ["$start", "$end"] }
      }
    }
  }
]);
```

Copiar
```
{ "_id" : 1, "delta" : 3 }
```

```
{ "_id" : 2, "delta" : 0 }
{ "_id" : 3, "delta" : 2 }
{ "_id" : 4, "delta" : 1 }
```

Para Fixar

Utilizando o banco de dados `storage` , faça o seguinte exercício:

1. Calcule o valor absoluto do lucro total de cada produto.

Expressão $multiply

A expressão `$multiply` multiplica dois valores numéricos. Esses valores devem ser passados num *array* , como nas outras expressões anteriores.

Vamos considerar os seguintes documentos na coleção `sales` :

Copiar

```
{ _id: 1, item: "abc", price: 10, quantity: 2, date: ISODate("2014-03-01T08:00:00Z") },

{ _id: 2, item: "jkl", price: 20, quantity: 1, date: ISODate("2014-03-01T09:00:00Z") },

{ _id: 3, item: "xyz", price: 5, quantity: 10, date: ISODate("2014-03-15T09:00:00Z") }
```

Na agregação a seguir, utilizamos o `$multiply` no estágio `$project` para projetar um novo campo chamado `total` , que conterá o valor da multiplicação entre os campos `price` e `quantity` :

Copiar

```
db.sales.aggregate([

  {

    $project: {

      date: 1,

      item: 1,
```

```
      total: {
        $multiply: ["$price", "$quantity"]
      }
    }
  }
]);
```

Retornando o seguinte resultado:

Copiar

```
{ "_id" : 1, "item" : "abc", "date" :
ISODate("2014-03-01T08:00:00Z"), "total" : 20 }
{ "_id" : 2, "item" : "jkl", "date" :
ISODate("2014-03-01T09:00:00Z"), "total" : 20 }
{ "_id" : 3, "item" : "xyz", "date" :
ISODate("2014-03-15T09:00:00Z"), "total" : 50 }
```

Para Fixar

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

1. Calcule qual o valor total em estoque de cada produto, considerando o preço de venda e a quantidade;
2. Calcule qual será o lucro total de cada produto caso todo o estoque seja vendido.

Expressão $divide

A expressão `$divide` , como o próprio nome sugere, divide dois valores. O primeiro argumento é o dividendo , e o segundo é o divisor .

Considere os seguintes documentos na coleção `planning` :

Copiar

```
{ _id: 1, name: "A", hours: 80, resources: 7 },
{ _id: 2, name: "B", hours: 40, resources: 4 }
```

A agregação abaixo utiliza o `$divide` para dividir o valor do campo `hours` por `8` e calcular o número de dias de trabalho (`workdays`):

Copiar

```
db.planning.aggregate([
  {
    $project: {
      name: 1,
      workdays: {
        $divide: ["$hours", 8]
      }
    }
  }
]);
```

Retornando os seguintes documentos:

Copiar

```
{ "_id" : 1, "name" : "A", "workdays" : 10 }
{ "_id" : 2, "name" : "B", "workdays" : 5 }
```

Para Fixar

Utilizando o banco de dados `storage` , faça o seguinte exercício:

1. Calcule qual será o preço de venda de cada produto caso haja uma promoção de 50% de desconto.

Estágio $addFields

O `$addFields` é um estágio que adiciona novos campos aos documentos. A saída desse estágio conterá todos os campos existentes nos documentos de entrada e adicionará os novos campos especificados.

Você pode incluir subdocumentos ou *arrays* de subdocumentos, utilizando o conceito de *dot notation* . Um *pipeline* pode conter mais de um estágio `$addFields` .

Considere os documentos abaixo na coleção `scores` :

Copiar

```
{
  _id: 1,
  student: "Maya",
  homework: [10, 5, 10],
  quiz: [10, 8],
  extraCredit: 0
},
{
  _id: 2,
  student: "Ryan",
  homework: [5, 6, 5],
  quiz: [8, 8],
  extraCredit: 8
}
```

A operação de agregação abaixo utiliza o `$addFields` duas vezes para incluir três novos campos nos documentos de saída:

Copiar

```
db.scores.aggregate([
  {
    $addFields: {
      totalHomework: { $sum: "$homework" } ,
```

```
      totalQuiz: { $sum: "$quiz" }
    }
  },
  {
    $addFields: {
      totalScore: {
        $add: [ "$totalHomework", "$totalQuiz", "$extraCredit" ]
      }
    }
  }
]);
```

O primeiro estágio adiciona o campo `totalHomework` somando os valores contidos no *array* `homework` . Também adiciona outro campo chamado `totalQuiz` somando os valores do *array* `quiz` .

O segundo estágio adiciona o campo `totalScore` , que, por sua vez, soma os valores dos campos `totalHomework` , `totalQuiz` e `extraCredit` .

Note que o resultado mantém os campos originais do documento de entrada, juntamente com os três novos campos adicionados:

Copiar

```
{
  "_id" : 1,
  "student" : "Maya",
  "homework" : [ 10, 5, 10 ],
  "quiz" : [ 10, 8 ],
  "extraCredit" : 0,
  "totalHomework" : 25,
```

```
  "totalQuiz" : 18,
  "totalScore" : 43
}
{
  "_id" : 2,
  "student" : "Ryan",
  "homework" : [ 5, 6, 5 ],
  "quiz" : [ 8, 8 ],
  "extraCredit" : 8,
  "totalHomework" : 16,
  "totalQuiz" : 16,
  "totalScore" : 40
}
```

Para Fixar

Utilizando o banco de dados `storage` , faça o seguinte exercício:

1. Calcule o valor total do estoque, considerando que cada produto valha o mesmo que seu preço de venda. Lembre-se da quantidade.

Cheat Sheet

A intenção deste conteúdo é fornecer uma base da sintaxe e proporcionar uma consulta rápida da estrutura para realização dos exercícios.

Na sessão de recursos adicionais, há um link para uma versão completa do Cheat Sheet.

$lookup (let/pipeline)

Template

Copiar

```
db.collection.aggregate([
  {
    $lookup:
      {
        from: <coleção para unir>,
        let: { <var_1>: <expressão>, …, <var_n>: <expressão> },
        pipeline: [ <pipeline a ser executada na coleção unida> ],
        as: <campo do array de saída>
      }
}
]);
```

Exemplo

Copiar

```
db.orders.aggregate([
   {
      $lookup:
         {
           from: "warehouses",
           let: { order_item: "$item", order_qty: "$ordered" },
           pipeline: [
              { $match:
                 { $expr:
                    { $and:
                       [
```

```
                        { $eq: [ "$stock_item",  "$$order_item" ]
},
                        { $gte: [ "$instock", "$$order_qty" ] }
                    ]
                }
            }
        },
        { $project: { stock_item: 0, _id: 0 } }
    ],
    as: "stockdata"
  }
}
])
```

Documentação

$add

Template

Copiar

```
db.collection.aggregate([
  {
    $project: {
      <campo>: {
        $add: [ <expressão1>, <expressão2>, ... ]
      },
    },
  },
```

```
]);
```

Exemplo

Copiar

```
db.products.aggregate([
  {
    $project: {
      item: 1,
      total: {
        $add: ["$price", "$fee"]
      },
    },
  },
]);
```

Documentação

$subtract

Template

Copiar

```
db.collection.aggregate([
  {
    $project: {
      <campo>: {
        $subtract: [
          <expression1>,
          <expression2>
```

```
        ]
      },
    },
  },
]);
```

Exemplo

Copiar

```
db.products.aggregate([
  {
    $project: {
      item: 1,
      total: {
        $subtract: [
          { $add: ["$price", "$fee"] },
          "$discount"
        ]
      },
    },
  },
]);
```

Documentação

$ceil

Template

Copiar

```
db.collection.aggregate([
  {
    $project: {
      roundedNumber: {
        $ceil: <numero>,
      },
    },
  },
]);
```

Exemplo

Copiar

```
db.movies.aggregate([
  {
    $project: {
      value: 1,
      ceilingValue: {
        $ceil: "$rating",
      },
    },
  },
]);
```

Documentação

$floor

Template

Copiar

```
db.collection.aggregate([
  {
    $project: {
      value: 1,
      roundedNumber: {
        $floor: <numero>,
      },
    },
  },
]);
```

Exemplo

Copiar

```
db.movies.aggregate([
  {
    $project: {
      value: 1,
      floorValue: {
        $floor: "$value",
      },
    },
  },
]);
```

Documentação

$abs

Template

Copiar

```
db.collection.aggregate([
  {
    project: {
      <campo>: {
        $abs: <numero>,
      },
    },
  },
]);
```

Exemplo

Copiar

```
db.operations.aggregate([
  {
    project: {
      delta: {
        $abs: { $subtract: ["$start", "$end"] },
      },
    },
  },
]);
```

Documentação

$multiply

Template

Copiar

```
db.collection.aggregate([
  {
    $project: {
      <campo>: {
        $multiply: [ <expressão1>, <expressão2>, ... ]
      },
    },
  },
]);
```

Exemplo

Copiar

```
db.operations.aggregate([
  {
    $project: {
      date: 1,
      item: 1,
      total: {
        $multiply: [
          "$price",
          "$quantity"
        ]
      },
```

```
    },
  },
]);
```

Documentação

$divide

Template

Copiar

```
db.collection.aggregate([
  {
    $project: {
      <campo>: {
        $divide: [ <expressão1>, <expressão2> ]
      },
    },
  },
]);
```

Exemplo

Copiar

```
db.employees.aggregate([
  {
    $project: {
      name: 1,
      workdays: {
        $divide: ["$hours", 8]
```

```
      },
    },
  },
]);
```

Documentação

$addFields

Template

Copiar

```
db.collection.aggregate([
  {
    $addFields: {
      <novoCampo1>: <valor> ,
      <novoCampo2>: <valor> ,
      ...
    },
  },
]);
```

Exemplo

Copiar

```
db.school.aggregate([
  {
    $addFields: {
      totalHomework: { $sum: "$homework" } ,
      totalQuiz: { $sum: "$quiz" }
```

```
    },
  },
  {
    $addFields: {
      totalScore: {
        $add: [ "$totalHomework", "$totalQuiz", "$extraCredit" ]
      },
    },
  },
]);
```

Documentação

Agora, a prática

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. Você continuará utilizando o banco de dados `erp` do dia anterior. Nos exercícios 1 a 8 , você utilizará o mesmo *pipeline* . A ideia é começar com um *pipeline* pequeno e ir adicionando estágios à medida que você for evoluindo nos exercícios. Vamos lá?
Exercício 1 : Utilize uma combinação das expressões aritméticas e adicione um campo chamado `idade` à coleção `clientes` . Algumas dicas:

- arredonde para baixo o valor da idade;
- calcule a idade usando a diferença entre a data corrente e a data de nascimento;
- 1 dia é igual a 86400000 milissegundos.

Exercício 2 : Utilizando o novo campo `idade` , conte quantos clientes têm entre `18` e `25` anos.

Exercício 3 : Remova os estágios `$count` e `$match` do exercício anterior e adicione um estágio no *pipeline* que coloque as compras do cliente no campo `compras` .
Exercício 4 : Selecione TODOS os clientes que compraram entre `Junho de 2019` e `Março de 2020` .
Exercício 5 : Confira o número de documentos retornados pelo *pipeline* com o método `itcount()` . Até aqui, você deve ter `486` documentos sendo retornados.
Exercício 6 : Ainda nesse *pipeline* , descubra os `5` estados com mais compras.
Exercício 7 : Descubra o cliente que mais consumiu `QUEIJO PRATO` . Retorne um documento com a seguinte estrutura:

Copiar

```
{
  "nomeCliente": "NOME",
  "uf": "UF DO CLIENTE",
  "totalConsumido": 100
}
```

Exercício 8 : Selecione todas as vendas do mês de `Março de 2020` , com `status EM SEPARACAO` . Acrescente um campo chamado `dataEntregaPrevista` com valor igual a três dias após a data da venda. Retorne apenas os campos `clienteId` , `dataVenda` e `dataEntregaPrevista` .

Bônus

Exercício 9 : Calcule a diferença absoluta em dias entre a data da primeira entrega prevista e a última, considerando o *pipeline* do exercício 8.