

# Aggregation Framework



Profa. Dra. Kelly Lais Wiggers

# Operadores de agregação

Operações de agregação processam diversos documentos e geram resultados calculados. Você pode usar operações de agregação para:

- Agrupar valores de diversos documentos.
- Executar operações nos dados agrupados para gerar um único resultado.
- Analisar alterações de dados ao longo do tempo.

# Operadores de agregação

Para realizar operações de agregação, você pode usar:

- Pipelines de agregação, considerado o método principal para realizar agregações.
- Métodos de agregação de finalidade específica, considerados simples, mas não possuem os mesmos recursos de um pipeline de agregação.

# Aggregation Pipelines

Na estrutura de agregação, pensamos em estágios em vez de comandos. E a "saída" do estágio são documentos.

- Um pipeline começa com documentos
- Esses documentos vêm de uma coleção, uma visualização ou um estágio especialmente projetado
- Em cada estágio, os documentos entram, o trabalho é feito e os documentos saem
- Os estágios em si são definidos usando a sintaxe do documento
- Um aggregation pipeline pode retornar resultados para grupos de documentos. Por exemplo, retornar o valor total, médio, máximo e mínimo.

# Estágios de agregação

- Há mais de 28 estágios de agregação
- Exemplos:
  - \$match, \$unwind, \$group, \$sort, \$limit, \$project, and finally a \$skip stage.
- Forma geral:

```
db.collectionName.aggregate(pipeline, options)
```

onde

- *collectionName* – é o nome de uma coleção,
- *pipeline* – é uma matriz que contém os estágios de agregação,
- *options* – parâmetros opcionais para a agregação

# Aggregation Pipelines

- Os pipelines de agregação executados com o método `db.collection.aggregate()` não modificam documentos em uma coleção, a menos que o pipeline contenha um estágio `$merge` ou `$out`.

# Exemplo com pipeline de agregação

```
db.orders.aggregate( [  
  // Stage 1: Filter pizza order documents by pizza size  
  {  
    $match: { size: "medium" }  
  },  
  // Stage 2: Group remaining documents by pizza name and calculate total  
  // quantity  
  {  
    $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }  
  }  
] )
```

# Exemplo com pipeline de agregação

O estágio `$match`:

- Filtra os documentos de pedido de pizza em pizzas com um `size` de `medium`.
- Repassa os documentos restantes para o estágio `$group`.

O estágio `$group`:

- Agrupa os documentos restantes por `name` da pizza.
- Utiliza `$sum` para calcular a `quantity` total do pedido para cada pizza de `name`. O `total` é armazenado no campo `totalQuantity` retornado pelo aggregation pipeline.



# Métodos de agregação de finalidade específica

Os métodos de agregação de finalidade específica agregam documentos a partir de uma única coleção. Os métodos são simples, mas não possuem os mesmos recursos de um pipeline de agregação.

Método	Descrição
<code>db.collection.estimatedDocumentCount()</code>	Gera uma contagem aproximada dos documentos em uma coleção ou visualização.
<code>db.collection.count()</code>	Gera uma contagem do número de documentos em uma coleção ou visualização.
<code>db.collection.distinct()</code>	Gera uma array de documentos com valores distintos para o campo especificado.

# Exemplo

Vamos criar uma collection

nomeada universities

Com os dados:

- país
- cidade
- nome
- localização
- estudantes (ano e numero)

```
test> use meubanco
switched to db meubanco
meubanco> db.universities.insertMany([
... {
...   country : 'Spain',
...   city : 'Salamanca',
...   name : 'USAL',
...   location : {
...     type : 'Point',
...     coordinates : [ -5.6722512, 17, 40.9607792 ]
...   },
...   students : [
...     { year : 2014, number : 24774 },
...     { year : 2015, number : 23166 },
...     { year : 2016, number : 21913 },
...     { year : 2017, number : 21715 }
...   ]
... },
... {
...   country : 'Spain',
...   city : 'Salamanca',
...   name : 'UPSA',
...   location : {
...     type : 'Point',
...     coordinates : [ -5.6691191, 17, 40.9631732 ]
...   },
...   students : [
...     { year : 2014, number : 4788 },
...     { year : 2015, number : 4821 },
...     { year : 2016, number : 6550 },
...     { year : 2017, number : 6125 }
...   ]
... }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6734831456e7bbf91cfe6911'),
    '1': ObjectId('6734831456e7bbf91cfe6912')
```

# Exemplo

Agora a collection courses  
com os dados:

- universidade
- nome
- nível

```
meubanco> db.courses.insertMany([
... {
...   university : 'USAL',
...   name : 'Computer Science',
...   level : 'Excellent'
... },
... {
...   university : 'USAL',
...   name : 'Electronics',
...   level : 'Intermediate'
... },
... {
...   university : 'USAL',
...   name : 'Communication',
...   level : 'Excellent'
... }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('6734833356e7bbf91cfe6913'),
    '1': ObjectId('6734833356e7bbf91cfe6914'),
    '2': ObjectId('6734833356e7bbf91cfe6915')
  }
}
```

# Exemplo

Iniciamos a agregação buscando por:

- O estágio **\$match** nos permite escolher apenas aqueles documentos de uma coleção com os quais queremos trabalhar. Ele faz isso filtrando aqueles que não seguem nossos requisitos.
  - No exemplo a seguir, queremos trabalhar apenas com aqueles documentos que especificam que **Spain é o valor do campo country, e Salamanca é o valor do campo city.**
- Para obter uma saída legível, adicionarei **.pretty()** no final de todos os comandos.

```
meubanco> db.universities.aggregate([  
...   { $match : { country : 'Spain', city : 'Salamanca' } }  
... ]).pretty()
```

## Exemplo

## Saída

```
{
  _id: ObjectId('6734831456e7bbf91cfe6911'),
  country: 'Spain',
  city: 'Salamanca',
  name: 'USAL',
  location: { type: 'Point', coordinates: [ -5.6722512, 17, 40.9607792 ] },
  students: [
    { year: 2014, number: 24774 },
    { year: 2015, number: 23166 },
    { year: 2016, number: 21913 },
    { year: 2017, number: 21715 }
  ]
},
{
  _id: ObjectId('6734831456e7bbf91cfe6912'),
  country: 'Spain',
  city: 'Salamanca',
  name: 'UPSA',
  location: { type: 'Point', coordinates: [ -5.6691191, 17, 40.9631732 ] },
  students: [
    { year: 2014, number: 4788 },
    { year: 2015, number: 4821 },
    { year: 2016, number: 6550 },
    { year: 2017, number: 6125 }
  ]
}
```

# Exemplo

- É raro que você precise recuperar todos os campos em seus documentos. É uma boa prática retornar apenas os campos necessários para evitar processar mais dados do que o necessário.
- O estágio **\$project** é usado para fazer isso e adicionar quaisquer campos calculados que você precise.

# Exemplo

- Neste exemplo, precisamos apenas dos campos **country**, **city** e **name**.

```
meubanco> db.universities.aggregate([  
...   { $project : { _id : 0, country : 1, city : 1, name : 1 } }  
... ]).pretty()
```

Observe que devemos escrever explicitamente `_id : 0` quando este campo não for obrigatório. **Saída:**

```
[  
  { country: 'Spain', city: 'Salamanca', name: 'USAL' },  
  { country: 'Spain', city: 'Salamanca', name: 'UPSA' }  
]
```

## Exemplo

Com o estágio **\$group**, podemos executar todas as consultas de agregação ou resumo que precisamos, como encontrar contagens, totais, médias ou máximos.

Neste exemplo, queremos saber **o número de documentos por universidade** em nossa coleção 'universities':

```
meubanco> db.universities.aggregate([  
...   { $group : { _id : '$name', totaldocs : { $sum : 1 } } }  
... ]).pretty()
```

- Saída:

```
[ { _id: 'USAL', totaldocs: 1 }, { _id: 'UPSA', totaldocs: 1 } ]
```



# 0 estágio group

- O estágio **\$group** suporta certas expressões (operadores) permitindo que os usuários executem operações aritméticas, de matriz, booleanas e outras como parte do pipeline de agregação.
  - **\$count** → Calcula a quantidade de documentos no grupo fornecido.
  - **\$max** → Exibe o valor máximo do campo de um documento na coleção.
  - **\$min** → Exibe o valor mínimo do campo de um documento na coleção.
  - **\$avg** → Exibe o valor médio do campo de um documento na coleção.
  - **\$sum** → Soma os valores especificados de todos os documentos na coleção.
  - **\$push** → Adiciona valores extras na matriz do documento resultante.

# Outros operadores

- **\$out:** este é um tipo incomum de estágio porque permite que você carregue os resultados da sua agregação para uma nova coleção, ou para uma existente após descartá-la, ou até mesmo adicioná-los aos documentos existentes.

O estágio \$out deve ser o último estágio no pipeline.

Exemplo:

```
db.universities.aggregate([
  { $group : { id : '$name', totaldocs : { $sum : 1 } } },
  { $out : 'aggResults' }
])
```

Podemos verificar o resultado da nova coleção usando:

```
db.aggResults.find().pretty()
```

# Outros operadores

- **\$unwind**: você não pode trabalhar diretamente nos elementos de um array dentro de um documento com estágios como \$group. O estágio \$unwind nos permite trabalhar com os valores dos campos dentro de um array.

Exemplo:

```
db.universities.aggregate([  
  { $match : { name : 'USAL' } },  
  { $unwind : '$students' }  
]).pretty()
```

Aqui aplicamos o estágio \$unwind, sobre o array do aluno, e verificamos se obtemos um documento para cada elemento do array.

# Outros operadores

- **\$sort**: para classificar seus resultados pelo valor de um campo específico.

```
meubanco> db.universities.aggregate([
...   { $match : { name : 'USAL' } },
...   { $unwind : '$students' },
...   { $project : { _id : 0, 'students.year' : 1, 'students.number' : 1 } },
...   { $sort : { 'students.number' : -1 } }
... ]).pretty()
```

Saída:

```
[
  { students: { year: 2014, number: 24774 } },
  { students: { year: 2015, number: 23166 } },
  { students: { year: 2016, number: 21913 } },
  { students: { year: 2017, number: 21715 } }
]
```

# Exemplo

```
meubanco> db.universities.aggregate([
...   { $match : { name : 'USAL' } },
...   { $unwind : '$students' },
...   { $project : { _id : 0, 'students.year' : 1, 'students.number' : 1 } },
...   { $sort : { 'students.number' : -1 } }
... ]).pretty()
```

```
[
  { students: { year: 2014, number: 24774 } },
  { students: { year: 2015, number: 23166 } },
  { students: { year: 2016, number: 21913 } },
  { students: { year: 2017, number: 21715 } }
]
```

# Usando Aggregation no Compass

localhost:27017 > meubanco > universities

Documents 2

Aggregations

Schema

Indexes 1

Validation



\$match

Untitled - modified

SAVE

+ CREATE NEW

EXPORT TO LANGUAGE

2 Documents in the collection

Preview of documents

```
{
  "_id": ObjectId('6734831456e7bbf91cfe6911'),
  "country": "Spain",
  "city": "Salamanca",
  "name": "USAL",
  "location": Object,
  "students": Array (4)
}
```

```
{
  "_id": ObjectId('6734831456e7bbf91cfe6912'),
  "country": "Spain",
  "city": "Salamanca",
  "name": "UPSA",
  "location": Object,
  "students": Array (4)
}
```

Stage 1 \$match



```
1 /**
2  * query: The query in MQL.
3  */
4 {
5   name: 'USAL'
6 }
```

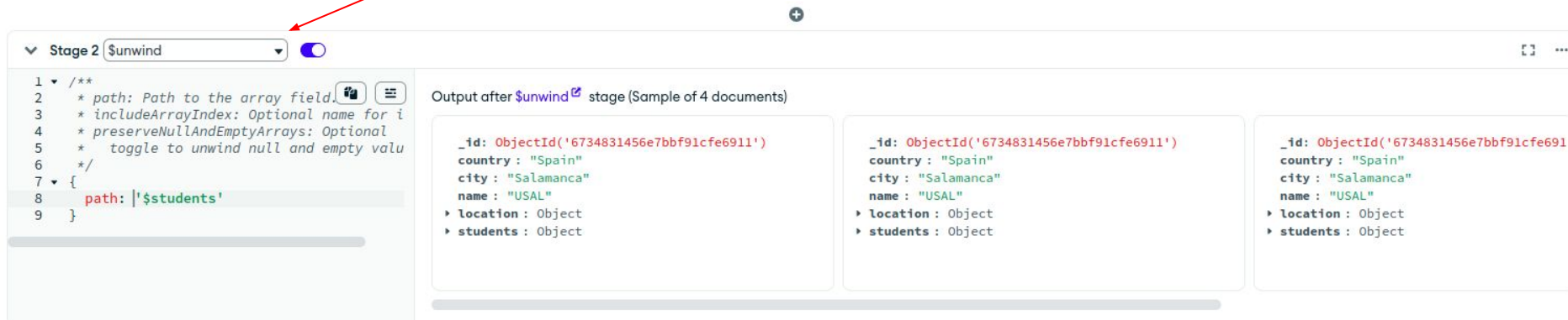


Output after \$match stage (Sample of 1 document)


```
{
  "_id": ObjectId('6734831456e7bbf91cfe6911'),
  "country": "Spain",
  "city": "Salamanca",
  "name": "USAL",
  "location": Object,
  "students": Array (4)
}
```

+ Add Stage

# Usando Aggregation no Compass



The screenshot shows the MongoDB Compass interface. At the top, a red arrow points to the 'Stage 2' dropdown menu, which is set to '\$unwind'. To the right of the dropdown is a toggle switch that is currently turned on. Below the dropdown, the aggregation pipeline is displayed in a code editor. The pipeline consists of a single stage: \$unwind, with the path '\$students' specified. To the right of the code editor, the output of the aggregation is shown. The output is titled 'Output after \$unwind stage (Sample of 4 documents)'. It displays three sample documents, each with the following fields: \_id, country, city, name, location, and students. The first document has \_id: ObjectId('6734831456e7bbf91cfe6911'), country: 'Spain', city: 'Salamanca', name: 'USAL', location: Object, and students: Object. The second document has \_id: ObjectId('6734831456e7bbf91cfe6911'), country: 'Spain', city: 'Salamanca', name: 'USAL', location: Object, and students: Object. The third document has \_id: ObjectId('6734831456e7bbf91cfe6911'), country: 'Spain', city: 'Salamanca', name: 'USAL', location: Object, and students: Object.

Stage 2 **\$unwind** 

```
1 /**
2  * path: Path to the array field.
3  * includeArrayIndex: Optional name for i
4  * preserveNullAndEmptyArrays: Optional
5  *   toggle to unwind null and empty values
6  */
7 {
8   path: '$students'
9 }
```

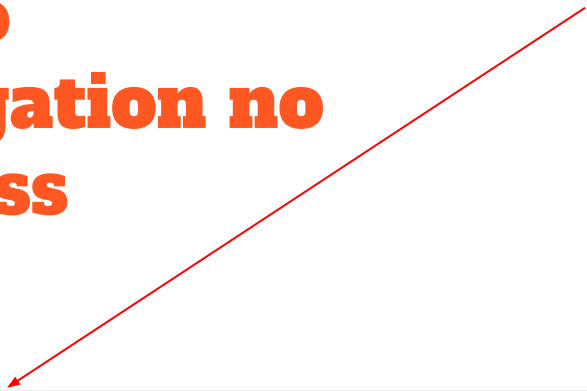
Output after **\$unwind** stage (Sample of 4 documents)


```
_id: ObjectId('6734831456e7bbf91cfe6911')
country: "Spain"
city: "Salamanca"
name: "USAL"
location: Object
students: Object
```

```
_id: ObjectId('6734831456e7bbf91cfe6911')
country: "Spain"
city: "Salamanca"
name: "USAL"
location: Object
students: Object
```

```
_id: ObjectId('6734831456e7bbf91cfe6911')
country: "Spain"
city: "Salamanca"
name: "USAL"
location: Object
students: Object
```

# Usando Aggregation no Compass



▼ Stage 3 \$project 

```
1  /**
2   * specifications: The fields to
3   * include or exclude.
4   */
5  {
6    _id : 0,
7    'students.year': 1,
8    'students.number': 1
9  }
```

Output after \$project stage (Sample of 4 documents)


► students : Object

► students : Object

► students : Object



# Usando Aggregation no Compass

▼ Stage 4 \$sort 

```
1  /**
2   * Provide any number of field/order pair
3   */
4  {
5    'students.number' : -1
6  }
```

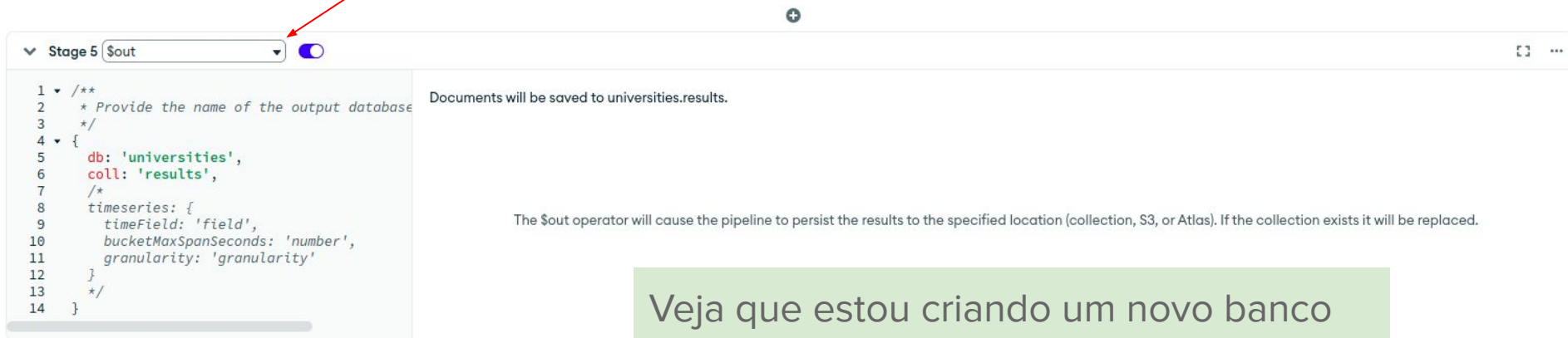
Output after \$sort stage (Sample of 4 documents)

► students : Object

► students : Object

► students : Object

# Usando Aggregation no Compass



The screenshot shows the MongoDB Compass interface. At the top, a dropdown menu is set to "Stage 5 \$out" with a toggle switch to its right. A red arrow points from the title "Usando Aggregation no Compass" to this dropdown. Below the dropdown, a code editor displays an aggregation pipeline. The pipeline starts with a comment and then defines a \$out stage. The \$out stage is configured with 'db: 'universities'', 'coll: 'results'', and a 'timeseries' object containing 'timeField: 'field'', 'bucketMaxSpanSeconds: 'number'', and 'granularity: 'granularity''. To the right of the code editor, a message states: "Documents will be saved to universities.results." Below this, a note explains: "The \$out operator will cause the pipeline to persist the results to the specified location (collection, S3, or Atlas). If the collection exists it will be replaced."

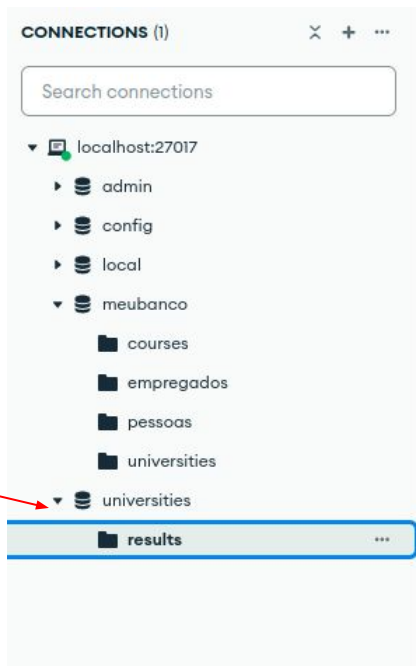
```
1  ▾ /**
2    * Provide the name of the output database
3    */
4  ▾ {
5    db: 'universities',
6    coll: 'results',
7    /*
8    timeseries: {
9      timeField: 'field',
10     bucketMaxSpanSeconds: 'number',
11     granularity: 'granularity'
12   }
13   */
14 }
```

Documents will be saved to universities.results.

The \$out operator will cause the pipeline to persist the results to the specified location (collection, S3, or Atlas). If the collection exists it will be replaced.

Veja que estou criando um novo banco chamado **universisites** e uma nova collection chamada **results**

# Usando Aggregation no Compass



Documents 4 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

[ADD DATA](#) [EXPORT DATA](#) [UPDATE](#) [DELETE](#)

`_id: ObjectId('67348be3df05ccd653237a1e')`  
▶ `students: Object`

`_id: ObjectId('67348be3df05ccd653237a1f')`  
▶ `students: Object`

`_id: ObjectId('67348be3df05ccd653237a20')`  
▶ `students: Object`

`_id: ObjectId('67348be3df05ccd653237a21')`  
▶ `students: Object`