MongoDB

Aggregation

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande

Agenda

MongoDB: Aggregation

O *aggregation framework* do MongoDB permite que você processe coleções de documentos de dados e retorne resultados computados. Você pode usar o *aggregation* para:

- **Agrupar dados:** Somar vendas por região, contar usuários ativos, etc.
- **Filtrar dados:** Selecionar apenas os documentos que atendem a certos critérios.
- **Transformar dados:** Renomear campos, calcular novos valores, criar *arrays*, etc.
- **Unir dados:** Combinar documentos de diferentes coleções (como um JOIN em SQL).
- **Realizar cálculos complexos:** Calcular médias, desvios padrão, etc.

Como funciona o *pipeline* de *aggregation*?

- O *aggregation framework* usa um conceito de *pipeline*, que é uma sequência de estágios.
- Cada estágio transforma os documentos à medida que eles passam pelo *pipeline*.
  - A saída de um estágio se torna a entrada para o próximo estágio.

Como funciona o *pipeline* de *aggregation*?

Alguns dos estágios mais comuns incluem:

- **match:** Filtra os documentos para passar apenas aqueles que correspondem aos critérios especificados para o próximo estágio.
  - É como a cláusula WHERE em SQL
- **group:** Agrupa os documentos de entrada por um campo especificado e aplica uma função de agregação para cada grupo (por exemplo, *sum*, *avg*, *min*, *max*, *count*).
  - É semelhante à cláusula GROUP BY em SQL, juntamente com funções agregadas.

Como funciona o *pipeline* de *aggregation*?

- **project:** Reformata os documentos. Você pode adicionar novos campos, remover campos existentes ou redefinir os valores dos campos.
  - É como a cláusula SELECT em SQL, permitindo que você escolha e transforme os campos.
- **sort:** Reordena os documentos de entrada por um ou mais campos especificados.
  - É como a cláusula ORDER BY em SQL.
- **limit:** Restringe o número de documentos que passam para o próximo estágio.
  - É como a cláusula LIMIT em SQL.

Como funciona o *pipeline* de *aggregation*?

- **unwind:** Desconstrói um campo de array para criar documentos separados para cada elemento do array.
- **lookup:** Realiza uma junção com outra coleção no mesmo banco de dados para combinar documentos. É semelhante ao JOIN em SQL.
- **out:** Escreve os resultados do pipeline para uma coleção especificada.

Exemplo

Imagine uma coleção chamada pedidos com documentos como este:

JSON

```
{ "_id": 1, "cliente": "Maria", "produto": "Camiseta", "preco": 25.00 }
{ "_id": 2, "cliente": "João", "produto": "Calça", "preco": 75.00 }
{ "_id": 3, "cliente": "Maria", "produto": "Tênis", "preco": 120.00 }
{ "_id": 4, "cliente": "Pedro", "produto": "Camiseta", "preco": 25.00 }
```

Observação

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**.

Como funciona

Se quisermos saber o total gasto por cada cliente, podemos usar o seguinte pipeline de *aggregation*:

JavaScript

```
db.pedidos.aggregate([
  { $group: { _id: "$cliente", totalGasto: { $sum: "$preco" } } }
])
```

1. **group:** Agrupamos os documentos pelo campo cliente
2. Para cada grupo (cada cliente), calculamos a soma (**sum**) dos valores do campo *preco* e armazenamos o resultado em um novo campo chamado *totalGasto*.

Como funciona

O resultado dessa operação seria algo como:

JSON

```
{ "_id": "Maria", "totalGasto": 145.00 }
{ "_id": "João", "totalGasto": 75.00 }
{ "_id": "Pedro", "totalGasto": 25.00 }
```

Como funciona

```
db.orders.insertMany( [
   { _id: 0, name: "Pepperoni", size: "small", price: 19,
     quantity: 10, date: ISODate( "2021-03-13T08:14:30Z" ) },
   { _id: 1, name: "Pepperoni", size: "medium", price: 20,
     quantity: 20, date : ISODate( "2021-03-13T09:13:24Z" ) },
   { _id: 2, name: "Pepperoni", size: "large", price: 21,
     quantity: 30, date : ISODate( "2021-03-17T09:22:12Z" ) },
   { _id: 3, name: "Cheese", size: "small", price: 12,
     quantity: 15, date : ISODate( "2021-03-13T11:21:39.736Z" ) },
   { _id: 4, name: "Cheese", size: "medium", price: 13,
     quantity:50, date : ISODate( "2022-01-12T21:23:13.331Z" ) },
   { _id: 5, name: "Cheese", size: "large", price: 14,
     quantity: 10, date : ISODate( "2022-01-12T05:08:13Z" ) },
   { _id: 6, name: "Vegan", size: "small", price: 17,
     quantity: 10, date : ISODate( "2021-01-13T05:08:13Z" ) },
   { _id: 7, name: "Vegan", size: "medium", price: 18,
     quantity: 10, date : ISODate( "2021-01-13T05:10:13Z" ) }
] )
```

Como funciona

```
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
   {
      $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }
   }

] )
```

Como funciona

```
db.orders.aggregate( [

   // Stage 1: Filter pizza order documents by date range
   {
      $match:
      {
         "date": { $gte: new ISODate( "2020-01-30" ), $lt: new ISODate( "2022-01
      }
   },

   // Stage 2: Group remaining documents by date and calculate results
   {
      $group:
      {
         _id: { $dateToString: { format: "%Y-%m-%d", date: "$date" } },
         totalOrderValue: { $sum: { $multiply: [ "$price", "$quantity" ] } },
         averageOrderQuantity: { $avg: "$quantity" }
      }
   },

   // Stage 3: Sort documents by totalOrderValue in descending order
   {
      $sort: { totalOrderValue: -1 }
   }
```

MongoDB

Aggregation

Prof. Igor Avila Pereira
igor.pereira@riogrande.ifrs.edu.br

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul (IFRS)
Câmpus Rio Grande