

Bancos de Dados Não-Relacionais

CET 0620 - Módulo 03e

Professor Franklin Amorim

MongoDB

Aggregation Framework

Aggregation Framework

Aggregation Framework

O Aggregation Framework é um recurso poderoso do MongoDB que permite aos usuários realizar operações de processamento e transformação de dados em documentos de uma coleção.

Ele fornece uma maneira flexível e expressiva de analisar dados, realizar cálculos e gerar resultados agregados.

A estrutura de agregação foi projetada para lidar com grandes volumes de dados de forma eficiente e é particularmente útil para tarefas complexas de processamento de dados.

Aggregation Framework

Processamento baseado em pipeline: as operações de agregação no MongoDB são construídas usando um pipeline de estágios. Cada estágio executa uma operação específica nos documentos de entrada e passa os resultados para o próximo estágio do pipeline.

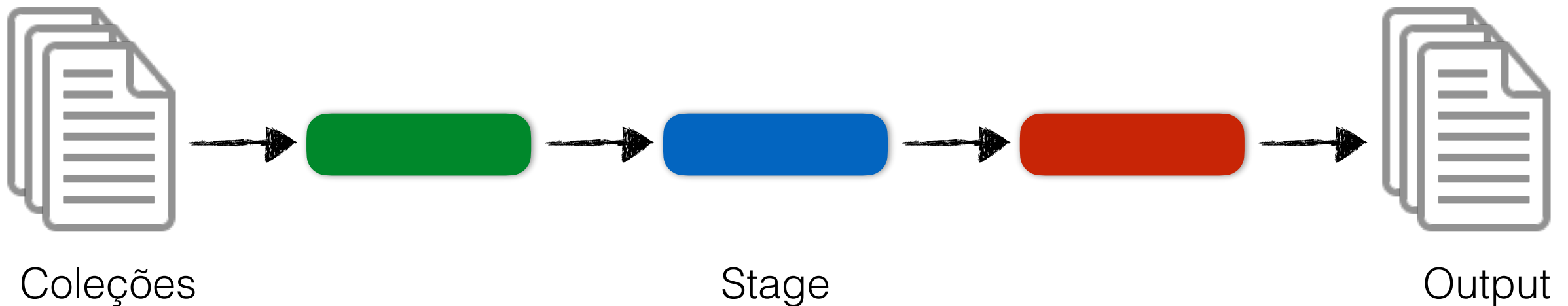
Operadores de estágio: o MongoDB fornece um rico conjunto de operadores de estágio que podem ser usados em pipelines de agregação. Esses operadores realizam diversas tarefas, como filtrar, agrupar, classificar, projetar, unir e realizar cálculos matemáticos nos dados.

Transformação de dados: A estrutura de agregação oferece suporte a operações de transformação de dados, permitindo aos usuários remodelar a estrutura de documentos, adicionar ou remover campos e realizar manipulações complexas de dados como parte do pipeline de agregação.

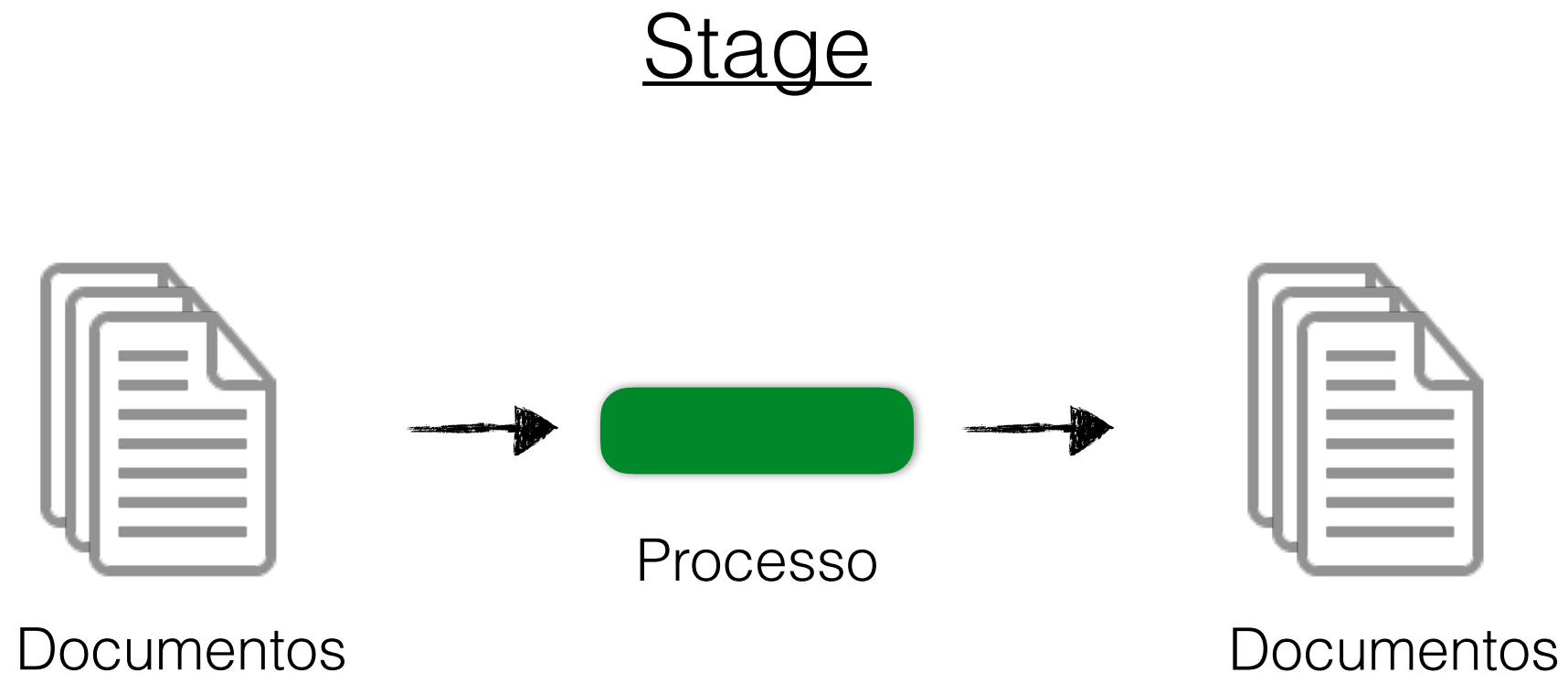
Otimização do pipeline de agregação: a estrutura de agregação do MongoDB foi projetada para ser executada com eficiência, aproveitando índices e outras otimizações para processar dados o mais rápido possível. Os usuários podem otimizar pipelines de agregação usando técnicas como uso de índice, projeção e filtragem para melhorar o desempenho.

Aggregation Framework

Pipeline



Aggregation Framework



Aggregation Framework

```
db.<collections>.aggregate([  
    {<stage>},  
    ...  
    {<stage>}  
])
```



Aggregation Framework

<stage>

{ \$operator: {<expression>} },



Documents



Data Processor



Documents

Agregação Simples

```
db.movies.aggregate([  
  {$match: {languages: "Portuguese"}}  
])
```

operator

expression

stage

Uma agregação com um único estágio: \$match

Dois Estágios

```
db.movies.aggregate([  
  {$match: {languages: "Portuguese"}},  
  {$project: {languages: 1, title: 1}}  
])
```

Agregação com dois estágios: \$match and \$project

\$match

\$match

```
db.collection.aggregate([  
  {$match: {...}}  
])
```

O operador \$match filtra documentos com base nas condições especificadas

\$match

```
db.movies.aggregate([  
  {$match: {languages: "Portuguese"}}  
])
```

```
SELECT *  
FROM movies  
WHERE "Portuguese" IN languages;
```

Equivalente em SQL

\$match com uma condição

\$match

```
db.movies.aggregate([  
  {$match: {languages: "Portuguese",  
            countries: "USA"}}  
])
```

```
SELECT *  
FROM movies  
WHERE "Portuguese" IN languages  
      AND "USA" IN countries;
```

Equivalente em SQL

\$match com múltiplas condições usando o AND

\$match

```
db.movies.aggregate([  
  {$match: {year: {$gte: 1960, $lt: 1970 } } }  
])
```

```
SELECT *  
FROM movies  
WHERE year >= 1960  
      AND year < 1970;
```

Equivalente em SQL

\$match com múltiplas condições usando o AND implícito

\$match

```
db.movies.aggregate([
  {$match: {$or: [
    { year: {$lt: 1931 }},
    { released: {$lt: ISODate("1931-01-01T00:00:00.000Z")} } ] }
  ]})
```

```
SELECT *
FROM movies
WHERE year <= 1931
      OR released < "1931-01-01T00:00:00.000Z";
```

Equivalente em SQL

Um exemplo com \$or

\$project

\$project

```
db.collection.aggregate([  
  {$project: {...}}  
])
```

\$project define os campos a serem passados para a próxima etapa.
Ele pode adicionar e remover campos de documentos, bem como criar novos campos

\$project

```
db.movies.aggregate([  
    {$project: {languages: 1, title: 1}}  
])
```

```
SELECT _id, languages, title  
FROM movies
```

Equivalente em SQL

Neste exemplo os campos "languages" e "title" serão os únicos atributos passados para a próxima etapa (com exceção de "_id")

\$project

```
db.movies.aggregate([  
  {$project: {_id: 0, languages: 1, title: 1}}  
])
```

```
SELECT languages, title  
FROM movies
```

Equivalente em SQL

Para remover o campo "_id" é necessário defini-lo explicitamente como 0.

\$project

```
db.movies.aggregate([  
  {$project: {title: 1, "awards.wins": 1}}  
])
```

```
SELECT _id, title, "awards.wins"  
FROM movies
```

Equivalente em SQL

Para referenciar atributos em subdocumentos (dotted fields) precisamos usar aspas (").

\$project

```
db.movies.aggregate([  
  {$project: {name: "$title", "awards.wins": 1}}  
])
```

```
SELECT _id, title as "name", "awards.wins"  
FROM movies
```

Equivalente em SQL

Também podemos renomear campos usando "field paths".

\$project

```
db.movies.aggregate([
  {$project: {
    name: "$title",
    wins: "$awards.wins"
  }}
])
```

```
SELECT _id, title as "name", "awards.wins" as "wins"
FROM movies
```

Equivalente em SQL

"field path" simples e com um atributo de um subdocumento.

\$project

```
db.movies.aggregate([
  {$project: {
    name: "$title",
    wins: "$awards.wins",
    size_cast: {$size: "$cast"}
  }}
])
```

```
SELECT _id, title as "name", "awards.wins" as "wins",
       len(cast) as size_cast
FROM movies
```

Equivalente em SQL

Criando um novo campo com uma expressão.

\$sort

\$sort

```
db.movies.aggregate([  
  {$project: {title: 1, year: 1, "awards.wins": 1}},  
  {$sort: {year: 1}}  
])
```

```
SELECT _id, title, year, "awards.wins"  
FROM movies  
ORDER BY year ASC
```

Equivalente em SQL

Ordena os documentos com base na lista de atributos especificada.

\$sort

```
db.movies.aggregate([  
  {$project: {title: 1, year: 1, "awards.wins": 1}},  
  {$sort: {year: 1, "awards.wins": -1}}  
])
```

```
SELECT _id, title, year, "awards.wins"  
FROM movies  
ORDER BY year ASC, "awards.wins" DESC
```

Equivalente em SQL

Use 1 para ordem crescente e -1 para ordem decrescente.

\$sort

```
db.movies.aggregate([  
  {$project: {title: 1, year: 1, "awards.wins": 1}},  
  {$sort: {year: 1, "awards.wins": -1}}  
])
```

Se o estágio de classificação não for capaz de usar um índice, ele executará uma classificação na memória. No entanto, o MongoDB limita esse uso de memória a 100 MB.

\$sort

```
db.movies.aggregate([  
  {$project: {title: 1, year: 1, "awards.wins": 1}},  
  {$sort: {year: 1, "awards.wins": -1}}  
, { allowDiskUse: true }])
```

Para executar uma classificação maior que 100M, use a opção allowDiskUse para ativar os estágios do pipeline de agregação para gravar dados em arquivos temporários.

\$sort

```
db.movies.aggregate([  
  {$project: {title: 1, year: 1, "awards.wins": 1}},  
  {$sort: {year: 1, "awards.wins": -1}}  
], {allowDiskUse: true})
```

A partir do MongoDB 6.0, os estágios do pipeline que exigem mais de 100M de memória para execução gravam arquivos temporários no disco por padrão.

\$limit e \$skip

\$limit

```
db.movies.aggregate([
  {$project: {title: 1, year: 1, "awards.wins": 1}},
  {$sort: {year: 1, "awards.wins": -1}},
  {$limit: 5}
])
```

```
SELECT _id, title, year, "awards.wins"
FROM movies
ORDER BY year ASC, "awards.wins" DESC
LIMIT 5
```

Equivalente em SQL

Limita o número de documentos passados para a próxima etapa.

\$skip

```
db.movies.aggregate([  
  {$project: {title: 1, year: 1, "awards.wins": 1}},  
  {$sort: {year: 1, "awards.wins": -1}},  
  {$skip: 100},  
  {$limit: 5}  
])
```

```
SELECT _id, title, year, "awards.wins"  
FROM movies  
ORDER BY year ASC, "awards.wins" DESC  
LIMIT 5 OFFSET 100
```

Equivalente em SQL

Pula o número de documentos que serão passados para a próxima etapa.

\$count

\$count

```
db.movies.aggregate([  
  {$match: {year: {$gte: 2000}}},  
  {$count: "total_2000"}  
])
```

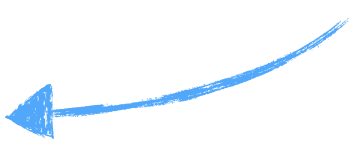
```
SELECT count(*) AS "total_2000"  
FROM movies  
WHERE year >= 2000
```

Equivalente em SQL

\$count retornar o número de documentos

\$group

\$group

```
db.collection.aggregate([  
  {$group: {  
    _id: <expression>   
  }}  
)
```

Agrupar documentos pela expressão "_id" especificada e para cada agrupamento distinto gera um documento.

\$group

```
db.movies.aggregate([  
  {$group: {  
    _id: "$year"  
  }}  
])
```

```
SELECT year AS "_id"  
FROM movies  
GROUP BY year
```

Equivalente em SQL

Vamos começar com um exemplo simples.
Neste caso vamos agrupar "year", criando um documento por ano distinto.

\$group

```
db.collection.aggregate([
  {$group: {
    _id: <expression>,
    field1: {<accumulator1> : <expression1>},
    ...
  }}
])
```

Computed field

accumulator expression

Os documentos gerados também podem conter campos computados por accumulator expressions (expressões de acumulador).

\$group

```
db.movies.aggregate([
  {$group: {
    _id: "$year",
    number_movies: {$sum : 1}
  }}
])
```

```
SELECT year AS "_id", count(*) AS "number_movies"
FROM movies
GROUP BY year
```

Equivalente em SQL

Neste exemplo usamos \$sum para contar os documentos por _id (neste caso "year")

\$group

```
db.movies.aggregate([
  {$group: {
    _id: "$year",
    number_movies: {$sum : 1}
  }},
  {$sort: {number_movies: -1}}
])
```

```
SELECT year AS "_id", count(*) AS "number_movies"
FROM movies
GROUP BY year
ORDER BY "number_movies" DESC
```

Equivalente em SQL

E agora podemos usar esses campos calculados na próxima etapa.
Aqui usamos para ordenar por número de filmes por ano

\$group

```
db.movies.aggregate([
  {$group: {
    _id: {year: "$year", type: "$type"},
    number_movies: {$sum : 1}
  }},
  {$sort: {number_movies: -1}}
])
```

```
SELECT year, type, count(*) AS "number_movies"
FROM movies
GROUP BY year, type
ORDER BY "number_movies" DESC
```

Equivalente em SQL

Também podemos agrupar por vários atributos.

\$group

```
db.movies.aggregate([
  {$group: {
    _id: "$rated",
    avg_rating: {$avg: "$imdb.rating"}
  }},
  {$sort: {avg_rating: -1}}
])
```

```
SELECT rated AS "_id", avg(imdb.rating) AS "avg_rating"
FROM movies
GROUP BY rated
ORDER BY "avg_rating" DESC
```

Equivalente em SQL

Aqui um exemplo com \$avg (média).

\$group

```
db.movies.aggregate([  
  {$group: {  
    _id: "$year",  
    avg_directors: {$avg: {$size: "$directors"}}  
  }}  
])
```

```
SELECT year AS "_id", avg(len(directors)) AS "avg_directors"  
FROM movies  
GROUP BY year
```

Equivalente em SQL

Importante: Se o campo utilizado na expressão do acumulador não estiver presente em todos os documentos um erro será gerado.

\$group

```
db.movies.aggregate([
  {$match: {directors: { $exists: true}}},
  {$group: {
    _id: "$year",
    avg_directors: {$avg: {$size: "$directors"}}
  }}
])
```

```
SELECT year AS "_id", avg(len(directors)) AS "avg_directors"
FROM movies
WHERE directors IS NOT NULL
GROUP BY year
```

Equivalente em SQL

Uma maneira de evitar erros é filtrar os documentos

\$unwind

\$unwind

```
db.collection.aggregate([  
  {$unwind: <field path> }  
])
```

Desconstrói documentos com array para gerar um documento para cada elemento do array.

\$unwind

```
db.movies.aggregate([  
  { $unwind: "$countries" }  
])
```

Neste exemplo, para cada país em cada documento, \$unwind irá gerar um novo documento

\$unwind

```
db.movies.aggregate([
  {$unwind: "$countries" },
  {$group: {
    _id: "$countries",
    Total: {$sum: 1}
  }}
])
```

Usando \$unwind podemos calcular, por exemplo, o número de filmes por país

\$lookup

\$lookup

```
db.collection.aggregate([
  {$lookup: {
    from: <collection to join>,
    localField: <field>,
    foreignField: <field>,
    as: <output array field>
  }}
])
```

Para realizar uma correspondência de igualdade entre um campo dos documentos de entrada com um campo dos documentos de outra coleção

\$lookup

```
db.orders.drop()
```

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

```
db.inventory.drop()
```

```
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 }  
)
```

Vamos criar duas novas coleções

\$lookup

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

E ejecutar o \$lookup