Aggregation pipeline

Método aggregate()

MongoDB provides the db.collection.aggregate() method in the mongo shell and the aggregate command to run the aggregation pipeline.

Sintaxis

db.collection.aggregate(pipeline, options)

options

{ explain: true }   Devuelve explain sobre el pipeline

{ allowDiskUse }   Evita el error por utilizar en memoria mas de 100 MB al escribir las etapas en archivos temporales.

{ cursor : { batchSize: number } } Establece el resultado en un cursor con un tamaño determinado para superar el límite de 16 MB por documento devuelto.

{ bypassDocumentValidation: <boolean> }   Solo para $out, para evitar requerimeintos de validación en la nueva colección.

{ readConcern: <document> }   Para modificar readConcern.

Stages Pipeline

The MongoDB aggregation pipeline consists of stages. Each stage transforms the documents as they pass through the pipeline. Pipeline stages do not need to produce one output document for every input document; e.g., some stages may generate new documents or filter out documents.

Pipeline stages can appear multiple times in the pipeline with the exception of $out, $merge, and $geoNear stages.

Sintaxis

{ <stage1> }, {<stage2>}...

$project

Passes along the documents with the requested fields to the next stage in the pipeline. The specified fields can be existing fields from the input documents or newly computed fields.

The $project stage has the following prototype form:

{ $project: { <specification(s)> } }
The $project takes a document that can specify the inclusion of fields, the suppression of the _id field, the addition of new fields, and the resetting of the values of existing fields. Alternatively, you may specify the exclusion of fields.

The $project specifications have the following forms:

Form	Description
<field>: <1 or true>	Specifies the inclusion of a field.
_id: <0 or false>	Specifies the suppression of the _id field.

To exclude a field conditionally, use the REMOVE variable instead. For details, see Exclude Fields Conditionally.

<field>: <expression>
Adds a new field or resets the value of an existing field.

Changed in version 3.6: MongoDB 3.6 adds the variable REMOVE. If the the expression evaluates to $$REMOVE, the field is excluded in the output. For details, see Exclude Fields Conditionally.

<field>:<0 or false>
Specifies the exclusion of a field.

To exclude a field conditionally, use the REMOVE variable instead. For details, see Exclude Fields Conditionally.

If you specify the exclusion of a field other than _id, you cannot employ any other $project specification forms. This restriction does not apply to conditionally exclusion of a field using the REMOVE variable.

See also the $unset stage to exclude fields.

Práctica

```
> db.titulos.aggregate([{$project:{_id: 0, titulo: 1, categorias: 1} }])
{ "titulo" : "El coronel no tiene quien le escriba", "categorias" : [ "amor", "novela",
"ficción" ] }
```

```
{ "titulo" : "El amor en los tiempos del cólera", "categorias" : [ "drama", "novela",
"ficción" ] }
> db.titulos.aggregate([{$project:{_id: 0, titulo: 1, categorias: 1} },{$project: {titulo: 1}}])
{ "titulo" : "El coronel no tiene quien le escriba" }
{ "titulo" : "El amor en los tiempos del cólera" }
```

Field reference

```
> db.titulos.aggregate([{$project:{_id: 0, title: "$titulo"} }])
{ "title" : "El coronel no tiene quien le escriba" }
{ "title" : "El amor en los tiempos del cólera" }
```

$sort

Sorts all input documents and returns them to the pipeline in sorted order.

The $sort stage has the following prototype form:

```
{ $sort: { <field1>: <sort order>, <field2>: <sort order> ... } }
```

$sort takes a document that specifies the field(s) to sort by and the respective sort order. <sort order> can have one of the following values:

Value	Description
1	Sort ascending.
-1	Sort descending.
{ $meta: "textScore" }	Sort by the computed textScore metadata in descending order. See Metadata Sort for an example.

Práctica

```
{ "_id" : 2, "nombre" : "Pedro", "alta" : ISODate("2017-11-18T00:00:00Z"), "actividades" :
[ "padel", "tenis", "esgrima" ] }
{ "_id" : 3, "nombre" : "Luis", "alta" : ISODate("2017-10-02T00:00:00Z"), "actividades" : [
"aquagym", "tenis", "step" ] }
{ "_id" : 4, "nombre" : "Carlos", "alta" : ISODate("2017-09-14T00:00:00Z"),
"actividades" : [ "aquagym", "padel", "cardio" ] }
{ "_id" : 5, "nombre" : "José", "alta" : ISODate("2017-11-15T00:00:00Z"), "actividades" : [
"pesas", "cardio", "step" ] }
```

```
> db.clientes.aggregate([
... {$project: {cliente: {$toUpper: "$nombre"}, _id: 0}},
... {$sort: {cliente: 1}}
... ])
{ "cliente" : "CARLOS" }
{ "cliente" : "JOSé" }
{ "cliente" : "LUIS" }
{ "cliente" : "PEDRO" }
```

```
> db.clientes.aggregate([ {$project: {mesAlta: {$month: "$alta"}, cliente: "$nombre",
_id: 0}}, {$sort: {mesAlta: -1}} ])
{ "mesAlta" : 11, "cliente" : "Pedro" }
{ "mesAlta" : 11, "cliente" : "José" }
{ "mesAlta" : 10, "cliente" : "Luis" }
{ "mesAlta" : 9, "cliente" : "Carlos" }
```

$group

Groups input documents by the specified _id expression and for each distinct grouping, outputs a document. The _id field of each output document contains the unique group by value. The output documents can also contain computed fields that hold the values of some accumulator expression.

NOTE $group does not order its output documents.

The $group stage has the following prototype form:

```
{
  $group:
    {
      _id: <expression>, // Group By Expression
      <field1>: { <accumulator1> : <expression1> },
      ...
    }
 }
```

```
> db.clientes.aggregate([
... {$project: {mesAlta: {$month: "$alta"}}},
... {$group: {_id: "$mesAlta", numeroAltas: {$sum: 1}}},
... {$sort: {_id: 1}}
... ])
{ "_id" : 1, "numeroAltas" : 5 }
{ "_id" : 9, "numeroAltas" : 1 }
{ "_id" : 10, "numeroAltas" : 1 }
{ "_id" : 11, "numeroAltas" : 2 }
```

Si no queremos el campo _id:

```
> db.clientes.aggregate([
{$project: {mesAlta: {$month: "$alta"}}},
{$group: {_id: "$mesAlta", numeroAltas: {$sum: 1}}},
{$project: {mes: "$_id", numeroAltas: "$numeroAltas", _id: 0}} ])

{ "mes" : 1, "numeroAltas" : 5 }
{ "mes" : 9, "numeroAltas" : 1 }
{ "mes" : 10, "numeroAltas" : 1 }
{ "mes" : 11, "numeroAltas" : 2 }
```

Comportamiento

$group Operator and Memory

The $group stage has a limit of 100 megabytes of RAM. By default, if the stage exceeds this limit, $group returns an error. To allow for the handling of large datasets, set the allowDiskUse option to true. This flag enables $group operations to write to temporary files. For more information, see the db.collection.aggregate() method and the aggregate command.

```
> use shop
switched to db shop
> db.pedidos.insert({sku: "v101", cantidad: 12, precio: 20, fecha:
ISODate("2020-01-19")})
WriteResult({ "nInserted" : 1 })
> db.pedidos.insert({sku: "v101", cantidad: 6, precio: 20, fecha:
ISODate("2020-01-11")})
WriteResult({ "nInserted" : 1 })
> db.pedidos.insert({sku: "v101", cantidad: 4, precio: 20, fecha:
ISODate("2020-01-21")})
WriteResult({ "nInserted" : 1 })
> db.pedidos.insert({sku: "v102", cantidad: 7, precio: 10.3, fecha:
ISODate("2020-01-21")})
WriteResult({ "nInserted" : 1 })
> db.pedidos.insert({sku: "v102", cantidad: 5, precio: 10.9, fecha:
ISODate("2020-01-21")})
WriteResult({ "nInserted" : 1 })
```

Ejemplo de total ventas por día de la semana

```
> db.pedidos.aggregate([
... {$group:
... {
... _id: {$dayOfWeek: "$fecha"},
...  totalVentas: {$sum: {$multiply: ["$cantidad","$precio"]}}
... }
... },
... {$sort: {totalVentas: -1}}
... ])
```

```
{ "_id" : 1, "totalVentas" : 240 }
{ "_id" : 3, "totalVentas" : 206.60000000000002 }
{ "_id" : 7, "totalVentas" : 120 }
```

Ejemplo de cantidad media de pedido por producto

```
> db.pedidos.aggregate([
{$group: {
            _id: "$sku",
           cantidadPromedio: { $avg: "$cantidad" }
}}, {$sort: {cantidadPromedio: -1}} ])

{ "_id" : "v101", "cantidadPromedio" : 7.333333333333333 }
{ "_id" : "v102", "cantidadPromedio" : 6 }
```

Ejemplo de creación de array

```
> db.titulos.insert({titulo: "El amor en los tiempos del cólera", autor: "Gabriel García Márquez"})
WriteResult({ "nInserted" : 1 })
> db.titulos.insert({titulo: "El Quijote", autor: "Miguel de Cervantes"})
WriteResult({ "nInserted" : 1 })
> db.titulos.insert({titulo: "Entremeses", autor: "Miguel de Cervantes"})
WriteResult({ "nInserted" : 1 })

> db.titulos.aggregate([ {$group: { _id: "$autor", libros: {$push: "$titulo"} }} ])
{ "_id" : "Miguel de Cervantes", "libros" : [ "El Quijote", "Entremeses" ] }
{ "_id" : "Gabriel García Márquez", "libros" : [ "El coronel no tiene quién le escriba",
"El amor en los tiempos del cólera" ] }
{ "_id" : null, "libros" : [ "El coronel no tiene quien le escriba", "El amor en los
tiempos del cólera" ] } // si no tienen el campo de agrupación crea sobre null
```

Con varios campos:

```
> db.participantes.aggregate([
   {$group: {
        _id: {nombre: "$nombre", edad: "$edad"}, total: {$sum: 1}
         }
    }
])
{ "_id" : { "nombre" : "Laura", "edad" : 10 }, "total" : 275 }
{ "_id" : { "nombre" : "María", "edad" : 75 }, "total" : 224 }
{ "_id" : { "nombre" : "Laura", "edad" : 75 }, "total" : 258 }
{ "_id" : { "nombre" : "Juan", "edad" : 8 }, "total" : 252 }
```

Pasarlos a un array:

```
> db.participantes.aggregate([ {$group: {_id: {nombre: "$nombre", edad: "$edad"},
docs: {$push: "$$ROOT"}}}, {$limit: 1} ])

{ "_id" : { "nombre" : "Laura", "edad" : 86 }, "docs" : [
        { "_id" : 172, "nombre" : "Laura", "apellido1" : "González", "apellido2" : "Fernández",
          "edad" : 86, "dni" : "61654363X" },
        { "_id" : ...
```

$unwind

Deconstructs an array field from the input documents to output a document for each element. Each output document is the input document with the value of the array field replaced by the element.

Syntax

```
{
  $unwind:
    {
      path: <field path>,
      includeArrayIndex: <string>,
      preserveNullAndEmptyArrays: <boolean>
    }
}
```

Práctica

```
> db.productos.insert({nombre: "Camiseta Nike", tallas: ["xs","s","m","l","xl"]})
WriteResult({ "nInserted" : 1 })
> db.productos.insert({nombre: "Camiseta Puma", tallas: null})
WriteResult({ "nInserted" : 1 })
> db.productos.insert({nombre: "Camiseta Adidas"})
WriteResult({ "nInserted" : 1 })
> db.productos.aggregate([
... {$unwind: "$tallas"}
... ])
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "xs" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "s" }
```

```
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "m" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "l" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "xl" }
> db.productos.aggregate([ {$unwind: {path:"$tallas"}} ])
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "xs" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "s" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "m" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "l" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "xl" }
> db.productos.aggregate([ {$unwind: {path:"$tallas", includeArrayIndex: "orden"}}
])
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "xs", "orden" : NumberLong(0) }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "s", "orden" : NumberLong(1) }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "m", "orden" : NumberLong(2) }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "l", "orden" : NumberLong(3) }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "xl", "orden" : NumberLong(4) }
> db.productos.aggregate([ {$unwind: {path:"$tallas",
preserveNullAndEmptyArrays: true }} ])
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "xs" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "s" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "m" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "l" }
{ "_id" : ObjectId("5e2f33f10a4428bb0b3d74d8"), "nombre" : "Camiseta Nike",
"tallas" : "xl" }
{ "_id" : ObjectId("5e2f33f90a4428bb0b3d74d9"), "nombre" : "Camiseta Puma",
"tallas" : null }
{ "_id" : ObjectId("5e2f34080a4428bb0b3d74da"), "nombre" : "Camiseta Adidas" }
```

```
> db.clientes.aggregate([
... {$unwind: "$actividades"},
... {$group: {
... _id: "$actividades", total: {$sum: 1}
... }},
... {$sort: {total: -1, _id: 1}}
... ])

{ "_id" : "cardio", "total" : 2 }
{ "_id" : "aquagym", "total" : 2 }
{ "_id" : "padel", "total" : 2 }
{ "_id" : "step", "total" : 2 }
{ "_id" : "tenis", "total" : 2 }
{ "_id" : "pesas", "total" : 1 }
{ "_id" : "esgrima", "total" : 1 }
```

$match

Filters the documents to pass only the documents that match the specified condition(s) to the next pipeline stage.

The $match stage has the following prototype form:

```
{ $match: { <query> } }
```

Práctica

```
> db.participantes.aggregate([
... {$match: {edad: {$gte: 40}, edad: {$lt: 50}}},
... {$group: {_id: "$nombre", totales: {$sum: 1}}},
... {$sort: {totales: -1}}
... ])
{ "_id" : "Juan", "totales" : 100481 }
{ "_id" : "Lucía", "totales" : 100311 }
{ "_id" : "Carlos", "totales" : 100243 }
{ "_id" : "Pedro", "totales" : 100100 }
{ "_id" : "María", "totales" : 99534 }
```

```
> db.participantes.explain("executionStats").aggregate([...
```

Ejemplo con texto

```
> db.reviews.insert({sku: "v102", user: "00012", review: "buen servicio pero producto en mal estado"})
WriteResult({ "nInserted" : 1 })
> db.reviews.insert({sku: "v102", user: "00013", review: "muy satisfecho con la compra"})
WriteResult({ "nInserted" : 1 })
> db.reviews.insert({sku: "v102", user: "00014", review: "muy mal, tuve que devolverlo"})
WriteResult({ "nInserted" : 1 })
> db.reviews.insert({sku: "v103", user: "00014", review: "perfecto en todos los sentidos"})
WriteResult({ "nInserted" : 1 })
> db.reviews.insert({sku: "v102", user: "00015", review: "mal, no volveré a comprar"})
WriteResult({ "nInserted" : 1 })
> db.reviews.createIndex({review: "text"})
{
	"createdCollectionAutomatically" : false,
	"numIndexesBefore" : 1,
	"numIndexesAfter" : 2,
	"ok" : 1
}

> db.reviews.aggregate([ { $match: {$text: {$search: "mal"}}}, { $group: {_id: "$sku", badReviews: {$sum: 1}}} ])
{ "_id" : "v102", "badReviews" : 3 }
{ "_id" : "v105", "badReviews" : 1 }

> db.reviews.aggregate([ { $match: {$text: {$search: "buen perfecto"}}}, { $group: {_id: "$sku", goodReviews: {$sum: 1}}} ])
{ "_id" : "v103", "goodReviews" : 1 }
{ "_id" : "v102", "goodReviews" : 1 }
```

$addFields

Adds new fields to documents. $addFields outputs documents that contain all existing fields from the input documents and newly added fields.

The $addFields stage is equivalent to a $project stage that explicitly specifies all existing fields in the input documents and adds the new fields.

$addFields has the following form:

```
{ $addFields: { <newField>: <expression>, ... } }
```

Práctica

```
> db.resultados.insert({nombre: "Juan", llegada: new Date("2020-01-30T16:31:40")})
WriteResult({ "nInserted" : 1 })
> db.resultados.insert({nombre: "Pedro", llegada: new
Date("2020-01-30T15:43:40")})
WriteResult({ "nInserted" : 1 })
> db.resultados.insert({nombre: "Pedro", llegada: new Date("2020-01-30T17:13:22")})
WriteResult({ "nInserted" : 1 })

> db.resultados.aggregate([
{$addFields: {tiempo: {$subtract: [ "$llegada", new Date("2020-01-30T12:00:00") ] }}}
,  {$addFields: {segundos: {$mod: [{$divide: ["$tiempo", 1000]}, 60]}}},
{$addFields: {minutos: {$floor: {$mod: [{$divide: ["$tiempo", 1000 * 60]}, 60]}}}},
{$addFields: {horas: {$floor: {$mod: [{$divide: ["$tiempo", 1000 * 60 * 60]}, 24]}}}},
{$project: {nombre: 1, horas: 1, minutos: 1, segundos: 1, _id: 0}}
])
```

$limit

Limits the number of documents passed to the next stage in the pipeline.

The $limit stage has the following prototype form:

```
{ $limit: <positive integer> }
```

$limit takes a positive integer that specifies the maximum number of documents to pass along.

```
> db.participantes.aggregate([
{$match: {edad: {$gte: 40}, edad: {$lt: 50}}},
{$group: {_id: "$nombre", totales: {$sum: 1}}},
{$sort: {totales: -1}}, {$limit: 3}
])
{ "_id" : "Juan", "totales" : 100481 }
{ "_id" : "Lucía", "totales" : 100311 }
{ "_id" : "Carlos", "totales" : 100243 }
```

$skip

Skips over the specified number of documents that pass into the stage and passes the remaining documents to the next stage in the pipeline.

The $skip stage has the following prototype form:

```
{ $skip: <positive integer> }
```

$skip takes a positive integer that specifies the maximum number of documents to skip.

$merge

Writes the results of the aggregation pipeline to a specified collection. The $merge operator must be the last stage in the pipeline.

The $merge stage:

- Can output to a collection in the same or different database.
- Creates a new collection if the output collection does not already exist.
- Can incorporate results (insert new documents, merge documents, replace documents, keep existing documents, fail the operation, process documents with a custom update pipeline) into an existing collection.
- Can output to a sharded collection. Input collection can also be sharded.
- For a comparison with the $out stage which also outputs the aggregation results to a collection, see Comparison with $out.

$merge has the following syntax:

```
{ $merge: {
    into: <collection> -or- { db: <db>, coll: <collection> },
    on: <identifier field> -or- [ <identifier field1>, ...],  // Optional
    let: <variables>,                                    // Optional
    whenMatched: <replace|keepExisting|merge|fail|pipeline>,  // Optional
    whenNotMatched: <insert|discard|fail>                // Optional
} }
```

Práctica

```
> db.participantes.aggregate([
        {$match: {edad: {$gte: 40}, edad: {$lt: 50}}},
        {$group: {_id: "$nombre", totales: {$sum: 1}}},
        {$sort: {totales: -1}},
        {$limit: 3},
        {$merge: "resumen"}
])
```

$lookup (aggregation)

Performs a left outer join to an unsharded collection in the same database to filter in documents from the "joined" collection for processing. To each input document, the $lookup stage adds a new array field whose elements are the matching documents from the “joined” collection. The $lookup stage passes these reshaped documents to the next stage.

Syntax
The $lookup stage has the following syntaxes:

Equality Match
To perform an equality match between a field from the input documents with a field from the documents of the “joined” collection, the $lookup stage has the following syntax:

```
{
  $lookup:
    {
      from: <collection to join>,
      localField: <field from the input documents>,
      foreignField: <field from the documents of the "from" collection>,
      as: <output array field>
    }
}
```

Práctica

```
db.pedidos.insert ( [
{ "_id" : 1, "item" : "a01", "precio" : 12, "cantidad" : 2 },
{ "_id" : 2, "item" : "j01", "precio" : 20, "cantidad" : 1 },
] )

db.inventario.insert ( [
{ "_id" : 1, "codigo" : "a01", descripcion: "product 1", "stock" : 120 },
{ "_id" : 2, "codigo" : "d01", descripcion: "product 2", "stock" : 80 },
{ "_id" : 3, "codigo" : "j01", descripcion: "product 3", "stock" : 60 },
{ "_id" : 4, "codigo" : "j02", descripcion: "product 4", "stock" : 70 },
] )

db.pedidos.aggregate([
   {
     $lookup:
       {
         from: "inventario",
         localField: "item",
         foreignField: "codigo",
         as: "inventarioItems"
       }
  }
])
```

```
{ "_id" : 1, "item" : "a01", "precio" : 12, "cantidad" : 2, "inventarioItems" : [ { "_id" : 1, "codigo"
: "a01", "descripcion" : "product 1", "stock" : 120 } ] }
{ "_id" : 2, "item" : "j01", "precio" : 20, "cantidad" : 1, "inventarioItems" : [ { "_id" : 3, "codigo"
: "j01", "descripcion" : "product 3", "stock" : 60 } ] }
```

En este ejemplo realizamos un $lookup sobre la colección pedidos. Esta etapa selecciona de la colección inventory el campo sku, cuyos valores coinciden con los del campo ítem de la colección orders, realizando un join con ítem y devolviendo cada documento de la colección orders con un nuevo campo inventory_docs que contiene un array con cada coincidencia de la colección inventory.