Base de Datos No SQL

Agregación

Las operaciones basadas en agregaciones nos permiten procesar la data que tenemos registrada para obtener resultados que aportan información útil.
Usualmente, estas operaciones las utilizamos para agrupar registros o contarlos.
Se comparan con los group by y count que usualmente utilizamos en bases de datos relacionales, aunque también incluyen otras operaciones como sort o limit, para ordenar y mostrar un límite de registros respectivamente.
Para poder realizar estas operaciones de agregación MongoDB nos da 3 alternativas: la tubería de agregación (aggregation pipeline), la función MapReduce, y operaciones de agregación con propósito simple.

Tubería de Agregación

Para utilizar la tubería de agregación de MongoDB, utilizamos el método aggregate de la colección sobre la cual deseemos realizar las operaciones.

```
db.collection.aggregate( [ { <etapa> }, { <etapa> } ] );
```

Dicho método recibe como parámetro un arreglo de “etapas”, en la que cada etapa consta de un objeto que representa una operación a realizar.

La data pasa por las etapas en el orden en que se pongan en el arreglo, es decir, la data resultante de la primera etapa, es la entrada de la segunda etapa, y así consecutivamente hasta que la última etapa retorna el resultado final obtenido.

Tubería de Agregación

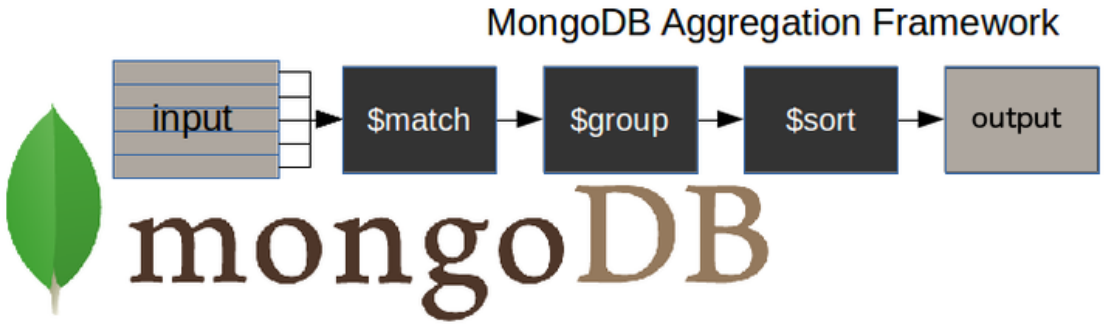
db.collection.aggregate([{ <etapa> }, { <etapa> }]);

Cada etapa está compuesta por un **operador de etapa** que representa una función y un objeto que representa una o varias **expresiones** que se pasa como parámetro a la función.

Tubería de Agregación

¿Cómo funciona el proceso de agregación de MongoDB?

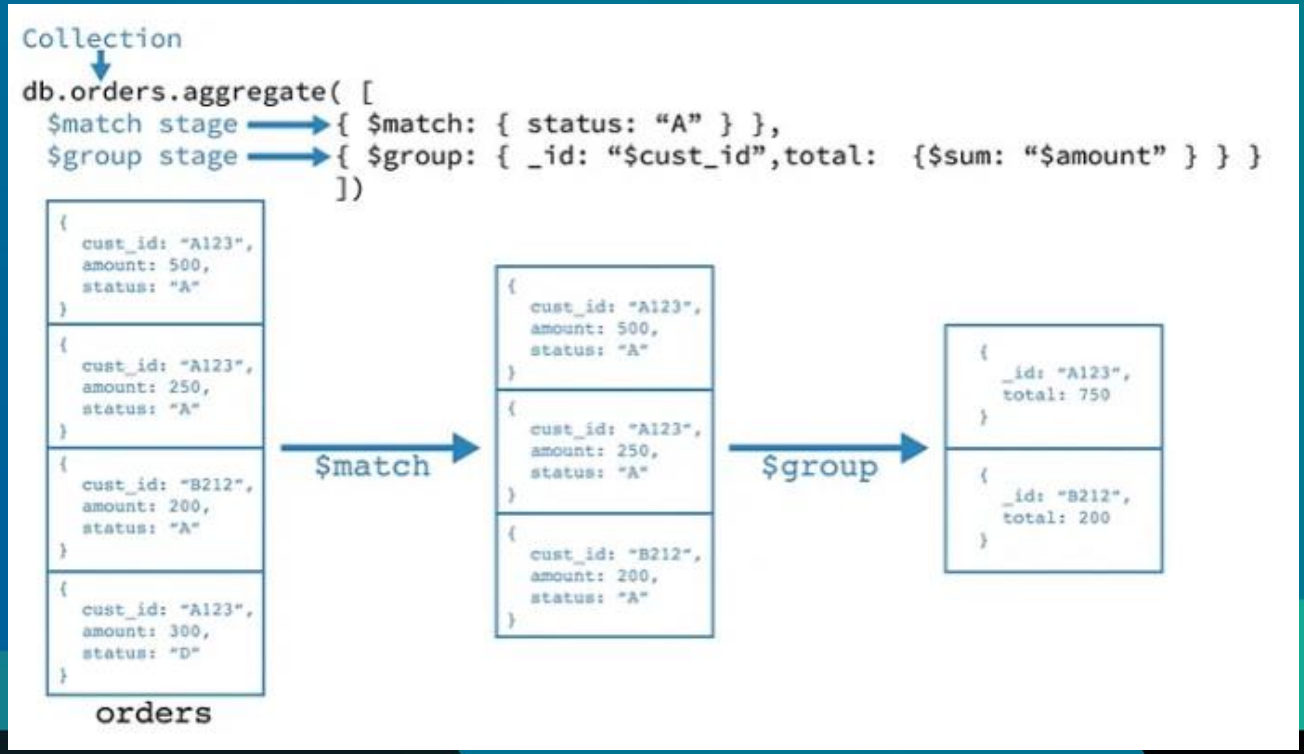
A continuación se muestra un diagrama que ilustra un proceso de agregación típico de MongoDB.



- `$match` etapa - filtra los documentos con los que necesitamos trabajar, los que se ajustan a nuestras necesidades
- `$group` etapa - realiza el trabajo de agregación
- `$sort` etapa - ordena los documentos resultantes de la forma que deseemos (ascendente o descendente)

La entrada de la cadena puede ser una sola colección, en la que se pueden fusionar otras más adelante.

Tubería de Agregación



Operadores de etapa

- {$match: { } }

Este operador funciona de forma similar al método find() que ya vimos en las operaciones de crud. Básicamente nos permite filtrar documentos de la colección según la consulta que pasemos como parámetro.

```
db.estudiantes.aggregate([{$match : {nombre : 'Juan'}}])
```

Operadores de etapa

- {$match: { } }

Permite obtener el **número de documentos**($count) que se tienen en la etapa específica de la agregación en la que se llama, pasamos como parámetro un string que servirá como alias a la consulta.

```
db.estudiantes.aggregate([{$match : {edad : { $lt : 20}}},
                          {$count : "Estudiantes menores a 20 años"}])
```

Operadores de etapa

- {$project: { <especificación(es)> } }

Permite seleccionar los campos de un documento para mostrar en el resultado.

Muestra : 1 No muestra : 0

```
db.estudiantes.aggregate([{$project : {nombre:1, _id:0}}])
```

El _id se muestra por defecto, por lo tanto se debe elegir cuando no mostrarlo.

Operadores de etapa

- {$project: { <especificación(es)> } }

Permite **crear alias** para los nombres de los campos, deben encerrarse entre “” con el signo $

```
db.estudiantes.aggregate([{$project : {_id:0, Nom:"$nombre", Ape:"$apellido" }}])
```

Operadores de etapa

- {$project: { <especificación(es)> } }

Permite utilizar **operadores de string**($concat) para formatear el resultado.

```
db.estudiantes.aggregate([{$project : {_id:0,
                                       nombre_Completo : {$concat : ["$nombre"," ","$apellido"]}
                                       }}])
```

Ver operadores de string para MongoDB

Operadores de etapa

- {$project: { <especificación(es)> } }

Permite **ordenar**($sort) los documentos obtenidos en la agregación. Como parámetro recibe un objeto con los diferentes campos sobre los cuales se ordenará, y un orden. El orden puede ser 1 si es ascendente o -1 si es descendente.

```
db.estudiantes.aggregate([{$project : {nombre:1,edad:1,_id:0}},
                          {$sort : {edad : 1}}])
```

Operadores de etapa

- {$project: { <especificación(es)> } }

Permite **limitar**($limit) el número de documentos obtenidos por la agregación en un entero que pasamos como parámetro.

```
db.estudiantes.aggregate([{$project : {nombre:1,edad:1,_id:0}},
                          {$sort : {edad : 1}},
                          {$limit : 3}])
```

Operadores de etapa

- {$project: { <especificación(es)> } }

Permite **saltar**($skip) los n primeros documentos que se obtengan en la agregación, donde n es un entero que se pasa como parámetro.

```
db.estudiantes.aggregate([{$project : {nombre:1,edad:1,_id:0}},
                          {$sort : {edad : 1}},
                          {$skip:2},
                          {$limit : 3}])
```

Operadores de etapa

- { $group: { _id: , : { : }, … } }

El commando group agrupa documentos según una determinada expresión y genera un documento por cada grupo el cual será la salida de esta etapa, y la entrada para la próxima etapa en la tubería.

Cada documento generado está compuesto por un **_id** que tendrá como valor el resultado de la expresión el cual será la clave primaria del documento, por lo tanto, este _id es obligatorio.

Operadores de etapa

- { $group: { _id: , : { : }, … } }

db.estudiantes.aggregate([{$group : { _id : '$programa', count : {$sum : 1}}}])
Agrupa los estudiantes por el programa que cursan

El id de cada documento generado será el mismo campo “programa”, y como campo adicional usamos un contador, el cual utiliza el **acumulador $sum**.

Operadores de etapa

- { $group: { _id: , : { : }, … } }

```
db.estudiantes.aggregate([{$group : { _id : {$lt : ['$edad',18]},
                                      edades : {$push : {edad : '$edad'}}}}])
```

La expresión ({$lt : [‘$edad’,18]}) retorna **true** o **false**, por lo que estos son los **ids** de cada grupo generado, y cada documento es clasificado en el grupo respectivo dependiendo si su valor cumple con el criterio.

Operadores de etapa

- { $unwind: {<expresión> }

Dada una colección universities con la siguiente estructura:

```
{
  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 }
  ]
}
```

Operadores de etapa

- { $unwind: {<expresión> }

Nos permite trabajar con los valores de los campos dentro de un array.

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

Si en los documentos de entrada hay un campo array , a veces tendrá que imprimir el documento varias veces, una por cada elemento de array.

Operadores de etapa

- { $count: {<expresión> }

Nos permite contabilizar los documentos de una colección o resultado.

```
db.universities.aggregate([
{ $match : { isActive : true } },
{ $count : 'usuariosactivos'}])
```

- Comenzamos con un grupo de usuarios.
- La primera etapa selecciona sólo los usuarios activos
- La segunda etapa cuenta cuántos usuarios quedan después del filtro.

Operadores de etapa

- {$sortByCount: <expresión> }

Permite hacer un $group dada la expresión que pasamos como parámetro y un $count de todos los elementos, y posteriormente, ordenarlo según el número de elementos de forma descendente.

```
db.estudiantes.aggregate([{$sortByCount : '$programa'}])
```

Operadores de etapa

- {$sort: <expresión> }

Permite hacer un ordenamiento según el criterio 1:ascendente -1:descendente

```
db.empleados.aggregate([
  {$sort:{'nombre':-1}},
  {$project : { _id: 0,  nombre: 1 } }
])
```

Resultado:

```
[ { nombre: 'Pedro' },
{ nombre: 'Juan' },
{ nombre: 'Ana' } ]
```

Operadores de etapa

- {$lookup: <expresión> }

Si necesitamos unir los datos de dos colecciones, emplearemos el operador $lookup, el cual realiza un *left outer join* a una colección de la misma base de datos para filtrar los documentos de la colección *joineada*.

El resultado es un nuevo campo array para cada documento de entrada, el cual contiene los documentos que cumplen el criterio del *join*.

Operadores de etapa

- {$lookup: <expresión> }

El operador $lookup utiliza cuatro parámetros:

from: colección con la que se realiza el join.

localField: campo de la colección origen, la que viene de la agregación db.origen.aggregate, que hace la función de clave ajena.

foreignField: campo en la colección indicada en from que permite la unión (sería la clave primaria de la colección sobre la que se realiza el join).

as: nombre del array que contendrá los documentos enlazados.

Operadores de etapa

- $lookup:

Dado el siguiente modelo:

-Buscar el “Nombre” de los empleados del Departamento 1 ordenado por nombre

empleados

_id	nombre	telefono	mail	idDepto
1	Juan	123	juan@gmail.com	1
2	Ana	456	ana@gmail.com	1
3	Pedro	789	pedro@gmail.com	2

departamentos

_id	nombre
1	RRHH
2	FINANZAS
3	SISTEMAS

```
db.empleados.insertOne(
   { _id: 1,
    nombre: 'Juan',
    telefono: '1234',
    mail: 'juan@gmail.com',
    idDepto : '1'
  })
db.empleados.insertOne(
   { _id: 2,
    nombre: 'Ana',
    telefono: '456',
    mail: 'ana@gmail.com',
    idDepto : '1'
  })
db.empleados.insertOne(
  {  _id: 3,
    nombre: 'Pedro',
    telefono: '789',
    mail: 'pedro@gmail.com',
    idDepto : '2'
  })
```

```
db.departamentos.insertOne(
  {  _id: 1,
    nombre: 'RRHH',
  })
db.departamentos.insertOne(
  {
    _id: 2,
    nombre: 'FINANZAS',
  })
db.departamentos.insertOne(
  {
    _id: 3,
    nombre: 'SISTEMAS',
  })
```

Operadores de etapa

- $lookup:

```
db.departamentos.aggregate([
 { $match : { _id : 1 }},
 { $lookup : {
    from : 'empleados',
    localField : '_id',
    foreignField : 'idDepto',
    as : 'EmpleadosDepto1'  }},
 {$unwind:'$EmpleadosDepto1'},
 {$sort:{'EmpleadosDepto1.nombre': 1}},
 {$project : { _id:0, 'EmpleadosDepto1.nombre': 1 } }])
```

→ _id departamento = 1

→ join con “empleados”

Como la relación siempre va a provocar la creación de un array, mediante $unwind lo podemos acceder

→ ordeno por nombre

→ muestro nombre del empleado

Operadores de etapa

- $lookup:

Dado el siguiente modelo:

-Buscar el “Nombre” de los empleados del Departamento 1 ordenado por nombre

Resultado:

```
[
 { EmpleadosDepto1: { nombre: 'Ana' } },
 { EmpleadosDepto1: { nombre: 'Juan' } }
]
```

empleados

_id	nombre	telefono	mail	idDepto
1	Juan	123	juan@gmail.com	1
2	Ana	456	ana@gmail.com	1
3	Pedro	789	pedro@gmail.com	2

```
db.empleados.insertOne(
   { _id: 1,
    nombre: 'Juan',
    telefono: '1234',
    mail: 'juan@gmail.com',
    idDepto : '1'
  })
db.empleados.insertOne(
   { _id: 2,
    nombre: 'Ana',
    telefono: '456',
    mail: 'ana@gmail.com',
    idDepto : '1'
  })
db.empleados.insertOne(
  {  _id: 3,
    nombre: 'Pedro',
    telefono: '789',
    mail: 'pedro@gmail.com',
    idDepto : '2'
  })
```

departamentos

_id	nombre
1	RRHH
2	FINANZAS
3	SISTEMAS

```
db.departamentos.insertOne(
  {  _id: 1,
    nombre: 'RRHH',
  })
db.departamentos.insertOne(
  {
    _id: 2,
    nombre: 'FINANZAS',
  })
db.departamentos.insertOne(
  {
    _id: 3,
    nombre: 'SISTEMAS',
  })
```