10-01-2022

AGGREGATION

MONGO DB

Consuelo Begines Roldán
GESTIÓN BASE DE DATOS

ÍNDICE

1. ¿Para qué sirve?

2. Operadores para usar en las etapas……
   2.1. $match……
   2.2. $group……
   2.3. $project……
   2.4. $sort……
   2.5. $limit……
   2.6. $round……

3. Operadores para utilizar en cada etapa……
   3.1. $sum……
   3.2. $multiply……
   3.3. $divide……
   3.4. $avg……
   3.5. $max……
   3.6. $min……
   3.7. $subtract……
   3.8. $expr……
   3.9. $year……

1 ¿Para qué sirve?

Para utilizar la tubería de agregación de MongoDB, utilizamos el método aggregate de la colección sobre la cual deseemos realizar las operaciones. Dicho método recibe como parámetro un arreglo de “etapas”, en la que cada etapa consta de un objeto que representa una operación a realizar. Funciona como una tubería, ya que 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.

La sintaxis para ejecutar un comando de agregación sería la siguiente:

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

Y 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.

Los operadores que podemos utilizar a este nivel se denominan operadores de etapa, y los describiremos a continuación:

2 Operadores que podemos utilizar en las etapas

2.1. $match:

{$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..

Ej: Para obtener los estudiantes de nombre “Juan” lo hacemos de la siguiente forma:

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

2.2. $group:

{ $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.

Y opcionalmente, puede contener campos adicionales obtenidos a través de operadores de acumulación o acumuladores, los cual veremos más detalladamente más adelante.

Tanto el _id como los acumuladores aceptan como parámetro una expresión válida.

Ej: Si deseamos agrupar los estudiantes separando los mayores de edad de los menores de edad. Debemos pasar una expresión

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

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

De la misma forma, una expresión válida puede ser el nombre de uno de los campos directamente.

Ej: Si queremos agrupar los estudiantes por el programa que cursan, hacemos lo siguiente:

db.estudiantes.aggregate([{$group : { _id : '$programa', count : {$sum : 1}}}])

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

2.3. $project:

{$project: { <especificación(es)> } }

Nos permite modificar la data de un documento al mostrarla como resultado, según unas especificaciones que pasamos como parámetro ya sea removiendo campos, o agregando nuevos mediante el uso de otras expresiones.

Ej: Si queremos solo ver el nombre de los estudiantes registrados, lo hacemos de la siguiente forma:

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

Ahora, también podemos agregar campos nuevos al documento utilizando expresiones. La más básica de ellas sería mostrar un campo ya existente pero como un alias.

Ej: para mostrar el apellido como “segundoNombre” lo haríamos de la siguiente forma:

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

Aquí utilizamos $ dentro de un string seguido del nombre del campo al que queremos hacer referencia.

Para ejemplos más útiles, MongoDB nos provee varios operadores de expresión que podemos utilizar para construir las etapas de las agregaciones.

Por ejemplo, si quisiéramos obtener el nombre y apellido del estudiante lo haríamos mediante una expresión que utilice el operador de strings $concat, de la siguiente forma:

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

2.4. $sort:

{ $sort: { : , : … } }

Nos permite ordenar 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.

Ej: Si queremos obtener los estudiantes ordenados según su edad de forma ascendente, lo haríamos de la siguiente forma:

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

2.5. $limit:

{ $limit: }

Nos permite limitar el número de documentos obtenidos por la agregación en un entero que pasamos como parámetro.

Ej: Si queremos obtener solo 3 estudiantes más jóvenes, lo haríamos así:

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

2.6. $round:

La función ROUND redondea los números hasta el valor entero o decimal más cercano. La función ROUND puede incluir, de forma opcional, un segundo argumento como un valor entero que indique la cantidad de lugares decimales para el redondeo, sea cual sea la dirección.

3 Operadores que podemos usar en cada etapa

3.1. $sum:

Este acumulador permite acumular los valores que retorne la expresión que pasamos como parámetro. En caso de que se use en el su segunda forma, retorna la sumatoria de todas las expresiones que se pasen como parámetro por cada uno de los documentos.

Ej: Para obtener la sumatoria y el promedio de todas las edades, lo hacemos de la siguiente forma.
db.estudiantes.aggregate([{$group : {_id: 'idAux', suma: {$sum: '$edad'}, promedio: {$avg:'$edad'}}}]);

NOTA: En este ejemplo como en los siguientes, como estamos utilizando un id auxiliar, toda la colección se considera un documento agrupado por este id, por lo que el operador está retornando la sumatoria o average te todo el documento conformado por la colección.

3.2. $multiply:

Multiplica números y devuelve el resultado. Pase los argumentos a $multiplyen una matriz.

La $multiplyexpresión tiene la siguiente sintaxis:

{ $multiply: [<expression1>, <expression2>, ...] }

Los argumentos pueden ser cualquier expresión válida siempre que se resuelvan en números.

3.3. $divide:

Divide un número por otro y devuelve el resultado. Pase los argumentos a $divideen una matriz.

La $divideexpresión tiene la siguiente sintaxis:

{ $divide: [<expression1>, <expression2>] }

El primer argumento es el dividendo y el segundo argumento es el divisor; es decir, el primer argumento se divide por el segundo argumento.

Los argumentos pueden ser cualquier expresión válida siempre que se resuelvan en números.

3.4. $avg:

Devuelve el valor medio de los valores numéricos. $avg ignora los valores no numéricos.

$avg está disponible en estas etapas:

$addFields (Disponible a partir de MongoDB 3.4)
$bucket
$bucketAuto
$group
$matchetapa que incluye una $expr expresión
$project
$replaceRoot (Disponible a partir de MongoDB 3.4)
$replaceWith (Disponible a partir de MongoDB 4.2)
$set (Disponible a partir de MongoDB 4.2)
$setWindowFields (Disponible a partir de MongoDB 5.0)

3.5. $max:

Permite obtener el valor máximo y el valor mínimo de la expresión que se evalúa para cada documento generado por el agrupamiento.

Ej: Obtengamos la edad máxima y la mínimo de la colección
db.estudiantes.aggregate([{$group : {_id: 'idAux', maximo: {$max: '$edad'}, minimo: {$min:'$edad'}}}]);

3.6. $min:

Permite obtener el valor máximo y el valor mínimo de la expresión que se evalúa para cada documento generado por el agrupamiento.

Ej: Obtengamos la edad máxima y la mínimo de la colección
db.estudiantes.aggregate([{$group : {_id: 'idAux', maximo: {$max: '$edad'}, minimo: {$min:'$edad'}}}]);

3.7. $subtract:

Resta dos números para devolver la diferencia, o dos fechas para devolver la diferencia en milisegundos, o una fecha y un número en milisegundos para devolver la fecha resultante.

La $subtractexpresión tiene la siguiente sintaxis:

{ $subtract: [<expression1>, <expression2>] }

El segundo argumento se resta del primer argumento.

Los argumentos pueden ser cualquier expresión válida siempre que se resuelvan en números y / o fechas. Para restar un número de una fecha, la fecha debe ser el primer argumento.

3.8. $expr:

Permite el uso de expresiones de agregación dentro del lenguaje de consulta.

$expr tiene la siguiente sintaxis:

{ $expr: { <expression> } }

Los argumentos pueden ser cualquier expresión de agregación válida

3.9. $year:

Devuelve la parte del año de una fecha.

La $yearexpresión tiene la siguiente sintaxis de expresión de operador :

{ $year: <dateExpression> }