Funciones de Agregación MongoDB.

En las bases de datos es muy importante disponer de consultas que permitan combinar diferentes elementos; por ejemplo, dada la colección de ventas en una tienda calcular el total vendido; o incluso los subtotales logrados por cada sucursal.

Las operaciones de agregación procesan varios documentos y devuelven resultados calculados. Puede utilizar operaciones de agregación para:

- Agrupe los valores de varios documentos juntos.
- Realice operaciones en los datos agrupados para devolver un único resultado.
- Analice los cambios de datos a lo largo del tiempo.

Para realizar operaciones de agregación, puede utilizar:

- Tuberías de agregación
- Métodos de agregación de propósito único
- Funciones de reducción de mapas

Canalizaciones (Pipeline) de agregación

Una canalización de agregación consta de una o más etapas que procesan documentos:

- Cada etapa realiza una operación en los documentos de entrada. Por ejemplo, una etapa puede filtrar documentos, agrupar documentos y calcular valores.
- Los documentos que salen de una etapa se ingresan a la siguiente etapa.
- Una canalización de agregación puede devolver resultados para grupos de documentos. Por ejemplo, devuelva los valores totales, promedio, máximo y mínimo.

Ejemplo de agregación.

El siguiente ejemplo de canalización de agregación contiene dos etapas y devuelve la cantidad total de pedidos urgentes para cada producto:

```
db.orders.aggregate( [
   { $match: { status: "urgent" } },
   { $group: { _id: "$productName", sumQuantity: { $sum: "$quantity" } } }
] )
```

El escenario **$match**:

Filtra los documentos a aquellos con una status de urgent.
Envía los documentos filtrados al $group escenario.

El escenario **$group**:

Agrupa los documentos de entrada por productName.
Se utiliza $sum para calcular el total quantity de cada uno productName, que se almacena en el sumQuantitycampo devuelto por la canalización de agregación. Link de Documentación de MongoDB: https://docs.mongodb.com/manual/aggregation/

Los elementos de la "tubería" se incluyen en un array y se ejecutarán por orden. Cada elemento puede repetirse y el orden puede variar.

$project: Su función es "recolocar" el documento. Selecciona las claves que se usarán, y puede "elevar" claves que están en subdocumentos al nivel superior. Tras un paso $project habrá tantos documentos como inicialmente; pero con un formato que puede haber variado. (1:1)
$match: Filtra documentos, dejando solo los que vamos a utilizar. (n:1)
$group: Realiza la agregación (n:1)
$sort: Ordena. 1:1
$skip: Saltarse algunos a elementos n:1
$limit: Número de elementos máximo. n:1
$unwind: "aplana" datos de arrays, produciendo tantos elementos como elementos tenga el array. 1:n
$out: crea una colección nueva a partir de los datos. 1:1
$redact: Seguridad. Impide que algunos usuarios vean algunos documentos. n:1
$geonear: Se utiliza para búsquedas por posición (ver índices geoespaciales). n:1
$sample: permite elegir al azar unos cuantos documentos a modo de muestra. n:1
$lookup: join the varias colecciones.

Vamos a estudiar más en detalle $group que necesitamos utilizar en los dos últimos sprint solicitados.

$ group (agregación)

Definición

$group

Agrupa los documentos de entrada por la expresión _id especificada y para cada agrupación distinta, genera un documento. El _idcampo de cada documento de salida contiene el grupo único por valor. Los documentos de salida también pueden contener campos calculados que contienen los valores de alguna expresión de acumulador. $group no ordena los documentos.

El formato de este comando es el siguiente:

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

Campo	Descripción
`_id`	*Requerido.* Si especifica un `_id` valor nulo o cualquier otro valor constante, la `$group` etapa calcula los valores acumulados para todos los documentos de entrada en su conjunto. Ver ejemplo de Group by Null .
`field`	*Opcional.* Calculado utilizando los operadores acumuladores .

Operador acumulador

El operador <accumulator> debe ser uno de los siguientes operadores de acumuladores:

Modificado en la versión 5.0 .

Nombre	Descripción
$accumulator	Devuelve el resultado de una función de acumulador definida por el usuario.
$addToSet	Devuelve una matriz de valores de expresión *únicos* para cada grupo. El orden de los elementos de la matriz no está definido.
$avg	Devuelve un promedio de valores numéricos. Ignora los valores no numéricos.
$count	Devuelve el número de documentos de un grupo. Distinto de la etapa de tubería $count.
$first	Devuelve un valor del primer documento para cada grupo. El orden solo se define si los documentos están clasificados. Distinto del operador de matriz $first.
$last	Devuelve un valor del último documento para cada grupo. El orden solo se define si los documentos están clasificados. Distinto del operador de matriz $last.
$max	Devuelve el valor de expresión más alto para cada grupo.
$mergeObjects	Devuelve un documento creado al combinar los documentos de

Nombre	Descripción
	entrada para cada grupo.
$min	Devuelve el valor de expresión más bajo para cada grupo.
$push	Devuelve una matriz de valores de expresión para documentos de cada grupo.
$stdDevPop	Devuelve la desviación estándar de la población de los valores de entrada.
$stdDevSamp	Devuelve la desviación estándar de la muestra de los valores de entrada.
$sum	Devuelve una suma de valores numéricos. Ignora los valores no numéricos.

Revisar el siguiente link:
https://docs.mongodb.com/manual/reference/operator/aggregation/group/

Ejemplo

Vamos a realizar un ejemplo utilizando la consola de mongodb, recordar que debemos iniciar mongod y después ejecutamos mongo para ingresar al Shell de mongo, usar los siguientes datos para nuestro ejemplo:

```
use running
db.sesiones.insert({nombre:"Bertoldo", mes:"Marzo", distKm:6, tiempoMin:42})
db.sesiones.insert({nombre:"Herminia", mes:"Marzo", distKm:10, tiempoMin:60})
db.sesiones.insert({nombre:"Bertoldo", mes:"Marzo", distKm:2, tiempoMin:12})
db.sesiones.insert({nombre:"Herminia", mes:"Marzo", distKm:10, tiempoMin:61})
db.sesiones.insert({nombre:"Bertoldo", mes:"Abril", distKm:5, tiempoMin:33})
db.sesiones.insert({nombre:"Herminia", mes:"Abril", distKm:42, tiempoMin:285})
db.sesiones.insert({nombre:"Aniceto", mes:"Abril", distKm:5, tiempoMin:33})
```

Supongamos que queremos saber el número de sesiones que ha realizado cada persona:

```
> db.sesiones.aggregate([{$group:{_id:'$nombre', num_sesiones: {$sum:1}}}])
{ "_id" : "Bertoldo", "num_sesiones" : 3 }
{ "_id" : "Aniceto", "num_sesiones" : 1 }
{ "_id" : "Herminia", "num_sesiones" : 3 }
>
```

También podemos agrupar por nombre y mes:

```
> db.sesiones.aggregate(
... [
... {$group:
... { _id:{nombre:"$nombre",
... mes: "$mes"},
... num_sesiones: {$sum:1}
... }
... }
... ]
... )
{ "_id" : { "nombre" : "Aniceto", "mes" : "Abril" }, "num_sesiones" : 1 }
{ "_id" : { "nombre" : "Herminia", "mes" : "Marzo" }, "num_sesiones" : 2 }
{ "_id" : { "nombre" : "Herminia", "mes" : "Abril" }, "num_sesiones" : 1 }
{ "_id" : { "nombre" : "Bertoldo", "mes" : "Marzo" }, "num_sesiones" : 2 }
{ "_id" : { "nombre" : "Bertoldo", "mes" : "Abril" }, "num_sesiones" : 1 }
>
```

Funciones de agregación

Ya hemos visto una función de agregación, $sum, pero hay muchas otras:

$sum: suma (o incrementa)
$avg : calcula la media
$min: mínimo de los valores
$max: máximo
$push: Mete en un array un valor determinado
$addToSet: Mete en un array los valore que digamos, pero solo una vez
$first: obtiene el primer elemento del grupo, a menudo junto con sort
$last: obtiene el último elemento, a menudo junto con sort

$sum

Ya la hemos visto como función para "contar" usando $sum:1, pero su propósito original es sumar:

```
> db.sesiones.aggregate(
... [
... {$group:
... { _id:{nombre:"$nombre"},
... num_km: {$sum:'$distKm'}
... }
... }
... ]
... )
{ "_id" : { "nombre" : "Bertoldo" }, "num_km" : 13 }
{ "_id" : { "nombre" : "Aniceto" }, "num_km" : 5 }
{ "_id" : { "nombre" : "Herminia" }, "num_km" : 62 }
>
```

$avg

Calcula la media. Por ejemplo: kilómetros que corre cada uno de media al mes

```
> db.sesiones.aggregate(
... [
... {$group:
... { _id:{nombre:"$nombre",
... mes: "$mes"},
... media: {$avg:'$distKm'}
... }
... }
... ]
... )
{ "_id" : { "nombre" : "Bertoldo", "mes" : "Marzo" }, "media" : 4 }
{ "_id" : { "nombre" : "Bertoldo", "mes" : "Abril" }, "media" : 5 }
{ "_id" : { "nombre" : "Aniceto", "mes" : "Abril" }, "media" : 5 }
{ "_id" : { "nombre" : "Herminia", "mes" : "Marzo" }, "media" : 10 }
{ "_id" : { "nombre" : "Herminia", "mes" : "Abril" }, "media" : 42 }
>
```

$addToSet

$addToSet crea arrays agrupando elementos.

Ejemplo: Supongamos que queremos saber qué distancias ha corrido cada persona.
Agrupamos por el nombre y "coleccionamos" las distancias distintas

```
> db.sesiones.aggregate(
...                         [
...                          {$group:
...                                  { _id:{nombre:"$nombre"},
...                                    distancias: {$addToSet:'$distKm'}
...                                  }
...                           }
...                          ]
...                        )
{ "_id" : { "nombre" : "Bertoldo" }, "distancias" : [ 2, 5, 6 ] }
{ "_id" : { "nombre" : "Aniceto" }, "distancias" : [ 5 ] }
{ "_id" : { "nombre" : "Herminia" }, "distancias" : [ 10, 42 ] }
```

$push

Análogo a $addToSet pero admite repeticiones.

Ejemplo, queremos saber en cada mes qué distancias se han hecho en alguna sesión. Si una distancia se ha corrido varias veces en ese mes debe aparecer varias veces:

```
> db.sesiones.aggregate(
...                         [
...                          {$group:
...                                  { _id:{mes:"$mes"},
...                                    distancias:{$push:'$distKm'}
...                                  }
...                           }
...                          ]
...                        )
{ "_id" : { "mes" : "Marzo" }, "distancias" : [ 6, 10, 2, 10 ] }
{ "_id" : { "mes" : "Abril" }, "distancias" : [ 5, 42, 5 ] }
>
```

$unwind

Es el reverso de $push; cuando tenemos documentos que contienen un array y queremos agrupar por valores del array, a veces conviene eliminar los arrays y convertirlos en múltiples documentos.

En realidad estamos "normalizando" (primera forma normal).

Ejemplo:

Volvemos al ejemplo de personas con aficiones:

Queremos saber el número de personas con el que cuenta cada afición. ¿Cómo hacerlo?

Para ello el primer paso es hacer $unwind:

```
> db.gustos.aggregate([   {$unwind:'$aficiones'}   ] )
{ "_id" : ObjectId("61b14c91c3fb2372b6a06ba0"), "nombre" : "Bertoldo", "aficiones" : "siesta" }
{ "_id" : ObjectId("61b14c91c3fb2372b6a06ba0"), "nombre" : "Bertoldo", "aficiones" : "cine" }
{ "_id" : ObjectId("61b14c91c3fb2372b6a06ba1"), "nombre" : "Herminia", "aficiones" : "correr" }
{ "_id" : ObjectId("61b14c91c3fb2372b6a06ba1"), "nombre" : "Herminia", "aficiones" : "cine" }
{ "_id" : ObjectId("61b14c91c3fb2372b6a06ba2"), "nombre" : "Aniceta", "aficiones" : "viajar" }
{ "_id" : ObjectId("61b14c91c3fb2372b6a06ba2"), "nombre" : "Aniceta", "aficiones" : "cine" }
{ "_id" : ObjectId("61b14c92c3fb2372b6a06ba3"), "nombre" : "Godofredo", "aficiones" : "correr" }
{ "_id" : ObjectId("61b14c92c3fb2372b6a06ba3"), "nombre" : "Godofredo", "aficiones" : "montaña" }
{ "_id" : ObjectId("61b14c92c3fb2372b6a06ba3"), "nombre" : "Godofredo", "aficiones" : "cine" }
```

Ahora es fácil pensar en la siguiente etapa: agrupar por aficiones

```
> db.gustos.aggregate([
...                     {$unwind:'$aficiones'},
...                     {$group:
...                         {_id:'$aficiones',
...                          total:{$sum:1} } }
...                    ] )
{ "_id" : "montaña", "total" : 1 }
{ "_id" : "correr", "total" : 2 }
{ "_id" : "viajar", "total" : 1 }
{ "_id" : "cine", "total" : 4 }
{ "_id" : "siesta", "total" : 1 }
```

$max, $min

El mayor, menor valor en el grupo.

Ejemplo:

```
> db.sesiones.aggregate(
...                     [
...                      {$group:
...                             { _id:{nombre:"$nombre"},
...                               maxdist:{$max:'$distKm'},
...                               mindist:{$min:'$distKm'}
...                             }
...                      }
...                     ]
...                    )
{ "_id" : { "nombre" : "Aniceto" }, "maxdist" : 5, "mindist" : 5 }
{ "_id" : { "nombre" : "Bertoldo" }, "maxdist" : 6, "mindist" : 2 }
{ "_id" : { "nombre" : "Herminia" }, "maxdist" : 42, "mindist" : 10 }
```

Implementación en Spring Boot

Vamos a implementar algunas de estas consultas especiales dentro del proyecto spring boot, vamos a desarrollar el ejemplo que nos permita mostrar el total de ventas por sucursal, para desarrollarlo debemos llevar el orden siguiente:

- Construir la clase que permita definir el modelo de datos a mostrar en la consulta.

  La clase debe contener los atributos que debemos mostrar, la consulta que queremos mostrar según lo solicitado en el sprint 5.

  La clase debe contener el nombre de la ciudad que representa la sucursal y el valor total de ventas.

  De tal forma que la clase debe contener el siguiente contenido:

```java
package com.mintic.ventas.modelo;

import org.springframework.data.annotation.Id;

public class VentasGrupo {

    @Id
    private String sucursal;
    private double totalventa;

    public VentasGrupo() {
    }

    public VentasGrupo(String sucursal, double totalventa) {
        this.sucursal = sucursal;
        this.totalventa = totalventa;
    }

    public String getSucursal() {
        return sucursal;
    }

    public void setSucursal(String sucursal) {
        this.sucursal = sucursal;
    }

    public double getTotalventa() {
        return totalventa;
    }
}
```

```
31    public void setTotalventa(double totalventa) {
32        this.totalventa = totalventa;
33    }
34
35
36
37 }
```

Debemos considerar que esta clase se va a mostrar en el frontend, por ello agregamos la anotación de @Id al atributo de sucursal.

- Crear la consulta dentro del repositorio.

  Vamos a conectarnos a la base de datos creada en el ejercicio de ventas anterior desde el Shell de mongo, para ello vamos a conectarnos con use nombrebasededatos.

  Revisemos las bases de datos existentes con show dbs

```
> show dbs
admin           0.000GB
config          0.000GB
db_clientes32   0.000GB
db_clientes33   0.000GB
db_productos    0.000GB
db_productos00  0.000GB
db_productos32  0.000GB
db_productos33  0.000GB
db_tutoriales   0.000GB
db_ventas00     0.000GB
examples        0.000GB
local           0.000GB
mystore         0.000GB
running         0.000GB
tienda          0.000GB
tiendaweb       0.000GB
```

  Nos conectamos a la base de datos, revisamos el nombre de las colecciones y mostramos los datos de la colección con la secuencia de los siguientes comandos:

```
> use db_ventas00
switched to db db_ventas00
> show collections
ventas
> db.ventas.find().pretty()
```

  El resultado de la consulta sería el siguiente:

Creamos la consulta de tipo agregación que realice la suma por sucursal como se muestra en la siguiente gráfica:

```
> db.ventas.aggregate( [ {$group: { _id:{sucursal:"$sucursal"}, totalventa: {$sum:'$total_venta'} } } ])
{ "_id" : { "sucursal" : "BOGOTA" }, "totalventa" : 28900 }
{ "_id" : { "sucursal" : null }, "totalventa" : 2500 }
>
```

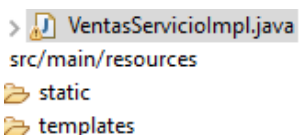
Ahora en el proyecto de spring boot agregamos la consulta que permita mostrar el total de ventas por sucursal.

```java
package com.mintic.ventas.dao;

import java.util.List;

import org.springframework.data.mongodb.repository.Aggregation;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

import com.mintic.ventas.modelo.Ventas;
import com.mintic.ventas.modelo.VentasGrupo;

@Repository
public interface IVentasRepositorio extends MongoRepository<Ventas, String> {

    @Aggregation("{ $group : { _id : {sucursal:'$sucursal'}, totalventa : { $sum : '$total_venta' } } }")
    List<VentasGrupo> sumSucursal();

}
```

- Crear el encabezado en la interface de servicio que defina el método a implementar.

```java
package com.mintic.ventas.servicio;

import java.util.List;

import com.mintic.ventas.modelo.Ventas;
import com.mintic.ventas.modelo.VentasGrupo;

public interface VentasServicio {

    // Cabeceras de métodos a utilizar en la colección Ventas
    // Crear
    Ventas crearVentas(Ventas ventas);

    //Actualizar
    Ventas updateVentas(Ventas ventas);

    //Listado de Ventas
    List<Ventas> getAllVentas();

    //Buscar Ventas por _id
    Ventas getVentasById(String ventasId);

    //Eliminar un Ventas
    void deleteVentas(String ventasId);

    //Ventas por Sucursal
    List<VentasGrupo> ventasSucursal();

}
```

- Implementar el método en la clase de servicio.




```java
    @Override
    public List<VentasGrupo> ventasSucursal() {
        return ventaRepo.sumSucursal();
    }
```

- Definir el endpoint que permite dar la respuesta a la petición.

```
@GetMapping("/sucursales")
public ResponseEntity<List<VentasGrupo>> getAllVentasxSucursal(){
    return ResponseEntity.ok().body(ventasServicio.ventasSucursal());
}
```

- Probar el método en Swagger

  Ejecutamos el proyecto y accedemos a la interfaz de swagger en el puerto 8081, dado que utilice el mismo backend de ventas

  El resultado es el siguiente:

  Se puede consultar en el siguiente link: https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#mongo.repositories

  En el ejemplo referenciado en:
  Example 161. Aggregating Repository Method