

Taller de Bases de Datos

Temas avanzados en MongoDB

Contenidos

- Agregaciones
 - MapReduce
- Modelamiento de relaciones
 - Uno a Uno
 - Uno a muchos
- Uso de driver Java

Aggregate

Aggregate



MongoDB tiene un framework de agregación, modelado como un canal de procesamiento de datos.

Los documentos entran a un **canal de múltiples pasos** que lleva a un resultado agregado.

```
db.orders.aggregate( [  
    { $match: { status: "A" } },  
    { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }  
])
```

<https://docs.mongodb.com/manual/aggregation/#aggregation-pipeline>

<pre>{ cust_id: "A123", amount: 500, status: "A" }</pre>
<pre>{ cust_id: "A123", amount: 250, status: "A" }</pre>
<pre>{ cust_id: "B212", amount: 200, status: "A" }</pre>
<pre>{ cust_id: "A123", amount: 300, status: "D" }</pre>

orders

\$match

<pre>{ cust_id: "A123", amount: 500, status: "A" }</pre>
<pre>{ cust_id: "A123", amount: 250, status: "A" }</pre>
<pre>{ cust_id: "B212", amount: 200, status: "A" }</pre>

\$group

<pre>{ _id: "A123", total: 750 }</pre>
<pre>{ _id: "B212", total: 200 }</pre>

Ejemplo de agregación

Consideremos un documento con este formato:

```
{
  "_id" : ObjectId("5f2f3dfafc817401512987e5"),
  "name" : "Moneda",
  "age" : NumberInt(10),
  "color" : "blanco",
  //...
  "juguetes" : [ "pelota", "peluche" ],
  "owner" : "Mario Hugo"
}
```

Ejemplo de agregación

Si quisiéramos hacer una etapa **\$match** para filtrar por dueño y luego contar juguetes en una etapa **\$group**, nos resulta:

```
db.dog.aggregate(  
  [  
    { $match: { "owner": "Mario Hugo" } },  
    {  
      $group: {  
        _id: "$_id",  
        juguetes_count: {  
          $first: { $size: "$juguetes" }  
        }  
      }  
    }  
  ]  
)
```

Ejemplo de agregación

Como las etapas son opcionales, podemos **eliminar la etapa \$match** y el proceso de agregación se aplica a todos los documentos.

```
db.dog.aggregate(  
  [  
    {  
      $group: {  
        _id: "$_id",  
        juguetes_count: {  
          $first: { $size: "$juguetes" }  
        }  
      }  
    }  
  ]  
)
```


Ejemplo de agregación

El resultado tiene el siguiente formato:

```
{ "_id" : ObjectId("5f3ed38103329a610d31c546"), "juguetes_count" : 0 }  
{ "_id" : ObjectId("5f2f3e15fc817401512987e6"), "juguetes_count" : 3 }  
{ "_id" : ObjectId("5f2f3dfafc817401512987e5"), "juguetes_count" : 2 }
```

¿cómo se logra lo mismo en SQL?

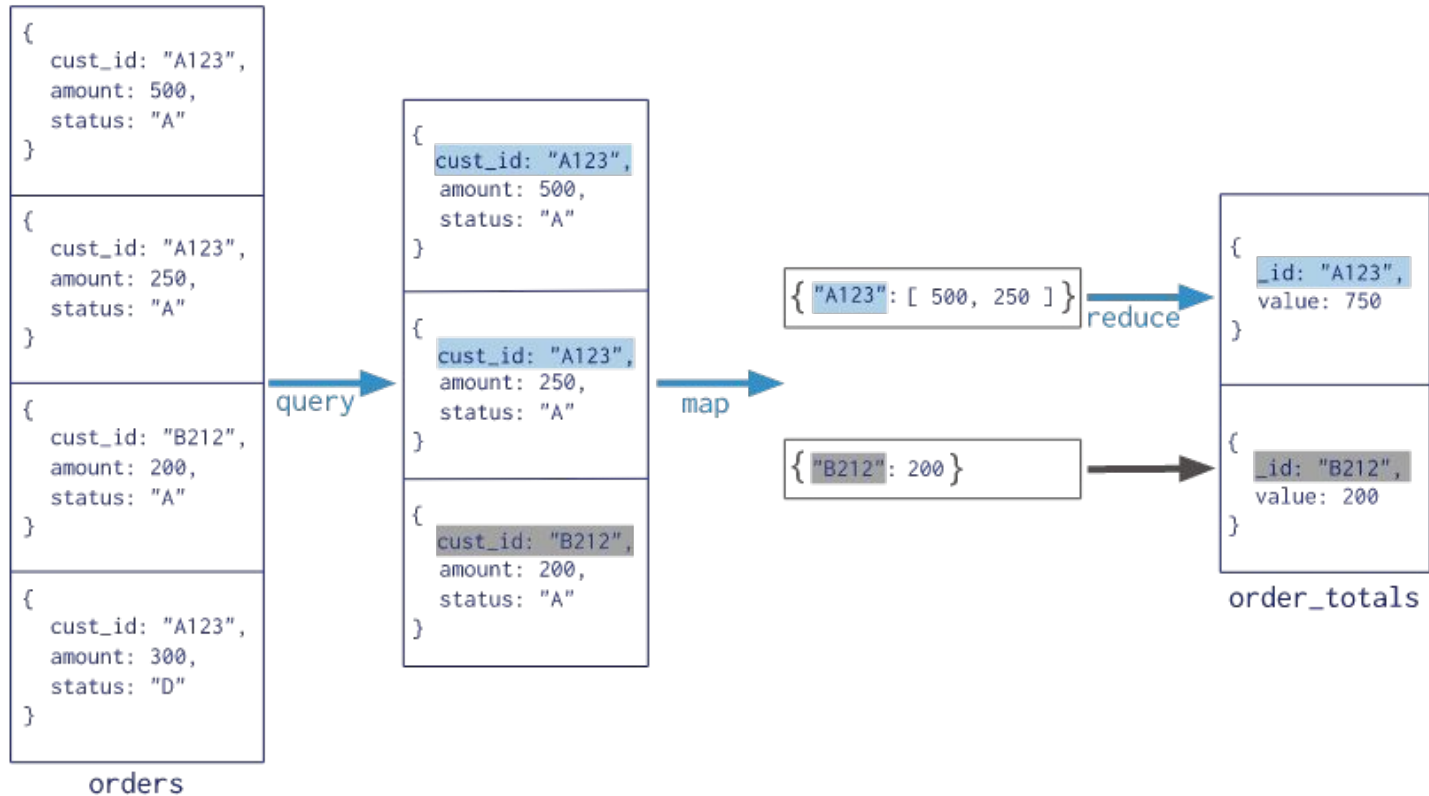
Map Reduce

Map-Reduce

En conjunto con el framework aggregate, MongoDB incluye el paradigma de **Map-Reduce** para procesamiento de grandes volúmenes de datos.

Tiene la siguiente forma:

```
Collection
↓
db.orders.mapReduce(
  map    → function() { emit( this.cust_id, this.amount ); },
  reduce → function(key, values) { return Array.sum( values ) },
  {
    query: { status: "A" },
    out: "order_totals"
  }
)
```



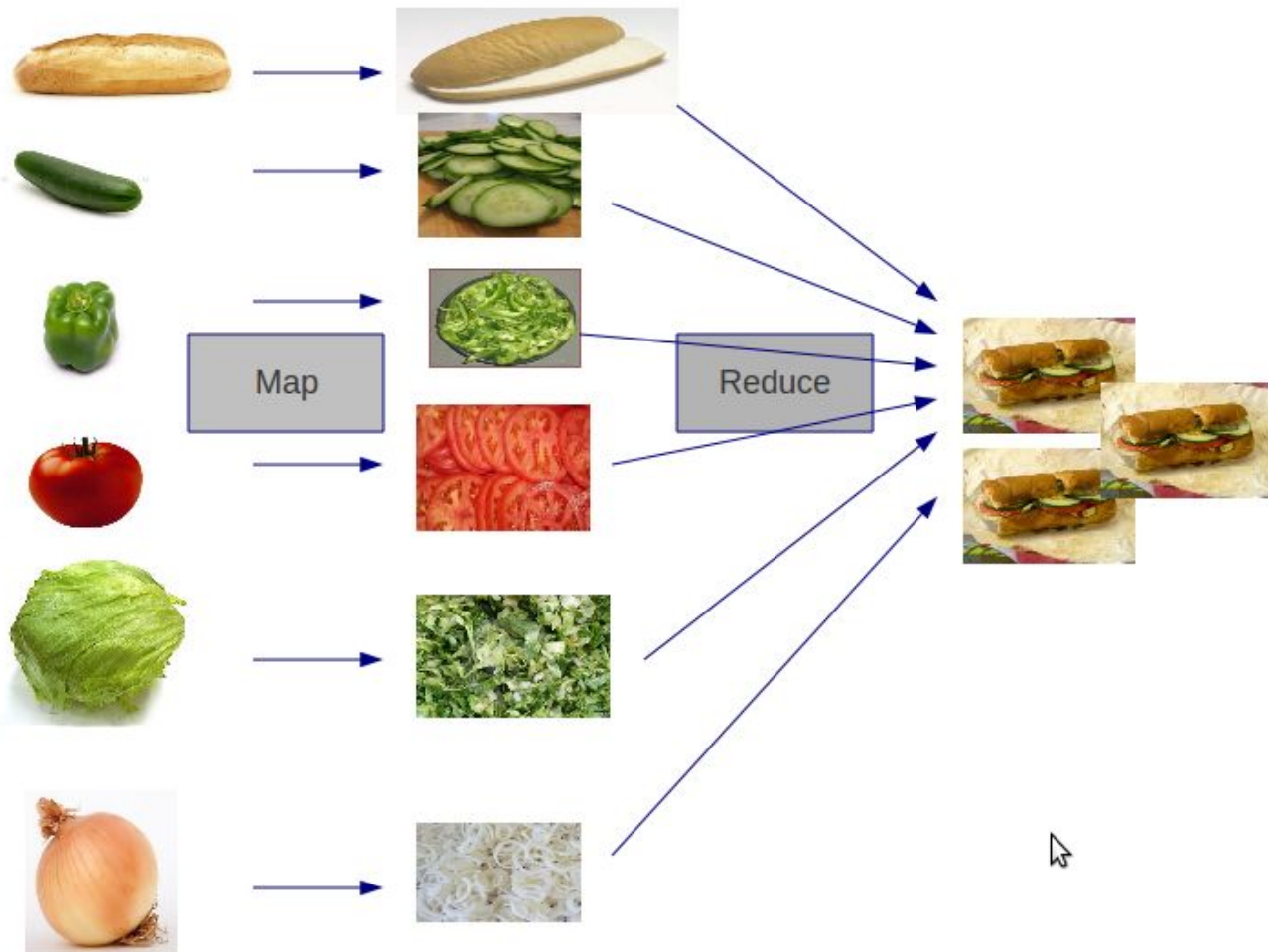
<https://docs.mongodb.com/manual/core/map-reduce/>

Map-Reduce



En este paradigma, la función **map** es una transformación a los datos y **reduce** es la función de agregación.

Para recordarlo visualmente:



Consideraciones de Map-Reduce

```
map    —> function() { emit( this.cust_id, this.amount ); },  
reduce —> function(key, values) { return Array.sum( values ) },
```

- Map tiene un retorno de tipo emit(key, value). El tipo de value (puede ser un valor simple o un nuevo documento) debe ser igual al retornado en la función Reduce.
- La documentación oficial recomienda preferir Aggregate debido a que **Aggregate tiene mejor desempeño que Map-Reduce.**

Ejemplo Map-Reduce

Cientes compran ítems por X precio

```
db.orders.insertMany([
{ _id: 1, cust_id: "Ant O. Knee", ord_date: new Date("2020-03-01"), price: 25, items: [ { sku: "oranges", qty: 5, price: 2.5 }, { sku: "apples", qty: 5, price: 2.5 } ] },
status: "A" },
{ _id: 2, cust_id: "Ant O. Knee", ord_date: new Date("2020-03-08"), price: 70, items: [ { sku: "oranges", qty: 8, price: 2.5 }, { sku: "chocolates", qty: 5, price: 10 } ] },
status: "A" },
{ _id: 3, cust_id: "Busby Bee", ord_date: new Date("2020-03-08"), price: 50, items: [ { sku: "oranges", qty: 10, price: 2.5 }, { sku: "pears", qty: 10, price: 2.5 } ] },
status: "A" },
{ _id: 4, cust_id: "Busby Bee", ord_date: new Date("2020-03-18"), price: 25, items: [ { sku: "oranges", qty: 10, price: 2.5 } ] }, status: "A" },
{ _id: 5, cust_id: "Busby Bee", ord_date: new Date("2020-03-19"), price: 50, items: [ { sku: "chocolates", qty: 5, price: 10 } ] }, status: "A"},
{ _id: 6, cust_id: "Cam Elot", ord_date: new Date("2020-03-19"), price: 35, items: [ { sku: "carrots", qty: 10, price: 1.0 }, { sku: "apples", qty: 10, price: 2.5 } ] },
status: "A" },
{ _id: 7, cust_id: "Cam Elot", ord_date: new Date("2020-03-20"), price: 25, items: [ { sku: "oranges", qty: 10, price: 2.5 } ] }, status: "A" },
{ _id: 8, cust_id: "Don Quis", ord_date: new Date("2020-03-20"), price: 75, items: [ { sku: "chocolates", qty: 5, price: 10 }, { sku: "apples", qty: 10, price: 2.5 } ] },
status: "A" },
{ _id: 9, cust_id: "Don Quis", ord_date: new Date("2020-03-20"), price: 55, items: [ { sku: "carrots", qty: 5, price: 1.0 }, { sku: "apples", qty: 10, price: 2.5 }, { sku:
"oranges", qty: 10, price: 2.5 } ] }, status: "A" },
{ _id: 10, cust_id: "Don Quis", ord_date: new Date("2020-03-23"), price: 25, items: [ { sku: "oranges", qty: 10, price: 2.5 } ] }, status: "A" }
])
```

Ejemplo Map-Reduce

```
var mapFunction1 = function() {  
  emit(this.cust_id, this.price);  
};
```

```
var reduceFunction1 = function(keyCustId,  
  valuesPrices) {  
  return Array.sum(valuesPrices);  
};
```

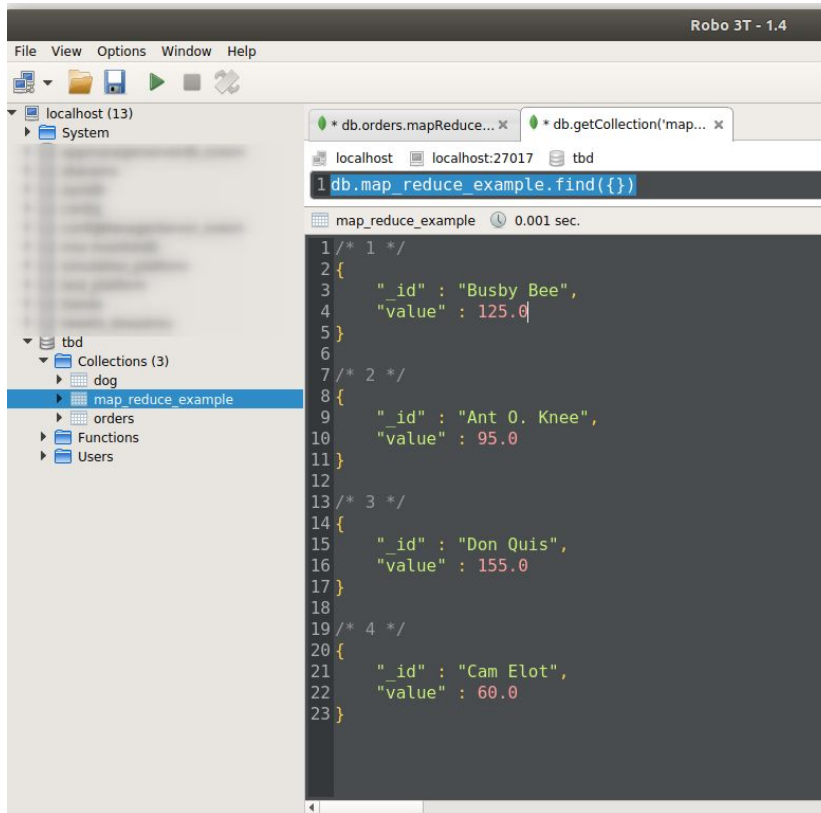
```
db.orders.mapReduce(  
  mapFunction1,  
  reduceFunction1,  
  { out: "map_reduce_example" }  
)
```



```
db.orders.mapReduce(  
  function() {  
    emit(this.cust_id, this.price);  
  },  
  function(keyCustId, valuesPrices) {  
    return Array.sum(valuesPrices);  
  },  
  { out: "map_reduce_example" }  
)
```

Ejemplo Map-Reduce

```
db.map_reduce_example.find({})
```



The screenshot shows the Robo 3T 1.4 application interface. On the left, a tree view shows the database structure: 'localhost (13)' contains 'System' and 'tbd'. 'tbd' contains 'Collections (3)' (dog, map_reduce_example, orders), 'Functions', and 'Users'. The 'map_reduce_example' collection is selected. The main window shows a query: `1 db.map_reduce_example.find({})`. Below the query, the results are displayed in a table format with 23 rows. The first row is a comment `1 /* 1 */`. The next three rows are a JSON document: `2 {`, `3 "_id" : "Busby Bee",`, `4 "value" : 125.0`, `5 }`. This is followed by another comment `6 1 /* 2 */` and a second JSON document: `7 {`, `8 "_id" : "Ant O. Knee",`, `9 "value" : 95.0`, `10 }`. Then a comment `11 1 /* 3 */` and a third JSON document: `12 {`, `13 "_id" : "Don Quis",`, `14 "value" : 155.0`, `15 }`. Finally, a comment `16 1 /* 4 */` and a fourth JSON document: `17 {`, `18 "_id" : "Cam Elot",`, `19 "value" : 60.0`, `20 }`. The results end with `21` and `22`.

id	value
Busby Bee	125.0
Ant O. Knee	95.0
Don Quis	155.0
Cam Elot	60.0

Modelamiento de relaciones

Modelamiento de relaciones

Es posible usar la estructura de documentos de MongoDB para generar distintos tipos de relaciones.

Por ejemplo **documentos incrustados** (address), o **arreglos** (hobbies)

```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  },
  "hobbies": ["surfing", "coding"]
}
```

Relaciones uno-a-uno

Considerando el tamaño de documento resultante, se puede incrustar un documento dentro de otro. Similar al ejemplo anterior, podemos agregar la dirección dentro del documento **dog**.

```
{
  "_id" : ObjectId("5f2f3dfafc817401512987e5"),
  "name" : "Moneda",
  "age" : 10,
  "color" : "blanco",
  "domicilio":{
    "calle":"Avenida Siempreviva 123",
    "ciudad":"Springfield"
  }
}
```

Relaciones uno-a-uno

Para buscar por atributos dentro de un objeto incrustado se debe usar el operador punto:

```
db.collection.find({"subdoc.att": "Value"});
```

Ejemplo:

```
db.dog.find({"domicilio.ciudad": "Springfield"});
```

Relaciones uno-a-uno

Consideraciones

- Se debe tomar en cuenta el tamaño del documento resultante de la colección en caso de incrustar uno o más documentos.
- El peso máximo de un documento es 16mb
- Si realmente es necesario tener una relación uno-a-uno con documentos de **tamaño considerable, se puede modelar mediante referencias.**

Relaciones uno-a-muchos

En casos simples de relaciones uno-a-muchos se recomienda usar **arreglos incrustados** en el documento principal.

```
{  
  "_id" : //..  
  "name" : "Moneda",  
  "age" : NumberInt(10),  
  "color" : "blanco",  
  "juguetes" : [  
    "pelota",  
    "peluche"  
  ]  
}
```

Relaciones uno-a-muchos

Consultar dentro de un arreglo.

Que contengan exactamente ese arreglo:

```
db.dog.find({"juguetes":["pelota", "peluche"]});
```

Que contengan los elementos del arreglo:

```
db.dog.find({"juguetes":{"$all: ["pelota", "peluche"] }});
```

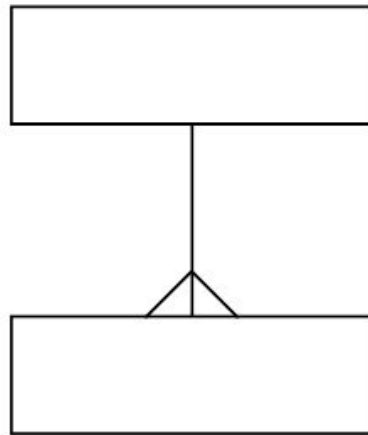
Que contengan el elemento:

```
db.dog.find({"juguetes":"pelota"});
```

Relaciones uno-a-muchos

Las relaciones uno-a-muchos también pueden hacerse por referencia.

Dependiendo de la cantidad estimada de documentos en la relación, las referencias pueden estar en el “**uno**” o en el “**muchos**” de la relación.



Referencias en “uno”

Si estimamos una **cantidad limitada** de documentos en la relación, podemos incluir las referencias en el “uno” de la relación.

Por ejemplo, definimos la colección owner con un arreglo de perros.

```
{
  "_id" : ObjectId("5f3ee01b9c993e7c1193e001"),
  "name" : "Mario Hugo",
  "dogs": ["Turrón", "Moneda"]
}
```

Referencias en “muchos”

Para casos donde la cantidad de elementos puede crecer de forma indefinida, resulta más apropiado guardar la referencia en el “muchos” de la relación.

```
{
  "_id" : ObjectId("5f2f3dfafc817401512987e5"),
  "name" : "Moneda",
  "age" :10,
  "color" : "blanco",
  "domicilio" : {
    "calle" : "Avenida Siempreviva 123",
    "ciudad" : "Springfield"
  },
  "juguetes" : [ "pelota","peluche"],
  "owner" : "Mario Hugo"
}
```



Patrón subset

- La documentación oficial de Mongo DB recomienda para ciertos casos el patrón subset, que consiste **en incrustar solamente los documentos necesarios y mantener la totalidad en una colección aparte.**
- Este patrón permite mejor desempeño en las consultas pero genera duplicidad de datos, lo cual debe ser manejado por la lógica de la aplicación.

<https://docs.mongodb.com/manual/tutorial/model-embedded-one-to-many-relationships-between-documents/#subset-pattern>

Consultas por referencia

Si quiero traer todos los documentos de una relación uno-a-muchos incrustados en el documento de origen tenemos que usar funciones de agregación (**aggregate**) con **\$lookup**.

Consultas usando \$lookup

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

El formato de es el siguiente.

- ▣ **from:** colección a la que se hace *join*
- ▣ **localField:** campo del documento de entrada.
- ▣ **foreignField:** campo del documento referenciado.
- ▣ **as:** nombre del campo generado.

\$lookup - Primer ejemplo

Teniendo la colección **dog**

```
{
  "name" : "Moneda",
  "age" :10,
  "color" : "blanco",
  //...
  "owner": "Mario Hugo"
}
```

Y la colección **owner**

```
{
  //...
  "name" : "Mario Hugo"
}
```

\$lookup - Primer ejemplo

```
db.owner.aggregate([
  {
    $lookup:{
      from:"dog",
      localField:"name",
      foreignField:"owner",
      as:"dogs"
    }
  }
]);
```

Desde la tabla **owner** hacemos “join” con la colección **dog**, donde “dog.owner” sea igual a “owner.name”.

\$lookup - Segundo ejemplo

Si tenemos las referencias
en el “uno” de la relación

```
{
  "_id" : ObjectId("5f3ee01b9c993e7c1193e001"),
  "name" : "Mario Hugo",
  "dogs": ["Turrón", "Moneda"]
}
```

\$lookup - Segundo ejemplo

Al tener las referencias dentro de un arreglo debemos **expandir este arreglo** con [\\$unwind](#). El paso siguiente en el canal de procesamiento es un \$lookup similar al anterior.

```
db.owner.aggregate([
  { $unwind: "$dogs" },
  {
    $lookup: {
      from: "dog",
      localField: "dogs",
      foreignField: "name",
      as: "dogDetail"
    }
  }
]);
```

Uso de driver Java

Uso de driver Java



En el repositorio de códigos del curso se agrega un ejemplo usando el driver oficial.

<https://github.com/citiaps/codigos-tbd/tree/master/ejemplo-mongo-avanzado>

Este ejemplo utiliza POJO para mapear documentos de la base de datos. También se puede utilizar la clase `org.bson.Document`.

<https://mongodb.github.io/mongo-java-driver/3.10/driver/getting-started/quick-start/>

Uso de driver Java

Para operaciones complejas como **aggregate**, se pueden utilizar herramientas como MongoDB Compass que ayudan tanto a generar la consulta como a convertirla al lenguaje necesario.

<https://www.mongodb.com/products/compass>

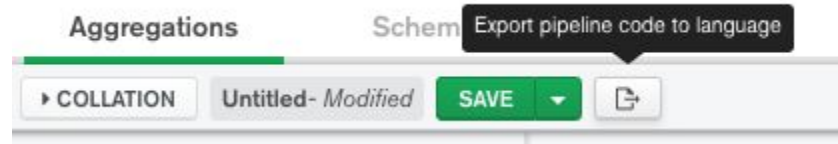
MongoDB Compass

The easiest way to explore and manipulate your MongoDB data

Try it now

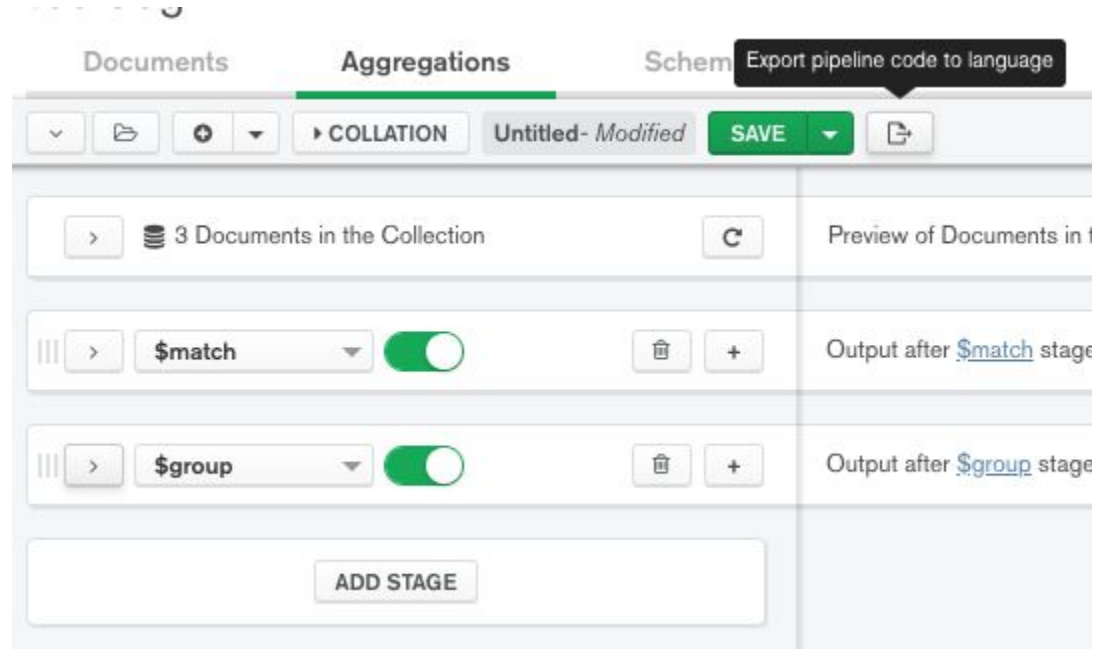
Convirtiendo aggregate a Java

Dentro de Compass, abrir colección y seleccionar aggregate.



Convirtiendo aggregate a Java

Se agregan las etapas necesarias y se hace click en en **Export pipeline to language**



Convirtiendo aggregate a Java

Export Pipeline To Language

My Pipeline:

Export Pipeline To:

JAVA

```
1 [{
2   $match: {
3     "owner": "Mario Hugo"
4   }
5 }, {
6   $group: {
7     _id: "$_id",
8     juguetes_count: {
9       $first: {
10        $size: "$juguetes"
11      }
12    }
13  }
14 }]
```

```
14 import org.bson.Document;
15
16 requires the MongoDB Java Driver.
17 https://mongodb.github.io/mongo-java-driver
18
19
20
21 MongoClient mongoClient = new MongoClient(
22   new MongoClientURI(
23     "mongodb://localhost:27017/?readPreference=pr
24   )
25
26   mongoDatabase database = mongoClient.getDatabase("tbd
27   mongoCollection<Document> collection = database.getCo
28
29   Iterable<Document> result = collection.aggregate()
```

☒ Include Import Statements

☒ Include Driver Syntax