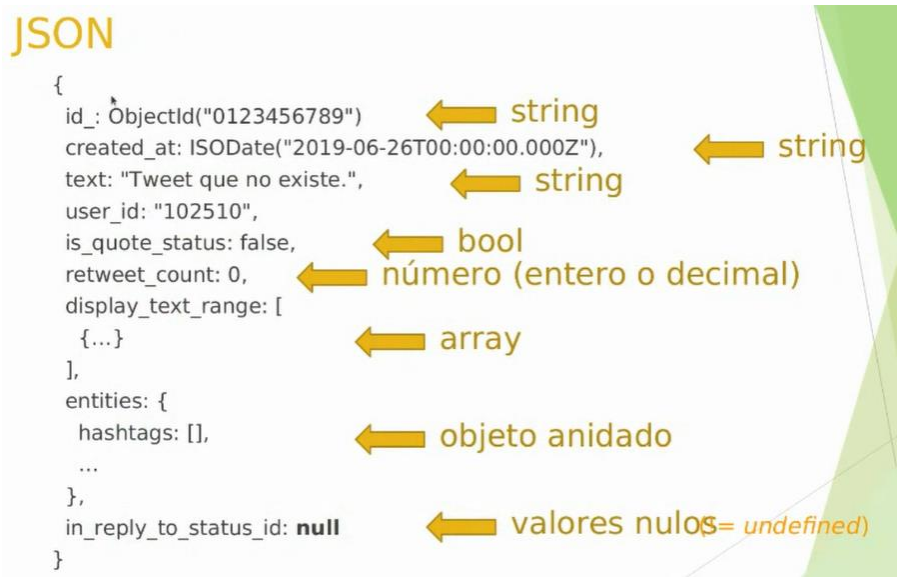
MongoDB: BDD Orientada a Documentos

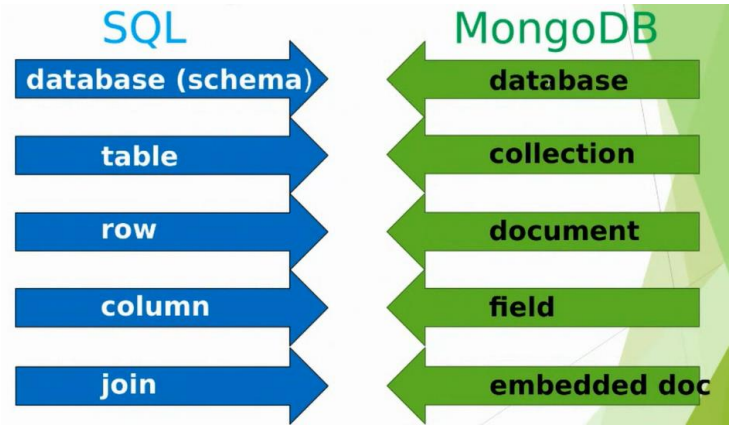
No vamos a trabajr con tablas como en SQL, sino colecciones de documentos. Un documento esta formado por varias entradas de tipo clave valor, como un JSON por ejemplo:



- El id es una especie de primary key para identificar a cada documento
- Los objetos anidados son las ***entidades***

Aclaración: Mongo no guarda exactamente JSONs sino BSONS (Binary JSON), por el tipo de almacenamiento. EN BSON los Nulls se representan con no existir

Comparativa de como se llama en una bdd relacional vs en MongoDB



En vez de usar joins, como mongodb no tiene una forma sencilla de juntar los datos, acá vamos a tener documento embebidos.

A diferencia de SQL, no tenemos que crear ninguna tabla. Simplemente decimos que queremos utilizar una colección en particular y accedemos a ella directamente.

- db.createCollection(“conductores”);
- db.createCollection(“autos”);

Cómo armamos documentos embebidos

En mongoDB NO existe el join. Lo que hacemos es poner un documento entero dentro de otro, en lugar de tener una referencia a otro documento.

Acá estoy guardando todo el documento de conductores en dueño

Consultas en Documentos

1. **Buscar documentos**

La palabra clave para buscar documentos en MongoDB es ***find.*** Se puede usar en conjunto con sort y limit para obtener una cantidad n de documentos nada más.

```
db.<collection>
  .find(<query>, <projection>)
  .sort(<order>)
  .limit(n)
```

Ejemplo: db.tweets.find().limit(5)

a. **Busqueda por igualdad**

Buscamos los documentos que tienen cierto campo igual al que indicamos. Podemos buscar de a más de un campo:

Ejemplos:

```
db.tweets.find(
  {_id: ObjectId("114367020929678540 8") }
)

db.user.find(
  {_id: "114367020929678540 8"},
  {_id: 0, text: 1}
)
```

a. Búsqueda por and y or

Otra cosa que podemos hacer es utilizar **AND y OR,** utilizando la notación $and: […], $or :[…]

```
db.tweets.find({ $or: [
    { user_id:“Juan” },
    { retweet_count: 2183 }
  ] })

  db.tweets.find(
    { user_id: “Juan”, retweet_count: 2183 })

db.tweets.find({ $and: [
    {user_id: “818839458”},
    {retweet_count: 2183}
  ] })
```

Acá vemos un ejemplo de or y dos notaciones posibles para el and.

b. Búsqueda por comparación: ==, >, >=, <, <=

Siguen la misma notación que el and y el or, con el $ y el [ ], teniendo las siguientes palabras claves:

- ==: eq
- >: gt
- >=: gte
- <: lt
- <=: lte

```
db.tweets.find(
  { retweet_count: { $gte: 2183 } })
```

Si queremos encontrar un documento donde cierto campo este entre dos valores, debemos combinar el <= y >= con un and.

Ejemplo:

```
collection.find(
    {
        "$and": [
            {"retweet_count": {"$gt": 1000}},
            {"retweet_count": {"$lt": 2000}},
            {"user.location": "Frias, Santiago del Estero"},
        ]
    },
```

c. Búsqueda de documentos embebidos

```
db.tweets.find(
  { retweet_count: { $gte: 2183 } })

db.tweets.find(
  {"user.name": "Juan"},
  {“name”: “$user.name”}
)


db.tweets.find(
  {"retweet_count": { $gt: “$fa} }
)
```

Para buscar en un campo de un documento embebido, debemos usar ña notación: “**doc.field”**

d. Búsqueda con regex

```
db.tweets.find( {“user.name”: /Juan/ } ) \

db.tweets.find( {“user.name”: { $regex: “Juan” } })

db.tweets.find( {“user.screen_name”: /juan/i } )
```

e. Búsqueda según in o nin (not in): palabra clave **$in** o **$nin**

```
db.tweets.find( {“user.name”:{$in:[“Juan”,“Laura”]} } )

db.tweets.find( {“user.name”: {$nin: [“Pablo”]} } )
```

f. Búsqueda por arrays

***Algún elemento en el array*:** palabra clave **$in**

```
db.tweets.find({ activities: "swimming" })

db.tweets.find({"entities.hashtags":
  { $elemMatch:
    {"text": {$in: ['futbol', 'copaamerica']}}
  }
})
```

***Todos los elementos del array*:** palabra clave **$all**

```
db.tweets.find(
  {activities: {$all: [“swimming“, “boxing”]}
} )
```

***Por tamaño de array*:** palabra clave **$size**

```
db.tweets.find( {activities: {$size: 3} } )
```

El find puede ser seguido por count, lo que colapsa la consulta en un valor numérico, por un limit, que trae una cantidad n de resultados, y también se pueden ordenar.

- *Count:* ***db-<collection>.find().count()***
- *Sort:* **db.<collection>.find().sort({fieldname: order})**

Ejemplos:

```
db.tweets.find().count()


db.tweets.find().sort( {created_at: 1} )

db.tweets.find().sort( {retweet_count: -1} )
```

2. Agregaciones de documentos

En MongoDB, se refiere a las agregaciones como un ***pipeline.*** Es como una tubería de procesamiento de datos en la que los documentos de una colección fluyen a través de una serie de etapas secuenciales

Cada pipeline de agregación consta de una o mas etapas. Cada etapa toma como entrada un conjunto de documentos y produce como salida otro conjunto de docs, que se pasa a la siguiente etapa en la pipeline.

Ejemplo de una pipeline:

Notación

```
db.collection.aggregate(
  [
    { $stage1 },
    { $stage2 },
    ...
    { $stageN },
  ]
)
```

agregation pipeline

Las etapas más comunes incluyen:

- **$match**: filtra los docs que coinciden con ciertos criterios.

Permite realizar una query al igual que find()

Notación: { $**match**: { <query> } }

Ejemplos:

```
db.tweets.aggregate( [ { $match: { “user.name”: “Juan” } } ] )

db.tweets.aggregate( [ { $match: {
  "entities.hashtags.text": {$eq: "futbol"}
}}])
```

- **$proyect**: redefine la estructura de los documentos, seleccionando campos específicos y/o añadiendo nuevos campos

Notación: { $**project**: { <field>: <**1, 0** or expression> … } }

Ejemplos:

```
db.tweets.aggregate( [ { $project: { user: 1 } } ] )
```

Acá estoy seleccionando el campo especifico user.

Y utilizando **$proyect** en conjunto con **$addFields** podemos crear campos nuevos

Notación:

{ $**project**: { <field>: <**1, 0** or expression> … } }

{ $**addFields**: { <field>: <**expression**> … } }

Ejemplos:

```
db.tweets.aggregate( [ { $project: { hashtag_count:
  {$size: "$entities.hashtags" }
}])

db.tweets.aggregate( [ { $addFields: { hashtag_count:
  {$size: "$entities.hashtags" }
}])
```

- **$limit** y **$skip**:limita la cantidad de docs que pasan a la siguiente etapa y skip omite un numero especificado de documentos

Ejemplo

```
db.tweets.aggregate(
  [
    { $limit: 1 }
  ]
)
```

```
db.tweets.aggregate(
  [
    { $skip: 32 }
  ]
)
```

- **$sort**:ordena los docs según un campo especificado

```
db.country.aggregate(
  [
    { $sort: { area: -1 } }
  ]
)
```

- **$group**: agrupa docs por un campo especificado, y puede realizar operaciones de agregación (suma, avg, etc.)

Notación:

{ $**group**: {

_id: <expression>,

<field>: <expression>

}}

Y se le pueden aplicar acumuladores: **avg, sum, max, min, first, last**

El _id de agrupación es **por que** voy a agrupar. Al colocar el $ lo que hace es usar ese campo, sino me lo toma como valor literal.

Es decir, si yo escribo _id: “user_name” me va a buscar un _id que sea user_name. En cambiom _id: “$user_name” agrupa por nombre de usuario.

Ejemplo:

```
db.country.aggregate( [ { $group: {
    _id: "$user.name",
    avg_retweeters: {
      $avg: "$retweet_count"
    }
  }])
```

- **$unwind**: “Deconstruye” una lista de elementos en sus elementos individuales

Notación:

db.country.aggregate([

{ **$unwind**: <field> }

])

- **$lookup**: simula operaciones de join con otras colecciones

3. ABM de documentos

3.1 Insertar documentos

Podemos insertar un elemento o muchos, con One o Many.

Notación:

db.<collection>.**insertOne**(<document>)

db.<collection>.**insertMany**([<document1>, <document2>, …])

Ejemplo:

```
db.user.insertOne(
  {user_id: “999999”, ...}
) -> ID
```

3.2 Actualizar documentos

La palabra clave para la actualización de documentos es Update. También podemos actualizar uno o muchos documentos a la vez.

Podemos agregar nuevos atributos (como si agregaramos una nueva columna en SQL) simplemente usando un **db.autos.update(…)**

En la sintaxis del update vamos a tener un filtro, entonces puede realizar la actualización sobre los cuales aplicamos el filtro

Es por ello, que para estas BDD, no necesariamente todos los objetos necesitan tener los mismos datos, porque algunos documentos van a poder tener un atributo q yo agregue, pero otros no.

Notación:

db.<collection>.**updateOne**(<filter>, <update>, {upsert: <boolean>})

db.<collection>.**updateMany**(<filter>, <update>, {upsert: <boolean>})

Ejemplo:

```
db.user.updateOne(
  {“user.name”: /Juan/i}, <documento_completo>
)
```

Palabras claves

- **$set y $unset**: Agrega o elimina un campo, sin afectar los otros del documento

Ejemplos:

```
db.tweets.updateOne( {id_: ID}, {$set: {is_fake: true}} )

db.tweets.updateMany(
  {is_fake: true},
  {
    $set: {is_real: false},
    $unset: {is_fake: 1}
  }
)
```

- **$inc**: modifica un campo numérico en la cantidad deseada

Ejemplos:

```
db.tweets.updateOne( {id_: ID}, {$inc: {retweet_count: 10}} )
```

3.3 Eliminar documentos

La palabra clave es **delete**, y también puedo eliminar uno o muchos documentos, con deleteOne o deleteMany.

Notacion:

db.<collection>.**deleteOne**(<filter>)

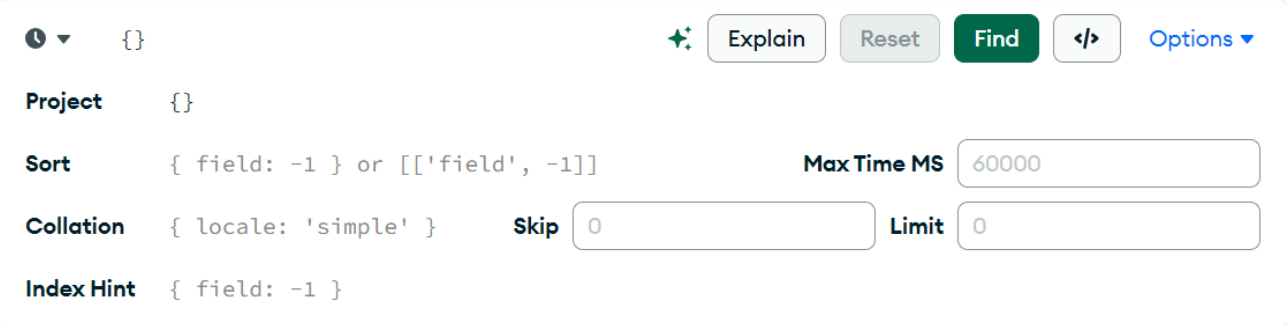
db.<collection>.**deleteMany**(<filter>)

Ejemplo:

```
db.tweets.deleteOne(
  {is_real: false}
)
```

Ejemplos de queries en MongoDB Compass

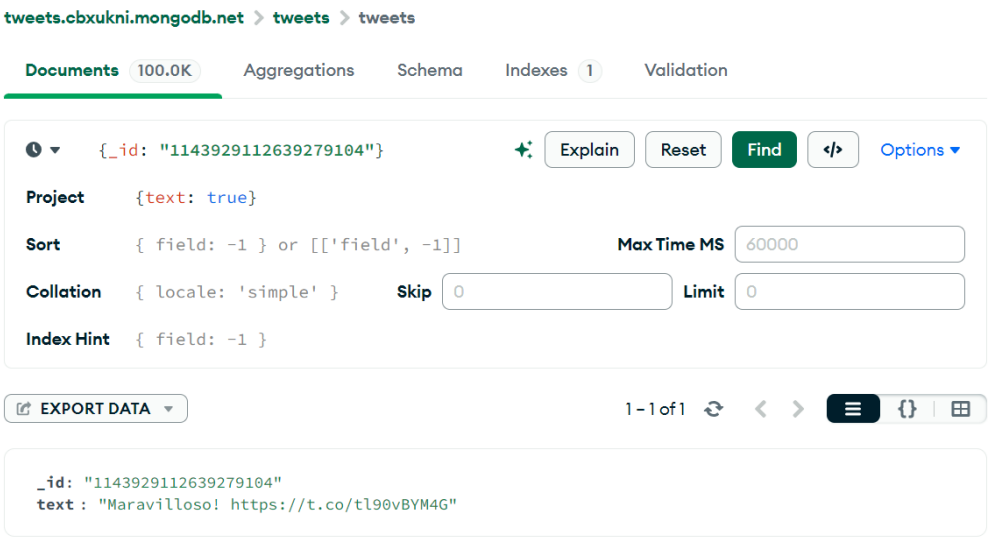
En la IDE que vamos a usar tenemos el siguiente formato:



El primer set de { } es el campo por el que queremos filtrar. Por ejemplo, podemos buscar un documento por su _id.

En Proyect es un **idem del Select en SQL,** donde digo con que campos del docs me quiero quedar cuando me devuelva los resultados

Ejemplo:



A menos que se indique lo contrario, el _id me lo devuelve siempre. Para que no aparezca, el Project hay que poner {_id: 0}

Ejercicios del Taller

1. Hallar los tweets del usuario con user id '818839458'.

```
db.tweets.find(
  {user_id: "818839458"}
)
```

2. Hallar aquellos tweets que tengan más de 500000 retweets.

```
db.tweets.find(
  {retweet_count: {$gte: 500000} }
)
```

Lo que ponemos en la IDE es lo siguiente:

```
{retweet_count: {$gte: 500000}}
```

3. Mostrar la cantidad de retweets de los tweets que se hayan hecho desde Argentina o Brasil.

```
collection.find(
  {$or:[
    {"place.country": "Argentina"},
    {"place.country": "Brasil"}
  ]},
  {"retweet_count": 1}
)
```

4. Hallar los usuarios que tengan tweets con 200000 o más retweets y sean en idioma espanol.

```
collection.find(
  {$and:[
    {"retweet_count": {$gte : 200000}},
    {"lang": "es"}
  ]},
  {"user": 1}
)
```

Formato en la IDE:

```
{retweet_count: {$gte : 200000},
lang: "es"}

Project    {"user": 1, "_id": 0}
```

5. Mostrar la cantidad de retweets para los tweets que no se hayan hecho en Argentina ni Brasil, pero si tengan un lugar definido y sean en español.

```
collection.find(
  {$and:[
    {"place.country": {$nin: ["Argentina", "Brasil"]}},
    {"place.country": {$ne: null}},
    {"lang": "es"}
  ]},
  {retweet_count: 1}
)
```

También puedo hacer directamente:

```
collection.find(
  {$and: [
    {"place.country": {$nin: ["Argentina", "Brasil", null]}},
    {"lang": "es"}
  ]},
  {retweet_count: 1}
)
```

6. Mostrar los screen name de aquellos usuarios que tengan “Juan” como parte de su nombre.

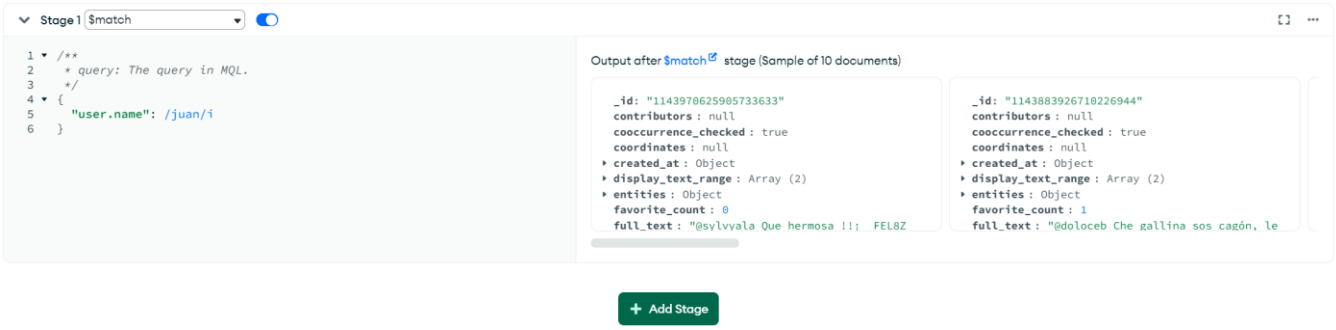
```
collection.find(
  {"user.name": {$regex: "Juan"}},
  {"user.screen_name": 1}
)
```

7. Mostrar de los 10 tweets con más retweets, su usuario y la cantidad de retweets.

```
collection.find(
  {},
  {user: 1, retweet_count: 1}
).sort(retweet_count: -1)
 .limit(10)
```

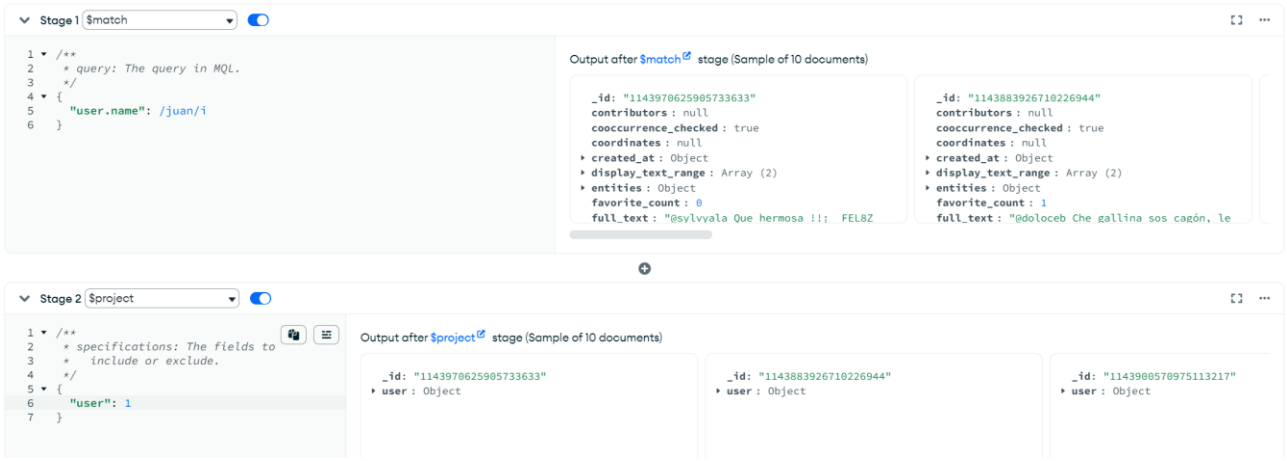
Consultas de Agregación

En la pestana de Aggregations vamos a ir agregando stages, cada uno con un Stage de aggregation



Y esto me va a ir devolviendo algunos documentos

Puedo agregar un 2do Stage de proyección por ejemplo:



Lo mas interesante que podemos hacer los las Aggregations es ***agrupar con $group***

1. Mostrar de los 10 tweets con más retweets, su usuario y la cantidad de retweets. Ordenar la salida de forma ascendente.

```
collection.aggregate([
  {
    $project: {
      user: 1,
      retweet_count: 1
    }
  },
  {
    $sort: { retweet_count: -1 }
  },
  {$limit: 10}
])
```

2. Encontrar los 10 hashtags más usados.

```
collection.aggregate([
  {
    $unwind:
      {
        path: "$entities.hashtags",
        preserveNullAndEmptyArrays: false
      }
  },
  {
    $group:
      {
        _id: "$entities.hashtags.text",
        cantUsos: {
          $count: {}
        }
      }
  },
  {
    $sort:
      {
        cantUsos: -1
      }
  },
  {
    $limit:
      10
  }
])
```

3. Por cada usuario obtener una lista de ids de tweets y el largo de la misma.

```
collection.aggregate([
  {
    $group: {
      _id: "$user",
      lista_de_tweets: {
        $push: "$_id"
      },
      largo: {
        $count: {}
      }
    }
  }
])
```

El operador push es como un append, lo que hace es armarme una lista.

4. Hallar para cada intervalo de una hora cuantos tweets realizo cada usuario.

```
collection.aggregate([
  {
    $match:
      //extraigo la hora de la fecha
      {
        created_at: {
          $gt: ISODate("2019-06-26T01:00:00.000")
        }
      }
  },
  {
    $project: //me quedo con el usuario y la hora del tweet
      {
        user: 1,
        hora: {
          $hour: "$created_at"
        }
      }
  },
  {
    $group: //agrupo por usuario y por hora. Hago un count y
      //me queda la cant de tweets por hora por usuario
      {
        _id: {
          user_name: "$user_name",
          hora: "$hora"
        },
        cantidad: {
          $count: {}
        }
      }
  }
])
```

5. Hallar la máxima cantidad de retweets totales que tuvo algún usuario.

```
collection.aggregate([
  {
    $group: {
      _id: "$user",
      cantidad_retweets: {
        $sum: "$retweet_count"
      }
    }
  },
  {
    $sort:
      {
        cantidad_retweets: -1
      }
  },
  {
    $limit:
      //dice algun usuario, me quedo con 1
      1
  },
  {
    $project: {
      cantidad_retweets: 1
    }
  }
])
```

6. Encontrar a los 5 usuarios más mencionados. (les hicieron @)

7. Hallar la cantidad de retweets promedio para los tweets que se hayan hecho desde Argentina y aquellos que no.