RESUMEN PRACTICAS

MONGO DB

Trabajamos con Binary JSON (BSON). La diferencia es que tienen tipos de datos predefinidos y son menos que la cantidad que acepta JSON.

NO hay joins
JOIN -> EMBEDDED DOCUMENT

A diferencia de SQL, para crear dos "tablas" (aqui llamadas COLECCIONES) simplemente alcanza con:

```
db.createCollection("conductores")
db.createCollection("autos")
```

Con estas instrucciones creo las colecciones de "autos" y "conductores". Es mas simple que SQL ya que no tengo que indicar ni keys, ni atributos, etc.. (schema-less)

> **ES FUNDAMENTAL EL CONCEPTO DE DOCUMENTOS EMBEBIDOS**

UN **SHARD** es un pedazo de la DB donde distribuyo los datos. Cada SHARD tiene un nodo primario y varios secundarios en el caso que se caiga el primero.

CONSULTAS BASICAS:

```
db.<collection>.find(<query>, <proejection>)
.sort(<order>)
.limit(N)


//Esto es como un SELECT * FROM tweets
db.tweets.find()

//Limito a 5 resultados
db.tweets.find().limit(5)

//Busco con una condicion (en este caso, por id)
db.tweets.find({"_id" : "11123464awefeaw"})

//Busco el mismo documento con ese _id, pero no quiero ver el _id en el
resultado. Uso la
//proyeccion. Ademas le indico que si quiero ver el campo "user"
db.tweets.find({"_id" : "11123464awefeaw"}, {"_id" : 0, "user" : 1})

//Con el '0' indico que no quiero un campo, con el '1' le digo que si.

//Con la proyeccion tambien puedo renombrar campos
db.tweets.find({"_id" : "11123464awefeaw"}, {"user_first_name" :
```

```
"$user.name"})
```

BUSQUEDA CONDICIONAL

```
//OR
db.tweets.find({ $or: [
    {"user_id : "Juan"},
    { retweet_count : 2183}
]})


//AND
db.tweets.find({ $and: [
    {"user_id : "19872394"},
    { retweet_count : 2183}
]})


//Para el AND tengo la alternativa de escribir directamente las condiciones
db.tweets.find({ user_id : "Juan", retweet_count : 2183})


//eq - gt - gte - lt - lte
db.tweets.find({ retweet_count : { $gte : 2183}})

//in, nin (not in) para busqueda en un conjunto de valores
db.tweets.find( {"user.name": {$in:["Juan","Laura"]} } )

db.tweets.find( {"user.name": {$nin:["Juan"]} } )
```

BUSQUEDA DOCUMENTOS

```
//Busca en un array de "hastags" todos aquellos campos que tengan como key
"text" y que contengan alguno de los //elementos del conjunto indicado

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


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

```
// tamaño del array
db.tweets.find( {activities: {$size: 3} } )
```

CONTAR

```
db.<collection>.find().count()
```

ORDENAR

```
db.<collection>.find().sort( {fieldName : order} )

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

//1 -> ascendente
//-1 -> descendente
```

AGREGACION

```
//$match: Permite realizar una query igual que el find()
db.tweets.aggregate( [ { $match: { “user.name”: “Juan” } } ] )

//$project: Permite elegir los campos al igual que la proyeccion del find
db.tweets.aggregate( [ { $project: { user: 1 } } ] )

//$group: Agrupa los documentos y establece una nueva clave
//Acumuladores
db.country.aggregate( [ { $group: {
  _id: "$user.name",
  avg_retweeters: {
    $avg: "$retweet_count"
        }
    }
}
])

{
  $group: {
    _id: <expression>,
    <field1>: { <accumulator1> : <expression1> },
    <field2>: { <accumulator2> : <expression2> },
    ...
  }
}
```

```
//$unwind: desconstruye un arreglo de elemntos. Agrega un id nuevo que
indica el indice del objeto original y los demas elementos del documento
los repite por cada elemento desanidado
```

TALLER MONGODB

1. Hallar los tweets del usuario con userid ‘818839458’.

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

2. Hallar aquellos tweets que tengan m´as de 500000 retweets.

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

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

```
db.tweets.find( {"place.country" : {$in : ["Argentina", "Brasil"] } },
{"retweet_count" : 1} )
```

4. Hallar los usuarios que tengan tweets con 200000 o m´as retweets y sean en idioma espa˜nol.

```
db.tweets.find({"retweet_count" : { $gte : 200000 }, "lang" : "es" })
```

5. Mostrar la cantidad de retweets para los tweets que no se hayan hecho en Argentina ni Brasil, pero s´ı tengan un lugar definido y sean en espa˜nol.

```
db.tweets.find({"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.

```
db.tweets.find( {"user.name": /Juan/ }, {"user.screen_name" : 1} )
```

7. Mostrar de los 10 tweets con m´as retweets, su usuario y la cantidad de retweets.

```

```

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

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

```
db.tweets.aggregate( [
    {
        $sort : {retweet_count : -1}
    },
    {
        $limit : 10
    },
    {
        $project : {user_id : 1, retweet_count : 1}
    },
    {
        $sort : {retweet_count : 1}
    }
])
```

2. Encontrar los 10 hashtags mas usados.

```
db.tweets.aggregate( [
    {
        $unwind : {path : "$entities.hastags"}
    },
    {
        $limit : 10
    }
])


db.tweets.aggregate([
  {
    $unwind:
      {
        path: "$entities.hashtags",
      },
  },
  {
    $group:
      {
        _id: "$entities.hashtags.text",
        counts: {
          $sum: 1,
        },
```

```
      },
  },
  {
    $sort:
      {
        counts: -1,
      },
  },
  {
    $limit:
      10,
  },
]
)
```

3. Encontrar a los 5 usuarios m´as mencionados. (les hicieron @)

```
[
  {
    $unwind: {
      path: "$entities.user_mentions",
      preserveNullAndEmptyArrays: false,
    },
  },
  {
    $group: {
      _id: "$entities.user_mentions.id",
      cantidad_menciones: {
        $count: {},
      },
    },
  },
  {
    $sort: {
      cantidad_menciones: -1,
    },
  },
  {
    $limit:
      5,
  },
]
```

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

```
db.tweets.aggregate([
    {
        $match : {"place.country" : "Argentina"}
    },
    {
        $group : {
        _id : "place.country",
        "avg_retweets" : {$avg : "$retweet_count"}
        },
    }
])


db.tweets.aggregate([
  {
    $group: {
      _id: {
        $cond: [
          {
            $eq: ["$place.country", "Argentina"],
          },
          "Argentina",
          "Otros",
        ],
      },
      promedio_retweet: {
        $avg: "$retweet_count",
      },
    },
  },
]
)
```

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

```
db.tweets.aggregate([
    {
        $group: {
            _id : "$user_id",
            nombre_usuario: {
                $first: "$user.name"
            },
            tweets:{
                $push : "$_id"
            },

        }

    },
    {
```

```
        $addFields: {
            cantidad : {
                $size : "$tweets"
            }
        }
    }

]
)
```

6. Hallar la m´axima cantidad de retweets totales que tuvo alg´un usuario.

```
db.tweets.aggregate([
    {
        $group : {
            _id : "$user_id",
            cantidad_retweets: {
                $max : "$retweet_count"
            }
        }
    }
]
)

[
    { $group: {
        _id: "$user.name",
        retweets_count: {$sum: "$retweet_count"}
    }},
    { $group: {
        _id: null,
        retweets_count: {$max: "$retweets_count"}
    }}
]
```

7. Hallar para cada intervalo de una hora cu´antos tweets realiz´o cada usuario.

```

```

ABM

Recordar el uso de _id. No puedo insertar un documento con un _id existentes

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

```
)


//Puedo incrementar campos numericos
db.tweets.updateOne(
    {id_: ID},
    {$inc: {retweet_count: 10}}
)
```

Neo4J

Base de grafos nativa. Almacenamiento modo grafo. Modelo de propiedades: los elementos tienen propiedades

Modelo de grafo etiqueta-propiedad: almacenados como clave-valor. Las relaciones tambien pueden tener propiedades y ademas, en el concepto de grafo, pueden ser direccionales o unidireccionales.

```
MATCH (p:Persona {nombre: "Daniel"}) - [:CASADO_CON] -> (esposa)

--es un select en SQL. Busca las personas que estan casads con Daniel
```

Puedo implementar varios algoritmos de grafos para realizar analisis en este tipo de DB.

```
MATCH (n:Person) WHERE n.surname = 'Smith' RETURN n.name ORDER BY n.name
DESC LIMIT 10
--Muestre en orden alfabético, los nombres de las primeras 10 personas
apellidadas ‘Smith’
```

```
--Muestre la marca y modelos de los vehículos de año 2013.
MATCH (v:Vehicle) WHERE v.year = 2013 RETURN v.make, v.model
```

```
--Muestre el nombre, apellido y rango de los oficiales cuyos apellidos
comiencen con ’Mc’, ordenados por rango (rank).

MATCH (o:Officer) WHERE o.surname START WITH 'Mc' RETURN o.name, o.surname,
o.rank  ORDER BY o.rank
```

Puedo hacer querys tambien acerca de los arcos

```
//Crimenes relacionados con el oficial McHan

MATCH (n:Officer) -- (m:Crime) WHERE n.surname = 'McHan' RETURN n,m
```

Podemos filtrar arcos por condiciones

```
--Personas que viven con Craig Gordon

MATCH (n:Person)-[d:KNOWS_LW]->(m:Person)
WHERE m.name = 'Craig' and m.surname = 'Gordon'
return n
```

Puedo filtrar tambien por los atributos de la relacion

```
--Obtener la informacion de las personas que sean hermanos

MATCH (a:Person)-[r:FAMILY_REL{rel_type:'SIBLING'}]-(b:Person)
RETURN a,b
```

Tambien puedo consultar por la cantidad de arcos que hay de distancia

```
-[*]-   -[*cant]-   -[*min..]-  -[*max..]-  -[*min..max]-
```

```
--Muestre el grafo de las locations en el área M30. Cuantos nodos hay?

MATCH (l:Location)-[:LOCATION_IN_AREA]-(a:Area)
WHERE a.areaCode = 'M30'
RETURN count(l)
```

```
--Muestre las personas que están a distancia 3 de Gordon Craig.
MATCH (p:Person) - [:KNOWS*3] -> (gc:Person) WHERE gc.name = 'Craig' and
gc.surname='Gordon' RETURN p
```

Se pueden definir alias de subgrafos para trabajar mas facil

```
MATCH s=((m:Person)-[HAS_PHONE]->(p:Phone)<-[]-(o:PhoneCall))
WHERE m.name = 'Craig' and m.surname = 'Gordon'
RETURN s
```

Acepta funciones Length(s), Relationships(s), Nodes(s)

```
MATCH (n:Person{surname:"Gordon", name:"Craig"})--(p:Person)
WHERE NOT (p)--(:Person{surname:"Robinson",name:"Bonnie"})
RETURN DISTINCT p.name, p.surname
```

Puedo usar la funcion shortestPath(a-[*]-b)

```
MATCH s = shortestPath((a)-[*]-(b))
RETURN s
```

```
--Muestre las personas conocidas de Roger Brooks que no participaron en
ningún crimen.

MATCH (rb:Person)-[:KWOWS]->(p:Person)
WHERE rb.name = 'Roger' and rb.surname = 'Brooks'
and NOT EXISTS ((p)-[:INVOLVED_IN]->(:Crime))
RETURN p

MATCH (roger:Person {name: 'Roger', surname: 'Brooks'})-[:KNOWS]->
(known:Person)
WHERE NOT EXISTS    ((known)-[:INVOLVED_IN]->(:Crime))
RETURN known
```

```
//Muestre el camino más corto de Judith Moore a Richard Green.
MATCH s=shortestPath((a:Person)-[*]-(b:Person))
WHERE a.name = 'Judith' and a.surname='Moore' and b.name='Richard' and
b.surname='Green'
return s
```

```
//Encuentre los oficiales que investigaron los crímenes cometidos en 165
Laurel Street.
MATCH (o:Officer)<-[:INVESTIGATED_BY]-(c:Crime)-[:OCCURRED_AT]->
(l:Location)
WHERE l.address = '165 Laurel Street'
RETURN o
```

```
//Obtenga el modelo, marca y año del auto más viejo de la base.

MATCH (v:Vehicle)
WITH min(v.year) as oldest
MATCH (v2:Vehicle)
WHERE v2.year = oldest
return v2
```

```
//¿A qué distancia se encuentra el auto más viejo de Roger Brooks?

MATCH (v:Vehicle)
WITH min(v.year) as oldest
MATCH d=shortestPath((p:Person)-[*]-(v2:Vehicle))
WHERE v2.year = oldest and p.name = 'Roger' and p.surname = 'Brooks'
RETURN d
```

```
//Devuelva el nombre y apellido de personas que conozcan más de 10
personas.
MATCH s=((p:Person)-[:KNOWS]-(p2:Person))
WITH p, count(p2) as conocidos
WHERE conocidos > 10
return p.name, p.surname
```

Para los 10 marcas de autos con más unidades mostrar el promedio de año y cantidad.
MATCH (a:Vehicle)
RETURN a.make, AVG(toInteger(a.year)), COUNT(a)
ORDER BY COUNT(a) DESC
LIMIT 10