Parte 1

Si bien las BBDD NoSQL tienen diferencias fundamentales con los sistemas de BBDD Relacionales o RDBMS, algunos conceptos comunes se pueden relacionar. Responda las siguientes preguntas, considerando MongoDB en particular como Base de Datos NoSQL.

1. ¿Cuáles de los siguientes conceptos de RDBMS existen en MongoDB? En caso de no existir, ¿hay alguna alternativa? ¿Cuál es?

- Base de Datos

  Si, existe. Solo que en MongoDB la base de datos está orientada a documentos.

- Tabla / Relación

  No, no existe. En lugar de guardar los datos en tablas lo hace en colecciones de documentos que no necesariamente deben tener la misma estructura.

- Fila / Tupla

  No aplica el concepto, ya que ahora la información se organiza en documentos (formato BSON).

- Columna

  Podemos entender a las columnas cómo claves del diccionario, pero se debe tener en cuenta que deja de tener una estructura estricta cómo tal.

2. MongoDB tiene soporte para transacciones, pero no es igual que el de los RDBMS. ¿Cuál es el alcance de una transacción en MongoDB?

El alcance de una transacción es hacia un documento en particular, es decir, cada vez que se modifique ese documento, se persiste en la BD, independientemente de los demás de la colección.

3. Para acelerar las consultas, MongoDB tiene soporte para índices. ¿Qué tipos de índices soporta?

Los índices nos permiten acceder a la información que necesitamos de forma directa, sin tener que recorrer toda la estructura consultado por una condición.

MongoDB nos proporciona varios tipos de índices:

- Single Field: además del por defecto que genera Mongo (_id), te permite generar índices de lo que quieras cómo indice por dni, alias, etc.
- Compound Index: o en español, indices compuestos. Permite ordenar los datos con varios sentidos, por ejemplo, con el índice { userid: 1, score: -1 } vemos cómo se ordenaría primero por userid y segundo por score.
- Multikey Index: permiten almacenar múltiples índices en la misma estructura. Al momento de realizar una consulta, se buscará aquellos que tengan incluido el índice buscado en el array de índices.
- Geospatial: permite guardar cómo indice todo valor soportado por GeoJson, el cual nos da herramientas para guardar valores de figuras geométricas, pares de valores X,Y, entre otros.
- Text Indexes: guarda cómo índice un String y realiza querys con palabras que lo contengan.
- Hashed Indexes: ordena los documentos mediante una función de hash, lo cual lo hace mucho más eficiente.

4. ¿Existen claves foráneas en MongoDB?

Si bien no existen las claves foráneas como tales, el concepto puede ser aplicado dentro del conjunto de herramientas que provee mongo, llamado referencia.
Para relacionar documentos se pueden usar las referencias de dos modos.

1. Referencias manuales. Básicamente se guarda un campo con el _id del documento a referenciar.
2. DBRefs. Se guarda el _id del documento, el nombre de la colección y opcionalmente el nombre de la bd.

En ambos casos se necesitan queries adicionales para recuperar la totalidad de la información.

Parte 2

▸ Descargue la última versión de MongoDB desde el sitio oficial. Ingrese al cliente de línea de comando para realizar los siguientes ejercicios.

5. Cree una nueva base de datos llamada ecommerce, y una colección llamada products. En esa colección inserte un nuevo documento (un producto) con los siguientes atributos:

{name:'Caldera Caldaia Duo', price:140000} recupere la información del producto usando el comando db.products.find() (puede agregar la función .pretty() al final de la expresión para ver los datos indentados). Notará que no se encuentran exactamente los atributos que insertó. ¿Cuál es la diferencia?

```
> use ecommerce
switched to db ecommerce
> db.createCollection("products")
{ "ok" : 1 }
> show dbs
admin      0.000GB
config     0.000GB
ecommerce  0.000GB
local      0.000GB
```

```
> db.products.insertOne( { name : "Caldera Caldaia Duo" , price : 140000 } )
{
        "acknowledged" : true,
        "insertedId" : ObjectId("60ae9ac181fb57ee89cf08b7")
}
```

```
> db.products.find().pretty()
{
        "_id" : ObjectId("60ae9ac181fb57ee89cf08b7"),
        "name" : "Caldera Caldaia Duo",
        "price" : 140000
}
```

Vemos cómo nos agregó el "_id" al documento, esa es la diferencia.

▸ Una característica fundamental de MongoDB y otras bases NoSQL es que los documentos no tienen una estructura definida, como puede ser una tabla en un RDBMS. En una misma colección pueden convivir documentos con diferentes atributos, e incluso atributos de múltiples valores y documentos embebidos.

6. Agregue los siguientes documentos a la colección de productos:

{name:'Caldera Orbis 230', price:77000, tags: ['gas', 'digital']}
{name:'Caldera Ariston Clas', price:127000, tags: ['gas envasado', 'termostato']}
{name:'Caldera Caldaia S30', price:133000}
{name:'Caldera Mural Orbis 225cto', price:100000, tags: ['gas', 'digital', 'termostato']}
Y busque los productos:
- de $100.000 o menos
- que tengan la etiqueta (tag) "digital"
- que no tengan etiquetas (es decir, que el atributo esté ausente)
- que incluyan la palabra 'Orbis' en su nombre
- con la palabra 'Orbis' en su nombre y menores de $100.000
vuelva a realizar la última consulta pero proyecte sólo el nombre del producto en los resultados, omitiendo incluso el atributo _id de la proyección.

Se agregan los productos:

```
> db.products.insertMany([{ name:"Caldera Orbis 230", price:77000, tags: ["gas", "digital"]}, {name:"
Caldera Ariston Clas", price:127000, tags: ["gas envasado", "termostato"]}, {name:"Caldera Caldaia S3
0", price:133000}, {name:"Caldera Mural Orbis 225cto", price:100000, tags: ["gas", "digital", "termos
tato" ]}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("60ae9d3981fb57ee89cf08b8"),
                ObjectId("60ae9d3981fb57ee89cf08b9"),
                ObjectId("60ae9d3981fb57ee89cf08ba"),
                ObjectId("60ae9d3981fb57ee89cf08bb")
        ]
}
```

Búsqueda de $100.000 o menos:

```
> db.products.find( { price:{$gt: 100000 }} ).pretty()
{
        "_id" : ObjectId("60ae9ac181fb57ee89cf08b7"),
        "name" : "Caldera Caldaia Duo",
        "price" : 140000
}
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08b9"),
        "name" : "Caldera Ariston Clas",
        "price" : 127000,
        "tags" : [
                "gas envasado",
                "termostato"
        ]
}
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08ba"),
        "name" : "Caldera Caldaia S30",
        "price" : 133000
}
```

Se busca las que tengan la etiqueta (tag) “digital” :

```
> db.products.find({tags: "digital"})
{ "_id" : ObjectId("60b4fe7f16447d3e4da1b87d"), "name" : "Caldera Orbis 230", "price" : 77000, "tags" : [ "gas", "digital" ] }
{ "_id" : ObjectId("60b4fe7f16447d3e4da1b880"), "name" : "Caldera Mural Orbis 225cto", "price" : 100000, "tags" : [ "gas", "digital", "termostato" ] }
```

Se busca que no tengan etiquetas (es decir, que el atributo esté ausente):

```
> db.products.find( { tags: { $exists: false }}).pretty()
{
        "_id" : ObjectId("60ae9ac181fb57ee89cf08b7"),
        "name" : "Caldera Caldaia Duo",
        "price" : 140000
}
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08ba"),
        "name" : "Caldera Caldaia S30",
        "price" : 133000
}
```

Se busca que incluyan la palabra ‘Orbis’ en su nombre:

```
> db.products.find( { name: / Orbis /}).pretty()
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08b8"),
        "name" : "Caldera Orbis 230",
        "price" : 77000,
        "tags" : [
                "gas",
                "digital"
        ]
}
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08bb"),
        "name" : "Caldera Mural Orbis 225cto",
        "price" : 100000,
        "tags" : [
                "gas",
                "digital",
                "termostato"
        ]
}
```

Se busca que contenga la palabra ‘Orbis’ en su nombre y menores de $100.000:

```
> db.products.find( { name: / Orbis /, price: {$lt:100000}}).pretty()
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08b8"),
        "name" : "Caldera Orbis 230",
        "price" : 77000,
        "tags" : [
                "gas",
                "digital"
        ]
}
```

Ralizar la última consulta pero proyecte sólo el nombre del producto en los resultados, omitiendo incluso el atributo _id de la proyección:

```
> db.products.find({ name: / Orbis /, price: {$lt:100000}},{name:1,_id:0}).pretty()
{ "name" : "Caldera Orbis 230" }
```

▸ En MongoDB hay diferentes maneras de realizar actualizaciones, de acuerdo a la necesidades del esquema flexible de documentos.

7. Actualice la “Caldera Caldaia S30” cambiándole el precio a $150.000.

```
> db.products.findOneAndUpdate({name:"Caldera Caldaia S30"},{$set: {price:150000}})
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08ba"),
        "name" : "Caldera Caldaia S30",
        "price" : 133000
}
```

```
> db.products.find({name:"Caldera Caldaia S30"}).pretty()
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08ba"),
        "name" : "Caldera Caldaia S30",
        "price" : 150000
}
```

8. Cree el array de etiquetas (tags) para la “Caldera Caldaia S30”.

```
> db.products.findOneAndUpdate({name:"Caldera Caldaia S30"},{$set: {tags:[]}})
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08ba"),
        "name" : "Caldera Caldaia S30",
        "price" : 150000
}
> db.products.find({name:"Caldera Caldaia S30"}).pretty()
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08ba"),
        "name" : "Caldera Caldaia S30",
        "price" : 150000,
        "tags" : [ ]
}
```

9. Agregue “digital” a las etiquetas de la “Caldera Caldaia S30”.

```
> db.products.findOneAndUpdate({name:"Caldera Caldaia S30"},{$addToSet: {tags:"digital"}})
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08ba"),
        "name" : "Caldera Caldaia S30",
        "price" : 150000,
        "tags" : [ ]
}
> db.products.find({name:"Caldera Caldaia S30"}).pretty()
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08ba"),
        "name" : "Caldera Caldaia S30",
        "price" : 150000,
        "tags" : [
                "digital"
        ]
}
```

10. Incremente en un 10% los precios de todas las calderas digitales.

```
> db.products.find({tags:"digital"}).pretty()
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08b8"),
        "name" : "Caldera Orbis 230",
        "price" : 77000,
        "tags" : [
                "gas",
                "digital"
        ]
}
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08ba"),
        "name" : "Caldera Caldaia S30",
        "price" : 150000,
        "tags" : [
                "digital"
        ]
}
{
        "_id" : ObjectId("60ae9d3981fb57ee89cf08bb"),
        "name" : "Caldera Mural Orbis 225cto",
        "price" : 100000,
        "tags" : [
                "gas",
                "digital",
                "termostato"
        ]
}
```

Parte 3

11. Busque en la colección de compras (purchases) si existe algún índice definido.

```
> db.products.getIndexes()
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
```

Los índices son una estructura de datos especial que almacena una pequeña parte del conjunto de datos de la colección de forma fácil de recorrer. MongoDB por defecto provee un índice por ID. Además, admite la creación de índices definidos por el usuario. Los índices de MongoDB se definen a nivel de colecciones y brindan soporte en el campo o subcampo de un documento.

12. Cree un índice para el campo productName. Busque los las compras que tengan en el nombre del producto el string “11” y utilice el método explain("executionStats") al final de la consulta, para comparar la cantidad de documentos examinados y el tiempo en milisegundos de la consulta con y sin índice.

```
> db.products.find({ name: /11/}).pretty().explain("executionStats")
```

Se ejecutó el siguiente comando para buscar productos que contengan el número 11 en el nombre y sin haber índices por el nombre previamente.
Se produjeron las siguientes estadísticas:

Tardó 71 milisegundos

Ahora se creó un índice por nombre y de manera descendente

```
> db.products.createIndex({"name" : 1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

```
> db.products.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "name" : 1
                },
                "name" : "name_1"
        }
]
```


```
      "executionStats" : {
            "executionSuccess" : true,
            "nReturned" : 2291,
            "executionTimeMillis" : 99,
            "totalKeysExamined" : 50000,
            "totalDocsExamined" : 2291,
            "executionStages" : {
                  "stage" : "FETCH",
                  "nReturned" : 2291,
                  "executionTimeMillisEstimate" : 35,
                  "works" : 50001,
                  "advanced" : 2291,
                  "needTime" : 47709,
                  "needYield" : 0,
                  "saveState" : 50,
                  "restoreState" : 50,
                  "isEOF" : 1,
                  "docsExamined" : 2291,
                  "alreadyHasObj" : 0,
                  "inputStage" : {
                        "stage" : "IXSCAN",
                        "filter" : {
                              "name" : {
                                    "$regex" : "11"
                              }
                        },
                        "nReturned" : 2291,
                        "executionTimeMillisEstimate" : 34,
                        "works" : 50001,
                        "advanced" : 2291,
                        "needTime" : 47709,
                        "needYield" : 0,
                        "saveState" : 50,
                        "restoreState" : 50,
                        "isEOF" : 1,
                        "keyPattern" : {
                              "name" : 1
                        },
                        "indexName" : "name_1",
                        "isMultiKey" : false,
                        "multiKeyPaths" : {
                              "name" : [ ]
                        },
                        "isUnique" : false,
                        "isSparse" : false,
                        "isPartial" : false,
```


Se puede observar que tarda más en milisegundos pero examina menos documentos.

13. Busque las compras enviadas dentro de la ciudad de Buenos Aires. Para esto, puede definir una variable en la terminal y asignarle como valor el polígono del archivo provisto caba.geojson (copiando y pegando directamente). Cree un índice geoespacial de tipo 2dsphere para el campo location de la colección purchases y, de la misma forma que en el punto 12, compare la performance de la consulta con y sin dicho índice.

```
> db.purchases.find({"location": { $geoWithin: { $geometry : caba }}}).pretty().explain("executionStats")
```

```
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 9899,
                "executionTimeMillis" : 139,
                "totalKeysExamined" : 0,
                "totalDocsExamined" : 44911,
                "executionStages" : {
                        "stage" : "COLLSCAN",
```

Vemos cómo sin un índice tarda 139 milisegundos, recorriendo 44911 documentos.

Ahora creamos el índice:

```
> db.purchases.createIndex({location : "2dsphere"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
```

Y ejecutamos nuevamente la consulta:

```
        "executionStats" : {
                "executionSuccess" : true,
                "nReturned" : 9899,
                "executionTimeMillis" : 92,
                "totalKeysExamined" : 12546,
                "totalDocsExamined" : 12527,
                "executionStages" : {
                        "stage" : "FETCH",
```

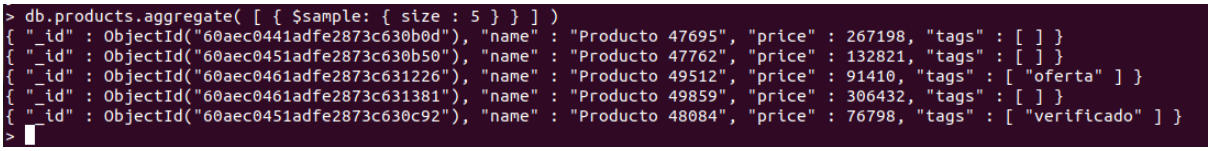
Vemos cómo esta vez tardó menos milisegundos (92) examinando menos documentos (12527)

Parte 4

▸MongoDB cuenta con un Aggregation Framework que brinda la posibilidad de hacer analítica en tiempo real del estilo OLAP (Online Analytical Processing), de forma similar a

otros productos específicos como Hadoop o MapReduce. En los siguientes ejercicios se verán algunos ejemplos de su aplicabilidad.

14. Obtenga 5 productos aleatorios de la colección.

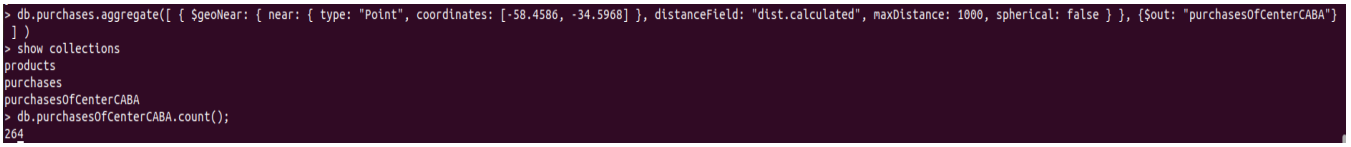


$sample sirve para devolver aleatorios y size para saber cuantos documentos agarrar de la colección.

15. Usando el framework de agregación, obtenga las compras que se hayan enviado a 1km (o menos) del centro de la ciudad de Buenos Aires ([-58.4586,-34.5968]) y guárdelas en una nueva colección.

La consulta reescrita por en el doc por si no se lee correctamente en la imagen:

```
db.purchases.aggregate([ {
        $geoNear: {
                near: {
                        type: "Point", coordinates: [-58.4586, -34.5968] },
                        distanceField: "dist.calculated",
                        maxDistance: 1000,
                        spherical: false } },
        {$out: "purchasesOfCenterCABA"}
        ]
        )
```



▪ ▪ ▪

16. Obtenga una colección de los productos que fueron enviados en las compras del punto anterior. Note que sólo es posible ligarlas por el nombre del producto.

```
db.purchasesOfCenterCABA.aggregate([
        {$group:
                {_id: "$productName"}
        },
        {$lookup:
                {from: "products",
                localField: "_id",
                foreignField: "name",
                as: "productsOnPurchase"}
        },
```

```
        {$unwind: "$productsOnPurchase"},
        {$replaceRoot:
                {newRoot: "$productsOnPurchase"}
        }
])
```

17. Usando la colección del punto anterior, obtenga una nueva en la que agrega a cada producto un atributo purchases que consista en un array con todas las compras de cada producto.

▸ Si la consulta se empieza a tornar difícil de leer, se pueden ir guardando los agregadores en variables, que no son más que objetos en formato JSON.

18. Obtenga el promedio de costo de envío pagado para cada producto del punto anterior.

```
> db.productsSoldOfCenterCABAWithPurchases.aggregate([{$unwind: "$purchases"},{$group: {_id: "$name", "averagePrice": {$avg:"$purchases.shippingCost"}}}])
{ "_id" : "Producto 2820", "averagePrice" : 300 }
{ "_id" : "Producto 42558", "averagePrice" : 2200 }
{ "_id" : "Producto 38202", "averagePrice" : 1900 }
{ "_id" : "Producto 16485", "averagePrice" : 2000 }
{ "_id" : "Producto 5635", "averagePrice" : 400 }
{ "_id" : "Producto 32010", "averagePrice" : 1300 }
{ "_id" : "Producto 29011", "averagePrice" : 1400 }
{ "_id" : "Producto 16063", "averagePrice" : 500 }
{ "_id" : "Producto 30723", "averagePrice" : 1700 }
{ "_id" : "Producto 44973", "averagePrice" : 1600 }
{ "_id" : "Producto 27300", "averagePrice" : 400 }
{ "_id" : "Producto 41929", "averagePrice" : 2200 }
{ "_id" : "Producto 27243", "averagePrice" : 1400 }
{ "_id" : "Producto 5977", "averagePrice" : 500 }
{ "_id" : "Producto 38192", "averagePrice" : 1300 }
{ "_id" : "Producto 22469", "averagePrice" : 400 }
{ "_id" : "Producto 22466", "averagePrice" : 1500 }
{ "_id" : "Producto 8790", "averagePrice" : 800 }
{ "_id" : "Producto 28015", "averagePrice" : 1600 }
{ "_id" : "Producto 3281", "averagePrice" : 1900 }
```

```
db.productsSoldOfCenterCABAWithPurchases.aggregate(
        [{$unwind: "$purchases"},
        {$group: {
                _id: "$name",
                "averagePrice": {$avg:"$purchases.shippingCost"}
                }
        }])
```