Parte 1: Bases de Datos NoSQL y Relacionales

▸ 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• Tabla / Relación• Fila / Tupla• Columna

Cada base de datos en MongoDB consta de colecciones que son equivalentes a una base de datos RDBMS que consta de tablas SQL.

MongoDB cuenta con **colecciones**. Cada colección almacena datos en forma de **documentos**, lo que equivale a tablas que almacenan datos en filas. Mientras que una fila almacena datos en su conjunto de columnas, un documento tiene una estructura similar a JSON (conocida como BSON en MongoDB)

Aca un ejemplo de un documento:

"_id": ObjectId ("5146bb52d8524270060001f3"), "age": 25, "city": "Los Angeles", "email": "mark@abc.com", "user_name": "Mark Hanks"

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

MongoDB nos trae transacciones ACID (Atomicity, Consistency, Isolation, Durability) entre múltiples documentos

Para entenderlas mejor hay que conocer los métodos:

startSession(), las transacciones están asociadas a una sesión, es por ello que siempre necesitaremos abrir una sesión.

endSession() , cuando cerremos la sesión, si hay alguna transacción ejecutándose, esta se abortará.

Las sesiones tienen un limite predeterminado de 60 segundos

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

Los indices en MongoDB utiliza se generan en forma de arbol Binario balanceado

Existen los siguientes tipos de indices

Índice _id

MongoDB crea un campo *_id* para cada documento. Con este campo se crea automáticamente un índice único que no podemos borrar. Todo documento insertado en una colección tendrá un campo *_id* único que no se podrá repetir.

Índices simples o de un solo campo: Estos índices se aplican a un solo campo de nuestra colección. Para declarar un índice de este tipo debemos usar un comando similar a este:

```
db.users.ensureIndex( { "user_id" : 1 } )
```

Índices compuestos: En este caso el índice se generará sobre varios campos.

```
db.users.ensureIndex( { "user_name" : 1, "age":-1 } )
```

El índice que se generará con la instrucción anterior, agrupará los datos primero por el campo *user_name* y luego por el campo*age*. Es decir, se generaría algo así:

"Antonio", 35

"Antonio", 18

"María", 56

"María", 30

Índices únicos

Los índices simples y múltiples, pueden estar obligados a contener valores únicos. Esto lo conseguimos añadiendo el parámetro *unique* a la hora de crearlos.

```
db.users.ensureIndex( { "user_id" : 1 }, {"unique":true} )
```

Índices sparse

Los índices que hemos mencionado antes, incluyen todos los documentos.

Para crear índices que solo incluyan los documentos cuyo campo indexado existe, utilizaremos la opción sparse.

```
db.users.ensureIndex( { "user_name" : 1 }, {"sparse":true} )
```

4. ¿Existen claves foráneas en MongoDB?

No existen claves foráneas en MongoDB, por lo que no hay "eliminaciones en cascada" o "actualizaciones en cascada".

Además, MongoDB tiene un DBRef estándar que ayuda a estandarizar la creación de estas referencias.

Parte 2: Primeros pasos con MongoDB

▶ 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 eccomerce

>db.products.insert({"name":"Caldera Caldaia Duo", "price":"140000"})

>db.products.find()

Nos devolvió:

> { "_id" : ObjectId("60b0632d52eb3d7ae4b18aa0"), "name" : "Caldera Caldaia Duo", "price" : "140000" }

este campo ObjectId fue generado por MongoDB

> db.products.find().pretty()

{

"_id" : ObjectId("60b0632d52eb3d7ae4b18aa0"),
"name" : "Caldera Caldaia Duo",
"price" : "140000”

}

▸ 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’]}

>db.products.insert({"name":"Caldera Orbis 230", "price":77000, "tags": ['gas', 'digital']})

> db.products.insert({'name':'Caldera Ariston Clas', 'price': 127000, 'tags': ['gas envasado', 'termostato']})

>db.products.insert({'name':'Caldera Caldaia S30', 'price':133000})

>db.products.insert({'name':'Caldera Mural Orbis 225cto', 'price':100000, 'tags': ['gas', 'digital', 'termostato']})

Y busque los productos:

- de $100.000 o menos

> db.products.find({"price":{$lte:100000}})
{ "_id" : ObjectId("60b06e7d379a56385b5bca66"), "name" : "Caldera

Orbis 230", "price" : 77000, "tags" : ["gas", "digital"] }
{ "_id" : ObjectId("60b06ed1379a56385b5bca69"), "name" : "Caldera Mural Orbis 225cto", "price" : 100000, "tags" : ["gas", "digital", "termostato"] }

- que tengan la etiqueta (*tag*) "digital"

> db.products.find({'tags':"digital"})
{ "_id" : ObjectId("60b06e7d379a56385b5bca66"), "name" : "Caldera Orbis 230", "price" : 77000, "tags" : ["gas", "digital"] }
{ "_id" : ObjectId("60b06ed1379a56385b5bca69"), "name" : "Caldera Mural Orbis 225cto", "price" : 100000, "tags" : ["gas", "digital", "termostato"] }

- que no tengan etiquetas (es decir, que el atributo esté ausente)

> db.products.find({"tags":null})
{ "_id" : ObjectId("60b06e8e379a56385b5bca68"), "name" : "Caldera Caldaia S30", "price" : 133000 } { "_id" : ObjectId("60b06eee379a56385b5bca6a"), "name" : "Caldera Caldaia Duo", "price" : "140000" }

- que incluyan la palabra 'Orbis' en su nombre

> db.products.find({"name": RegExp('Orbis', 'i')})
{ "_id" : ObjectId("60b06e7d379a56385b5bca66"), "name" : "Caldera Orbis 230", "price" : 77000, "tags" : ["gas", "digital"] }
{ "_id" : ObjectId("60b06ed1379a56385b5bca69"), "name" : "Caldera Mural Orbis 225cto", "price" : 100000, "tags" : ["gas", "digital", "termostato"] }

/*"Orbis"*/

- con la palabra 'Orbis' en su nombre y menores de $100.000

> db.products.find({"name": RegExp('Orbis', 'i') , "price":{$lte:100000}})
{ "_id" : ObjectId("60b06e7d379a56385b5bca66"), "name" : "Caldera Orbis 230", "price" : 77000, "tags" : ["gas", "digital"] }
{ "_id" : ObjectId("60b06ed1379a56385b5bca69"), "name" : "Caldera Mural Orbis 225cto", "price" : 100000, "tags" : ["gas", "digital", "termostato"] }

vuelva a realizar 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": RegExp('Orbis', 'i') , "price":{$lte:100000}}, { _id: 0, price: 0, tags: 0 })

{ "name" : "Caldera Orbis 230" } {
{"name" : "Caldera Mural Orbis 225cto" }

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

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

> db.products.update({"name":"Caldera Caldaia S30"}, {$set: {"price":150000}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.products.find({"name":"Caldera Caldaia S30"}).pretty()
{

"_id" : ObjectId("60b13091379a56385b5bca6c"),
"name" : "Caldera Caldaia S30",
"price" : 150000 }

}

8. Cree el *array* de etiquetas (*tags*) para la "Caldera Caldaia S30".

> db.products.update({"name":"Caldera Caldaia S30"}, {$set: {"tags":[]}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.products.find({"name":"Caldera Caldaia S30"}).pretty()
{
"_id" : ObjectId("60b13091379a56385b5bca6c"),
"name" : "Caldera Caldaia S30",
"price" : 150000,
"tags" : [ ]
}

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

> db.products.update({"name":"Caldera Caldaia S30"}, {$addToSet: {tags:"digital"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 0 })
> db.products.find({"name":"Caldera Caldaia S30"}).pretty()
{
"_id" : ObjectId("60b13091379a56385b5bca6c"),
"name" : "Caldera Caldaia S30",
"price" : 150000,
"tags" : [
"digital"
]
}

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

```
> db.products.update({"name": RegExp('Caldera', 'i') ,
"tags":"digital"}, { $mul: { price: 1.1 }},{multi:true})
WriteResult({ "nMatched" : 4, "nUpserted" : 0, "nModified" : 4 })
> db.products.find({"name": RegExp('Caldera', 'i') ,
"tags":"digital"}, { _id: 0, })
{ "name" : "Caldera Orbis 230", "price" : 84700, "tags" : [ "gas",
"digital" ] }
{ "name" : "Caldera Mural Orbis 225cto", "price" : 110000, "tags"
: [ "gas", "digital", "termostato" ] }
{ "name" : "Caldera Mural Orbis 225cto", "price" : 110000, "tags"
: [ "gas", "digital", "termostato" ] }
{ "name" : "Caldera Caldaia S30", "price" : 165000, "tags" : [
"digital" ] }
```

Parte 3: Índices

▸ Elimine todos los productos de la colección. Guarde en un archivo llamado 'generador.js' el siguiente código JavaScript y ejecútelo con: load(<ruta del archivo 'generador.js'>).

Si utiliza un cliente que lo permita (ej. Robo3T), se puede ejecutar directamente en el espacio de consultas.

```
for (var i = 1; i <= 50000; i++) {

var randomTags = ['envio express', 'oferta', 'cuotas',
'verificado'].sort( function()

{ return 0.5 - Math.random() } ).slice(1, Math.floor(Math.random()
* 5));

var randomPrice = Math.ceil(110000+(Math.random() * 300000 -
100000));

db.products.insert({

     name:'Producto '+i,

     price:randomPrice,

     tags: randomTags

});

}

for (var i = 1; i <= 50000; i++) {

     if (Math.random() > 0.7) {

          var randomPurchases = Math.ceil(Math.random() * 5);

               for (var r = 1; r <= randomPurchases; r++){
```

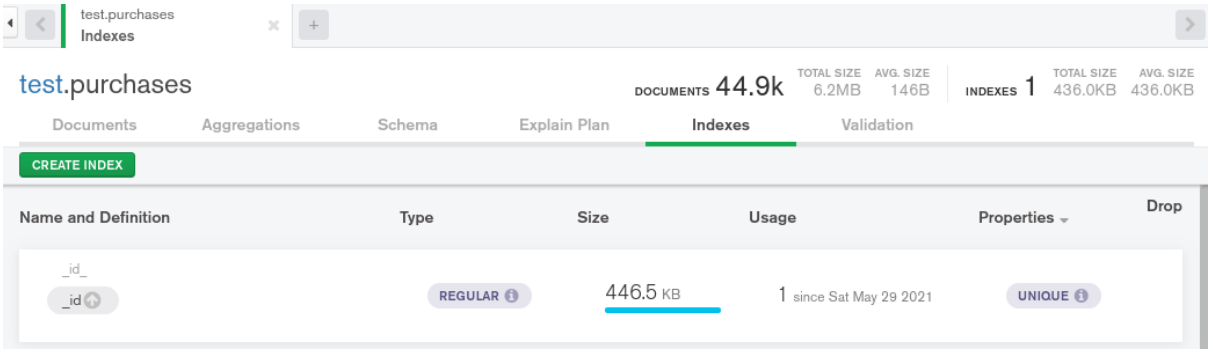
```
                    var randomLong = -34.56 - (Math.random() *
.23);

                    var randomLat = -58.4 - (Math.random() *
.22);

                    var shippingCost =
200+Math.ceil(Math.random()*20) * 100;

                    db.purchases.insert({

                    productName:'Producto '+i,

                    shippingCost: shippingCost,

                    location: {type: "Point",coordinates:
[randomLat, randomLong]}

                });

          }

     }

}
```

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

En Consola:

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

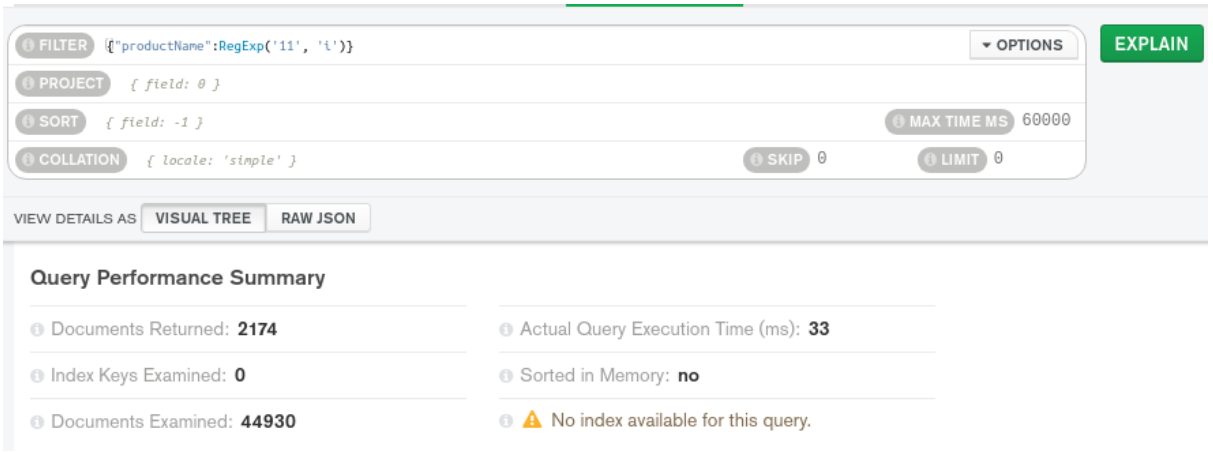
En Compass:



Podemos observar que existe un campo _id. Este fue generado por MongoDB automaticamente pero no vemos ningún otro definido

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.

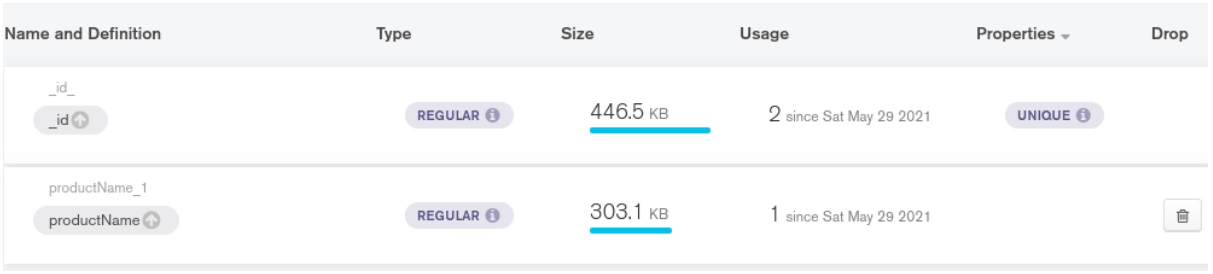
Antes de crear el indice en productName



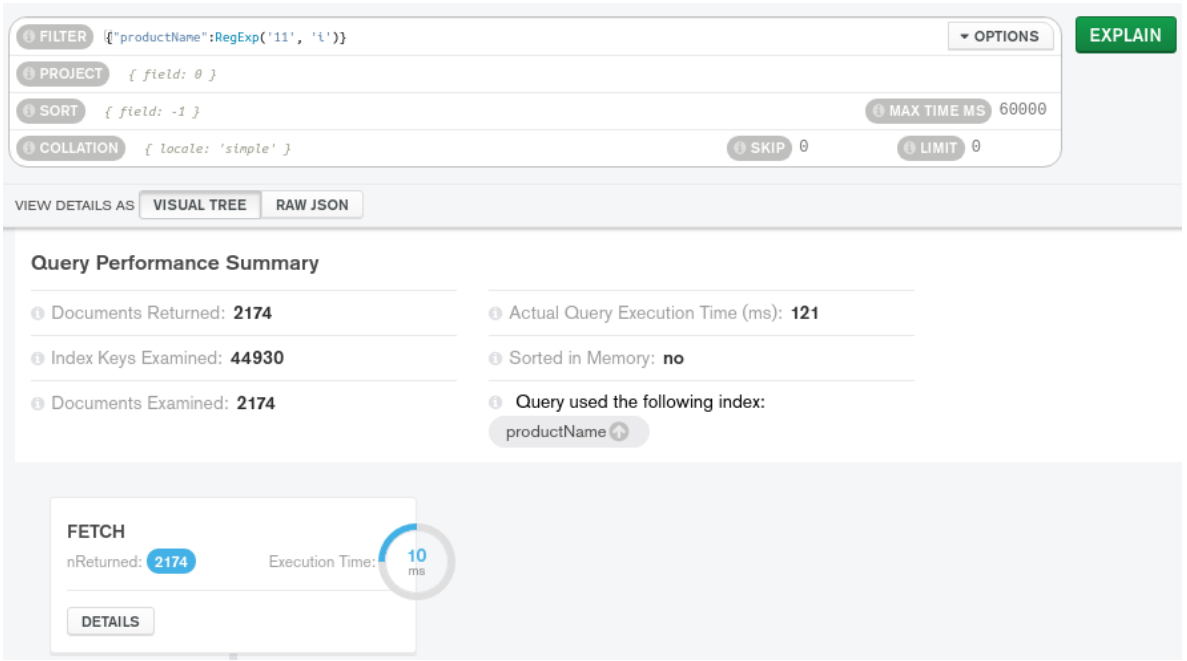
tiro el comando
> db.purchases.getIndexes()

```
[
  {
        "v" : 2,
        "key" : {
              "_id" : 1
        },
        "name" : "_id_",
        "ns" : "test.purchases"
  },
  {
        "v" : 2,
```

```
        "key" : {
                "productName" : 1
        },
        "name" : "productName_1",
        "ns" : "test.purchases"
    }
]
```



con index productName_1 vamos a observar que la busqueda ya no se realiza en todo el documento. Sino que realiza una busqueda por indices y luego en los documentos que obtiene los consulta

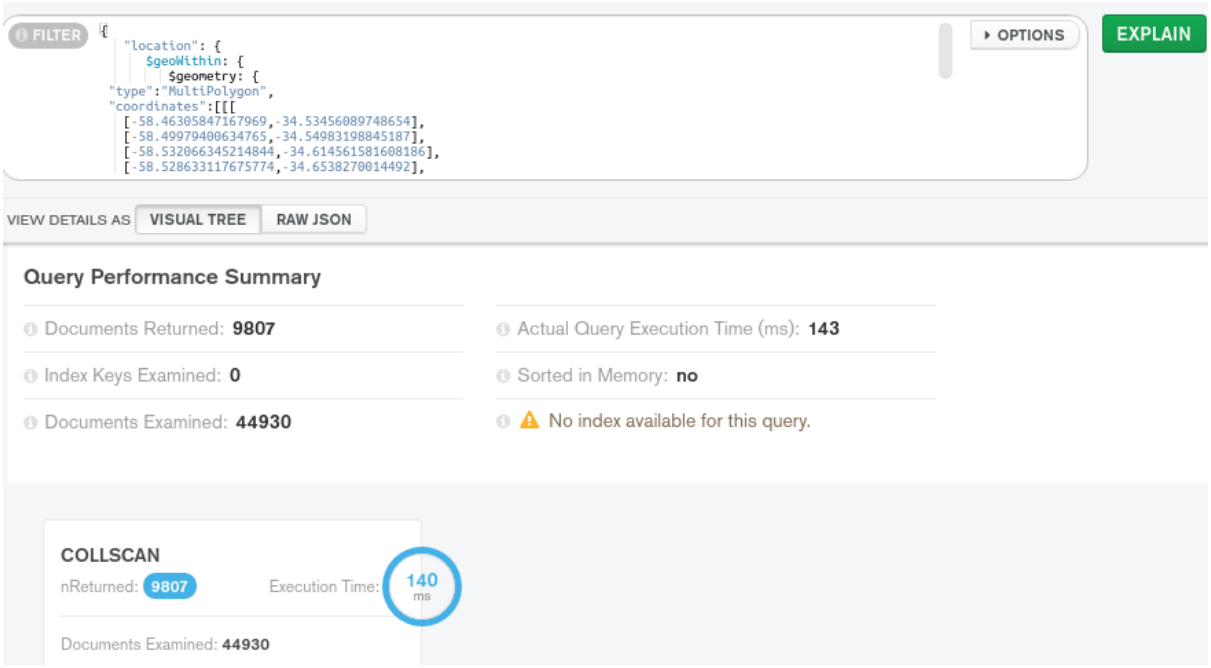


13. Busque las compras enviadas dentro de la ciudad de Buenos Aires. Para esto, puede definir una variable en la terminal y asignarle como valor el polígono del archivo provisto **caba.geojson** (copiando y pegando directamente). Cree un índice geoespacial de tipo **2dsphere** para el campo **location** 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.

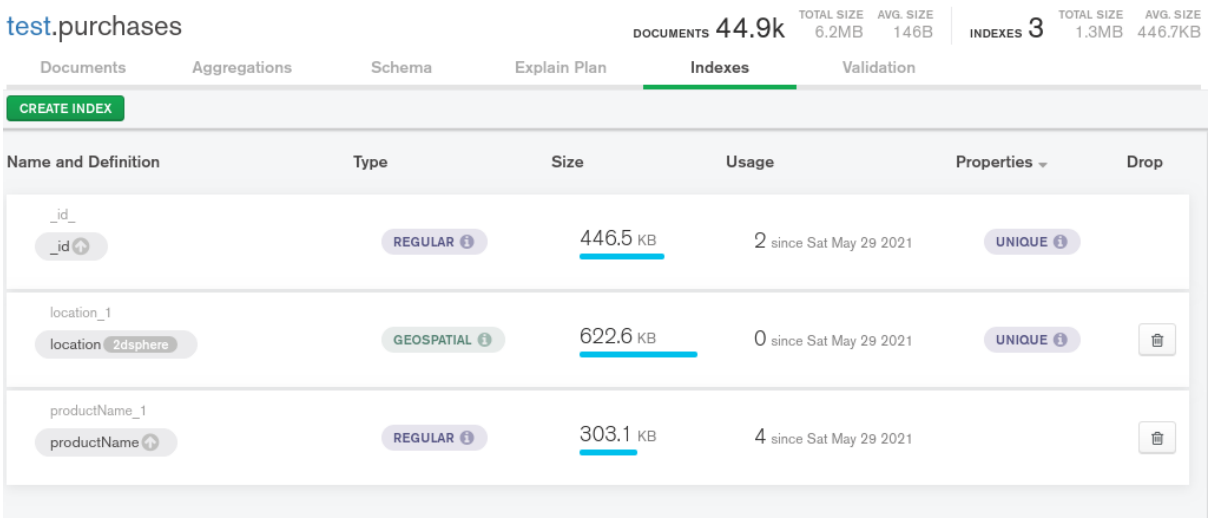
```
> db.purchase.find{
  "location": {
        $geoWithin: {
```

```
        $geometry: { $coords
        }
  }
}
```

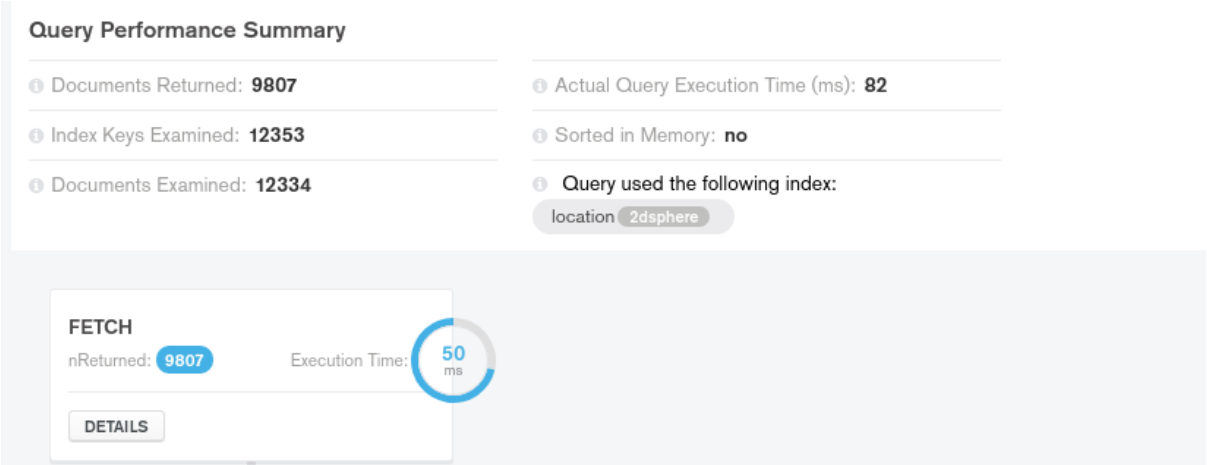
Sin el indice en location.coordinates



> db.purchases.createIndex({"location":2dsphere})
luego de aplicar el indice 2dsphere



en el explain plan:



Parte 4: Aggregation Framework

▸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.

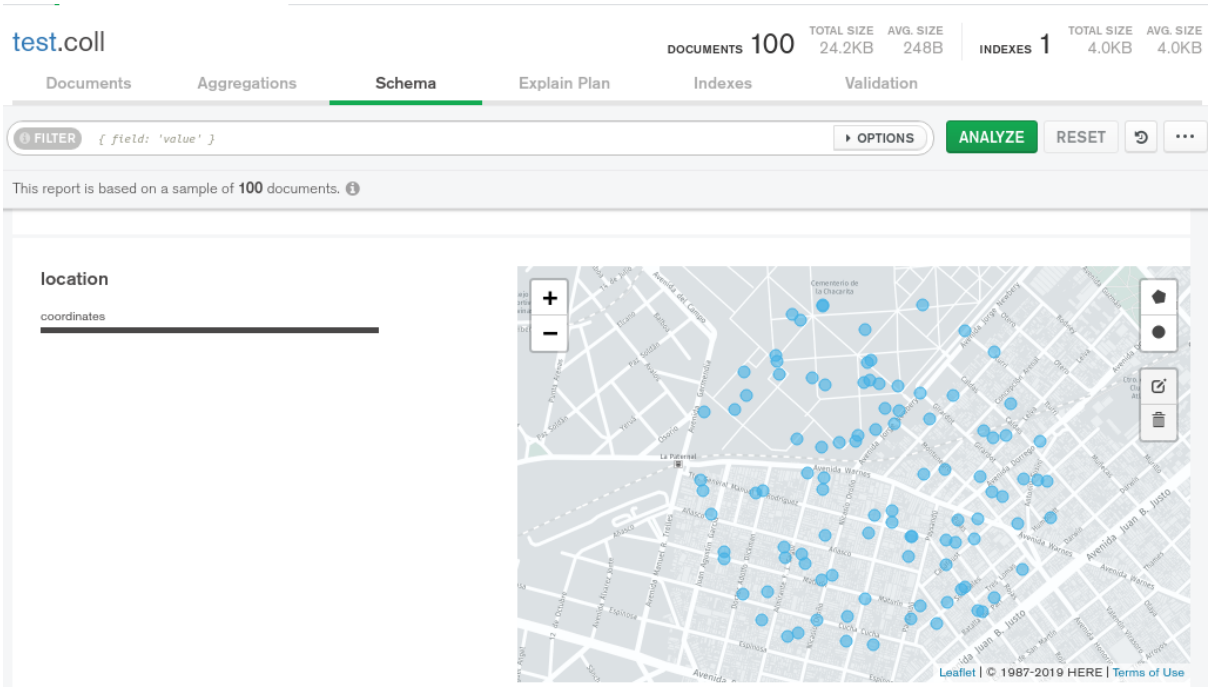
14. Obtenga 5 productos aleatorios de la colección.

> db.purchases.aggregate([{ $sample: { size: 5 } }])

esta consulta nos devuelve 5 elementos aleatorios

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.

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



Usando la funcionalidad de MongoDB Compass podemos ver en el mapa los puntos según cada documento

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.coll.aggregate(
{ $lookup: {
      from: "products",
      localField: "productName",
      foreignField: "name",
      as: "resul" }},
{$unwind: "$resul"},
{ "$replaceRoot": { "newRoot": "$resul" }},
{$out:"punto16"}
)
```

▸ 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.

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.

```
> db.punto16.aggregate(

{ $lookup: {

      from: "purchases",

      localField: "name",

      foreignField: "productName",

      as: "purchases" }},

{$out:"punto17"}

)
```

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

```
> db.punto17.aggregate([{
    $unwind: {
        path: '$purchases',
    }
 }, {
    $group: {
        _id: {
            id: "$_id",
            name: "$name"
        },

        envio: {
            $avg: "$purchases.shippingCost"
        }
    }
}],
)
```