Facultad de Informática – Universidad Nacional de La Plata
Cátedra de Bases de Datos II

Bases de Datos 2 2022 -TP3

Bases de Datos NoSQL / Práctica con MongoDB

entrega: 13/6

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: el concepto existe en MongoDB.
   - Tabla / Relación:el concepto de tabla/relación no existe en MongoDB,es reemplazado por colecciones que son un grupo de documentos a diferencia de las tablas que son un grupo de filas.
   - Fila / Tupla:el concepto de fila/tupla no existe en MongoDB,es reemplazado por el concepto de documento.Un documento es una entidad que respeta el formato JSON y por ende está conformada por claves y valores.
   - Columna: el concepto de columna no existe en MongoDB,es reemplazado por el concepto de campo Estos campos pueden variar dentro de los documentos de una misma colección.
2. MongoDB tiene soporte para transacciones, pero no es igual que el de los RDBMS. ¿Cuál es el alcance de una transacción en MongoDB?

   Hasta la versión de mongo 4.0 , que agrega la implementación de modelos de transacción A.C.I.D sobre diferentes documentos.
   Mongo trabaja de forma atómica sobre cada uno de sus documentos , ofreciéndoles a ellos un modelo transaccional BA.S.E. La forma de modelar las referencias entre documentos para asegurar la integridad de datos se realiza mediante anidación de documentos.

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

Los tipos de índices que soporta MongoDB son:

- **Single Field index**: estos tipos de índices se aplican a un solo campo de un documento y facilitan el ordenamiento ya sea ascendente o descendente de los documentos.
- **Compound index**: estos tipos de índices permiten crear índices utilizando varios campos del documento.El orden de estos campos del índice son importantes ya que ordena en el orden que el índice fue creado.
- **Multikey index**: estos tipos de índices se utilizan cuando un campo tenga asociado como valor un array y se necesite encontrar todos los documentos que tengan un valor dentro del array mencionado anteriormente.Para esto MongoDB crea una clave de índice para cada elemento del array.
- **Geospatial index**:estos tipos de índices son utilizados para permitir consultas eficientes para datos de coordenadas geoespaciales.
- **Text index**: estos tipos de índices son proporcionados por MongoDB para permitir la búsqueda de texto dentro de una colección.
- **Hashed index**: estos tipos de índices son proporcionados por MongoDB para permitir indexar campos utilizando una función de hash.Estos solo admiten coincidencia por igualdad,no por rangos.

4. ¿Existen claves foráneas en MongoDB?

En mongo no existen las claves foráneas.

Ya que no es Relacional , debemos modelar la bd de forma adecuada los datos que almacena y a las consultas que pretende ejecutar.

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 **vaccination**, y una colección llamada **nurses**. En esa colección inserte un nuevo documento (una enfermera) con los siguientes atributos:

```
{name:'Morella Crespo', experience:9}
```

recupere la información de la enfermera usando el comando `db.nurses.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?

Para crear la base de datos se utilizó el comando use vaccination.

```
> use vaccination
switched to db vaccination
```

Para crear la colección nurses se usó el comando db.createCollection("nurses") y para

agregarle una enfermera se utilizó el comando:db.nurses.insertOne({name:”Morella Crespo”,experience:9}).Después se usó el comando db.nurses.find().pretty() y la diferencia es que se agregó un atributo id.

```
> db.createCollection("nurses")
{ "ok" : 1 }
> db.nurses.insertOne({name:"Morella Crespo",experience:9})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("629fccc0c2bbf63467c388db")
}
> db.nurses.find().pretty()
{
        "_id" : ObjectId("629fccc0c2bbf63467c388db"),
        "name" : "Morella Crespo",
        "experience" : 9
}
```

► 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 enfermeros:

```
{name:’Gale Molina', experience:8, vaccines: ['AZ', ‘Moderna']}
{name:’Honoria Fernández', experience:5, vaccines: ['Pfizer', ‘Moderna’, ‘Sputnik V’]} {name:’Gonzalo Gallardo', experience:3}
{name:’Altea Parra', experience:6, vaccines: ['Pfizer']}
```

Para agregar los enfermeros se usó el comando db.nurses.insertMany()

```
> db.nurses.insertMany([{name:"Gale Molina",experience:8,vaccines:["AZ","Moderna"]},
... {name:"Honoria Fernández",experience:5,vaccines:["Pfizer","Moderna","Sputnik V"]},
... {name:"Gonzalo Gallardo",experience:3},
... {name:"Altea Parra",experience:6,vaccines:["Pfizer"]}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("629ff90004ad360550ff9c57"),
                ObjectId("629ff90004ad360550ff9c58"),
                ObjectId("629ff90004ad360550ff9c59"),
                ObjectId("629ff90004ad360550ff9c5a")
        ]
}
```

Y busque los enfermeros:

- de 5 años de experiencia o menos > **db.nurses.find({experience: {$lt: 5}})** <

- que hayan aplicado la vacuna “Pfizer”

> **db.nurses.find({vaccines: {$elemMatch: {$eq: 'Pfizer'}}})** <

- que no hayan aplicado vacunas (es decir, que el atributo *vaccines* esté ausente)

> **db.nurses.find({vaccines: { $exists: false}})** <

- de apellido ‘Fernández’

Debemos crear un índice , para buscar sobre name

> **db.nurses.createIndex({name:"text",line:"text"})** <

ahora si podemos buscar por el campo name

> **db.nurses.find({$text: {$search: "Fernandez" }})** <

- con 6 o más años de experiencia y que hayan aplicado la vacuna ‘Moderna’

db.nurses.find
({$and: [{experience: {$gte: 6} }, {vaccines: {$elemMatch: {$eq: 'Moderna'}} }] })

vuelva a realizar la última consulta pero proyecte sólo el nombre del enfermero/a en los resultados, omitiendo incluso el atributo **_id** de la proyección.

Se agrega al final los campos que quiero y los que no.

db.nurses.find({$and: [{experience: {$gte: 6} }, {vaccines: {$elemMatch: {$eq: 'Moderna'}} }] },{name:1,_id:0})

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

7. Actualice a “Gale Molina” cambiándole la experiencia a 9 años.

db.nurses.update({name: {$eq: "Gale Molina"}}, {$set: {experience:9}})

8. Cree el *array* de vacunas (*vaccines*) para ”Gonzalo Gallardo".

db.nurses.update({name: {$eq: "Gonzalo Gallardo"}}, {$set: {vaccines: []}})

9. Agregue “AZ” a las vacunas de “Altea Parra”.

db.nurses.update({name: {$eq: "Altea Parra"}}, {$push: {vaccines: 'AZ'}})

10. Duplique la experiencia de **todos** los enfermeros que hayan aplicado la vacuna “Pfizer”

db.nurses.updateMany({vaccines: {$elemMatch: {$eq: 'Pfizer'}}}, {$mul: {experience: 2}})

Parte 3: Índices

▸ Elimine a todos los enfermeros 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.

```
var vaccines = ['AZ', 'Pfizer', 'Moderna', 'Sputnik V',
"Johnson"]; for (var i = 1; i <= 2000; i++) {
      var randomVax = vaccines.sort( function() { return 0.5 -
Math.random() } ).slice(1, Math.floor(Math.random() * 5));
      var randomExperience = Math.ceil(2+(Math.random() * 20 - 2));
```

```
        db.nurses.insert({
                name:'Enfermero '+i,
                experience:randomExperience,
                tags: randomVax
        });}
var patientNumber = 0;
for (var i = 1; i <= 2000; i++) {
                var dosesCount = 50 + Math.ceil(Math.random() * 100);
                for (var r = 1; r <= dosesCount; r++){
                      var randomLong = -34.56 - (Math.random() * .23);
                       var randomLat = -58.4 - (Math.random() * .22);
                       db.patients.insert({
                        name:'Paciente '+patientNumber,
                          address: {type: "Point",coordinates: [randomLat, randomLong]}
                       });
                       patientNumber++;
                       var year = 2020 + Math.round(Math.random());
                       var month = Math.ceil(Math.random() * 12);
                       var day = Math.ceil(Math.random() * 28);
                       db.doses.insert({
                        nurse:'Enfermero '+i,
                        patient:'Paciente '+patientNumber,
                         vaccine: vaccines[Math.floor(Math.random()*vaccines.length)],
                        date: new Date(year+'-'+month+'-'+day)
                       });
                }}
```

Para eliminar a todos los enfermeros de la colección se utilizó el comando db.nurses.remove({}) y se ejecutó el load:

```
> db.nurses.remove({})
WriteResult({ "nRemoved" : 5 })
> load("C:\Users\PC\Desktop\BBDD2\generador.js")
{"t":{"$date":"2022-06-08T21:36:27.363Z"},"s":"E",  "c":"-",        "id":22779,   "ctx":"js","msg":"file [{filename}] d
esn't exist","attr":{"filename":"C:UsersPCDesktopBBDD2generador.js"}}
Error: error loading js file: C:UsersPCDesktopBBDD2generador.js :
@(shell):1:1
> load("C:/Users/PC/Desktop/BBDD2/generador.js")
true
```

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

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

12. Cree un índice para el campo `nurse` de la colección `doses`. Busque las dosis que tengan en el nombre del enfermero 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.

    Se crea el índice ejecutando el comando db.doses.createIndex({nurse:1})

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

Sin índice la consulta db.doses.find({nurse: /.*11.*/}) se ejecutó en 389 milisegundos y se analizaron 200069 documentos:

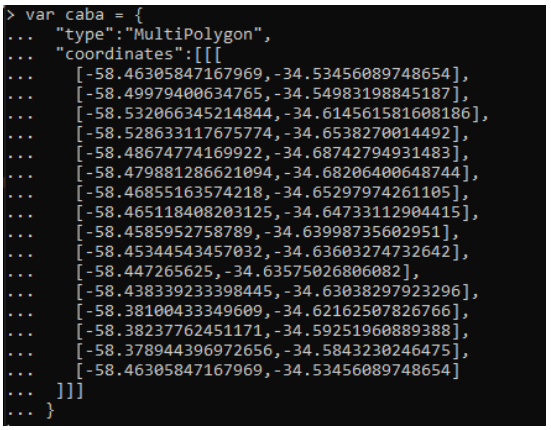
```
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 11963,
        "executionTimeMillis" : 389,
        "totalKeysExamined" : 0,
        "totalDocsExamined" : 200069,
```

Con índice la consulta db.doses.find({nurse:/.*11.*/}) se ejecutó en 528 milisegundos y se analizaron 11963 documentos:

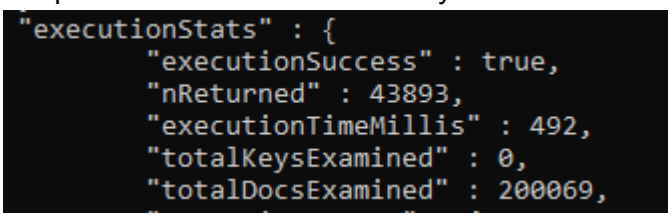
```
"executionStats" : {
        "executionSuccess" : true,
        "nReturned" : 11963,
        "executionTimeMillis" : 528,
        "totalKeysExamined" : 200069,
        "totalDocsExamined" : 11963,
```

13. Busque los pacientes que viven 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 **`patients`** y, de la misma forma que en el punto 12, compare la performance de la consulta con y sin dicho índice.

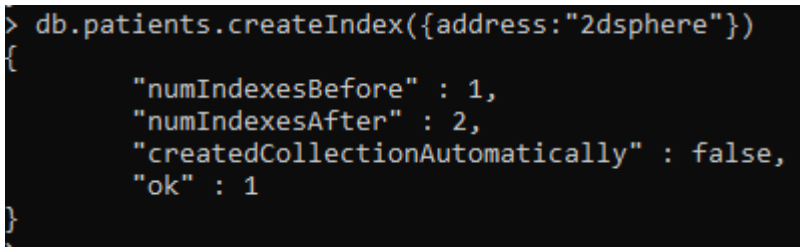
Se define la variable en la terminal:



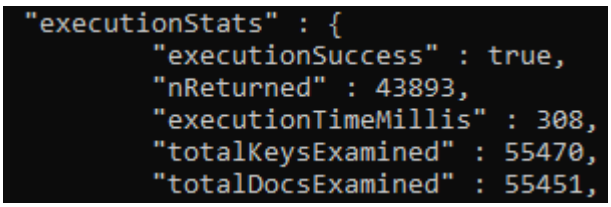
Para buscar los pacientes que viven en la ciudad de Buenos Aires se ejecutó la consulta: db.patients.find({address:{$geoWithin: {$geometry: caba}}}).explain(“executionStats”).Sin indice se compararon 200069 documentos y lo hizo en 492 milisegundos.



Para crear el índice 2dsphere se ejecutó db.patients.createIndex({address:”2dsphere”})



Luego de crear el índice se volvió a realizar db.patients.find({address:{$geoWithin: {$geometry: caba}}}).explain(“executionStats”) y con índice se compararon 55451 y lo hizo en 308 milisegundos.



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 pacientes aleatorios de la colección.

> **db.patients.aggregate([{ $sample: { size: 5 } }])** <

15. Usando el framework de agregación, obtenga los pacientes que vivan a 1km (o menos) del centro geográfico de la ciudad de Buenos Aires (`[-58.4586,-34.5968]`) y guárdelos en una nueva colección.

Creamos la nueva colección donde guardamos el resultado

> **db.createCollection("patientsBsAs")** <

Anteriormente se creó el índice 2dsphere (ejercicio 13) para el campo address

> **db.patients.createIndex({address:"2dsphere"})** <

En el siguiente paso realizamos la consulta utilizando $geoNear , que genera un orden de documentos a partir de un punto específico.

```
db.patients.aggregate([
      {
        $geoNear: {
          near: { type: "Point", coordinates: [ -58.4586 , -34.5968  ] },
          distanceField: "address",
          maxDistance: 1000,
          includeLocs: "address",
          spherical: true
        }
      }, {$out: "patientsBsAs"}
    ])
```

Si luego hacemos ejecutamos > **db.patientsBsAs.find()** < nos dará como resultado :

16. Obtenga una colección de las dosis aplicadas a los pacientes del punto anterior. Note que sólo es posible ligarlas por el nombre del paciente.

Primero debemos crear la nueva collection , donde se guardaran los documentos resultantes, en nuestro caso la nombramos dosesAplicadas.

> **db.createCollection(“dosesAplicadas”)** <

Utilizamos $lookup para realizar el “join entre las dos collections y unificarlas a partir de la coincidencia de los campos name de patients y patients de doses”. Luego agrupamos los documentos por patients (en este caso es el nombre) y agregamos una array con todos sus doses que le han sido aplicadas. Finalmente estos documentos son almacenados en la collection “doses aplicadas”.

```
db.patientsBsAs.aggregate([
 { $lookup:
     { from: "doses",
       localField: "name",
       foreignField: "patient",
       as: "doses"
     }
 },
 { $group: {
    _id : "$name" ,
    doses: { $push : "$doses" } }
  } ,
  { $out: "dosesAplicadas" }
])
```

▸ 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. Obtenga una nueva colección de *nurses*, cuyos nombres incluyan el string “111”. En cada documento (cada *nurse*) se debe agregar un atributo *doses* que consista en un array con todas las dosis que aplicó después del 1/5/2021.

En este ejercicio decidimos separar la operación en 3 pasos:

primer paso: creamos una nueva collection que almacena los nurses con name que contienen 111

> **db.createCollection(“nurses111”)** <

> **db.nurses.aggregate([{$match: { name:/.*111.*/}} , {$out: "nurses111"}])** <

segundo paso : creamos una collection que almacena las doses que se aplicaron luego de la fecha 1/5/2021

> **db.createCollection("dosesAfter")** <

> **db.doses.aggregate([{$match: {date: { $gt: new Date("2021-5-1T00:00:00Z") }}}, {$out: "dosesAfter"}])** <

tercer paso: creamos una collection que almacenará a a los nurses con todas sus vacunas aplicadas. Utilizamos $lookup con las collections creadas en el primer y segundo paso. Agrupamos los nurses por su nombre. Con el $unset eliminamos el campo nurse de las dosis que se repite y el id de la dosis. Finalmente guardamos los documentos en nursesConDosisAplicadas

> **db.createCollection("nursesConDosisAplicadas")** <

```
db.nurses111.aggregate([
        { $lookup:
          { from: "dosesAfter",
            localField: "name",
            foreignField: "nurse",
            as: "doses"  }
        } ,
        { $group:
            {  _id : "$name" ,
               doses: { $push : "$doses" } }
        } ,
        {$unset: ["doses.nurse" , "doses._id" ]
        },
        { $out: "nursesConDosisAplicadas" }
])
```

Links de ayuda : https://www.mongodb.com/es/basics/create-database

comandos de mongoDB:

https://geekflare.com/es/mongodb-queries-examples/#:~:text=Para%20buscar%20el%20documento%20almacenado,documentos%20almacenados%20en%20una%20colecci%C3%B3n.

Como armar una Query en el find de command line:

https://www.mongodb.com/docs/manual/reference/operator/query/

crear indice 2dsphere https://www.mongodb.com/docs/manual/core/2dsphere/

busqúeda como el like de sql
https://stackoverflow.com/questions/3305561/how-to-query-mongodb-with-like

crear índice

https://www.mongodb.com/docs/manual/reference/method/db.collection.createIndex/