Facultad de Informática – Universidad Nacional de La Plata

Cátedra de Bases de Datos II

Bases de Datos 2 2022 -TP3 - Grupo 3

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.
   - **Tabla / Relación:** El concepto de tabla en sí no existe, pero es reemplazado por el de **colecciones**. Las relaciones pueden ser modeladas como **documentos embebidos** o a través de **referencias**.
   - **Fila / Tupla:** A lo que se conoce como fila en los RDBMS, en MongoDB se lo conoce como **documento**.
   - **Columna:** En MongoDB, no existe el concepto de columnas, ya que las colecciones de MongoDB no tienen un esquema por defecto, es decir, **los documentos de una colección no necesitan tener el mismo conjunto de campos** y el tipo de datos para un campo puede diferir entre los documentos dentro de una colección.

Fuente:
https://www.mongodb.com/docs/manual/core/databases-and-collections/

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

En MongoDB las operaciones realizadas sobre un solo documento son atómicas, aún si la operación involucra múltiples documentos embebidos dentro del mismo.
Por otra parte si se realizara una modificación sobre varios documentos (ej: db.collection.updateMany()), la modificación de cada uno de los documentos es atómica, pero no será atómica la operación como un todo.
Por otra parte, MongoDB tiene soporte para transacciones atómicas a nivel multi-documento a partir de la versión 4.0. Desafortunadamente las transacciones de este tipo conllevan un gran costo en cuanto al rendimiento de las operaciones realizadas.

Fuentes:

https://docs.mongodb.com/manual/core/transactions/

https://www.mongodb.com/docs/manual/core/data-modeling-introduction/

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

- **Single Field Index**: De forma predeterminada, todas las colecciones tienen un índice en el campo _id, y las aplicaciones y los usuarios pueden agregar índices adicionales en cualquier campo del documento para admitir consultas y operaciones importantes.
- **Compound Index:** Permite la creación de un índice basado en múltiples campos del documento. El orden de los campos de un índice compuesto tiene importancia. Por ejemplo, si un índice compuesto consiste en **{ userid: 1, score: -1 }**, el índice ordena primero por userid y luego, dentro de cada valor de userid, ordena por score.
- **Multikey index**: Son índices compuestos por matrices, MongoDB crea además índices para cada uno de los elementos de la matriz. Los índices multiclave se pueden construir sobre matrices que contienen valores escalares (por ejemplo, strings, números) y documentos anidados.
- **Geospatial index:** Este tipo de índices se utiliza para indexar coordenadas geoespaciales.
- **Text index:** Como el nombre lo indica permite crear índices para campos que tienen strings en su contenido o una matriz de strings.
- **Hashed index:** Son índices formados por el valor del hash de un dato en un campo.

Fuente:

https://www.mongodb.com/docs/manual/indexes/

4. ¿Existen claves foráneas en MongoDB?

No, en MongoDB no existen claves foráneas. Dado esto nunca habrá restricciones al borrado o a la modificación, o una operación "en cascada" como resultado.

Sin embargo, en MongoDB se pueden utilizar los documentos embebidos para representar las relaciones entre los datos, pero también se pueden utilizar mecanismos de multi-documentos para obtener modelos normalizados. En este último caso, las relaciones se representan mediante referencias entre los documentos, parecidas al sistema de claves foráneas de las bases de datos relacionales.

Fuentes:

https://docs.mongodb.com/manual/core/data-model-design/#data-modeling-embedding
https://docs.mongodb.com/manual/core/data-model-design/#data-modeling-referencing
https://docs.mongodb.com/manual/reference/database-references/

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.

Nota
Para los ejercicios, realizamos los siguientes pasos:

1. Iniciamos el servicio de MongoDB:

```
$ sudo systemctl start mongodb
```

2. Accedemos al shell de MongoDB:

```
$ mongo
```

3. Ingresamos los comandos para realizar las operaciones correspondientes.

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?

```
> use vaccination
switched to db vaccination

> db.nurses.insertOne({name:"Morella Crespo", experience:9})
{
   "acknowledged" : true,
   "insertedId" : ObjectId("628c7ff29d3be00896eb442a")
}


> db.nurses.find().pretty()
{
      "_id" : ObjectId("628c7ff29d3be00896eb442a"),
      "name" : "Morella Crespo",
      "experience" : 9
}
```

   Nota:

Las consultas pueden ser ejecutadas ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ5/EJ5-queries.js
```

Luego de crear el documento, se agregó el campo “_id”. Esto se debe a que cada documento almacenado en una colección requiere un campo _id único que actúa como clave principal. Si se omite el campo _id al insertar el documento, MongoDB genera automáticamente un ObjectId para el campo _id.

▶ 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']}
```

Y busque los enfermeros:

- de 5 años de experiencia o menos
- que hayan aplicado la vacuna “Pfizer”
- que no hayan aplicado vacunas (es decir, que el atributo *vaccines* esté ausente)
- de apellido ‘Fernández’
- con 6 o más años de experiencia y que hayan aplicado la vacuna ‘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.

Ejecutamos la siguiente **consulta** para insertar los documentos en la colección nurses:

```
use vaccination
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"
        ]
    }
])
```

Nota:
Las consultas pueden ser ejecutadas ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ6/EJ6-insertMany.js
```

Como **resultado** obtuvimos:

```
{
	"acknowledged" : true,
	"insertedIds" : [
		ObjectId("628ef073afb9e45858ab869e"),
		ObjectId("628ef073afb9e45858ab869f"),
		ObjectId("628ef073afb9e45858ab86a0"),
		ObjectId("628ef073afb9e45858ab86a1")
	]
}
```

A. Enfermeros de 5 años de experiencia o menos

Consulta

```
use vaccination
db.nurses.find( { experience: { $lt: 5} })
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ6/EJ6-queryA.js
```

Resultado

```
{ "_id" : ObjectId("628ef073afb9e45858ab86a0"), "name" : "Gonzalo Gallardo",
"experience" : 3 }
```

B. Enfermeros Que hayan aplicado la vacuna “Pfizer”

Consulta

```
use vaccination
db.nurses.find( { vaccines: "Pfizer" } )
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ6/EJ6-queryB.js
```

Resultado

```
{ "_id" : ObjectId("628ef073afb9e45858ab869f"), "name" : "Honoria Fernández",
"experience" : 5, "vaccines" : [ "Pfizer", "Moderna", "Sputnik V" ] }
{ "_id" : ObjectId("628ef073afb9e45858ab86a1"), "name" : "Altea Parra",
"experience" : 6, "vaccines" : [ "Pfizer" ] }
```

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

Consulta

```
use vaccination
db.nurses.find( { vaccines: { $exists: false } } )
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ6/EJ6-queryC.js
```

Resultado

```
{ "_id" : ObjectId("628ef073afb9e45858ab86a0"), "name" : "Gonzalo Gallardo",
"experience" : 3 }
```

D. Enfermeros de apellido ‘Fernández’

Consulta

```
use vaccination
db.nurses.find( { name: { $regex: '\s*Fernández' } } )
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ6/EJ6-queryD.js
```

Resultado

```
{ "_id" : ObjectId("628ef073afb9e45858ab869f"), "name" : "Honoria Fernández",
"experience" : 5, "vaccines" : [ "Pfizer", "Moderna", "Sputnik V" ] }
```

E. Enfermeros con 6 o más años de experiencia y que hayan aplicado la vacuna 'Moderna'

Consulta

```
use vaccination
db.nurses.find( { $and: [ { experience: { $gte: 6} }, { vaccines: "Moderna" }
] } )
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ6/EJ6-queryE.js
```

Resultado

```
{ "_id" : ObjectId("628ef073afb9e45858ab869e"), "name" : "Gale Molina",
"experience" : 8, "vaccines" : [ "AZ", "Moderna" ] }
```

F. 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.

Consulta

```
use vaccination
db.nurses.find( { $and: [ { experience: { $gte: 6} }, { vaccines: "Moderna" }
] }, {name:1, _id:0} )
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ6/EJ6-queryF.js
```

Resultado

```
{ "name" : "Gale Molina" }
```

▶ 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.

Consulta

```
use vaccination
db.nurses.updateOne( { name: "Gale Molina" }, { $set: { experience: 9 } })
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ7/EJ7-query.js
```

Resultado

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

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

Consulta

```
use vaccination
db.nurses.updateOne( { name: "Gonzalo Gallardo" }, { $set: { vaccines: [] } })
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ8/EJ8-query.js
```

Resultado

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

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

Consulta

```
use vaccination;
db.nurses.updateOne( { name: "Altea Parra" }, { $push: { vaccines: "AZ" } })
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ9/EJ9-query.js
```

Resultado

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
```

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

Consulta

```
use vaccination;
db.nurses.updateMany( { vaccines: "Pfizer" }, {  $mul: { experience: 2 } } )
```

Nota:
La consulta puede ser ejecutada ingresando el siguiente comando en la terminal:

```
mongo < ./tp3/queries/EJ1/EJ10-query.js
```

Resultado

```
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 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)
                  });
            }}
```

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

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.
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.

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.
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.
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.

▸ 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.