Grupo 7:
Cristóbal Santiago
Levy Andrés
Pugliese Alejo

Bases de Datos 2 2022 -TP3

Bases de Datos NoSQL / Práctica con MongoDB

entrega: 13/6

Parte 1: Bases de Datos NoSQL y Relacionales

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

MongoDB es un sistema gestor de bases de datos no relacionales orientado a documentos y lo mejor de todo es que es código abierto.
MongoDB utiliza una estructura dinámica en formato JSON (Que en mongoDB se llama BSON pero es básicamente lo mismo), que significa que aunque tengas objetos definidos en la misma colección no necesariamente tendrán las mismas características. Esto es interesante porque el programador tiene la libertad de manejar su información como más le convenga mientras no cometa un error.
La forma en que MongoDB almacena datos también es diferente. El concepto de base de datos existe, en cambio, en lugar de usar tablas, columnas y filas, MongoDB usa colecciones, objetos y campos.

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

Las operaciones en MongoDB son atómicas, y con la utilización de documentos embebidos y el uso de arrays para capturar relaciones entre multiples colecciones permite obviar el uso de transacciones entre multiple documentos.
Aunque en caso de necesitar el uso de transacciones las mismas no pueden escribir o leer de las bases de datos “admin”, “config” o locales. Tampoco pueden escribir a las bases de datos de system. Toda transacción que requiera múltiples shards va a ser abortada si alguno de los shards posee un árbitro.

El alcance de las transacciones en MongoDB es definido por el “read concern”, que específica el alcance de las operaciones de la transacción. El mismo puede ser seteado al inicio de la misma. En caso de no ser seteado de manera explícita se utiliza el de la session. Si la session tampoco lo tiene seteado se utiliza por defecto el “read concern” “local.
Además del “read concern” mongo tiene limitaciones en las transacciones. Las más destacables son:

No se puede crear colecciones nuevas en transacciones de escritura en múltiples shards.

No se puede escribir en colecciones de tamaño fijo.

Entre otras limitaciones.

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

Los índices en MongoDB se generan en forma de Árbol B. Es decir, que los datos se guardan en forma de árbol, pero manteniendo los nodos balanceados. Esto incrementa la velocidad a la hora de buscar y también a la hora de devolver resultados ya ordenados. De hecho MongoDB es capaz de recorrer los índices en ambos sentidos, por lo que con un solo índice, podemos conseguir ordenación tanto ascendente como descendente.
Los tipos de índices soportados por MongoDB son:

- Single field
- Compound index
- Multikey index
- Geospatial index
- Text index
- Hashed index
- Existen también otros tipos de índices como son los índices para geoposicionamiento.
- Indexación de subdocumentos
- Indexación de arrays

4. ¿Existen claves foráneas en MongoDB?

No, no existe el concepto de claves foráneas. Aunque se pueden especificar referencias entre colecciones. Existen las relaciones manuales o “DBRefs”.
DBRef es diminutivo de “Database Reference” y hace referencia a una convención para referenciar documentos sin tipo específico. Se nombra la colección de origen junto con los nombres de uno o varios campos, y a veces también el de la base de datos.
Las relaciones manuales pueden ser creadas con el _id, y las relaciones de DBRefs con los campos $ref, $id y $db, donde el último es opcional y denota la base de datos donde el documento referenciado reside. Además se pueden crear inner joins con la operación de agregación $lookup, vinculando por alguno de los campos referencia nombrados anteriormente.

Parte 2: Primeros pasos con MongoDB

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

db.vaccination.nurses.insertOne( { name: "Morella Crespo" ,
experience: 9 } )

db.nurses.find( { name: "Morella Crespo" , experience: 9 }
).pretty()
```

{ _id: ObjectId("629e7dd6b91c7dee507df01b"),
name: 'Morella Crespo',
experience: 9 }

La diferencia es que se agrega el atributo _id que utiliza el identificador UUID del objeto para su identificación.

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

```
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:
{ '0': ObjectId("629e7df6b91c7dee507df01c"),
'1': ObjectId("629e7df6b91c7dee507df01d"),
'2': ObjectId("629e7df6b91c7dee507df01e"),
'3': ObjectId("629e7df6b91c7dee507df01f") } }

Y busque los enfermeros:
- de 5 años de experiencia o menos

```
db.nurses.find( { experience: { $lte: 5 } } )
```

```
{ _id: ObjectId("629e7df6b91c7dee507df01d"),
  name: 'Honoria Fernández',
  experience: 5,
  vaccines: [ 'Pfizer', 'Moderna', 'Sputnik V' ] }
{ _id: ObjectId("629e7df6b91c7dee507df01e"),
  name: 'Gonzalo Gallardo',
  experience: 3 }
```

- que hayan aplicado la vacuna “Pfizer”

```
db.nurses.find( { vaccines: { $in: ["Pfizer"] } } )
```

```
{ _id: ObjectId("629e7df6b91c7dee507df01d"),
  name: 'Honoria Fernández',
  experience: 5,
  vaccines: [ 'Pfizer', 'Moderna', 'Sputnik V' ] }
{ _id: ObjectId("629e7df6b91c7dee507df01f"),
  name: 'Altea Parra',
  experience: 6,
  vaccines: [ 'Pfizer' ] }
```

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

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

```
{ _id: ObjectId("629e7dd6b91c7dee507df01b"),
  name: 'Morella Crespo',
  experience: 9 }
{ _id: ObjectId("629e7df6b91c7dee507df01e"),
  name: 'Gonzalo Gallardo',
  experience: 3 }
```

- de apellido ‘Fernández’

```
db.nurses.find( { name: { $regex: "Fernández" , $options: "xi" } }
)
```

```
{ _id: ObjectId("629e7df6b91c7dee507df01d"),
```

name: 'Honoria Fernández',
experience: 5,
vaccines: ['Pfizer', 'Moderna', 'Sputnik V'] }

- 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.

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

{ _id: ObjectId("629e7df6b91c7dee507df01c"),
name: 'Gale Molina',
experience: 8,
vaccines: ['AZ', 'Moderna'] }

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

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

{ acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0 }

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

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

{ acknowledged: true,
insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0 }

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

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

{ acknowledged: true,

insertedId: null,
matchedCount: 1,
modifiedCount: 1,
upsertedCount: 0 }

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

```
db.nurses.updateMany( { vaccines: { $in: [ "Pfizer" ] } }, { $mul:
{ experience: Int32( 2 ) } } )
```

{ acknowledged: true,
insertedId: null,
matchedCount: 2,
modifiedCount: 2,
upsertedCount: 0 }

Parte 3: Índices

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

```
db.doses.getIndexes()
```

[{ v: 2, key: { _id: 1 }, name: '_id_' }]

Esto indica que solamente tiene el índice de _id, que por defecto crea Mongo en la creación de cada documento. El cual no se puede eliminar y es utilizado para controlar la unicidad de este campo.

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.

Sin Indexes:

```
db.doses.find( { nurse: { $regex: "11" } }
).explain("executionStats")
```

nReturned: 12570,
executionTimeMillis: 201,
totalKeysExamined: 0,
totalDocsExamined: 202162,

Con:

```
db.doses.createIndex( { nurse: 1 } , { name: "nurse_indexes" } )
```

```
db.doses.find( { nurse: { $regex: "11" } }
).explain("executionStats")
```

nReturned: 12570,
executionTimeMillis: 450,
totalKeysExamined: 202162,
totalDocsExamined: 12570,

La cantidad de tiempo en milisegundos que tarda es mayor con índices. Esto se debe a que, aunque la cantidad de documentos examinados haya sido menor, el overhead creado al revisar primero los índices y después buscar los documentos terminó siendo mayor al costo normal de buscar entre esta cantidad tan baja de

documentos. En cambio si el número de documentos fuese exponencialmente superior al actual, la consulta con índices mantendría un tiempo de ejecución menor.

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.

```
var caba = ( {
 "type":"MultiPolygon",
 "coordinates": [ [ [
   [-58.46305847167969,-34.53456089748654],
   [-58.49979400634765,-34.54983198845187],
   [-58.532066345214844,-34.614561581608186],
   [-58.528633117675774,-34.6538270014492],
   [-58.48674774169922,-34.68742794931483],
   [-58.479881286621094,-34.68206400648744],
   [-58.46855163574218,-34.65297974261105],
   [-58.465118408203125,-34.64733112904415],
   [-58.4585952758789,-34.63998735602951],
   [-58.45344543457032,-34.63603274732642],
   [-58.447265625,-34.63575026806082],
   [-58.438339233398445,-34.63038297923296],
   [-58.38100433349609,-34.62162507826766],
   [-58.38237762451171,-34.59251960889388],
   [-58.378944396972656,-34.5843230246475],
   [-58.46305847167969,-34.53456089748654]
 ] ] ]
} )

db.patients.find( { address: { $geoWithin: { $geometry:caba } } } )
```

nReturned: 44204,
executionTimeMillis: 790,
totalKeysExamined: 0,
totalDocsExamined: 202162,

```
db.patients.createIndex( { address: "2dsphere" } )
```

nReturned: 44204,
executionTimeMillis: 535,
totalKeysExamined: 55784,
totalDocsExamined: 55765,

En este caso es más eficiente la utilización de índices dado que 2dsphere es un campo múltiple por lo que tienen que comparar con cada uno de los campos y, con la cantidad de datos actuales, es más eficiente buscarlo en el índice a comparar entre ambos campos por cada documento.

Parte 4: Aggregation Framework

14. Obtenga 5 pacientes aleatorios de la colección.

```
db.patients.aggregate( [ { $sample: { size: 5 } } ] )
```

{ _id: ObjectId("629e8507e69dbd73a026be39"),
name: 'Paciente 19879',
address:
{ type: 'Point',
coordinates: [-58.58222838748483, -34.71296927687901] } }
{ _id: ObjectId("629e859be69dbd73a028bf0f"),
name: 'Paciente 85522',
address:
{ type: 'Point',
coordinates: [-58.51192085374275, -34.76913638220495] } }
{ _id: ObjectId("629e8507e69dbd73a026bc9f"),
name: 'Paciente 19674',
address:
{ type: 'Point',
coordinates: [-58.595665411180306, -34.66801648551575] } }
{ _id: ObjectId("629e8622e69dbd73a02acbb3"),
name: 'Paciente 152676',
address:
{ type: 'Point',
coordinates: [-58.46213862271545, -34.77887395587042] } }
{ _id: ObjectId("629e84e2e69dbd73a0263be3"),
name: 'Paciente 3196',
address:
{ type: 'Point',
coordinates: [-58.49531822166145, -34.7885434851647] } }

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

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

```
    $out: "closeByPatients"
  }
])
```

El operador “geoNear” se utiliza para ubicar geográficamente objetos, al pasarle dentro del parámetro “near” las coordenadas del centro de Buenos Aires y “maxDistance” 1000, estamos pidiendo los pacientes que estén dentro de los 1000 metros (1km) del centro de la Ciudad.
A la salida de esta consulta, que es una colección con los pacientes que cumplen la condición, utilizando el operador “out” los guardamos en la nueva colección “closeByPatients”.

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.

```
db.doses.aggregate( [
  {
    $lookup:
    {
      from: "closeByPatients",
      localField: "patient",
      foreignField: "name",
      as: "patientData"
   }
  },{
      $match:
        {
          "patientData": {$ne: []}
        }
  },{
      $project:
        {
          patientData: 0
        }
  }
])
```

A la colección de dosis, la unimos con la colección creada en el punto anterior (“closeByPatients”) mediante el operador “lookup”, realizamos la vinculación por el campo local “patient”, el cual es “name” en la otra colección, y guardamos la información en el campo nuevo “patientData”.
Esta unión va a poblar el campo “patientData” de cada dosis con un arreglo que contiene a todos los pacientes cuyo “name” coincide con “patient” de la dosis. Y será vacío en caso de no haber ninguno.
Para descartar aquellos casos donde “patientData” está vacío, utilizamos “match” para descartarlos.

Luego y por último, quitamos de la proyección el campo “patientData” para quedar únicamente con la colección solicitada de dosis que cumplieran la condición dada por el enunciado.

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.

```
db.nurses.aggregate( [
      {
            $match: {name:/111/}
      },{
            $lookup:
{
      from: "doses",
      pipeline: [
        {
$match:
            {
  "date":{
$gt:new ISODate("2021-05-01")
}
}
        }
      ],
      localField: "name",
      foreignField: "nurse",
      as: "doses"
            }
      }
] )
```

En primer lugar, utilizamos el operador “match” para quedarnos únicamente con aquellos enfermeros que tengan “111” en su nombre.
Luego, utilizamos “lookup” para unirlo con la colección de dosis, pero, a diferencia del anterior uso de “lookup”, esta vez utilizamos el parámetro “pipeline” para ejecutar operaciones sobre la colección con la que se va a unir. A la colección de dosis, antes de unirla, vamos a ejecutarle un operador “match” para quedarnos únicamente con aquellas que cumplan con la condición de fecha (ser posteriores al 01/05/2021).
Luego el “lookup” sigue de la misma forma que se vio antes, trayendo las dosis cuyo campo “nurse” coincida con el campo “name” de la colección de enfermeros. Y el campo agregado a los enfermeros será un arreglo con todas las dosis que hayan aplicado.