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: este concepto existe también en MongoDB.
- Tabla / Relación: este concepto no existe en MongoDB. La alternativa son las 'Colecciones' y la diferencia con las 'Tablas' es que son grupos de documentos en lugar de grupos de registros.
- Fila / Tupla: este concepto no existe en MongoDB. La alternativa es 'Documento'.
- Columna: este concepto no existe en MongoDB. La alternativa son los 'Campos' que son pares clave/valor.

RDBMS	MongoDB
Base de datos	Base de datos
Tabla	Colección
Fila	Documento
Columna	Campo
Índice	Índice
Join	Documento embebido
Clave foránea	Referencia

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 base de datos relacionales te bloquea toda la tabla cuando se realiza alguna operación CRUD, en Mongodb se bloquea solo el documento a modificar, por lo que es un sistema de base de datos compatible con ACID a nivel de documento.
Con la versión 4.0 se habilitan transacciones ACID a nivel multi-documento en MongoDB. Se estima que cerca de un 90% de modelos que utilicen un modelo de documentos no necesitaran transacciones en MongoDB. Aun así, existen casos donde transacciones multi-documento o multi-colección son la mejor alternativa. Las transacciones en MongoDB trabajan de forma similar a las transacciones en otras bases de datos. Para usarlas primero hay que abrir una sesión y utilizarla para ejecutar alguna operación CRUD entre documentos, colecciones o incluso shards.

Fuente:
https://www.mongodb.com/basics/acid-transactions
https://www.iteramos.com/pregunta/8978/Que-significaba-realmente-que-MongoDB-no-fuera-compatible-con-ACID-antes-de-la-v4

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

- Single Field: Estos índices se aplican a un solo campo de una colección.
  Para declarar un índice de este tipo debemos usar un comando similar a este:

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

El número indica que queremos que el índice se ordene de forma ascendente. Si quisiéramos un orden descendente, el parámetro será un -1.

- Compound index: 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
  "María", 21
  "Pedro", 19
  "Unai", 34
  "Unai", 27

  Lo bueno de los índices compuestos, es que podemos usarlos para consultar uno o varios de los campos, sin que sea necesario incluirlos todos. En el ejemplo anterior el índice se puede utilizar siempre que se hagan consultas sobre user_name o sobre user_name y age. Al fin y al cabo este índice compuesto ordenará el campo user_name de la misma manera que si creamos un índice simple. Lo que no podemos hacer es buscar sólo sobre el campo “age”. Para ese caso tendríamos que crear un índice específico.
  Si el índice compuesto tuviera tres campos, podríamos utilizarlo al consultar sobre el primer campo, sobre el primer y segundo campo, o sobre los tres campos.

- Multikey index: Son aplicados a campos que tienen información contenida en arreglos. Al usar estos índices, MongoDB crea una entrada para cada uno de los elementos del arreglo.

- Geospatial index: La indexación geoespacial de MongoDB permite ejecutar de manera eficiente consultas espaciales en una colección que contiene formas y puntos geoespaciales.

- Text index: MongoDB proporciona índices de texto para admitir consultas de búsqueda de texto sobre el contenido de un string. Los índices de texto pueden incluir cualquier campo cuyo valor sea un string o una matriz de strings. Una colección solo puede tener un índice de búsqueda de texto, pero ese índice puede cubrir varios campos.

- Hashed index: Los índices hash mantienen entradas con hash de los valores del campo indexado.

Los índices hash admiten la fragmentación mediante claves de fragmentación hash. La fragmentación basada en hash utiliza un índice hash de un campo como clave de partición para dividir los datos en su clúster fragmentado.

El uso de una clave fragmentada con hash para fragmentar una colección da como resultado una distribución más uniforme de los datos.
Los índices hash utilizan una función hash para calcular el hash del valor del campo de índice. La función hash colapsa los documentos incrustados y calcula el hash para el valor completo, pero no admite índices de varias claves (es decir, matrices). Específicamente, crear un índice hash en un campo que contiene una matriz o intentar insertar una matriz en un campo indexado hash devuelve un error.

4. ¿Existen claves foráneas en MongoDB?
El término clave foránea no existe en MongoDB, si existe el término referencia.
Necesitamos mantener la integridad de los datos por nuestra cuenta .
Por ejemplo:

```
student
{
  _id: ObjectId(...),
  name: 'Jane',
  courses: ['bio101', 'bio102']   // <= ids de los courses
}

course
{
  _id: 'bio101',
  name: 'Biology 101',
  description: 'Introduction to biology'
}
```

En courses, el campo contiene _id de cursos. Es fácil definir una relación de uno a muchos. Sin embargo, si queremos recuperar los nombres de los cursos de los estudiantes Jane necesitamos realizar otra operación para recuperar el documento course a través de _id .
Si el curso bio101 es eliminado, necesitamos realizar otra operación para actualizar el courses en el campo del documento de student.

Parte 2: Primeros pasos con MongoDB

A PARTIR DE ESTA PARTE EMPIEZAN LAS CONSULTAS, EJECUTAR LAS QUE ESTÁN EN EL TEXTO PLANO SUBIDO, NO ESTAS. SUBIMOS LAS CONSULTAS EN LOS DOS ARCHIVOS PORQUE HAY IMÁGENES Y EXPLICACIONES EN ALGUNOS PUNTOS. EJECUTANDO LAS CONSULTAS DE ESTE ARCHIVO NO CORREN PORQUE EL FORMATO CAMBIA AL HACER COPY PASTE, CORRER LAS DEL TEXTO PLANO.

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?

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Crear una nueva base de datos llamada vaccination:

- use vaccination

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Crear la colección nurses:

- db.createCollection("nurses")

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Insertar una enfermera en la colección:

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

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Recuperar la información de la enfermera:

- db.nurses.find().pretty()

A parte de los atributos que se insertaron, aparece el campo _id que es de tipo ObjectId.

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']}

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

- db.nurses.insertMany([{name:'Gale Molina', experience:8, vaccines:['AZ','Moderna']},{name:'Honoria Fernandez', experience:5, vaccines:['Pfizer','Moderna','Sputnik V']},{name:'Gonzalo Gallardo',experience:3},{name:'Altea Parra',experience:6,vaccines:['Pfizer']}])

Y busque los enfermeros:

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

De 5 años de experiencia o menos:

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

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Que hayan aplicado la vacuna “Pfizer”:

- db.nurses.find({vaccines: "Pfizer"})

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

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

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

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

De apellido ‘Fernández’

- db.nurses.find({name:/Fernandez$/})

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

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

- db.nurses.find({experience:{$gte:6},vaccines:'Moderna'})

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

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:'Moderna'},{"name":1,_id:0})

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

7. Actualice a “Gale Molina” cambiándole la experiencia a 9 años.
db.nurses.updateOne({name: "Gale Molina"}, {$set: {experience: 9}})

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

8. Cree el array de vacunas (vaccines) para ”Gonzalo Gallardo".
db.nurses.updateOne({name: 'Gonzalo Gallardo'},{$set: {vaccines:[]}})

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

9. Agregue “AZ” a las vacunas de “Altea Parra”.
db.nurses.updateOne({name: 'Altea Parra'},{$addToSet: {vaccines:'AZ'}})

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

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

db.nurses.updateMany({vaccines: '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(). Si utiliza un cliente que lo permita (ej. Robo3T), se puede ejecutar directamente en el espacio de consultas

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Eliminar a todos los enfermeros de la colección:

- db.nurses.deleteMany({})

Guarde en un archivo llamado 'generador.js' el siguiente código JavaScript y ejecútelo con: load():

- load("<ruta>/generador.js")

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

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Obtener los índices de la colección doses:

- db.doses.getIndexes()

Hay un índice _id que se crea por defecto cuando se crea la collection. Este índice controla la unicidad del valor del _id. No se puede eliminar este índice.

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.

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Crear el índice para el campo nurse de la colección doses:

- db.doses.createIndex({nurse:1})

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Busque las dosis que tengan en el nombre del enfermero el string "11" y utilice el método explain("executionStats"):

- db.doses.find({nurse:/11/}).explain("executionStats")

Búsqueda con índice:

Se analizaron 13002 documentos (valor en totalDocsExamined) y tardo 193 milisegundos (valor en executionTimeMillis)

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Para borrar el índice se usa db.doses.dropIndex({nurse:1})
Búsqueda sin indice:

Se analizaron 199946 documentos (valor en totalDocsExamined) y tardo 130 milisegundos (valor en executionTimeMillis)

Con el índice la búsqueda tarda más que sin el índice.

Está bien que pase esto ya que no queda otra que buscar en cada documento si el nombre del enfermero tiene incluido el string 11. No está utilizando ningún campo en particular para que el índice se utilice apropiadamente, es como el LIKE %% en la búsqueda con SQL. La búsqueda con índice va a implicar más operaciones que sin el índice, por eso tarda más.

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.

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)
Creamos el índice pedido:

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

Nos guardamos en una variable el multipoligono con las coordenadas de BsAs

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

- var ubicaciones ={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]]]]}

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Realizamos la búsqueda de los paciente que vivan en BsAs:

- db.patients.find({address: {$geoIntersects: {$geometry: ubicaciones}}}).explain("executionStats")

Se analizaron 55574 documentos (valor en totalDocsExamined) y tardo 379 milisegundos (valor en executionTimeMillis)

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Para borrar el índice se usa db.patients.dropIndex({address: "2dsphere" })

Búsqueda sin índice:

Se analizaron 199946 documentos (valor en totalDocsExamined) y tardo 507 milisegundos (valor en executionTimeMillis)

Con el índice tarda menos que realizar la búsqueda sin el índice, además de analizar menos documentos que sin tener el índice. Esto es correcto ya que se está utilizando un campo en particular que matchea en los documentos de la colección para realizar una búsqueda más eficiente.

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.

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

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.

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

Como en el punto 13) eliminamos el índice para hacer las pruebas que pedía, para este punto se debe volver a crear:

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

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

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

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.

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

db.doses.aggregate([{$lookup: {from:"patientsBsAs", localField:"patient", foreignField:"name", as: "array"}}, {$match: {"array": {$ne:[]}}}, {$project: {"nurse": 1, "patient": 1, "vaccine": 1, "date": 1}},{$out: "dosesPatientsBsAs"}])

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

(USAR EL COMANDO EN EL ARCHIVO DE TEXTO PLANO)

db.nurses.aggregate([{$match:{name:/111/}},{$lookup:{from:"doses",localField:"name",foreignField:"nurse",as:"doses"}} , {$project: {name:1,experience:1,tags:1,doses: {$filter: {input: "$doses", as: "dose", cond:{$gte: ["$$dose.date", new ISODate("2021-05-01T00:00:00Z")]}} }}}, {$out:"nursesWithDoses"}])