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

Bases de datos → existen en Mongo

Tabla/Relación → Las tablas serían las “colecciones” y relaciones en si no hay. Pero se pueden establecer relaciones entre documentos embebidos (documentos de distintas colecciones, hasta incluso de distintas bases).

Fila/Tupla → Serían los “documentos”

Columna → Serían (en Mongo) los campos/atributos.

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

El alcance era a nivel de un único documento, incluso si la operación modifica múltiples documentos embebido en uno solo.

A partir de la versión 4.0.0 Mongo provee soporte para transacciones multi-documento (varios documentos de una colección).

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

soporta?

Single field → Además del \_id definido por MongoDB, este admite la creación de índices ascendentes/descendentes definidos por el usuario en un solo campo de un documento.

Compound index → MongoDB también admite índices definidos por el usuario en múltiples campos de un documento. (el orden de los campos de un índice compuesto es importante)

Multikey index → Para indexar arrays. Si se indexa un campo que contiene como valor un arreglo, Mongo crea índices separados para cada elemento del arreglo.

Geospatial index → Para admitir eficientemente consultas de coordenadas geoespaciales. Mongo proporciona dos índices especiales: “2d” que usan geometría plana para devolver resultados y “2dsphere” que utiliza geometría esférica.

Text index → Índices que admiten la búsqueda de strings en una colección.

Hashed index → Para admitir fragmentación basada en hash , MongoDB proporciona un tipo de índice hash, que indexa el valor del hash sobre los campos.

4/ ¿Existen claves foráneas en MongoDB?

En MongoDB no existen claves foráneas sino que se utilizan referencias entre documentos. Hay dos maneras de relacionar documentos:

Referencias manuales → Se guarda el campo \_id del documento referenciado en el documento que contiene la referencia.

DBRefs → son referencias desde un documento a otro usando el valor del campo \_id del primer documento, el nombre de la colección y, opcionalmente el nombre de la BD.

“Mejor usar referencias manuales que DBRefs”

Parte 2: Primeros pasos con MongoDB

5/ Cree una nueva base de datos llamada airbdb, y una colección llamada apartments. En esa

colección inserte un nuevo documento (un departamento) con los siguientes atributos:

{name:’Apartment with 2 bedrooms’, capacity:4}

recupere la información del departamento usando el comando db.apartments.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 →

1° use airbdb; (resultado en consola: switched to db airbdb) → con esto indicamos que vamos a usar esa base de datos.

2° db.apartments.insertOne({name:”Apartment with 2 bedrooms”,

capacity:4})

Con este último creamos airbdb (además de insertar la colección y el documento)

Si busco con el documento con el comando db.apartments.find().pretty() → el documento que nos muestra también tiene el campo “\_id” : ObjectId(“5eb632...”). Campo que identifica unívocamente al documento.

6/ Agregue los siguientes documentos a la colección de departamentos:

{name:’New Apartment', capacity:3, services: ['wifi', 'ac']}

{name:’Nice apt for 6‘, capacity:6, services: ['parking']}

{name:’1950s Apartment‘, capacity:3}

{name:’Duplex Floor’, capacity:4, services: ['wifi', ‘breakfast’, ‘laundry’]}

Y busque los departamentos:

- con capacidad para 3 personas.

- con capacidad para 4 personas o más

- con wifi

- que incluyan la palabra ‘Apartment’ en su nombre

- con la palabra ‘Apartment’ en su nombre y capacidad para más de 3 personas

- sin servicios (es decir, que el atributo esté ausente)

vuelva a realizar la última consulta pero proyecte sólo el nombre del departamento en los

resultados, omitiendo incluso el atributo \_id de la proyección.

**Para agregar los documentos** →

db.apartments.insertOne({name:’New Apartment', capacity:3, services: ['wifi', 'ac']});

db.apartments.insertOne({name:’Nice apt for 6‘, capacity:6, services: ['parking']});

db.apartments.insertOne({name:’1950s Apartment‘, capacity:3});

db.apartments.insertOne({name:’Duplex Floor’, capacity:4, services: ['wifi', ‘breakfast’, ‘laundry’]});

**Buscar con capacidad para 3 personas** → db.apartments.find({capacity:3}).pretty();

Esto retorna a “New Apartment” y a “1950s Apartment”.

**Buscar con capacidad para 4 o más** → db.apartments.find({capacity:{$gte:4}}).pretty();

Esto retorna a "Apartment with 2 bedrooms", "Nice apt for 6" y a "Duplex Floor".

**Con wifi** → db.apartments.find({ services:'wifi'}).pretty();

Retorna "New Apartment" y "Duplex Floor".

**Que incluyan la palabra ‘Apartment’ en su nombre** → db.apartments.find({name:{$regex:/Apartment/}}).pretty();

**Con la palabra ‘Apartment’ en su nombre y capacidad para más de 3 personas** →

db.apartments.find({ name: { $regex: /Apartment/ }, capacity:{$gt:3}}).pretty();

**Sin servicios (es decir, que el atributo esté ausente)** →

db.apartments.find({ services: {$exists:false} }).pretty();

**Vuelva a realizar la última consulta pero proyecte sólo el nombre del departamento en los**

**resultados, omitiendo incluso el atributo \_id de la proyección** →

db.apartments.find({ services: {$exists:false} }, {\_id:0, name:1}).pretty();

7/ Actualice el “Duplex Floor” asignándole capacidad 5 →

db.apartments.updateOne({name:“Duplex Floor”}, {$set : {capacity:5}})

8/ Agregue “laundry” al listado de services del “Nice apt for 6” →

db.apartments.updateOne({ name: "Nice apt for 6" }, { $push: { services:"laundry"}});

9/ Agregue una persona más de capacidad a todos los departamentos con wiﬁ →

db.apartments.updateMany({services: {$in: ['wifi']}},{$inc: {'capacity':1}});

Parte 3: Índices

Eliminar todos los departamentos de la colección → db.apartments.remove({})

Guarde en un archivo llamado ‘generador.js’

el siguiente código JavaScript y ejecútelo con: load(<ruta del archivo ‘generador.js’>) →

load(“c:\\scriptmongo\\generador.js”);

10/ Busque en la colección de departamentos si existe algún índice definido →

db.apartments.getIndexes();

Retorna:

{

“v”:2,

“key”:{

“\_id”:1

},

“name”:”\_id\_”,

“ns”:”airbdb.apartments”

}

11/ Cree un índice para el campo name. Busque los departamentos que tengan en su nombre 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 →

db.apartments.find({ name:{$regex:/11/}}).explain("executionStats"); →

Hasta acá executionTimeMillis : 49

db.apartments.createIndex({name:1});

db.apartments.find({ name:{$regex:/11/}}).explain("executionStats"); →

Acá executionTimeMillis : 115

“Tardó más con índice”

12/ Busque los departamentos dentro de la ciudad de Londres. Para esto, puede definir una

variable en la terminal y asignarle como valor el polígono del archivo provisto

greaterlondon.geojson (copiando y pegando directamente). Cree un índice geoespacial de

tipo 2dsphere para el campo location de la colección apartments y, de la misma forma que

en el punto 11, compare la performance de la consulta con y sin dicho índice →

Creamos un índice “2dsphere” → explicación ejer 14.

db.apartments.createIndex({location:"2dsphere"})

Hay que usar $geoWithin →

$geoWithin

Selecciona documentos con datos geoespaciales que existen completamente dentro de una forma específica.

La forma especificada puede ser un [Polígono](https://docs.mongodb.com/manual/reference/geojson/#geojson-polygon) GeoJSON (de un solo anillo o de varios anillos), un [MultiPolygon](https://docs.mongodb.com/manual/reference/geojson/#geojson-multipolygon) GeoJSON o una forma definida por pares de coordenadas heredados. El [$geoWithin](https://docs.mongodb.com/manual/reference/operator/query/geoWithin/#op._S_geoWithin)operador usa el [$geometry](https://docs.mongodb.com/manual/reference/operator/query/geometry/#op._S_geometry) operador para especificar el objeto [GeoJSON](https://docs.mongodb.com/manual/reference/geojson/#geospatial-indexes-store-geojson) .

var greaterlondon = “pegamos el contenido del archivo greaterlondon.geojson”

db.apartments.find({location:{$geoWithin:{$geometry:greaterlondon}}}).explain(“executionStats”)

con indice → 173

sin indice → 107

Parte 4: Aggregation Framework

load(“c:\\scriptmongo\\generadorReservas.js”);

“Aunque MapReduce es más potente, también es más difícil de utilizar. Para realizar cálculos sencillos es más conveniente utilizar Aggregation Framework”.

13/ Obtenga 5 departamentos aleatorios de la colección →

Randomly selects the specified number of documents from its input.

{ $sample: { size: <positive integer> } }

db.reservations.aggregate([{$group : {\_id : "$apartmentName"}}, {$sample:{size:5}}]);

14/ Usando el framework de agregación, obtenga los departamentos que estén a 15km (o menos) del centro de la ciudad de Londres ([-0.127718, 51.507451]) y guárdelos en una nueva colección.

“Acá es necesario un índice “2dsphere” para luego usar la consulta $geoNear” → y especificar puntos como GeoJSON.

“El 2dsphere índice admite datos almacenados como [objetos GeoJSON](https://docs.mongodb.com/manual/geospatial-queries/#geospatial-geojson) *y* [pares de coordenadas heredados](https://docs.mongodb.com/manual/geospatial-queries/#geospatial-legacy) (consulte también [Restricciones de campo indexadas de 2dsphere](https://docs.mongodb.com/manual/core/2dsphere/#dsphere-data-restrictions) ). Para los pares de coordenadas heredados, el índice convierte los datos en GeoJSON [Point](https://docs.mongodb.com/manual/reference/geojson/#geojson-point).” → hay que agregar un índice al campo “location” (que es type Point) →

db.apartments.createIndex({location:”2dsphere”});

Para **$geoNear** usamos:

“near para indicar el punto → [-0.127718, 51.507451]”

“distanceField para contener la distancia calculada”

“maxDistance para quedarnos con los dep que estén a 15km (o menos)”

“spherical en true para calcular distancias usando geo esférica”

Finalmente usamos **$out** que → “Toma los documentos devueltos por la canalización de agregación y los escribe en una colección específica”

db.apartments.aggregate([{$geoNear: {near: {type: "Point", coordinates: [-0.127718, 51.507451] }, distanceField:"dist.calculated", maxDistance:15000, spherical:true}} ,{ $out: "quincekmapartmentscenterlondon" }]);

15/ Para los departamentos hallados en el punto anterior, obtener una colección con cada departamento agregando un atributo reservas que contenga un array con todas sus reservas. Note que sólo es posible ligarlas por el nombre del departamento.

Con $lookup hacemos una especie de IN entre dos consultas (SQL).

(Entre la colección de la consulta y la que está en “from” - macheando entre el “localField” de quincekmapartmentscenterlondon y el “foreignField” de reservations - en “as” guardo las coincidencias)

db.quincekmapartmentscenterlondon.aggregate([ { $lookup: { from: "reservations", localField: "name", foreignField: "apartmentName", as: "reservas" }} , { $out: "apartments14withreservations" }]);

16/ Usando la colección del punto anterior, obtenga el promedio de precio pagado por reserva

(precio completo, no dividir por la cantidad de noches) de cada departamento.

db.apartments14withreservations.aggregate([{$match:{"reservas":{$exists:true, $ne:[]} }}, {$project:{\_id:0, name:1, averagePrice:{$avg:"$reservas.amount"}}}])