Base de Datos II
Grupo 24
Luiggi Franceschi
Juan Francisco Mannino Sanchez

TP2

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

Base de datos existe en MongoDB y tiene el mismo significado que en los RDBMS relacionales.
Tabla/Relacion no existe, y su equivalente seria una Coleccion.
Fila/Tupla no existe, y su equivalente seria un Documento
Columna tampoco existe, y su equivalente es un Campo

2- MongoDB tiene soporte para transacciones, pero no es igual que el de los RDBMS. ¿Cuál es el alcance de una transacción en MongoDB?
CAP:
Consistency
las réplicas tienen la misma versión de los datos.
El cliente siempre ve la misma información sin importar el nodo.

Availability:
El sistema permanece activo aún con nodos fallando.
Todos los clientes tienen la posibilidad de escribir y de leer.

Partition Tolarance:
El sistema presenta múltiples puntos de entrada debido a fallas en la red que conecta los nodos.

Para el modelo transaccional mongo utiliza la idea BASE, diferente al ACID de base datos relacional.
Basically Available
Soft state
Eventually consistent

Basically Available:
Esta restricción establece que el sistema garantiza la disponibilidad de los datos con respecto al teorema CAP; Habrá una respuesta a cualquier solicitud. Sin embargo, esa respuesta aún podría ser "fracaso" para obtener los datos solicitados o los datos pueden

estar en un estado inconsistente o cambiante, como esperar a que se borre un cheque en su cuenta bancaria.

Soft state:
El estado del sistema podría cambiar con el tiempo, por lo que incluso durante los momentos sin entrada puede haber cambios debido a la "consistencia eventual", por lo que el estado del sistema siempre es "suave".

Eventually consistent:
El sistema eventualmente se volverá consistente una vez que deje de recibir información. Los datos se propagarán a todas partes, tarde o temprano, pero el sistema continuará recibiendo información y no verificará la coherencia de cada transacción antes de pasar a la siguiente.

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

Tipos de indices:

Single field: son indices sobre un solo campo.

Compound index: son indices sobre 2 o mas campos.

Multikey index: son usados para indexar contenido almacenados en arreglos.

Geospatial index: usados para indexar coordenadas geoespaciales. Pueden ser de tipo 2d o 2dsphere, para obtener resultados usando geometria plana o esferica respectivamente.
Text index: permite buscar cadenas de texto dentro de una coleccion.

Hashed index: indexan el hash del valor de un campo, es necesario para el sharding basado en hash.

4- ¿Existen claves foráneas en MongoDB?
Mongo puede referenciar a otros documentos, a traves de un atributo(ej: id), o tener embebido directamente el documento al que se hace referencia., pero no es restrictivo como sql, que al crear una tabla exige que se le agregue la clave que referencia a otra tabla.

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}

use airbdb
db.apartments.insertOne({*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?

db.apartments.find().pretty() → Devuelve

```
{
   "_id" : ObjectId("5eb58f37dcffb18324bf8b64"),
   "name" : "Apartment with 2 bedrooms",
   "capacity" : 4
}
```

El "_id" es un atributo que genera automáticamente mongo, para poder identificar unívocamente cada documento dentro de cada colección dentro de la base de datos.

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']}

db.apartments.insertMany([{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.
db.apartments.find({"capacity": 3}).pretty()
-con capacidad para 4 personas o más
db.apartments.find({"capacity": {$gte: 4 }}).pretty()
-con wifi
db.apartments.find({ "services":"wifi" }).pretty()
-que incluyan la palabra 'Apartment' en su nombre
db.apartments.find({ "name":/Apartment/}).pretty()
-con la palabra 'Apartment' en su nombre y capacidad para más de 3 personas
db.apartments.find({$and: [{ "name":/Apartment/}, {"capacity": {$gt: 3 }}]}).

-sin servicios (es decir, que el atributo esté ausente)
db.apartments.find({"services": {$exists:false}})

- vuelva a realizar la última consulta pero proyecte sólo el nombre del departamento en los resultados, omitiendo incluso el atributo _id

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

7.Actualice el “Duplex Floor” asignándole capacidad 5

db.apartments.update({"name": "Duplex Floor"},{$set: {"capacity": 5}})

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

db.apartments.update({"name": "Nice apt for 6"},{$addToSet: {"services": "laundry"}}) -> <u>añade “laundry” si existe “services”</u>

db.apartments.update({"name": "Nice apt for 6"},{$push {"services": "laundry"}}) -> <u>si no existe “services” lo crea y agrega “laundry”</u>

db.apartments.update({"name": "1950s Apartment"},{$unset: {"services":""}})-> <u>para eliminar un atributo</u>

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

db.apartments.update({ "services":"wifi" }, {$inc: {"capacity": 1}}, {multi: true})

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

db.apartments.getIndexes()

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.createIndex({name:1})
db.apartments.find({ "name":/11/}).explain("executionStats")

CON INDICE
"executionSuccess" : true,
"nReturned" : 2291,
"executionTimeMillis" : 123,
"totalKeysExamined" : 50000,
"totalDocsExamined" : 2291,
}

db.apartments.dropIndexes() -> Borra los indices creados, en este caso “name”

SIN INDICE
executionStats" : {
"executionSuccess" : true,
"nReturned" : 2291,
"executionTimeMillis" : 155,
"totalKeysExamined" : 0,

"totalDocsExamined" : 50000

}

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.

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

db.apartments.find({"location": {$geoWithin: {$geometry: poligono}}}).explain("executionStats")

CON INDICE

"executionStats" : {
"executionSuccess" : true,
"nReturned" : 12263,
"executionTimeMillis" : 821,
"totalKeysExamined" : 18361,
"totalDocsExamined" : 18340,
}

db.apartments.dropIndexes() -> Borra los indices creados, en este caso "location"

SIN INDICE

"executionStats" : {
"executionSuccess" : true,
"nReturned" : 12263,
"executionTimeMillis" : 424,
"totalKeysExamined" : 0,
"totalDocsExamined" : 50000,

}

13.Obtenga 5 departamentos aleatorios de la colección.

db.reservations.aggregate([{$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.

db.apartments.aggregate([
{$geoNear: {
near: { type: "Point", coordinates: [-0.127718, 51.507451] },

```
        distanceField: "dist.calculated",
        maxDistance: 15000,
        spherical: true,
$limit:1000000}},
        {  $out: “nearLondon"        }
])
```

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.

```
db.nearLondon.aggregate([
        {$lookup:{
                from: "reservations", localField: "name",foreignField:
        "apartmentName",as: "reservas"}
        }
])
```

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.

```
var lookup = {$lookup:{
from: "reservations", localField: "name",foreignField: "apartmentName",as:
"reservas"}
};
var unwind = {$unwind: "$reservas"};
var avg = {$group: {_id: "$name", average: {$avg: "$reservas.amount"}}};
db.nearLondon.aggregate([lookup, unwind, avg])
```

para comprobar que funciona se utilizo {$match: {"name": "Apartment 39367"}} para calcular el promedio de sus reservas.
(697.8 + 242.5 + 685.68 + 648.84)/4 = 568.705