



Bases de Datos 2 2022 -TP3

Bases de Datos NoSQL / Práctica con MongoDB
Grupo #26 - Colla, Agustín - Piñero, Fausto, - Mena, María Elena

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?

Analogías entre los conceptos

RDBMS	MongoDB
Base de Datos	existe y se conoce con la misma denominación: Base de Datos
Tabla / Relación	Existe y se conoce como Colección : conjunto de documentos similares, cuya estructura es similar o que representan el mismo concepto de dominio.
Fila / Tupla	existe y se conoce como Documento : son documentos con formato JSON como formato requerido y éste se entiende como una entidad que agrupa los campos y sus valores
Columna	existe y se conoce como Campo : “name” según el estándar JSON, vinculado a un documento, como pares “clave/valor” sin ningún orden en particular

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

MongoDB por defecto garantiza atomicidad a nivel documento. Como generalmente cada documento suele tener asociado datos que en un modelo relacional se distribuirá en varias tablas, es deseable garantizar atomicidad en operaciones complejas que abarcan varios documentos, incluso de distintas colecciones y registros. A partir de la versión 4.0, MongoDB provee transacciones entre múltiples documentos.¹

¹ Referencia: <https://platzi.com/tutoriales/1533-mongodb/3757-transacciones-en-mongodb-2/>

Basándonos en la Teoría: En el mundo de NoSQL, se busca trabajar en un escenario donde se tienen tres elementos que compiten entre ellos: Disponibilidad, Consistencia, Soporte de Fallos y considerando el Teorema de Brewer (que aplica para MongoDB: “es imposible para un sistema computacional distribuido ofrecer simultáneamente las siguientes tres garantías”: CAP: Consistency, Availability, Partition).

Al traducir un sistema transaccional a NoSql, MongoDB se basa en el esquema de Consistencia:

Basically Available: está operativo la mayoría del tiempo.

Soft state: los datos en las diferentes réplicas no tienen que ser mutuamente consistentes en todo momento.

Eventually consistent: la información va a ser consistente. Se asegura la consistencia solo después de que pase cierto tiempo.

y ante una eventual transacción MongoDB garantiza la Consistencia (Consistency): todos los nodos ven los mismos datos al mismo tiempo y Tolerancia a la partición (Partition): el sistema continúa funcionando a pesar de la pérdida de mensajes o fallos parciales del sistema.²

² Referencia: Bases de datos 05_20220505.pdf

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

Tipos de índices

- **Single Field:** por un solo campo. Este tipo de índice se aplica sobre un campo de un documento. Ordena la colección en base a dicho campo y su criterio de orden (Ascendente o Descendente), así las búsquedas por igualdad o rangos se vuelven muy eficientes.
- **Compound Index:** por campos compuestos. Extiende la idea del “Single Field Index” a más de un campo, es decir por más de una clave. Por cada campo del índice compuesto se indica criterio de orden, y es el especificado por el programador. Esta característica es fuerte al momento de utilizar este tipo de índices.
- **Multikey Index:** por un campo que está accediendo a un elemento interno de ese atributo. MongoDB introduce los índices multi-clave, que permiten indexar la información de campos de tipo Array, para su posterior búsqueda eficiente.
- **Geospatial Index:** para consultas geográficas. Con el tipo de dato geopunto, se define índices que hacen uso de este tipo de datos, permite utilizarse para consultas de tipo geográficas tales como distancias, contenimiento, no contenimiento de manera eficiente, viene en forma nativa. Restricción: son puntos en dos dimensiones. Para admitir consultas eficientes de datos de coordenadas geoespaciales, MongoDB proporciona dos índices especiales: índices 2d que usan geometría plana al devolver resultados e índices 2dsphere que usan geometría esférica para devolver resultados.
- **Text index:** permite la búsqueda de strings dentro de una colección
- **Hash Index:** se utiliza para dar soporte al sharding basado en Hashing. Indexan el hash del valor del campo usado como clave. Como beneficio, permite hacer una especie de “load-balancing” de cada shard, evitando que claves muy próximas o similares siempre residan en el mismo shard, y agregando un nivel de dispersión mayor a las claves. Como contrapartida, hashing no permite búsquedas por rango, solo de acceso directo (por igualdad).

4. ¿Existen claves foráneas en MongoDB?

Se pueden representar haciendo en un documento una **referencia** a otro documento. Es decir, para no repetir mucha información en un documento, se puede tener un documento que apunta, que tiene una **referencia** a otro documento, esto sirve para que no repetir información y también sirve para cuando se realiza una modificación en un solo lugar todo el resto de las referencias se actualicen, esto es lo más parecido a tener una referencia a una clave foránea de otro elemento, en relacional.

Cuando en algunos casos sea conveniente tener toda la información en un mismo documento, por más que esté referenciando algunas veces a otras entidades, entonces tengo lo que se denominan **documentos embebidos** (join) (un json y que dentro en alguna parte tiene otro json) y la única manera de acceder a ese embebido es accediendo a través del acceso al documento padre, es decir acceder al documento que lo está conteniendo, aunque esto hace que sea más difícil hacer una actualización dado que se deberá acceder en cascada para llegar al documento que quiero hacer la actualización en el que se encuentra embebido.

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.

```
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
> db.version()
5.0.9
> db
test
> help
    db.help()                help on db methods
    db.mycoll.help()         help on collection methods
    sh.help()                sharding helpers
    rs.help()                replica set helpers
    help admin               administrative help
    help connect             connecting to a db help
    help keys                key shortcuts
    help misc                misc things to know
    help mr                  mapreduce

    show dbs                 show database names
    show collections         show collections in current database
    show users               show users in current database
    show profile             show most recent system.profile entries with time >= 1ms
    show logs                show the accessible logger names
    show log [name]          prints out the last segment of log in memory, 'global' is default
    use <db name>            set current database
    db.mycoll.find()          list objects in collection mycoll
    db.mycoll.find( { a : 1 } ) list objects in mycoll where a == 1
    it                       result of the last line evaluated; use to further iterate
    DBQuery.shellBatchSize = x set default number of items to display on shell
    exit                     quit the mongo shell
>
```

Métodos:

```
C:\Windows\System32\cmd.exe - mongo
> db.help()
DB methods:
    db.adminCommand(nameOrDocument) - switches to 'admin' db, and runs command [just calls db.runCommand(...)]
    db.aggregate([pipeline], {options}) - performs a collectionless aggregation on this database; returns a cursor
    db.auth(username, password)
    db.commandHelp(name) returns the help for the command
    db.createUser(userDocument)
    db.createView(name, viewOn, [{operator: {...}}, ...], {viewOptions})
    db.currentOp() displays currently executing operations in the db
    db.dropDatabase(writeConcern)
    db.dropUser(username)
    db.eval() - deprecated
    db.fsyncLock() flush data to disk and lock server for backups
    db.fsyncUnlock() unlocks server following a db.fsyncLock()
    db.getCollection(cname) same as db['cname'] or db.cname
    db.getCollectionInfos([filter]) - returns a list that contains the names and options of the db's collections
    db.getCollectionNames()
    db.getLastErrorMessage() - just returns the err msg string
    db.getLastErrorMessageObj() - return full status object
    db.getLogComponents()
    db.getMongo() get the server connection object
    db.getMongo().setSecondaryOk() allow queries on a replication secondary server
    db.getName()
    db.getProfilingLevel() - deprecated
    db.getProfilingStatus() - returns if profiling is on and slow threshold
    db.getReplicationInfo()
    db.getSiblingDB(name) get the db at the same server as this one
    db.getWriteConcern() - returns the write concern used for any operations on this db, inherited from server object if set
    db.hostInfo() get details about the server's host
    db.isMaster() check replica primary status
    db.hello() check replica primary status
    db.killOp(opid) kills the current operation in the db
    db.listCommands() lists all the db commands
    db.loadServerScripts() loads all the scripts in db.system.js
    db.logout()
    db.printCollectionStats()
    db.printReplicationInfo()
    db.printShardingStatus()
    db.printSecondaryReplicationInfo()
    db.rotateCertificates(message) - rotates certificates, CRLs, and CA files and logs an optional message
    db.runCommand(cmdObj) run a database command. if cmdObj is a string, turns it into {cmdObj: 1}
    db.serverStatus()
    db.setLogLevel(level,<component>)
    db.setProfilingLevel(level,slowms) 0=off 1=slow 2=all
```

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}
```

Para crear la base de datos utilizamos el comando: **use** nombre_de_la_base

MongoDB por defecto se encuentra trabajando en la base test

```
> db
test
```

Al crear la nueva base, en la siguiente línea luego del comando **use**, nos indica que ha cambiado a la nueva base

```
> use vaccination
switched to db vaccination
>
```

→ crea la base
→ indica que ha cambiado a la base vaccination

y confirmamos que la base ha sido creada y que **vaccination** es la base sobre la que estamos trabajando:

```
> db.dropDatabase()
{ "ok" : 1 }
> db
vaccination
>
```

Luego creamos la colección **nurses** utilizando el comando:
db.createCollection("nombre_de la_colección")

```
> db.createCollection("nurses")
{ "ok" : 1 }
>
```

Ahora, agregamos a la colección **nurses**, una enfermera con los atributos solicitados, esto lo realizamos con el siguiente comando:

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

y recuperamos la información usando el comando db.nurses.find() + .pretty()

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

y 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?

La siguiente captura muestra lo solicitado en el párrafo anterior:

```
> db.nurses.insertOne({name:"Morella Crespo", experience:9})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("62a3baeaefa68f0bbe01d79e")
}
```

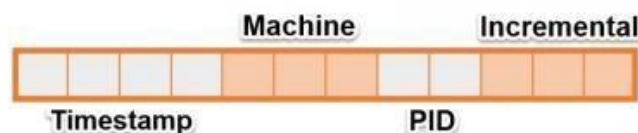
```
> db.nurses.find({name:"Morella Crespo", experience:9}).pretty()
{
  "_id" : ObjectId("62a3baeaefa68f0bbe01d79e"),
  "name" : "Morella Crespo",
  "experience" : 9
}
```

```
> db.nurses.find().pretty()
{
  "_id" : ObjectId("62a3baeaefa68f0bbe01d79e"),
  "name" : "Morella Crespo",
  "experience" : 9
}
```

Al insertar un documento enfermera a la colección, le asignó un campo `_id` a ese documento.

Este campo lo genera MongoDB automáticamente, si es que no se le especificó al momento de la inserción. Todos los documentos tienen un `_id` y tiene que ser único y por defecto es un `ObjectId` y una vez insertado así, ese campo `id` no se puede modificar. Si quisiéramos que fuera otro se debe eliminar y volver a insertar con un `id` distinto.

Este objeto `_id` del tipo `ObjectId`, así definido garantiza unicidad en entornos distribuidos como MongoDB. El campo está compuesto por 12 bytes: Los cuatro primeros bytes son un timestamp con los segundos desde el epoch de Unix; los tres siguientes bytes representan el identificador único de la máquina; los dos siguientes el identificador del proceso; y para finalizar los últimos tres bytes, son un campo incremental.



En definitiva los nueve primeros bytes nos garantizan un identificador único por segundo, máquina y proceso. Los tres últimos bytes, garantizan que cada segundo podemos insertar $2^{24} = 16.777.216$ documentos con un identificador distinto. Aunque técnicamente un `_id` podría repetirse, en la práctica es un número tan alto que es muy difícil que eso suceda.³

³ Referencias:

<https://www.mongodb.com/developer/products/mongodb/bson-data-types-objectid/>,

<https://www.genbeta.com/desarrollo/mongodb-empezando-por-el-principio-insertando-datos>

► 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:"Honoría Fernández", experience:5, vaccines: ["Pfizer", "Moderna", "Sputnik V"]}
{name:"Gonzalo Gallardo", experience:3}
{name:"Altea Parra", experience:6, vaccines: ["Pfizer"]}
```

En este caso usamos el método que inserta múltiples documentos a una colección:

```
db.nurses.insertMany(
[
{name: "Gale Molina", experience:8, vaccines: ["AZ", "Moderna"]},
{name: "Honoría Fernández", experience:5, vaccines: ["Pfizer", "Moderna", "Sputnik V"]},
{name:"Gonzalo Gallardo", experience:3},
{name:"Altea Parra", experience:6, vaccines: ["Pfizer"]}
])
```

```
> db.nurses.insertMany([
... {name:"Gale Molina", experience:8, vaccines: ["AZ", "Moderna"]},
... {name:"Honoría Fernández", experience:5, vaccines: ["Pfizer", "Moderna", "Sputnik V"]},
... {name:"Gonzalo Gallardo", experience:3},
... {name:"Altea Parra", experience:6, vaccines: ["Pfizer"]}
... ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("62a40a0b8d5e716dbf70fb91"),
    ObjectId("62a40a0b8d5e716dbf70fb92"),
    ObjectId("62a40a0b8d5e716dbf70fb93"),
    ObjectId("62a40a0b8d5e716dbf70fb94")
  ]
}
```

Y busque los enfermeros:

- de 5 años de experiencia o menos

```
> db.nurses.find({experience: {$lte:5}}).pretty()
{
  "_id" : ObjectId("62a40a0b8d5e716dbf70fb92"),
  "name" : "Honoría Fernández",
  "experience" : 5,
  "vaccines" : [
    "Pfizer",
    "Moderna",
    "Sputnik V"
  ]
}
{
  "_id" : ObjectId("62a40a0b8d5e716dbf70fb93"),
  "name" : "Gonzalo Gallardo",
  "experience" : 3
}
```

- que hayan aplicado la vacuna "Pfizer"

```
> db.nurses.find({vaccines:"Pfizer"}).pretty()
{
  "_id" : ObjectId("62a40a0b8d5e716dbf70fb92"),
  "name" : "Honorio Fernández",
  "experience" : 5,
  "vaccines" : [
    "Pfizer",
    "Moderna",
    "Sputnik V"
  ]
}
{
  "_id" : ObjectId("62a40a0b8d5e716dbf70fb94"),
  "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}}).pretty()
{
  "_id" : ObjectId("62a407888d5e716dbf70fb90"),
  "name" : "Morella Crespo",
  "experience" : 9
}
{
  "_id" : ObjectId("62a40a0b8d5e716dbf70fb93"),
  "name" : "Gonzalo Gallardo",
  "experience" : 3
}
>
```

- de apellido "Fernández"

```
> db.nurses.find({"name": {$regex:/Fernández/}}).pretty()
{
  "_id" : ObjectId("62a40a0b8d5e716dbf70fb92"),
  "name" : "Honorio Fernández",
  "experience" : 5,
  "vaccines" : [
    "Pfizer",
    "Moderna",
    "Sputnik V"
  ]
}
>
```

- con 6 o más años de experiencia y que hayan aplicado la vacuna "Moderna"

```
> db.nurses.find({$and: [{experience: {$gte:6}}, {vaccines: "Moderna"}]}).pretty()
{
  "_id" : ObjectId("62a40a0b8d5e716dbf70fb91"),
  "name" : "Gale Molina",
  "experience" : 8,
  "vaccines" : [
    "AZ",
    "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({$and: [{experience: {$gte:6}}, {vaccines: "Moderna"}]}, {name: true, _id: false})
{ "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.

Aquí se muestran los documentos antes de ser actualizados:

```
> db.nurses.insertMany([
... {name:"Morella Crespo", experience:9},
... {name:"Gale Molina", experience:8, vaccines: ["AZ", "Moderna"]},
... {name:"Honoraria Fernández", experience:5, vaccines: ["Pfizer", "Moderna", "Sputnik V"]},
... {name:"Gonzalo Gallardo", experience:3},
... {name:"Altea Parra", experience:6, vaccines: ["Pfizer"]}
... ])
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("62a429598d5e716dbf70fb95"),
    ObjectId("62a429598d5e716dbf70fb96"),
    ObjectId("62a429598d5e716dbf70fb97"),
    ObjectId("62a429598d5e716dbf70fb98"),
    ObjectId("62a429598d5e716dbf70fb99")
  ]
}
```

Para actualizar utilizamos el comando **updateOne()** dado que es solo un caso

```
> db.nurses.updateOne({name:"Gale Molina"},{$set: {experience:9}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Luego con el comando **find()** confirmamos que la actualización se haya realizado en el documento correspondiente.

```
> db.nurses.find().pretty()
{
  "_id" : ObjectId("62a429598d5e716dbf70fb95"),
  "name" : "Morella Crespo",
  "experience" : 9
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb96"),
  "name" : "Gale Molina",
  "experience" : 9,
  "vaccines" : [
    "AZ",
    "Moderna"
  ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb97"),
  "name" : "Honoraria Fernández",
  "experience" : 5,
  "vaccines" : [
    "Pfizer",
    "Moderna",
    "Sputnik V"
  ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb98"),
  "name" : "Gonzalo Gallardo",
  "experience" : 3
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb99"),
  "name" : "Altea Parra",
  "experience" : 6,
  "vaccines" : [
    "Pfizer"
  ]
}
>
```

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

```
> db.nurses.updateOne({name:"Gonzalo Gallardo"},{$set: {vaccines: []}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

```
{
  "_id" : ObjectId("62a429598d5e716dbf70fb98"),
  "name" : "Gonzalo Gallardo",
  "experience" : 3,
  "vaccines" : [ ]
}
```

9. Agregue "AZ" a las vacunas de "Altea Parra".

```
> db.nurses.updateOne({name: "Altea Parra"}, {$push: {vaccines: "AZ"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
>
```

Confirmamos la modificación en el documento:

```
> db.nurses.find({name: "Altea Parra"}).pretty()
{
  "_id" : ObjectId("62a429598d5e716dbf70fb99"),
  "name" : "Altea Parra",
  "experience" : 6,
  "vaccines" : [
    "Pfizer",
    "AZ"
  ]
}
```

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

```
> db.nurses.updateMany({vaccines: "Pfizer"},{$mul: {experience: 2}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
>
```

Antes de modificarse:

```
> db.nurses.find().pretty()
{
  "_id" : ObjectId("62a429598d5e716dbf70fb95"),
  "name" : "Morella Crespo",
  "experience" : 9
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb96"),
  "name" : "Gale Molina",
  "experience" : 9,
  "vaccines" : [
    "AZ",
    "Moderna"
  ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb97"),
  "name" : "Honorina Fernández",
  "experience" : 5,
  "vaccines" : [
    "Pfizer",
    "Moderna",
    "Sputnik V"
  ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb98"),
  "name" : "Gonzalo Gallardo",
  "experience" : 3,
  "vaccines" : [ ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb99"),
  "name" : "Altea Parra",
  "experience" : 6,
  "vaccines" : [
    "Pfizer"
  ]
}
```

Luego de la modificación solicitada:

```
> db.nurses.find().pretty()
{
  "_id" : ObjectId("62a429598d5e716dbf70fb95"),
  "name" : "Morella Crespo",
  "experience" : 9
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb96"),
  "name" : "Gale Molina",
  "experience" : 9,
  "vaccines" : [
    "AZ",
    "Moderna"
  ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb97"),
  "name" : "Honorina Fernández",
  "experience" : 10,
  "vaccines" : [
    "Pfizer",
    "Moderna",
    "Sputnik V"
  ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb98"),
  "name" : "Gonzalo Gallardo",
  "experience" : 3,
  "vaccines" : [ ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb99"),
  "name" : "Altea Parra",
  "experience" : 12,
  "vaccines" : [
    "Pfizer",
    "AZ"
  ]
}
>
```

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.*

Antes de eliminar los documentos de la colección, tenemos:

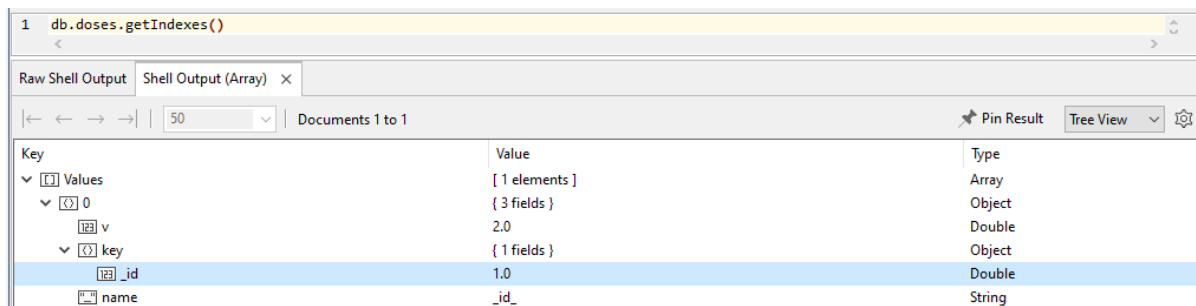
```
> db
vaccination
> show collections
nurses
> db.nurses.find().pretty()
{
  "_id" : ObjectId("62a429598d5e716dbf70fb95"),
  "name" : "Morella Crespo",
  "experience" : 9
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb96"),
  "name" : "Gale Molina",
  "experience" : 9,
  "vaccines" : [
    "AZ",
    "Moderna"
  ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb97"),
  "name" : "Honoría Fernández",
  "experience" : 10,
  "vaccines" : [
    "Pfizer",
    "Moderna",
    "Sputnik V"
  ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb98"),
  "name" : "Gonzalo Gallardo",
  "experience" : 3,
  "vaccines" : [ ]
}
{
  "_id" : ObjectId("62a429598d5e716dbf70fb99"),
  "name" : "Altea Parra",
  "experience" : 12,
  "vaccines" : [
    "Pfizer",
    "AZ"
  ]
}
```

Ahora, eliminamos a todos los enfermeros con el comando `db.nurses.remove({})` y podemos ver que ha eliminado los 5 elementos de nuestra colección y al listar vemos que ya no hay documentos en nuestra colección `nurses`, ha quedado vacía.

```
> db.nurses.remove({})
WriteResult({ "nRemoved" : 5 })
> db.nurses.find()
>
```

11. Busque en la colección de ~~empresas~~ dosis (doses) si existe algún índice definido.

En la siguiente captura, con el comando `getIndexes()` sobre la colección *doses*, se encuentra el índice definido por defecto por MongoDB, como una clave primaria, a todo documento

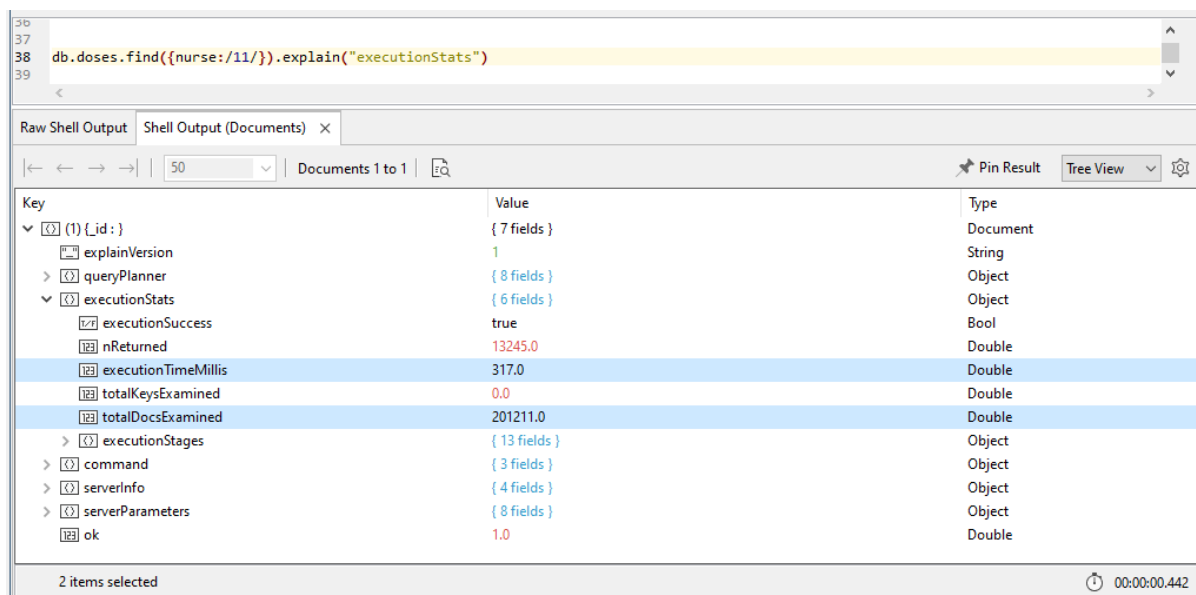


Key	Value	Type
Values	[1 elements]	Array
0	{ 3 fields }	Object
v	2.0	Double
key	{ 1 fields }	Object
_id	1.0	Double
name	_id_	String

MongoDB crea un índice sobre el campo `_id` durante la creación de una colección. Este índice controla la unicidad del valor del `_id`. No se puede eliminar este índice.⁴

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.**

Consulta sin índice:

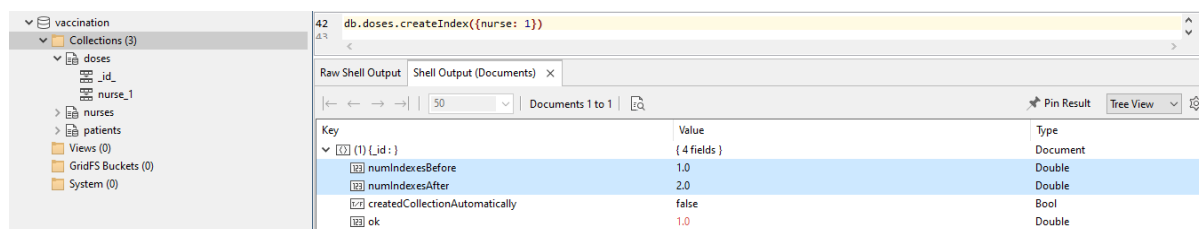


Key	Value	Type
(1) { _id : }	{ 7 fields }	Document
explainVersion	1	String
queryPlanner	{ 8 fields }	Object
executionStats	{ 6 fields }	Object
executionSuccess	true	Bool
nReturned	13245.0	Double
executionTimeMillis	317.0	Double
totalKeysExamined	0.0	Double
totalDocsExamined	201211.0	Double
executionStages	{ 13 fields }	Object
command	{ 3 fields }	Object
serverInfo	{ 4 fields }	Object
serverParameters	{ 8 fields }	Object
ok	1.0	Double

Podemos observar que se examinaron la totalidad de los documentos y la consulta se realizó en 317 milisegundos. (disco HDD como soporte para recolección de datos)

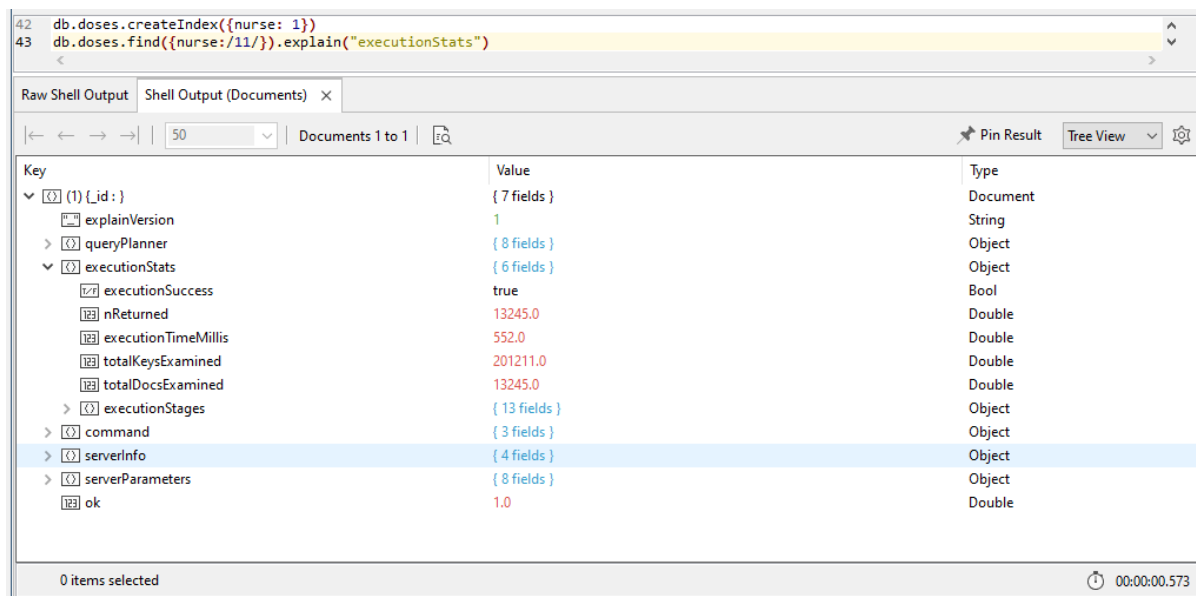
⁴ Referencia: teoría [Bases de datos 05_20220505.pdf](#)

Creamos el índice para la búsqueda por nombre y poder comparar. Con `db.doses.createIndex({nurse: 1})`, ahora vemos que se ha generado un nuevo índice:



Key	Value	Type
(1) {id: }	{ 4 fields }	Document
numIndexesBefore	1.0	Double
numIndexesAfter	2.0	Double
createdCollectionAutomatically	false	Bool
ok	1.0	Double

Y volvemos a realizar la consulta **con** índice:



Key	Value	Type
(1) {id: }	{ 7 fields }	Document
explainVersion	1	String
queryPlanner	{ 8 fields }	Object
executionStats	{ 6 fields }	Object
executionSuccess	true	Bool
nReturned	13245.0	Double
executionTimeMillis	552.0	Double
totalKeysExamined	201211.0	Double
totalDocsExamined	13245.0	Double
executionStages	{ 13 fields }	Object
command	{ 3 fields }	Object
serverInfo	{ 4 fields }	Object
serverParameters	{ 8 fields }	Object
ok	1.0	Double

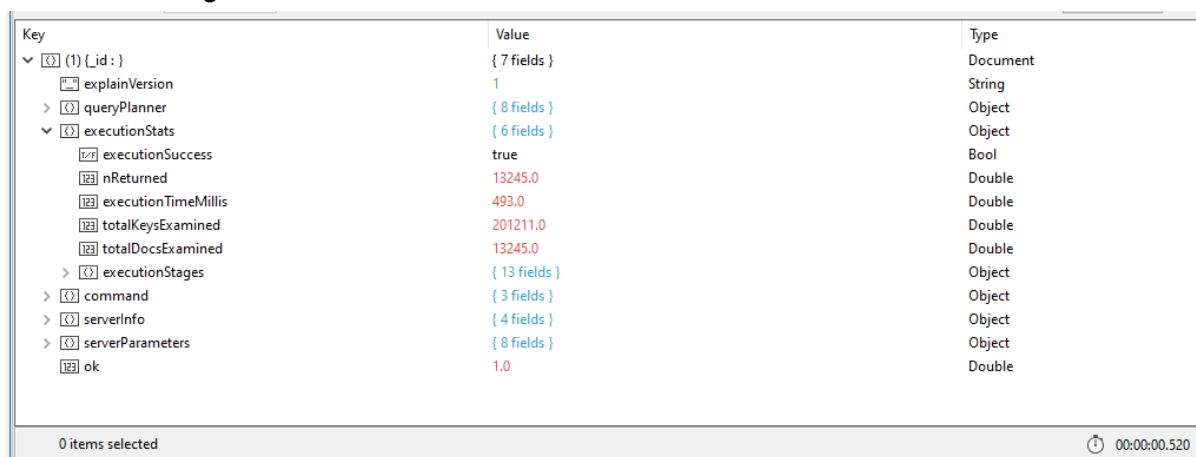
Observamos que redujo la cantidad de documentos examinados a la cantidad de documentos retornados según la consulta.

y a medida que reiteramos la consulta los tiempo de ejecución se reducen:

1ra - 552 milisegundos

2da - 493 milisegundos

3ra - 463 milisegundos



Key	Value	Type
(1) {id: }	{ 7 fields }	Document
explainVersion	1	String
queryPlanner	{ 8 fields }	Object
executionStats	{ 6 fields }	Object
executionSuccess	true	Bool
nReturned	13245.0	Double
executionTimeMillis	493.0	Double
totalKeysExamined	201211.0	Double
totalDocsExamined	13245.0	Double
executionStages	{ 13 fields }	Object
command	{ 3 fields }	Object
serverInfo	{ 4 fields }	Object
serverParameters	{ 8 fields }	Object
ok	1.0	Double

Raw Shell Output		
Shell Output (Documents) X		
<div> <div> <div>←</div> <div>→</div> <div>50</div> </div> <div>Documents 1 to 1</div> <div> <div>🔍</div> <div>Pin Result</div> <div>Tree View</div> <div>⚙️</div> </div> </div>		
Key	Value	Type
<div> <div>⌵</div> <div>(1) { _id : }</div> </div>	{ 7 fields }	Document
<div> <div>⌵</div> <div>explainVersion</div> </div>	1	String
<div> <div>⌵</div> <div>queryPlanner</div> </div>	{ 8 fields }	Object
<div> <div>⌵</div> <div>executionStats</div> </div>	{ 6 fields }	Object
<div> <div>⌵</div> <div>executionSuccess</div> </div>	true	Bool
<div> <div>⌵</div> <div>nReturned</div> </div>	13245.0	Double
<div> <div>⌵</div> <div>executionTimeMillis</div> </div>	463.0	Double
<div> <div>⌵</div> <div>totalKeysExamined</div> </div>	201211.0	Double
<div> <div>⌵</div> <div>totalDocsExamined</div> </div>	13245.0	Double
<div> <div>⌵</div> <div>executionStages</div> </div>	{ 13 fields }	Object
<div> <div>⌵</div> <div>command</div> </div>	{ 3 fields }	Object
<div> <div>⌵</div> <div>serverInfo</div> </div>	{ 4 fields }	Object
<div> <div>⌵</div> <div>serverParameters</div> </div>	{ 8 fields }	Object
<div> <div>⌵</div> <div>ok</div> </div>	1.0	Double
<div>0 items selected</div> <div>⌚ 00:00:00.484</div>		

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~~ *address* 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.

Definimos nuestra variable “cc” asignándole el polígono del archivo *caba.geojson*

Consulta sin índice:

Raw Shell Output

Shell Output (Documents) ×

Documents 1 to 1

Key	Value	Type
(1) { _id : }	{ 7 fields }	Document
explainVersion	1	String
queryPlanner	{ 8 fields }	Object
executionStats	{ 6 fields }	Object
executionSuccess	true	Bool
nReturned	43805.0	Double
executionTimeMillis	1246.0	Double
totalKeysExamined	0.0	Double
totalDocsExamined	201211.0	Double
executionStages	{ 13 fields }	Object
command	{ 3 fields }	Object
serverInfo	{ 4 fields }	Object
serverParameters	{ 8 fields }	Object
ok	1.0	Double

0 items selected

00:00:01.280

Se observa que sobre un escaneo de la totalidad de los documentos, la consulta se realizó en 1246 milisegundos. (disco HDD como soporte para recolección de datos)

Ahora creamos el índice con `db.patients.createIndex({address: "2dsphere"})`

Raw Shell Output

Shell Output (Documents) ×

Documents 1 to 1

Key	Value
(1) { _id : }	{ 4 fields }
numIndexesBefore	1.0
numIndexesAfter	2.0
createdCollectionAutomatically	false
ok	1.0

y realizamos nuevamente la consulta luego de aplicar un índice en el campo *address*

`db.patients.find({address: {$geoWithin: {$geometry: cc}}}).explain("executionStats")`

Arrojó los siguientes datos:

db.patients.find({address: {\$geoWithin: {\$geometry: cc}}}).explain("executionStats")

Raw Shell Output | Shell Output (Documents) X

← ← → → | 50 | Documents 1 to 1 | [?] Pin Result Tree View [⚙]

Key	Value	Type
✓ (1) {_id: }	{ 7 fields }	Document
explainVersion	1	String
> queryPlanner	{ 8 fields }	Object
✓ executionStats	{ 6 fields }	Object
executionSuccess	true	Bool
nReturned	43805.0	Double
executionTimeMillis	743.0	Double
totalKeysExamined	55440.0	Double
totalDocsExamined	55421.0	Double
> executionStages	{ 14 fields }	Object
> command	{ 3 fields }	Object
> serverInfo	{ 4 fields }	Object
> serverParameters	{ 8 fields }	Object
ok	1.0	Double

1 item selected | 00:00:00.779

Reduce considerablemente el tiempo de la consulta: 743 milisegundos.

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.

Utilizamos del framework de agregación la función [\\$sample](#)

```
78
79 db.patients.aggregate([{$sample: {size: 5}}])
RA
```

Raw Shell Output | Aggregate Query (line 79) ×

← → | 50 | Documents 1 to 5 | 🔍

Pin Result | Query Code | Explain Query | Table View ⌵

patients > name

_id	name	address
62a79ed765875e...	Paciente 41100	{ 2 fields }
62a79ec865875e...	Paciente 31991	{ 2 fields }
62a79ec365875e...	Paciente 28646	{ 2 fields }
62a79eb665875e...	Paciente 20437	{ 2 fields }
62a79eb565875e...	Paciente 20057	{ 2 fields }

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.

Utilizamos la función [\\$geoNear](#): Genera documentos en orden del más cercano al más lejano desde un punto específico. A partir de un punto de coordenadas le determinamos la distancia en metros (1 km=1000 mts)

```
81 db.patients.aggregate([
82 {
83   $geoNear: {
84     near: { type: "Point", coordinates: [-58.4586, -34.5968] },
85     distanceField: "dist.distance",
86     maxDistance: 1000,
87     spherical: true
88   },
89 },
90 { $out: "patients_caba" }
91 ])
```

Raw Shell Output | Aggregate Query (line 81) ×

← → | 50 | Documents 1 to 50 | 🔍

Pin Result | Query Code | Explain Query | Table View ⌵

patients > name

_id	name	address	dist
62a79f6365875e...	Paciente 129663	{ 2 fields }	{ 1 fields }
62a79ea465875e...	Paciente 9191	{ 2 fields }	{ 1 fields }
62a79f7f65875e...	Paciente 147391	{ 2 fields }	{ 1 fields }
62a79fc565875e...	Paciente 190490	{ 2 fields }	{ 1 fields }
62a79f3b65875e...	Paciente 104276	{ 2 fields }	{ 1 fields }
62a79f6165875e...	Paciente 128496	{ 2 fields }	{ 1 fields }
62a79fca65875e...	Paciente 193165	{ 2 fields }	{ 1 fields }
62a79ad465875e...	Paciente 44965	{ 2 fields }	{ 1 fields }

1,219 documents | 00:00:00.471

Como resultado obtuvimos que se ha creado la colección “patients_caba” y se obtuvieron 1219 pacientes que responden a esta consulta

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.

```

93 db.applied_doses.aggregate([
94   {
95     $lookup: {
96       from: "patients",
97       localField: "patient",
98       foreignField: "name",
99       as: "patient_doc"
100     },
101   },
102   { $match: { patient_doc: { $not: { $size: 0 } } } },
103   { $project: { patient_doc: 0 } },
104   { $out: "applied_doses" }
105 ])
  
```

_id	nurse	patient	vaccine	date
62a79e9665875e...	Enfermero 2	Paciente 119	AZ	2020-07-20T00:...
62a79e9665875e...	Enfermero 2	Paciente 153	Moderna	2020-05-10T00:...
62a79e9665875e...	Enfermero 2	Paciente 168	Sputnik V	2021-07-08T00:...
62a79e9665875e...	Enfermero 5	Paciente 435	Sputnik V	2020-07-02T00:...
62a79e9665875e...	Enfermero 5	Paciente 448	Moderna	2021-09-01T00:...
62a79e9665875e...	Enfermero 7	Paciente 653	Johnson	2021-05-11T00:...

Aquí se utilizaron [\\$lookup](#) y [\\$project](#)

► 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.

```

115 db.nurses.aggregate([
116   { $match: { name: /111/ } },
117   {
118     $lookup: {
119       from: "doses",
120       localField: "name",
  
```

(Document id)	_id	nurse	patient	vaccine	date
62a79e9365875e...	62a79ea765875e...	Enfermero 111	Paciente 11288	Moderna	2020-11-20T00:00:00.00...
62a79e9465875e...	62a79f4665875e...	Enfermero 1110	Paciente 111512	Pfizer	2020-08-11T00:00:00.00...
62a79e9465875e...	62a79f4665875e...	Enfermero 1111	Paciente 111626	Pfizer	2021-02-28T00:00:00.00...
62a79e9465875e...	62a79f4765875e...	Enfermero 1112	Paciente 111747	Pfizer	2021-07-01T00:00:00.00...
62a79e9465875e...	62a79f4765875e...	Enfermero 1113	Paciente 111863	Johnson	2021-06-16T00:00:00.00...
62a79e9465875e...	62a79f4765875e...	Enfermero 1114	Paciente 111954	Sputnik V	2020-06-01T00:00:00.00...
62a79e9465875e...	62a79f4765875e...	Enfermero 1115	Paciente 112047	Sputnik V	2021-08-13T00:00:00.00...
62a79e9465875e...	62a79f4765875e...	Enfermero 1116	Paciente 112176	Moderna	2021-04-13T00:00:00.00...
62a79e9465875e...	62a79f4765875e...	Enfermero 1117	Paciente 112240	Moderna	2021-04-24T00:00:00.00...
62a79e9465875e...	62a79f4765875e...	Enfermero 1118	Paciente 112295	Pfizer	2021-07-15T00:00:00.00...
62a79e9465875e...	62a79f4865875e...	Enfermero 1119	Paciente 112407	Pfizer	2021-07-11T00:00:00.00...