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:** Existe el concepto.
- **Tabla / Relación:** No existe el concepto. Sí, hay alternativa, son las colecciones.
- **Fila / Tupla:** No existe el concepto. Sí, hay alternativa, son los documentos.
- **Columna:** No existe el concepto. Sí, hay alternativa, son los 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?

Para situaciones que requieren atomicidad de lecturas y escrituras en varios documentos (en una o varias colecciones), MongoDB admite transacciones de varios documentos. Con transacciones distribuidas, las transacciones se pueden usar en múltiples operaciones, colecciones, bases de datos, documentos y fragmentos.

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

- Single Field Index: son índices que referencian a un solo campo dentro de los documentos de una colección.
- Compound Index: estos índices referencian a varios campos dentro de los documentos de una colección.
- Multikey Index: mejoran el rendimiento de las consultas que especifican un campo con un índice que contiene un valor de un arreglo.
- Text Index: permiten consultas de búsqueda de texto en el contenido de strings.
- Geospatial Index: permite hacer consultas de datos de coordenadas geoespaciales utilizando índices 2dsphere. Con un índice 2dsphere podemos consultar los datos geoespaciales para inclusión, intersección y proximidad.
- Unique Index: aseguran que los campos indexados no almacenen valores duplicados. Por defecto, MongoDB crea un índice único en el campo _id durante la creación de una colección.

4. ¿Existen claves foráneas en MongoDB?

MongoDB no es relacional, por lo tanto no tiene el concepto de clave foránea, pero brinda dos métodos para relacionar documentos.

El primero son las **referencias manuales:** para las cuales se guarda el campo _id de un documento en otro documento a modo de referencia. La aplicación debe realizar una segunda consulta para obtener los datos. Este tipo de referencias son suficientes para los casos de uso más comunes.

El segundo método son las **DBRefs:** estas son referencias de un documento a otro que utilizan los valores de los campos _id, nombre de colección y (opcionalmente) el nombre de la base de datos del primer documento, así como cualquier otro campo necesario. Las DBRefs permiten referenciar documentos almacenados en diferentes colecciones o bases de datos de forma más sencilla ya que brindan un formato y tipo común para representar

relaciones entre documentos. Además proveen una semántica común para representar conexiones entre documentos en caso de que la base de datos deba interactuar con múltiples framework y/o herramientas.

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

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}

use vaccination
db.createCollection('nurses')
db.nurses.insert({name:'Morella Crespo', experience:9})

```
> db.nurse.insert({name:'Morella Crespo', experience:9})
< 'DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany, or bulkWrite.'
< { acknowledged: true,
    insertedIds: { '0': ObjectId("62915a023eea8430b86fa291") } }
```

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?

```
> db.nurse.find().pretty()
< { _id: ObjectId("62915a023eea8430b86fa291"),
    name: 'Morella Crespo',
    experience: 9 }
```

La diferencia es que se agregó el campo _id, como se mencionaba antes en el apartado de "Unique Index", por defecto, MongoDB crea un índice único en el campo _id durante la creación de una colección.

▸ 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:'Honoria Fernández', experience:5, vaccines: ['Pfizer', 'Moderna', 'Sputnik V']}
{name:'Gonzalo Gallardo', experience:3}
{name:'Altea Parra', experience:6, vaccines: ['Pfizer']}

```
> db.nurse.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("62915cfb3eea8430b86fa292"),
       '1': ObjectId("62915cfb3eea8430b86fa293"),
       '2': ObjectId("62915cfb3eea8430b86fa294"),
       '3': ObjectId("62915cfb3eea8430b86fa295") } }
```

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']}])

Y busque los enfermeros:

- **de 5 años de experiencia o menos**

```
> db.nurses.find({'experience':{$lte:5}}).pretty()
< { _id: ObjectId("62915c59f39fe6c176685e42"),
    name: 'Honoria Fernandez',
    experience: 5,
    vaccines: [ 'Pfizer', 'Moderna', 'Sputnik V' ] }
  { _id: ObjectId("62915c89f39fe6c176685e43"),
    name: 'Gonzalo Gallardo',
    experience: 3 }
```

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

- **que hayan aplicado la vacuna “Pfizer”**

```
> db.nurses.find({vaccines:{$in:['Pfizer']}}).pretty()
< { _id: ObjectId("62915c59f39fe6c176685e42"),
    name: 'Honoria Fernandez',
    experience: 5,
    vaccines: [ 'Pfizer', 'Moderna', 'Sputnik V' ] }
  { _id: ObjectId("62915ca6f39fe6c176685e44"),
    name: 'Altea Parra',
    experience: 6,
    vaccines: [ 'Pfizer' ] }
```

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

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

```
> db.nurses.find({ vaccines: { $exists: false }})
< { _id: ObjectId("62915a2d717f76c28130c836"),
    name: 'Morella Crespo',
    experience: 9 }
  { _id: ObjectId("62915d53717f76c28130c83a"),
    name: 'Gonzalo Gallardo',
    experience: 3 }
```

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

- **de apellido 'Fernández'**

```
> db.nurses.find({'name': /Fernandez/})
< { _id: ObjectId("62915c59f39fe6c176685e42"),
    name: 'Honoria Fernandez',
    experience: 5,
    vaccines: [ 'Pfizer', 'Moderna', 'Sputnik V' ] }
```

db.nurses.find({name: /Fernández/ })

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

```
> db.nurses.find({experience:{$gte:6}, vaccines: {$in: ['Moderna']}})
< { _id: ObjectId("62915c33f39fe6c176685e41"),
    name: 'Gale Molina',
    experience: 8,
    vaccines: [ 'AZ', 'Moderna' ] }
```

db.nurses.find({experience:{$gte:6}, vaccines: {$in: ['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']}}, {name: 1, _id: 0})
< { name: 'Gale Molina' }
```

db.nurses.find({experience:{$gte:6}, vaccines: {$in: ['Moderna']}}, {name: 1, _id: 0})

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

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

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

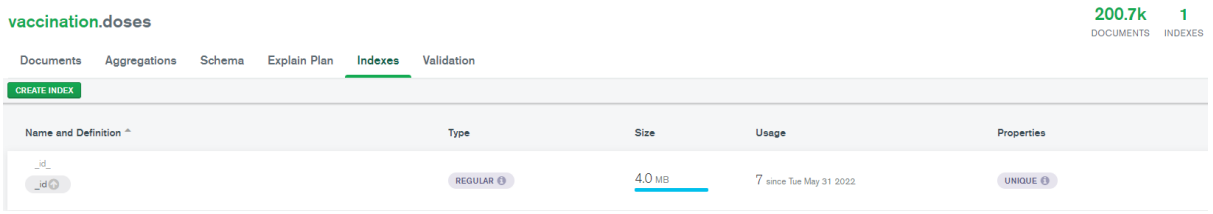
10. Duplique la experiencia de todos los enfermeros que hayan aplicado la vacuna “Pfizer”
db.nurses.updateMany({ vaccines: { $in: ["Pfizer"]}}, { $mul: {experience: 2}})

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(<ruta del archivo ‘generador.js’>). Si utiliza un cliente que lo permita (ej. Robo3T), se puede ejecutar directamente en el espacio de consultas.
CÓDIGO DE LA PRÁCTICA

```
> db.nurses.deleteMany({})
< { acknowledged: true, deletedCount: 5 }
```

db.nurses.deleteMany({})

11. Busque en la colección de compras (doses) si existe algún índice definido.
Si, existe el “_id”



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 ÍNDICE:

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

```
> db.doses.find({nurse: /11/}).explain("executionStats")
< { explainVersion: '1',
    queryPlanner:
     { namespace: 'vaccination.doses',
       indexFilterSet: false,
       parsedQuery: { nurse: BSONRegExp("11", "") },
       maxIndexedOrSolutionsReached: false,
       maxIndexedAndSolutionsReached: false,
       maxScansToExplodeReached: false,
       winningPlan:
        { stage: 'COLLSCAN',
          filter: { nurse: BSONRegExp("11", "") },
          direction: 'forward' },
       rejectedPlans: [] },
    executionStats:
     { executionSuccess: true,
       nReturned: 12492,
       executionTimeMillis: 175,
       totalKeysExamined: 0,
       totalDocsExamined: 200662,
```

CON ÍNDICE:
db.doses.createIndex({ nurse: 1 })
db.doses.find({nurse: /11/}).explain("executionStats")

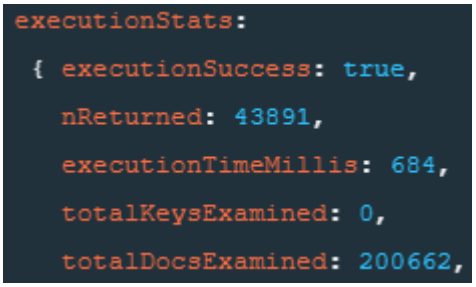
```
> db.doses.find({nurse: /11/}).explain("executionStats")
< { explainVersion: '1',
    queryPlanner:
     { namespace: 'vaccination.doses',
       indexFilterSet: false,
       parsedQuery: { nurse: BSONRegExp("11", "") },
       maxIndexedOrSolutionsReached: false,
       maxIndexedAndSolutionsReached: false,
       maxScansToExplodeReached: false,
       winningPlan:
        { stage: 'FETCH',
          inputStage:
           { stage: 'IXSCAN',
             filter: { nurse: BSONRegExp("11", "") },
             keyPattern: { nurse: 1 },
             indexName: 'nurse',
             isMultiKey: false,
             multiKeyPaths: { nurse: [] },
             isUnique: false,
             isSparse: false,
             isPartial: false,
             indexVersion: 2,
             direction: 'forward',
             indexBounds: { nurse: [ '["", {})', '[/11/, /11/]' ] } } },
       rejectedPlans: [] },
    executionStats:
     { executionSuccess: true,
       nReturned: 12492,
       executionTimeMillis: 244,
       totalKeysExamined: 200662,
       totalDocsExamined: 12492,
```

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.

```
coordenadas_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]
 ]]]
}
```

SIN ÍNDICE:

```
db.patients.find({address:{$geoWithin:{$geometry:coordenadas_CABA}}}).explain("execution
Stats")
```



CON ÍNDICE:

```
db.patients.createIndex( { address: "2dsphere" } )
db.patients.find({address:{$geoWithin:{$geometry:coordenadas_CABA}}}).explain("execution
Stats")
```

```
executionStats:
 { executionSuccess: true,
   nReturned: 43891,
   executionTimeMillis: 327,
   totalKeysExamined: 55428,
   totalDocsExamined: 55409,
```

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.

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

```
> db.patients.aggregate([{ $sample: { size: 5 } }])
< { _id: ObjectId("6296727c8d00dd3764a662c9"),
    name: 'Paciente 84586',
    address:
     { type: 'Point',
       coordinates: [ -58.580221866310715, -34.74658981887896 ] } }
  { _id: ObjectId("629671c98d00dd3764a552f9"),
    name: 'Paciente 49794',
    address:
     { type: 'Point',
       coordinates: [ -58.611638092870066, -34.76729972117141 ] } }
  { _id: ObjectId("629671af8d00dd3764a52999"),
    name: 'Paciente 44498',
    address:
     { type: 'Point',
       coordinates: [ -58.5657356919065, -34.604011147278314 ] } }
  { _id: ObjectId("6296721f8d00dd3764a5d2f9"),
    name: 'Paciente 66178',
    address:
     { type: 'Point',
       coordinates: [ -58.40580831764848, -34.68698051102152 ] } }
  { _id: ObjectId("629671e48d00dd3764a57cd1"),
    name: 'Paciente 55150',
    address:
     { type: 'Point',
       coordinates: [ -58.60112679174261, -34.67935598773915 ] } }
```

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.

```
db.patients.aggregate([
  {
    $geoNear: {
      near: { type: "Point", coordinates: [-58.4586,-34.5968] },
      distanceField: "distancia_del_centro",
      maxDistance: 1000,
      spherical: true
    }
  },
  {$out: 'center_patients'}
])
```

distanceField es un parámetro (obligatorio) que en este caso agrega a la colección el nuevo campo “distancia_del_centro” que especifica a qué distancia del punto [-58.4586,-34.5968] está el campo “address” de cada paciente.

```
  _id: ObjectId('629672ce0345432b63342d4b')
  name: "Paciente 48466"
> address: Object
  distancia_del_centro: 42.78285302253771
```

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.center_patients.aggregate([
{
  $lookup:{
     from: "doses",
     localField: "name",
     foreignField: "patient",
     as: "doses"
  }
},
{
  $project: {
    _id: 0,
    name: 0,
    address: 0,
    distancia_del_centro: 0
  }
},
{
  $replaceRoot: {newRoot: {$mergeObjects: [ {$arrayElemAt: ["$doses", 0] }, "$$ROOT" ] } }
},
{
  $project: {doses: 0}
```

```
},
{
  $out: "center_doses"
}
])
```

▸ **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.

```
db.nurses.aggregate([
  {
    $match: { name: /111/ }
  },
  {
    $lookup:{
      from: "doses",
      pipeline: [
        { "$match": { date: {$gt: ISODate("2021-05-01")} }}
      ],
      localField: "name",
      foreignField: "nurse",
      as: "vacunas"
    }
  },
  {
      $out: "nurses111"
  }
])
```