# Bases de Datos 2 2020 -TP2
# Bases de Datos NoSQL / Práctica con MongoDB

Entrega: 11/5

---

## 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, sin embargo, no es como en SQL que hay tablas, sino **conjuntos de colecciones**.
    - Tabla/Relación: el concepto tal cual de tabla SQL no existe en MongoDB. En cambio, hay **colecciones**, es decir, grupos de documentos. A diferencia de una tabla en SQL tradicional, en donde todos los registros tienen las mismas columnas, en MongoDB cada documento puede contener diferentes llaves (columnas).
    - Fila/Tupla: el concepto de tupla en MongoDB se conoce como **documento**, esto es, un conjunto de llaves y valores que se almacena en un formato llamado BSON (Binary JSON). BSON es muy similar a JSON (JavaScript Object Notation) pero con algunos tipos adicionales que no son soportados en JSON.
    - Columna: en MongoDB se conocen como **llaves** y cada documento podría contener diferentes.

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 problema de MongoDB con las transacciones es que se garantizan transacciones ACID **a nivel de documento**, esto quiere decir que las operaciones realizadas en los subdocumentos dentro de un documento cumplían con la condición de que si un error ocurría la base de datos se encargaba de hacer rollback al estado anterior al inicio de la operación.
    Si bien el 80%-90% de las aplicaciones **no necesitan transacciones entre documentos**, y que los problemas de consistencia se pueden solucionar con un buen modelamiento en la base de datos empleando subdocumentos, lo cierto es que en los casos en que se requiere contar con transaccionalidad entre documentos, es necesario **implementar la lógica transaccional dentro del código**, lo cual lleva indefectiblemente a soluciones negativas que presentan variados problemas para la escritura escalable, aumenta la complejidad del código y, en algunos casos, la latencia de la ejecución de las operaciones.

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

    Los índices soportan la ejecución eficiente de consultas en MongoDB. Sin índices, MongoDB debería realizar un sondeo en la coleccion para, por ejemplo, seleccionar aquellos documentos que cumplan con la condición de la sentencia. Si existe un índice apropiado para cada búsqueda, MongoDB puede usarlo para limitar el número de documentos que debe inspeccionar.

Los índices son estructuras de datos especiales que guardan una pequeña porción del data-set de las colecciones en una forma fácil de atravezar. El índice guarda un valor de un campo específico o un conjunto de valores de un conjunto de campos, ordenados por el alor de campo. Tal ordenamiento es el que permite matcheos por igualdad eficientes y operaciones range-based de búsqueda. Además, es posible que MongoDB devuelva el resultado ordenado utilizando la ordenación que defina ese índice.

Existen varios tipos:

- Índices simples o de un solo campo: Además del índice *_id* definido por MongoDB, MongoDB admite la creación de índices ascendentes/descendentes definidos por el usuario en un solo campo de un documento. Para un índice de campo único y operaciones de clasificación, el orden de clasificación (es decir, ascendente o descendente) de la clave de índice no importa porque MongoDB puede atravesar el índice en cualquier dirección.
- Índices compuestos: MongoDB también admite índices definidos por el usuario en múltiples campos, es decir, índices compuestos. El orden de los campos que figuran en un índice compuesto tiene importancia. Para índices compuestos y operaciones de clasificación, el orden de clasificación (es decir, ascendente o descendente) de las claves de índice puede determinar si el índice puede admitir una operación de clasificación.
- Índices multikey: MongoDB utiliza índices de múltiples claves para indexar el contenido almacenado en arrays. Si se indexa un campo que contiene un valor de array, MongoDB crea entradas de índice separadas para cada elemento del array. Estos índices multikey permiten que las consultas seleccionen documentos que contienen matrices matcheando por elemento o elementos del array. MongoDB determina automáticamente si se crea un índice multikey si el campo indexado contiene un valor de array; no es necesario que se especifique explícitamente el tipo multikey.
- Índices geoespaciales: Para admitir consultas eficientes de datos de coordenadas geoespaciales, MongoDB proporciona dos índices especiales: índices 2d que utilizan geometría plana al devolver resultados e índices 2dsphere que utilizan geometría esférica para devolver resultados.
- Índices de texto: MongoDB proporciona índices de texto para admitir consultas de búsqueda de texto en contenido de cadena. Los índices de texto pueden incluir cualquier campo cuyo valor sea una cadena o una matriz de elementos de cadena.
- Índices hash: Para admitir fragmentación basada en hash, MongoDB proporciona un tipo de índice hash, que indexa el hash del valor de un campo. Estos índices tienen una distribución de valores más aleatoria a lo largo de su rango, pero solo admiten matcheos de igualdad y no pueden admitir consultas basadas en rangos.

4. ¿Existen claves foráneas en MongoDB?

   MongoDB no tiene claves foráneas. Cuando es necesario, por ejemplo, insertar un valor que sea coherente con el campo de otra tabla, es necesario, definir manualmente tales reglas de integridad referencial.

---

## 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 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?

   Al usar `db.apartments.find()` se puede apreciar que hay un atributo más que los que se insertó anteriormente con nombre _id que es un identificador que mongo generó automaticamente al crear el documento.

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 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})`

   - Con capacidad para 4 personas o más

     `db.apartments.find({capacity:{$gte:4}})`

- Con wifi-que incluyan la palabra 'Apartment' en su nombre

  ```
  db.apartments.find({$and:[{services:"wifi"},{name:{$regex: /Apartment$/}}]})
  ```

- Con la palabra 'Apartment' en su nombre y capacidad para más de 3 personas

  ```
  db.apartments.find({$and:[{capacity:{$gt:3}},{name:{$regex: /Apartment$/}}]})
  ```

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

  ```
  db.apartments.find({services:null})
  ```

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:null}, {name :1, _id:0})
```

En MongoDB hay diferentes maneras de realizar actualizaciones, de acuerdo a las necesidades del esquema flexible de documentos.

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"}, {$addToSet:{services:"laundry"}})
   ```

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

   ```
   db.apartments.update({services:{$in:["wifi"]}}, {$inc: {capacity:1}})
   ```

---

## Parte 3: Índices

Elimine todos los departamentos de la colección. Guarde en un archivo llamado 'generador.js' el siguiente código JavaScript y ejecútelo con: `load(<ruta del archivo 'generador.js'>)`.

```
for (var i = 1; i <= 50000; i++)
{
    var randomServices = ['wifi', 'pool', 'parking', 'breakfast'].sort( function()
    { return 0.5 - Math.random() } ).slice(1, Math.floor (Math.random() * 5));
    var randomCapacity = Math.ceil(Math.random() * 5);
    var randomLong = ((Math.random()/1.3)+51);
    var randomLat = Math.random() - .4;
    db.apartments.insert({
        name:'Apartment '+i,
        capacity:randomCapacity,
        services: randomServices,
        location: { type: "Point", coordinates: [randomLat, randomLong] }
    });
}
```

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

    Usando la consulta `db.apartments.getIndexes()`, se puede apreciar que solo existe el índice por defecto *_id*.

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.

    Con el comando que sigue, en apartments, estoy creando un índice de la llave *name* y le defino el nombre *apartments_name*.

    ```
    db.apartments.createIndex({name:1}, {name:"apartments_name"})
    ```

    Para buscar los documentos que tengan en su nombre el string *11*, es necesario usar **regex** dentro del campo que se busca. Luego se le aplica el método *explain("executionStats")* para conocer el tiempo. La consulta quedaría así:

    ```
    db.apartments.find({name: /.*11.*/}).explain("executionStats")
    ```

    Con el índice tarda 44ms, sin el índice tarda 30ms. Esto podría darse ya que la cantidad de documentos es relativamente poca.

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.

    Primero es necesario definir la variable *coords*:

    ```
    var coords={
    "type":"MultiPolygon",
    "coordinates":[[[
                    [0.210307, 51.485877],
                    [0.203326, 51.454328],
                    [0.171111, 51.441565],
                    [0.151133, 51.420431],
                    [0.159891, 51.394648],
                    [0.136931, 51.344174],
                    [0.085001, 51.316023],
                    [0.085665, 51.293085],
                    [0.058483, 51.289355],
                    [0.032881, 51.307521],
                    [0.014982, 51.291787],
                    [0.002266, 51.329138],
                    [-0.037918, 51.338705],
                    [-0.047855, 51.326511],
                    [-0.084781, 51.315885],
                    [-0.094352, 51.299355],
                    [-0.12432, 51.28676],
                    [-0.154371, 51.310254],
    ```

```
                [-0.16314, 51.330243],
                [-0.217289, 51.343388],
                [-0.226898, 51.362595],
                [-0.260841, 51.379556],
                [-0.285462, 51.364251],
                [-0.319307, 51.327812],
                [-0.330534, 51.348421],
                [-0.308695, 51.37545],
                [-0.31772, 51.393668],
                [-0.419277, 51.432353],
                [-0.456488, 51.438107],
                [-0.460465, 51.457035],
                [-0.506834, 51.471057],
                [-0.483194, 51.506646],
                [-0.495488, 51.538333],
                [-0.477367, 51.555254],
                [-0.496983, 51.601163],
                [-0.497247, 51.631654],
                [-0.457152, 51.612291],
                [-0.438195, 51.619892],
                [-0.401441, 51.613166],
                [-0.316696, 51.640532],
                [-0.296151, 51.635444],
                [-0.252275, 51.646561],
                [-0.249881, 51.654611],
                [-0.19097, 51.663982],
                [-0.163632, 51.6824],
                [-0.109283, 51.691743],
                [-0.066368, 51.683843],
                [-0.011092, 51.680867],
                [-0.012286, 51.646227],
                [0.062972, 51.60691],
                [0.087118, 51.604465],
                [0.136079, 51.6236],
                [0.168874, 51.621417],
                [0.22406, 51.631734],
                [0.264299, 51.60783],
                [0.290262, 51.564298],
                [0.313007, 51.565816],
                [0.331194, 51.540761],
                [0.265319, 51.532149],
                [0.263655, 51.517869],
                [0.214128, 51.496039],
                [0.210307, 51.485877]
                ]]]
}
```

Luego probamos la consulta `db.apartments.find({ location: { $geoIntersects: { $geometry: coords } } }).explain("executionStats")`

Tarda 147ms.

Para contrastar con un índice, primero hay que crearlo:

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

Luego procedemos a probar la consulta con `db.apartments.find({ location: { $geoIntersects: { $geometry: coords } } }).explain("executionStats")`

En este caso, la primera vez tarda 300ms. Pero luego, tarda 135ms, logrando una mejor performance.

---

## 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. Al igual que en la parte 3, guarde en un archivo llamado 'generadorReservas.js' el siguiente código JavaScript y ejecútelo con la función `load()`:

```
Date.prototype.addDays=function(d){return new Date(this.valueOf()+864E5*d);};
function randomDate(start, end)
{
    return new Date(start.getTime()+Math.random()*(end.getTime()-
start.getTime()));
}
for (var i = 1; i <= 50000; i++)
{
    if (Math.random() > 0.7)
    {
        var randomReservations = Math.ceil(Math.random() * 5);
        for (var r = 1; r <= randomReservations; r++)
        {
            var startDate = randomDate(new Date(2012, 0, 1), new Date());
            var days = Math.ceil(Math.random()*8);
            var toDate = startDate.addDays(days);
            var randomAmount = days * ((Math.random() * 100) + 80).toFixed(2);
            db.reservations.insert({
                apartmentName:'Apartment '+i,
                from: startDate,
                to: toDate,
                amount: randomAmount
            });
        }
    }
}
```

13. Obtenga 5 departamentos aleatorios de la colección.

    ```
    db.apartments.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.createIndex( { location : "2dsphere"  })
db.apartments.aggregate([
   {$geoNear: {
        near: { type: "Point", coordinates: [-0.127718, 51.507451] },
        distanceField: "distance",
        maxDistance: 15000,
        spherical: true}},
   {$out: "centerOfLondon"}
])
```

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.centerOfLondon.aggregate([
    {$lookup: {
            from: "reservations",
            localField: "name",    // field in the centerOfLondon collection
            foreignField: "apartmentName",  // field in the reservations
collection
            as: "myReservations"}},
        {$out: "aparWithRes"}
])
```

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.

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.aparWithRes.aggregate([{
    $project: {
            name: 1,
            avgReservations:{$avg: "$myReservations.amount" }}}
])
```