MONGO

MongoDB (del inglés humongous, "enorme") es un sistema de base de datos NoSQL orientado a documentos. Es la base dedatos no relacional más popular. Su gran fortaleza viene de su flexibilidad.

Instalación

Pre-requisitos:

```
$ sudo apt-get install libcurl4 openssl
```

Para obtener los binarios se puede correr el comando:

```
$ wget https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-ubuntu2004-5.0.6.tgz
```

Y para descomprimir:

```
$ tar -xzvf mongodb-linux-x86_64-ubuntu2004-5.0.6.tgz
```

Instalación

Desde la nueva carpeta en *bin* se puede correr mongo, pero seguramente prefieren copiarlo al *path*.

```
$ cd mongodb-linux-x86_64-ubuntu2004-5.0.6/
$ sudo cp bin/* /usr/local/bin/
```

Instalación

Antes de correr la base de datos, se necesita crear un directorio en el que se va a guardar la información, sin cambiar configuraciones, mongo espera uno por defecto:

```
$ sudo mkdir -p /data/db
$ sudo chown `whoami` /data/db
```

OJO con esas últimas comillas, provocan que la terminal ejecute el comando, si quieren pueden escribir allí su nombre de usuario en lugar de esa instrucción.

mongo shell

También aquí tenemos un comando para comenzar el servidor:

$ mongod

Y otro para el cliente:

$ mongo

mongo shell

Primero lo primero, elegir/crear una base de datos:

```
mongo> use basePrueba;
```

A partir de ese momento "db" se va a referir a la base de datos que elegimos, y vamos a utilizarlo para prácticamente todos los comandos.

Para insertar un elemento, debemos elegir una colección en la que se va a añadir, no hace falta declararla, simplemente la usamos como si ya existiera:

```
mongo> db.coleccion.insertOne({llave:"valor"})
```

mongo shell

Para buscar elementos, se usa la función *find*. OJO, por defecto, *mongo shell* limitará el resultado a 20 elementos (aunque se pueden pedir más). Pero en los drivers, la consulta regresa iteradores.

mongo> db.coleccion.find()

Cuando no recibe ningún argumento, la función *find* regresa todos los elementos.

mongo shell

Así como se puede insertar un documento, se pueden insertar muchos, y recuerden que son documentos completos, no solo llave valor:

```
mongo> db.coleccion.insertMany([
   {
      nombre: "Alejandro" ,
      asistencia: 40 ,
      rol: "profesor" ,
   } , {
      nomre: "X" ,
      asistencia: 35 ,
      rol: "alumno" ,
   }
])
```

mongo shell

También es importante poder especificar las búsquedas, y aquí lo importante es recordar: Cada parámetro de la función es un documento, y cada criterio a buscar es la entrada de un documento:

```
mongo> db.coleccion.find(
  {
    nombre: "Alejandro"
  } , {
    nombre: 1 ,
    rol: 1
  }
)
```

```
SELCT nombre,rol FROM coleccion
  WHERE nombre = "Alejandro"
```

mongo shell

Por supuesto, hay mas condiciones que se pueden buscar:

```
mongo> db.coleccion.find(
   {
      nombre: { $in : [ "X" , "Y" ] }
   }
)
```

```
SELCT * FROM coleccion WHERE nombre in ("X",""Y")
```

mongo shell

OR:

```
mongo> db.coleccion.find(
   {
      $or: [
            { nombre : "Alejandro"} ,
            { rol: "profesor"}
      ]
   }
)
```

```
SELCT * FROM coleccion WHERE nombre = "Alejandro"
  OR rol = "profesor"
```

mongo shell

Rangos:

```
mongo> db.coleccion.find(
   {
      asistencia : { $lt : 30 } ,
   }
)
```

```
SELCT * FROM coleccion WHERE asistencia < 30
```

mongo shell

```
mongo> db.coleccion.find(
   {
      nombre: "Alejandro" ,
      $or: [
            { asistencia : { $gt : 30 } } ,
            { rol: "profesor"}
      ]
   }
)
```

Intenten encontrar el equivalente de esta consulta (la respuesta en la siguiente diapositiva).

mongo shell

```
mongo> db.coleccion.find(
   {
      nombre: "Alejandro" ,
      $or: [
            { asistencia : { $gt : 30 } } ,
            { rol: "profesor"}
      ]
   }
)
```

Intenten encontrar el equivalente de esta consulta (la respuesta en la siguiente diapositiva).

```
SELCT * FROM coleccion WHERE nombre = "Alejandro"
  AND ( rol = "profesor" OR asistencia > 30 )
```

mongo shell

Sin duda ya todos notaron el `"_id":ObjectId("...")`. Cada que se agrega un nuevo valor, se le asigna automáticamente un ID. Así que se necesita algo diferente para cambiar un valor.

```
mongo> db.coleccion.updateOne(
   { apellido: "Y" } ,
   { $set : { apellido : "Z" } }
)
```

mongo shell

O muchos.

```
mongo> db.coleccion.updateMany(
   { asistencia: {$lt:40} } ,
   { $set : { exento : false } }
)
```

mongo shell

Borrar documentos funciona igual.

```
mongo> db.coleccion.deleteOne(
   {
      llave: "valor"
   }
)

mongo> db.coleccion.deleteMany( )
```

mongo shell

Debemos recordar que en mongo no son solo pares llave-valor. Las llaves pueden contener cualquier cosa, desde listas hasta otros documentos a cualquier profundidad.

```
mongo> db.coleccion.insertMany( [
    { p: "carpeta", d: [14, 21], u: "cm", i:{b:"A", c:25 }, s:53 },
    { p: "cuaderno", d: [8.5, 11], u: "in", i:{b:"C", c:75 }, s:26 },
    { p: "hojas", d: [8.5, 11], u: "in", i:{b:"A", c:100 }, s:87 },
    { p: "hojas", d: [8.5, 11], u: "in", i:{b:"B", c:500 }, s:61 },
    { p: "agenda", d: [21, 14], u: "cm", i:{b:"E", c:50 }, s:43 },
    { p: "sobre", d: [10, 15.25], u: "cm", i:{b:"E", c:25 }, s:85 },
    { p: "linterna", i:{b:"B", c:12 }, s:80 }
] )
```

mongo shell

Si se hace una búsqueda igual a las anteriores, tendríamos que especificar la lista/diccionario completo para que coincida, si queremos buscar por elemento, se necesita otra forma.
Comenzando con las listas:

```
mongo> db.coleccion.find( { d: [8.5, 11] } )

mongo> db.coleccion.find({ d: { $all: [8.5, 11] } } )

mongo> db.coleccion.find( { d: 10 } )

mongo> db.coleccion.find( { "d.0": 14 } )
```

mongo shell

En cuanto a los documentos:

mongo> db.coleccion.find({ i: {b:"A", c:100} })
OJO: Con el orden.
mongo> db.coleccion.find({ i: {c:100 , b:"A"} })

mongo> db.coleccion.find({ "i.b": "E" })

Índices

Se pueden crear índices en mongo:

```
mongo> db.coleccion.createIndex(
    { s: 1 },
    { name: "score" }
)
```

Los índices son árboles ordenados. Ese '1' es para indicar que el índice es ascendente, con un '-1' sería descendente.
Para índice múltiple:

```
mongo> db.coleccion.createIndex(
    { p: 1, s: -1 },
    { name: "producto-score" }
)
```

Índices

Texto

Si se trata de texto, el índice no funciona igual, así que hay que especificar:

```
mongo> db.coleccion.createIndex(
    { p: "text" }
)
```

Si se quiere tener más índices de texto para una colección, se pueden crear todos de una vez:

```
mongo> db.coleccion.createIndex(
    { p: "text" , u: "text" },
)
```

Índices

Muy probablemente ya se toparon con un error que no les deja crear un índice porque uno de sus elementos ya tiene un índice:

```
mongo> db.coleccion.getIndexes()
```

Pueden eliminar un índice ya que conocen su nombre:

```
mongo> db.coleccion.dropIndex("p_text")
```

Índices

OJO, los índices de texto sirven para hacer búsquedas de texto pero se toman como todo un conjunto para la colección.
Es decir, cuando se usan, se usan todos juntos.

```
mongo> db.coleccion.find(
    { $text : { $search : "hojas" } }
)
```

Con la instrucción `$text` se le indica que se va a hacer una búsqueda sobre los índices de texto. Esto les permite buscar palabras, todas las que pongan separadas por espacios.

Índices

Otro OJO, si hacen una búsqueda como esta:

```
mongo> db.coleccion.find(
    { $text : { $search : "in" } }
)
```

NO VAN A ENCONTRAR NADA!.

Porque.... "in" es una palabra funcional del inglés.

Índices

Para cambiar eso, se tiene que hacer al momento de crear el índice:

```
mongo> db.coleccion.createIndex(
    { p: "text" },
    { default_language: "spanish" }
)
```

O, si quieren que no considere ninguna palabra funcional:

```
mongo> db.coleccion.createIndex(
    { p: "text" , u: "text" },
    { default_language: "none" }
)
```

GeoJSON

Hay un índice más que vale la pena revisar:

```
mongo> db.collection.createIndex(
    { lugar: "2dsphere" },
)
```

Este índice esta diseñado para hacer consultas relacionadas con consultas espaciales (coordenadas).

GeoJSON

El índice solo funcionará con un tipo de objeto especial llamado GeoJSON:

{ lugar: {type: "Point" , coordinates: [-73.94 , 40.77] } }

Puede ser de diferentes tipos, los mas importantes son:

- Point
- LineString
- Polygon

GeoJSON

Se pueden hacer consultas especiales, por ejemplo:

```
mongo> db.collection.find( {
      lugar: {
            $near: {
                  $geometry: { type: "Point",
                               coordinates: [ -73.9667, 40.78 ] },
                  $maxDistance: 5000
            }
      }
} )
```

Agregación

Con esto se puede hacer una cadena de operaciones una tras otra.
Lo más común (no siempre) es que se usen dos etapas:

- **Match:** Etapa en la que se filtran documentos según algún criterio. Igual que `find()`.
- **Group:** Agrupa los documentos según un identificador y aplica una función para obtener un resultado único.

```
mongo> db.coleccion.aggregate( [
    { $match: { p : "hojas" } },
    { $group: { _id : "$p" , score : { $avg : "$s" } } }
] )
```

Agregación

Existen muchas operaciones para que se pueden agregar a una cadena de `aggregate`.
Por ejemplo, se tiene una instrucción especial para hacer consultas espaciales:

```
mongo> db.collection.aggregate( [ {
      $geoNear: {
            near: { type: "Point",coordinates: [ -73.9667, 40.78 ] },
            query: { categoria: "Universidades"},
            distanceField: "distancia"
      }
} ] )
```

Réplicas

Para trabajar con réplicas, se deben hacer cambios en la configuración. Pueden usar el archivo básico de la plataforma, y cambiar los siguientes campos:

```
replication:
    replSetName: "rs0"
net:
    bindIp: 127.0.0.1,<direccion ip>
```

Revisen también el `path` del `log`, seguramente no existe la carpeta por defecto, cámbienla o agréguenla.

Réplicas

Recuerden usar al menos tres máquinas. En cada una deben cambiar la IP del archivo de configuración y correr:

```
$> mongod --config mongo.conf
```

Réplicas

Desde un cliente se pueden conectar al servidor que quieran que sea el servidor primario y correr:

```
mongo> rs.initiate( {
      _id:"rs0",
      members: [
            {_id: 0, host: "<ip0>"},
            {_id: 1, host: "<ip1>"},
            {_id: 2, host: "<ip2>"},
      ]
} )
```

Réplicas

Ya que se arma la configuración, se puede ver su `status`:

```
mongo> rs.status( )
```

Además, se puede verificar si el nodo al que se está conectado es primario o es réplica:

```
mongo> db.isMaster( )
```

Réplicas

Por supuesto, se puede uno conectar siempre al primary y que el cluster sea el encargado de manejar los cambios:

```
$> mongo --host
rs0/192.168.56.14:27017,192.168.56.15:27017,192.168.56.16:27017
```

Sharding

Mongo necesita tres componentes:

shard: Cada una de las partes en las que se dividen los datos.

mongos: Actúa como el intermediario entre el cliente y el cluster.

config server: Configuración e información del cluster.

El config server se compone a su vez de un set de réplicas.

```
replication:
    replSetName: "rs0"
sharding:
    clusterRole: configsvr
net:
    bindIp: 127.0.0.1,<direccion ip>
```

Esta configuración va a necesitar una carpeta adicional:

```
$ sudo mkdir -p /data/configdb
$ sudo chown `whoami` /data/configdb
```

Réplicas

Recuerden que se debe correr el servicio con la configuración correspondiente:

```
$> mongod --config mongo.conf
```

Para echar a andar el set de configuración, se hace igual que el de réplicas, pero se debe indicar el tipo:

```
mongo> rs.initiate( {
        _id: "rs0",
        configsvr: true,
        members: [
                {_id: 0, host: "<ip0>"},
                {_id: 1, host: "<ip1>"},
                {_id: 2, host: "<ip2>"},
        ]
} )
```

Cada elemento de cada `shard server` debe tener su configuración correspondiente, recuerden sobre todo cambiar el `id`:

```
replication:
    replSetName: "rs1"
sharding:
    clusterRole: shardsvr
net:
    bindIp: 127.0.0.1,<direccion ip>
```

Para echar a andar el set, ya no se debe indicar el tipo, pero si recuerden poner el `_id` correcto:

```
mongo> rs.initiate( {
        _id: "rs1",
        members: [
                {_id: 0, host: "<ip0>"},
                {_id: 1, host: "<ip1>"},
                {_id: 2, host: "<ip2>"},
        ]
} )
```

Y lo mismo para un set más (por lo menos).

Nuestro `mongos` también debe tener una configuración particular:

```
sharding:
    configDB: rs0/<ip1:puerto>,<ip2:puerto>,<ip3:puerto>
```

Correr `mongos` es igual que correr `mongod`:

```
$> mongos --config mongo.conf
```

Ya que se está conenctado con `mongos` se puden agregar los shards:

mongos>
sh.addShard("rs1/<ip1:puerto>,<ip2:puerto>,<ip3:puerto>")

Sharding

Para que las colecciones efectivamente sean distribuidas, se debe habilitar la base de datos:

```
mongos> sh.enableSharding("shardedBase")
```

Y para las colecciones, se especifica que llave sobre la que se hará la distribución:

```
mongos>
sh.shardCollection("shardedBase.coleccion1",{llave_hashed:"hashed"})
```

```
mongos>
sh.shardCollection("shardedBase.coleccion2",{llave_rango:1})
```

Actividades

Datos:
`https://s3.amazonaws.com/tripdata/201307-201402-citibike-tripdata.zip`

Desarrollar un sistema de paseos en el que:

- Un usuario se pueda dar de alta junto con información de ubicación (coordenadas), debe usar al menos un lugar (casa por ejemplo) pero con opción de agregar más (trabajo, escuela, etc.)
- El usuario debe poder buscar las estaciones de bicicleta que le queden más cerca a sus lugares.

Actividades

- También debe poder planear viajes, dado un tiempo que quiere viajar, el sistema debe recomendar viajes usando como salida sus estaciones mas cercanas (o estaciones específicas seleccionadas por el usuario) y los destinos que le tomen mas o menos ese tiempo.
- El usuario debe poder dar la opción de que su viaje sea redondo (mismo punto de partida y salida). En este caso, solo se debe tomar en cuenta los datos que pasan por otras estaciones, a menos que el tiempo sea muy corto.
- Si el usuario busca una ruta específica que el sistema no tenga, la debe intentar armar de forma indirecta.

- EXTRA: Los resultados deben tomar en cuenta la hora a la que se hace la consulta y la hora de los viajes registrados para dar sus resultados con mayor confianza.