TEMA 11
INTRODUCCIÓN A MONGO DB

1.1. ANTECEDENTES BÁSICOS.

1.1.1. Principales categorías de modelos de datos NoSQL

- Key-Value
- Document
- Column-Family
- Graph

Los primeros 3 hacen uso del concepto de "***agregaciones***", o modelos orientados a agregaciones.

1.1.2. Concepto de agregación

- Unidades de datos de estructuras complejas comparadas con una tupla en el modelo relacional.
- Permiten anidar estructuras de datos.
- Su origen viene de los conceptos empleados en DDD (Domain Driven Design):
    - Una agregación es una colección de objetos relacionados entre sí que se desea tratarlos como una sola unidad de datos.
- Se desea realizar operaciones CRUD en términos de agregaciones.
- Las agregaciones son adecuadas para bases de datos en clusters, representan la unidad básica de replicación.

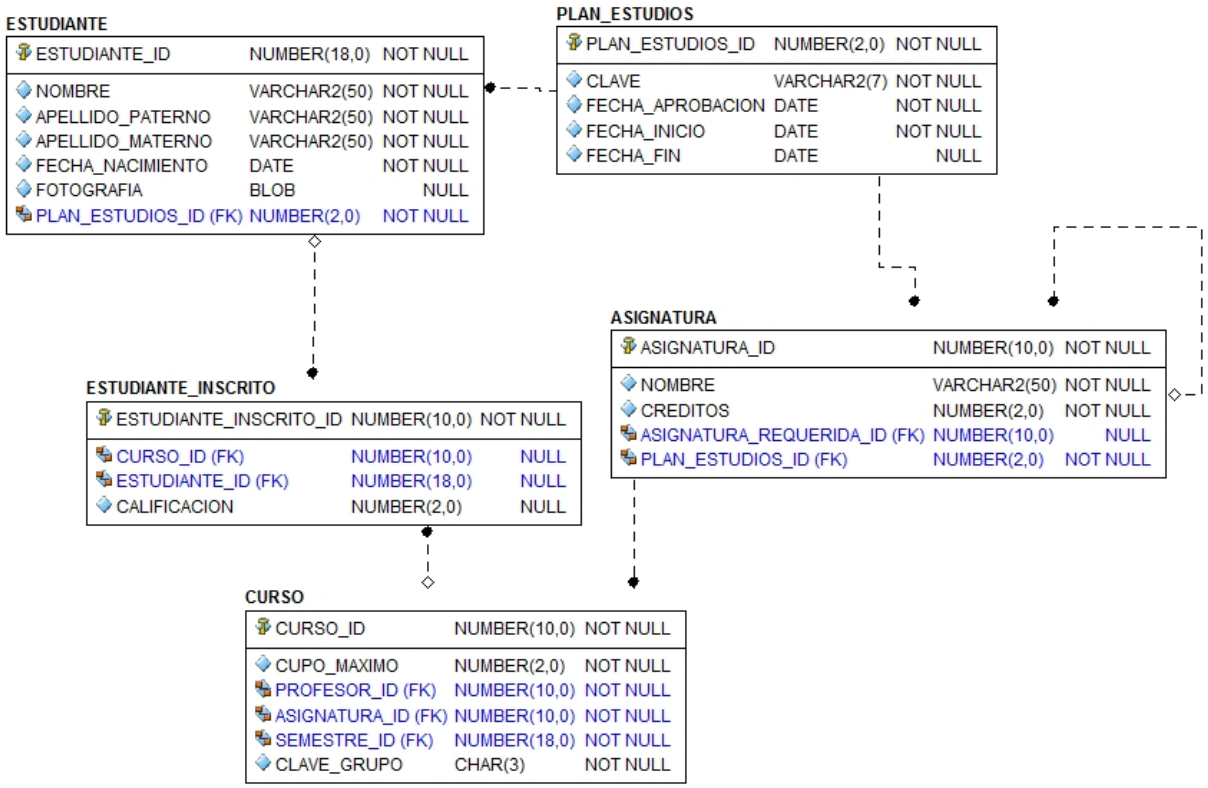
La manera en la que se diseñan las agregaciones es totalmente particular a cada solución.

- En el modelo relacional las tablas se normalizan a nivel general en 3FN como solución general para todos los casos.
- En el modelo relacional las agregaciones se modelan empleando un conjunto de tablas relacionadas entre sí.
- Sin embargo, no todas las relaciones son agregaciones, no hay manera de distinguir cuales si y cuales no (relación de agregación).
- Por lo anterior al modelo relacional se le llama “aggregate-ignorant".
- En NoSQL, graph Databases también son "aggregate-ignorant".
- Esta característica permite representar unidades de datos en diferentes contextos, por ejemplo:

    - Representar al historial de las órdenes de compra de un cliente como una agregación, para facilitar su análisis para unos casos, ó
    - Representar al historial de forma separada, donde importa más la orden actual y no la historia (procesamiento). En este caso si se desea analizar el historial, cada entrada se tendría que extraer uno a uno.

Ejemplo:

En el modelo relacional se tienen los datos de los alumnos en estas 4 relaciones.



Para un diseño en particular, se define una agregación estudiante que contiene los siguientes datos.

- Datos generales
- Datos de su plan de estudios
- Lista de cursos a los que se inscribe
- Calificaciones obtenidas por curso.

Un ejemplo de agregación para un estudiante es la que se muestra a continuación.

```
{
  "nombre": "Gerardo",
  "apellidoPaterno": "Martinez",
  "apellidoMaterno": "Lopez",
  "fechaNacimiento": "1984-09-03T00:00:00.000Z",
  "planEstudios": {
    "clave": "PL-002",
    "fechaAprobacion": "2017-04-03T00:00:00.000Z",
    "fechaInicio": "2018-01-01T00:00:00.000Z",
    "fechaFin": null
  },
  "cursos": [
    {
      "claveGrupo": "001",
      "cupoMaximo": 40,
      "semestre": "2008-1",
      "numProfesor": 1,
      "asignatura": {
        "nombre": "Algebra",
        "creditos": 8
      },
      "calificacion": 8.5
    },
    {
      "claveGrupo": "001",
      "cupoMaximo": 40,
      "semestre": "2008-1",
      "numProfesor": 10,
      "asignatura": {
        "nombre": "Algebra Lineal",
        "creditos": {},
        "antecedente": {
          "nombre": "Algebra",
          "creditos": 8
        }
      },
      "calificacion": 10
    }
  ]
}
```

`planEstudios` representa un ejemplo de un documento embebido.

Es importante mencionar que diversos aspectos de diseño deben ser tomados en cuenta para decidir y definir el diseño de agregaciones. A nivel general se pueden considerar los siguientes aspectos:

- Los datos deben estar almacenados de tal forma que represente la forma más adecuada y útil para la aplicación que accede a ellos.
- Se debe pensar en términos de "***Application data Patterns***" (Patrones de los datos identificados en una aplicación).
- Lo anterior implica identificar comportamientos en la aplicación como los siguientes:
  - ¿Qué datos se consultan o se usan juntos?, por ejemplo: los datos generales de los estudiantes y los datos de sus cursos siempre se consultan a la vez.
  - ¿Qué conjunto de datos se consideran como de "*solo de lectura*"?
  - ¿Qué conjunto de datos se escriben o actualizan con alta frecuencia?
- Estos conjuntos de datos deben organizarse en una BD NoSQL para satisfacer estos patrones o comportamientos.

- En un RDBMS se procura que el diseño de la estructura o esquema sea ***agnóstico*** con respecto a la aplicación.

En resumen:

El factor más importante en el diseño del esquema de datos de una aplicación es hacer posible la correspondencia entre los patrones de acceso a datos que fueron identificados con la forma en la que estos son almacenados.

1.1.2.1. ¿Cómo vivir sin constraints, transacciones, o niveles de aislamiento?

Algunas opciones:

- Operaciones atómicas deben ser representadas por un solo documento.
- Implementar estos requerimientos en el software
- Tolerancia: los requerimientos de la aplicación indican que no hay problema si existen lecturas sucias.

1.1.2.2. Algunas ideas para implementar tipos de relaciones.

One-to-One

Opción 1: Documentos separados
Opción 2: Documento embebido.

- Lo anterior puede depender de la forma en la que se acceden los datos (de forma separada o en conjunto).
- Otro aspecto es verificar atomicidad (consistencia) de la relación. Si es importante, la mejor opción es documento embebido.
- Si uno de los documentos es muy grande, mantenerlo embebido puede traer problemas de memoria.

One-to-Many

Opción generalmente viable: Ligar 2 documentos empleando un id (similar a una FK, notar que no existe constraint de referencia).

One-to-Few

Opción generalmente viable: Documento embebido

Few-to-few

Ligar 2 documentos, pero de forma bidireccional.

Many-to-many

Misma estrategia, ligar 2 documentos de forma bidireccional, pero empleando arreglos de Ids.

1.1.3. Transaccionalidad.

- En cuanto a las transacciones, ***NO*** garantiza la implementación de las propiedades ACID para un conjunto de agregaciones, ***se garantiza para una sola agregación*** en un instante de tiempo.
- Si se desea garantizar ACID para un grupo de agregaciones, esto se debe implementar en el código.
- Por lo anterior, esta característica debe ser considerada para diseñar agregaciones.

Nota: MongoDB 4.0 soportará transacciones con múltiples documentos.

1.1.4. Key-Value, Document Data Models

1.1.4.1. Similitudes en estos 2 modelos:

- En este tipo de modelos, a una agregación se le conoce como "Documento".
- Cada agregación es representada por un ***id*** para recuperar datos.

1.1.4.2. Diferencia entre modelos Key – Value y Document

Key-Value:

- La agregación se trata como bonche de datos sin saber de su estructura.
- Solo se puede recuperar la agregación por ID y completa.

Document:

- La BD conoce la estructura de la agregación
- Se define una estructura y tipos de datos para realizar el almacenamiento
- Se pueden hacer búsquedas empleando como criterios el contenido de la agregación.
- Se puede consultar solo parte de la agregación.
- Se pueden crear índices aplicados a algún campo de la agregación.

1.1.5. Column family Stores

- Bases de datos con estructura "BigTable-style" data model llamadas comúnmente ***column stores***
- En esta técnica la unidad básica de almacenamiento ***NO*** es el registro (tupla).
- Se almacena un conjunto de columnas (grupos de columnas) para todos los registros. Estos grupos representan la unidad básica de almacenamiento.
    - Grupos de columnas = Familias de columnas.
    - Este tipo de BDs organizan sus columnas en familias.
    - Cada columna actúa como unidad de acceso y se recuperan todos los datos de la familia.

<u>Ejemplo:</u>

```
row key 1234
     profile(familia 1) -> name, address, payment
     orders (familia 2) -> ord001, ord002, 0rd003
```

- A su vez cada columna puede contener más datos.

Más características:

- Row oriented: Cada registro es una agregación, el registro 1234 con sus 2 familias representa un registro.
- Column oriented: cada familia representa un "record type": profile y orders
- Cada registro es un join de sus familias de columnas.
- Se pueden agregar columnas libremente a cada familia
- Cada registro puede tener columnas diferentes.
- Lo que no es común es definir nuevas familias.
- Esta agrupación de datos es conocida por la BD y puede ser empleada para almacenamiento y acceso.

1.2. CARACTERÍSTICAS GENERALES DE MONGO DB

Mongo DB es una de las bases de datos NoSQL más populares.
Analogías con una BD relacional:

- Base de Datos ⇔ Base de Datos
- Tabla ⇔ Colección
- Registro ⇔ Agregación o Documento
- Columna ⇔ Campo
- Join ⇔ Documentos embebidos o referencias.

1.2.1. Llaves primarias en Mongo DB

- En el modelo relacional la PK puede estar formada por 1 o más columnas.
- En MongoDB existe un campo llamado `_id` que se asigna en automático con un valor único para identificar al documento (similar a una columna auto incrementable).
- Este valor se puede sobrescribir asignando un valor de forma explícita.

Ejemplo:

```
{
  _id: "12345asdfghj7890666",
  nombre: "Gerardo",
  apellidoPaterno: "Martinez",
  apellidoMaterno: "Lopez",
  fechaNacimiento: "1984-09-03T00:00:00.000Z",
  planEstudios: {
    clave: "PL-002",
    fechaAprobacion: "2017-04-03T00:00:00.000Z",
    fechaInicio: "2018-01-01T00:00:00.000Z",
    fechaFin: null
  },
  cursos: [
    {
      claveGrupo: "001",
      cupoMaximo: 40,
      semestre: "2008-1",
      numProfesor: 1,
      asignatura: {
        nombre: "Algebra",
        creditos: 8
      },
      calificacion: 8.5
    },
    {
      claveGrupo: "001",
      cupoMaximo: 40,
      semestre: "2008-1",
      numProfesor: 10,
      asignatura: {
        nombre: "Algebra Lineal",
        creditos: {},
        antecedente: {
          nombre: "Algebra",
          creditos: 8
        }
      },
      calificacion: 10
    }
  ]
}
```

- Observar que en MongoDB se eliminan las comillas que contiene los nombres de los atributos.
- Internamente, Mongo almacena estos documentos en formato BJSON (versión en binario de un documento JSON).
- El tamaño máximo de un documento que soporta MongoDB es de 16 MB.

Algunos beneficios del uso de documentos JSON

- Formato altamente empleado en diversos lenguajes de programación
- Documentos embebidos o anidados reducen la cantidad de operaciones Join.
- Permite el concepto de "***esquema dinámico***"
    - Posibilidad de agregar campos en cualquier momento.

- 2 documentos pueden tener diferente número de campos, por ejemplo, puede existir un documento para un estudiante que defina campos adicionales como: teléfono, dirección.

Algunas características adicionales de MongoDB

- El soporte de documentos embebidos y arreglos reduce las operaciones I/O en la BD.
- Uso de índices que contienen etiquetas para un manejo eficiente de documentos embebidos.
- Lenguaje de acceso a datos robusto:
  - Operaciones CRUD
  - Acceso eficiente a los datos de la agregación
  - Búsqueda de texto
  - Consultas geoespaciales.
- Alta disponibilidad
  - Hace uso del concepto de "Replica Set" formado por un grupo de servidores que contienen los mismos datos (espejos) que ofrecen redundancia y por lo tanto proporcionan una alta disponibilidad.
- Escalamiento horizontal
  - MongoDB fue concebido considerando esta característica como parte de su funcionalidad central
  - Implementa el concepto de "***Sharding***": Método para distribuir grandes volúmenes de datos hacia múltiples servidores optimizando el uso de recursos, en especial, recursos de red.
  - A partir de la versión 3.4 MongoDB soporta la creación de las llamadas "***zonas de datos***" empleando como criterio de construcción un "***shard key***". En un cluster balanceado, MongoDB realiza escrituras y lecturas únicamente hacia los servers que pertenecen a una determinada zona.
- Soporte de múltiples estrategias de almacenamiento
  - WiredTiger Storage Engine (Con soporte para cifrado)
  - In-Memory Storage Engine
  - MMAPv1 Storage Engine

Como se mencionó anteriormente, el diseño de agregaciones es importante. No solo se trata de hacer un mapeo de tablas a documentos.
El siguiente articulo representa una guía inicial para comenzar a diseñar empleado agregaciones: Iniciar pensando o diseñando en términos de ***documentos***.

https://www.mongodb.com/blog/post/thinking-documents-part-1?jmp=docs&_ga=2.146911543.1039731747.1527110438-1230747958.1527110438

1.3. INSTALACIÓN MONGODB COMMUNITY EDITION

- Existe un paquete proporcionado por Ubuntu llamado `mongodb` el cual no es mantenido por MongoDB.
- Por tal razón no se hará uso de este paquete, se hará uso del paquete oficial `mongodb-org`. El uso de dicho paquete permite realizar actualizaciones periódicas.
- Para Ubuntu, únicamente se ofrece soporte para versiones LTS a 64 bits.
- Principales paquetes a instalar:

Paquete	Descripción
mongodb-org	Meta-package que instala de forma automática los siguientes paquetes
mongodb-org-server	Contiene el proceso de background (deamon) `mongod` junto con archivos de configuración
mongodb-org-mongos	Contiene el proceso de background `mongos`
mongodb-org-shell	Contiene el Shell de mongoDB
mongodb-org-tools	Contiene algunas herramientas para trabajar con MongoDB: `mongoimport bsondump, mongodump, mongoexport, mongofiles, mongoperf, mongorestore, mongostat, and mongotop`

1.3.1. Instalación en Ubuntu/Mint

- Para detalles en cuanto al proceso de instalación leer: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/

1. Importar llave pública para verificar consistencia de los paquetes a instalar.

```
sudo apt-key adv --keyserver \
hkp://keyserver.ubuntu.com:80 --recv 2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5
```

2. Crear un archivo `list`. En este archivo se guardará la configuración de los repositorios de código de donde se obtendrán los paquetes.

```
echo "deb [ arch=amd64,arm64 ] \
https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.6 \
multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.6.list
```

Modificar el nombre de la distribución marcada en negritas en caso de emplear una versión diferente.

3. Actualizar la base de datos de paquetes.

```
sudo apt-get update
```

4. Instalar los paquetes

```
sudo apt-get install -y mongodb-org
```

1.3.2. Instalación en Fedora.

1. Crear el siguiente archivo para agregar los repositorios de mongodb

```
sudo touch  /etc/yum.repos.d/mongodb-org-3.6.repo
```

2. Agregar el siguiente contenido al archivo

```
[mongodb-org-3.6]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/redhat/7/mongodb-org/3.6/x86_64/
```

```
gpgcheck=1
enabled=1
gpgkey=https://www.mongodb.org/static/pgp/server-3.6.asc
```

Para abrir el archivo: `sudo nano /etc/yum.repos.d/mongodb-org-3.6.repo`

3. Instalar paquetes:

```
sudo yum install -y mongodb-org
```

1.4. INICIAR MONGODB

- El proceso de instalación anterior crea un usuario en el sistema operativo llamado `mongodb`.
- La instancia de MongoDB guarda datos por default en `/var/lib/mongodb`, los archivos log en `/var/log/mongodb` por default.
- Esta configuración se puede cambiar en el archivo `/etc/mongod.conf`
- La instancia se inicia con el usuario root, pero el proceso es ejecutado por el usuario `mongodb`.
- Para iniciar la instancia, ejecutar el siguiente comando:

```
sudo service mongod start
```

- Para verificar que la instancia se ha iniciado, ejecutar los siguientes comandos:

```
cat /var/log/mongodb/mongod.log
ps -ef | grep mongo
```

Deberá mostrar un mensaje al final indicando que el inicio fue correcto y se está en espera de peticiones.

- Comando para detener la instancia:

```
sudo service mongod stop
```

1.5. SHELL DE MONGODB

- Ejecutar la siguiente instrucción para iniciar el Shell de MongoDB

```
mongo --host 127.0.0.1:27017
```

1.6. COLECCIONES Y BASES DE DATOS

MongoDB guarda los documentos en colecciones, y una base de datos puede tener múltiples colecciones.

- Para emplear una base de datos en particular se emplea el comando `use`.

```
use <nombreBaseDeDatos>
```

- Si la base de datos no existe, se creará una nueva.
- Si no se emplea el comando `use`, por default se accederá a una base de datos llamada `test`.
- El comando `show dbs` muestra las bases de datos existentes en el servidor.

Ejemplo:

```
use tema11
```

1.6.1. Operaciones CRUD

1.6.1.1. Inserción de documentos.

```
db.controlEscolar.insertOne(
    {
       nombre: "Jorge",
       apellidoPaterno: "Rodriguez",
       apellidoMaterno: "Campos",
       email: "jorgerdc@gmail.com",
       semestre: 10
    }
);
```

- En este ejemplo se crea un nuevo documento en la colección `controlEscolar`. Si la colección no existe, esta se creará.
- Por default, los documentos pueden tener cualquier estructura (esquema). Pueden tener diferentes atributos.

Ejemplo:

```
db.controlEscolar.insertOne(
    {
       nombre: "Lucy",
       apellidoPaterno: "Lara",
       RFC: "LALURO980301LZ1"
    }
);
```

- A Partir de la versión 3.6 es posible validar la estructura (esquema) de un documento JSON.

Ejemplo:

```
db.createCollection("students", {
   validator: {
      $jsonSchema: {
         bsonType: "object",
         required: [ "name", "year", "major", "gpa" ],
         properties: {
            name: {
               bsonType: "string",
               description: "must be a string and is required"
            },
            gender: {
               bsonType: "string",
               description: "must be a string and is not required"
            },
            year: {
               bsonType: "int",
               minimum: 2017,
               maximum: 3017,
               exclusiveMaximum: false,
               description: "must be an integer in [ 2017, 3017 ] and is required"
            },
            major: {
               enum: [ "Math", "English", "Computer Science", "History", null ],
               description: "can only be one of the enum values and is required"
            },
            gpa: {
               bsonType: [ "double" ],
               minimum: 0,
               description: "must be a double and is required"
            }
         }
      }
   }
})
```

- Observar el uso de `db.createCollection` . En este caso se creará una colección llamada `students` cuyos documentos deben cumplir con el esquema (estructura) que se define en el código anterior.

Ejemplo:

- Crear una colección de planes de estudio en el que se valide la siguiente estructura:
  - Nombre del plan de estudios: requerido, string
  - Descripción del plan de estudios: opcional string
  - Tipo de plan: solo puede tener los valores ordinario, y extraordinario
  - Fecha de aprobación: debe ser una fecha válida, requerido
  - Numero de asignaturas del plan: no debe ser mayor a 150, requerido

- Insertar un documento que cumpla con el esquema anterior.

Solución:

```
db.createCollection("planEstudios",
   {
      validator: {
         $jsonSchema:{
            bsonType: "object",
            required: ["nombre","tipo","fechaAprobacion","numeroAsignaturas"],
            properties: {
               nombre: {
                  bsonType: "string",
                  description: "Nombre del plan requerido, tipo string",
               },
               description: {
                  bsonType: "string"
               },
               tipo: {
                  enum: ["ordinario","extraordinario"]
               },
               fechaAprobacion: {
                  bsonType: "string",
                  description: "Fecha de aprobacion del plan de estudios"
               },
               numeroAsignaturas: {
                  bsonType: "int",
                  maximum: 150
               }
            }
         }
      }
   }
```

- Ejemplo de un documento válido:

```
db.planEstudios.insertOne(
   {
      nombre: "PL-001",
      descripcion: "Plan de estudios moderno",
      tipo: "ordinario",
      fechaAprobacion: "2010-01-01",
      numeroAsignaturas: NumberInt(100)
   }
)
```

1.6.1.2. Eliminar objetos.

- Para eliminar una colección

```
db.<collection_name>.drop() Elimina todos los documentos más rápido.
db.<collection_name>.remove({<predicado>})
db.<collection_name>.deleteOne({<predicado>})
db.<collection_name>.deleteMany({<predicado>})
```

1.6.2. Búsquedas y consultas de documentos.

Para ilustrar los ejemplos de esta sección, importar el archivo `estudiantes.json`:

```
mongoimport --db tema11 --collection estudiantes --file estudiantes.json
```

Notar que el comando `mongoimport` se ejecuta a nivel de sistema operativo.

1.6.2.1. Búsqueda de documentos.

Uso de `find`. Sintaxis general.

```
db.collection.find( <query filter>, <projection> )
```

- Regresa un documento al azar

```
db.estudiantes.findOne({name:"Marcus Blohm"})
```

- Búsqueda con criterios de búsqueda.

```
db.estudiantes.find({name:"Marcus Blohm"})
```

- Búsqueda indicando los campos que serán mostrados.

```
db.estudiantes.find({name:"Bao Ziglar"},{name:true,_id:false})
```

- Uso de un cursor interno. Por default se crea un cursor empleado para recuperar el resultado de una búsqueda. Muestra hasta 20 documentos por cada página.
- Se emplea la instrucción `it` para mostrar el siguiente conjunto de resultado.

```
db.estudiantes.find()
```

- Uso de operaciones de comparación.

```
$lte less than or equal
$gte
$lt

db.estudiantes.find({_id:{$gte:190}})

db.estudiantes.find({name:{$gte:"A",$lt:"C"}},{name:true})
```

- Expresiones regulares

```
db.estudiantes.find({name:{$regex:"y$"}},{name:true})

db.estudiantes.find({name:{$regex:"^A"}},{name:true})

db.estudiantes.find({name:{$regex:"w"}},{name:true})
```

- And, Or

```
$or
$and
$in
$all

db.estudiantes.find({$or:[{name:{$regex:"^A"}},
       {name:{$regex:"^B"}}]},{name:true})
```

En general la estructura de estos operadores es:

```
$or: [{},{},{},{},…]
```

- Búsqueda en arreglos.

```
db.estudiantes.find({"scores.score":{$gte:90}},
       {name:true,"scores.score":true})
```

En este ejemplo, al menos un elemento del arreglo debe tener calificación mayor o igual a 90

- Conteo de documentos.

```
db.estudiantes.count()
```

1.6.2.2. Uso de cursores.

- Obtiene un cursor sin mostrar resultados empleando `null`;

```
cur = db.estudiantes.find(); null;
```

- Verifica si existen más elementos.

```
cur.hasNext()
```

- Obtiene el siguiente elemento.

```
cur.next()
```

- Ordenamiento: >0 ascendente, <= descendente.

```
cur.sort({name:-1});null
```

1.6.3. Actualización de documentos

- Reemplazo de documentos:

```
db.profesores.insertOne({
  nombre: "Juan Aguirre",
  asignaturas: ["algebra","calculo"],
  tipo: 1
})

db.profesores.update(
  {nombre: "Juan Aguirre"},
  {nombre: "Juan Aguirre",tipo: 1, numProfesor: 1501}
)
```

  - Primer parámetro: criterios de búsqueda
  - Segundo parámetro: Documento que se va a reemplazar. ¿Qué le pasó al arreglo de asignaturas después del hacer `update`?
    El documento original es reemplazado por el documento indicado en el segundo parámetro el cual no contiene el arreglo de asignaturas. Para evitar reemplazo de documentos, emplear `$set`.

- Uso de `$set`

```
db.profesores.update(
   {numProfesor: 1501},
   {$set:{asignaturas: ["algebra","calculo"]}}
)
```

```
db.profesores.find({numProfesor: 1501})
```

En este caso, se modifica o se agrega en caso de no existir, el campo asignaturas al documento cuyo número de profesor sea el 1501

- Uso de `$push, $pop, $pull, $addToSet`

En general se emplean para manipular los elementos de un arreglo.

- Actualizar los datos del primer elemento:

```
db.profesores.update(
   {numProfesor: 1501},
   {$set:{"asignaturas.0":"algebra lineal"} }
)
```

- Agregar un elemento al arreglo

```
db.profesores.update(
   {numProfesor: 1501},
   {$push:{"asignaturas":"bases de datos"} }
)
```

- Eliminar el último elemento del arreglo.

```
db.profesores.update(
   {numProfesor: 1501},
   {$pop:{"asignaturas":1}}
)
```

- Eliminar el primer elemento del arreglo.

```
db.profesores.update(
   {numProfesor: 1501},
   {$pop:{"asignaturas":-1}}
)
```

- Agregar N elementos a un arreglo existente.

```
db.profesores.update(
   {numProfesor: 1501},
   {
      $push:{"asignaturas": {$each: ["estadistica","quimica"]}}
   }
)
```

- Agregar un elemento solo en caso de no existir.

```
db.profesores.update(
   {numProfesor: 1501},
   {$addToSet:{asignaturas:"estadistica"}}
)
```

- Eliminar elementos de un arreglo por valor.

```
db.profesores.update(
    {numProfesor: 1501},
    {
        $pull:{asignaturas:"quimica"}
    }
)
```

1.6.3.1. Multi - Updates

- ¿Qué debería hacer la siguiente instrucción?

```
db.profesores.update({},{$set:{"tipo": 2}})
```

La instrucción solo actualiza el primer documento, a pesar de que se espera que se actualicen todos. Para lograrlo, se debe indicar la siguiente opción:

```
db.profesores.update({},{$set:{"tipo": 2}},{multi: true})
```

O de forma adicional:

```
db.profesores.updateMany({},{$set:{"tipo": 1}})
```

1.7. TEXT SEARCH

MongoDB soporta la ejecución de operaciones de búsqueda sobre texto. Para ello se hace uso de un índice especial para manejo de texto y el operador `$text`.

- Para realizar los ejercicios de esta sección, descargar el archivo `tweets.zip` de la carpeta del tema 11.
- Descomprimir, cambiarse a la carpeta `dump/twitter` , observar la existencia del archivo `tweets.bson`
- Importar la colección en `mongodb` empleando el siguiente comando:

```
mongorestore -d twitter -c tweets tweets.bson
```

- Comprobar el número de documentos almacenados.

```
use twitter
db.tweets.count()
```

- Revisar los índices que contiene la colección:

```
db.tweets.getIndexes()
```

- Observar la salida del comando anterior para obtener el nombre del índice que fue asociado a un campo llamado “text”, borrar el índice.

```
db.tweets.dropIndex("text_text")
```

- La colección anterior contiene 2 campos: `text y user.description`
- Crear un índice sobre estos 2 campos para realizar búsquedas:

```
db.tweets.createIndex({ text: "text", "user.description": "text" })
```

1.7.1. Operador $text

Básicamente el operador `$text` hace un "String tokenizer" del texto empleando como separadores puntos, espacios, etc., y aplica operaciones OR considerando el predicado proporcionado para encontrar los resultados solicitados.

Sintaxis general:

```
{
  $text:
    {
      $search: <string>,
      $language: <string>,
      $caseSensitive: <boolean>,
      $diacriticSensitive: <boolean>
    }
}
```

Mostrar el valor del campo `texto` y el valor del campo `created_at` para los siguientes ejercicios:

- Mostrar tweets que tengan las palabras Apple, IPhone

```
db.tweets.find(
    {$text:{$search: "Apple iPhone"}},
    {created_at:true,text:true, _id:false}
)
```

- Mostrar tweets que contengan la palabra "the Apple Watch"

```
db.tweets.find(
    {$text:{$search: "\"the Apple Watch\""}},
    {created_at:true,text:true,_id:false}
)
```

- Mostrar tweets que contengan HTML pero que no contengan palabras negativas como : 'hate'

```
db.tweets.find(
    {$text:{$search: "-hate HTML"}},
    {created_at:true,text:true,_id:false}
)
```

1.8. ROBO 3T

- Robo 3T es una herramienta gráfica que permite visualizar el contenido de las bases de datos de un servidor.
- Existe otro producto llamado Studio 3T con mayores funcionalidades (comercial).

1.8.1. Instalar Robo 3T

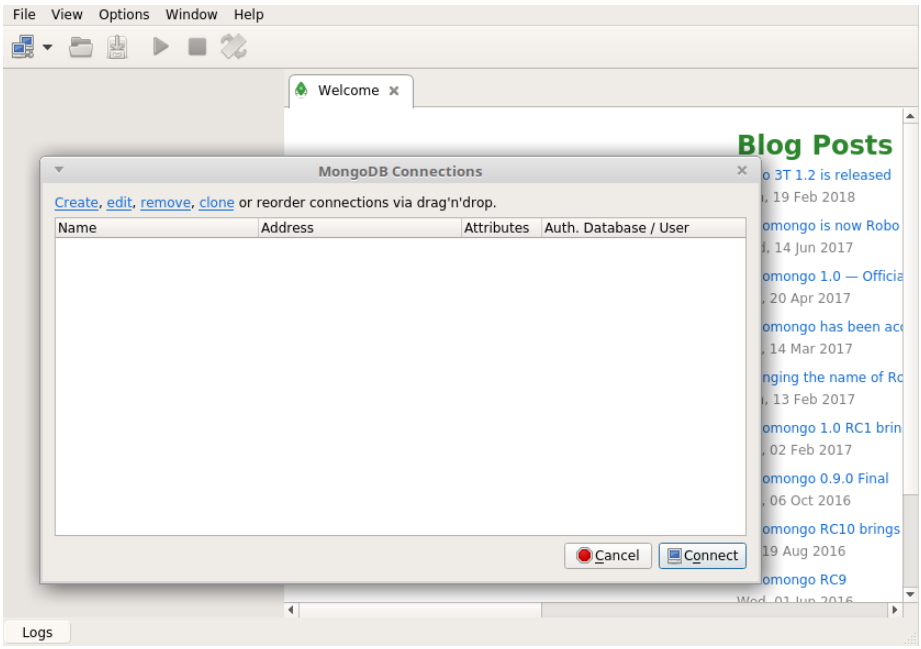
- Descargar el archivo correspondiente al sistema operativo de https://robomongo.org/download

Para el caso de Linux:

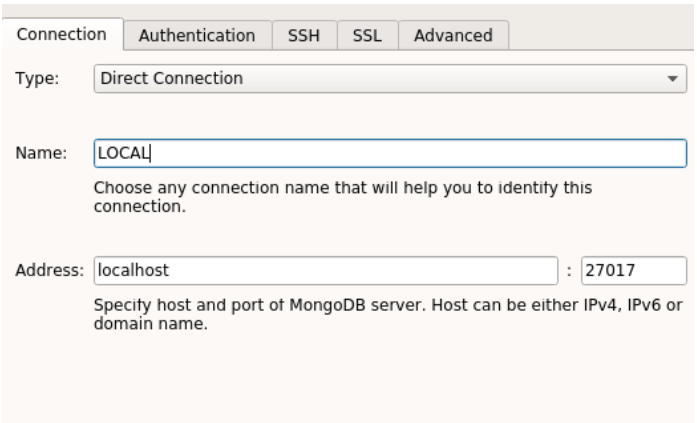
- Descomprimir el archivo `robo3t-X.X.X-linux-x86_64-XxXXxXx.tar.gz`
- Ejecutar el siguiente archivo dentro de la carpeta `bin`:

```
cd robo3t-X.X.X-linux-x86_64-XxXXxXx/bin
./robot3t
```

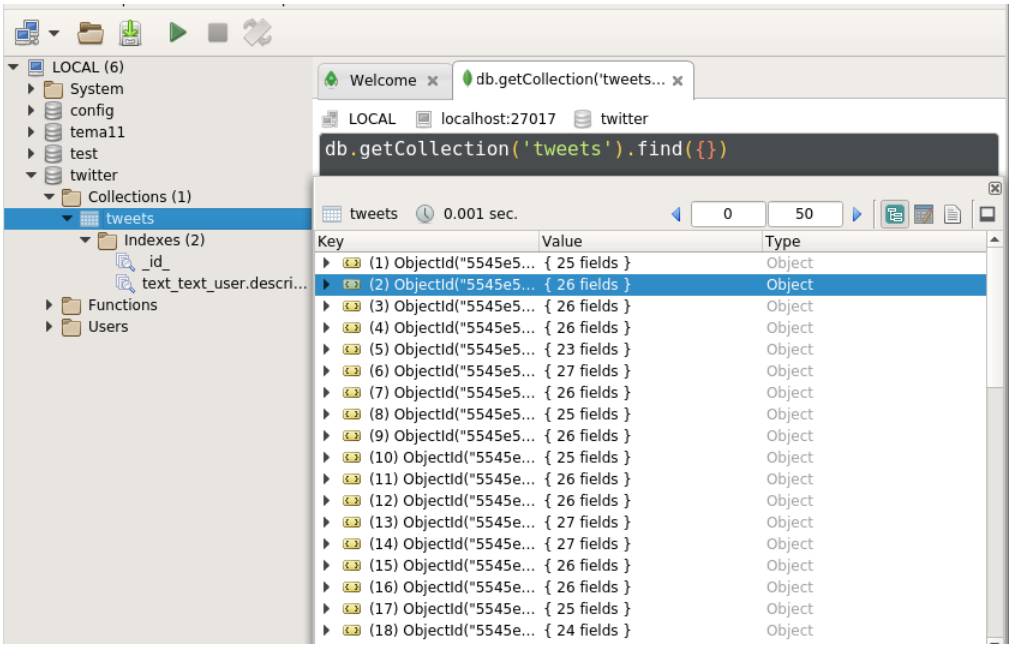
La interfaz gráfica se muestra a continuación.



- Crear una conexión local como se muestra en la siguiente figura:



- Verificar la conexión, visualizar las bases de datos creadas y explorar sus colecciones y documentos.
- Explorar las 3 vistas disponibles para mostrar el contenido de un documento.
- Realizar algunas consultas empleando el Shell embebido.


LOCAL (6)
System
config
tema11
test
twitter
Collections (1)
tweets
Indexes (2)
_id_
text_text_user.descri...
Functions
Users
Welcome
db.getCollection('tweets...
LOCAL
localhost:27017
twitter
db.getCollection('tweets').find({})
tweets
0.001 sec.
0
50
Key
Value
Type
(1) ObjectId("5545e5... { 25 fields } Object
(2) ObjectId("5545e5... { 26 fields } Object
(3) ObjectId("5545e5... { 26 fields } Object
(4) ObjectId("5545e5... { 26 fields } Object
(5) ObjectId("5545e5... { 23 fields } Object
(6) ObjectId("5545e5... { 27 fields } Object
(7) ObjectId("5545e5... { 26 fields } Object
(8) ObjectId("5545e5... { 25 fields } Object
(9) ObjectId("5545e5... { 26 fields } Object
(10) ObjectId("5545e... { 25 fields } Object
(11) ObjectId("5545e... { 26 fields } Object
(12) ObjectId("5545e... { 26 fields } Object
(13) ObjectId("5545e... { 27 fields } Object
(14) ObjectId("5545e... { 27 fields } Object
(15) ObjectId("5545e... { 26 fields } Object
(16) ObjectId("5545e... { 26 fields } Object
(17) ObjectId("5545e... { 25 fields } Object
(18) ObjectId("5545e... { 24 fields } Object