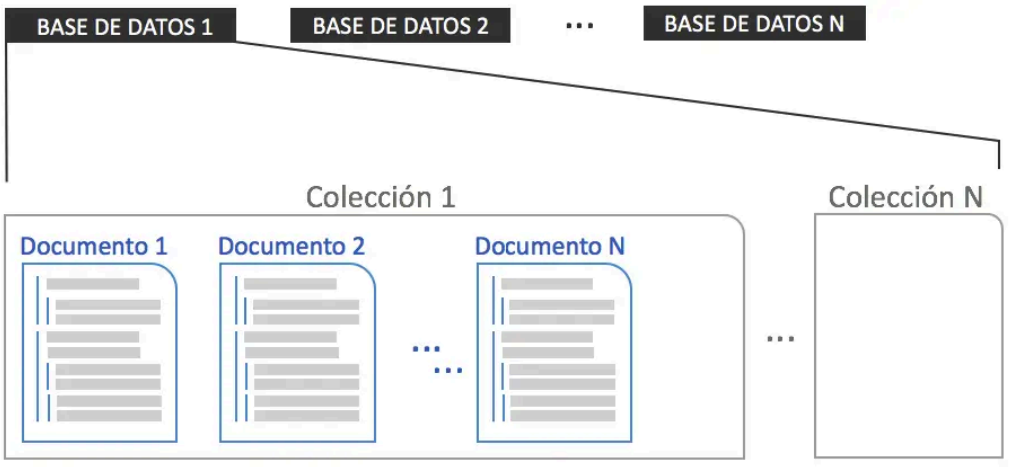
4. BASES DE DATOS ORIENTADAS A DOCUMENTOS

4.1. Bases de datos orientadas a documentos

- Utilizan documentos para almacenar datos.
- Pensadas para almacenar **datos semi-estructurados**.
- Ofrecen una gran versatilidad.
- Se le asigna un identificador único a cada documento de tal forma que si se quieren consultar no es necesario recorrer todas las columnas de una tabla como en una base de datos relacional.
- Por lo general, se ocupan documentos JSON o XML.
- Algunos casos de uso: Administración de contenido, catálogos.



4.2. Key-Value & Document Data Models

4.2.1. Similitudes

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

4.2.2. Diferencias.

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.

4.3. Concepto de agregación

Los 3 patrones de almacenamiento de bases de datos NoSQL pueden hacer uso del concepto de agregación

- Key-Value
- Document
- Column-Family

Una agregación representa 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***.

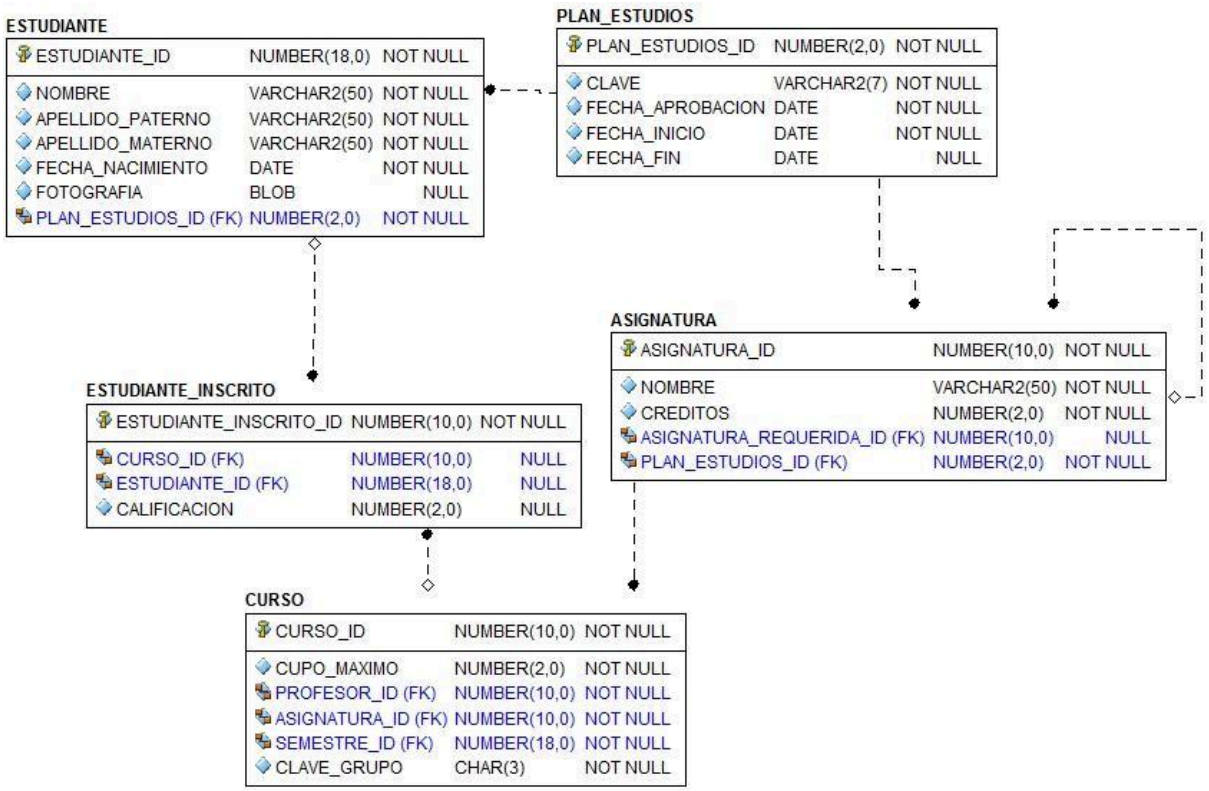
La manera en la que se diseñan las agregaciones es totalmente particular a cada solución.

- 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
    }
  ]
}
```


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.

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

Algunas opciones:

- Las 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.

4.3.2. Algunas ideas para implementar tipos de relaciones.

4.3.2.1. 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 la 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.

4.3.2.2. One-to-Many

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

4.3.2.3. One-to-Few

Opción generalmente viable: Documento embebido

4.3.2.4. Few-to-few

Ligar 2 documentos, pero de forma bidireccional.

4.3.2.5. Many-to-many

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

4.3.3. 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.

4.4. Características generales de MongoDB

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

BD Relacional	Mongo DB
Base de Datos	Base de Datos
Tabla	Colección
Registro	Agregación o Documento
Columna	Campo
Join	Documentos embebidos o referencias

- 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 o 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.
En este artículo representa una guía inicial para comenzar a diseñar empleando agregaciones: Iniciar pensando o diseñando en términos de documentos

4.4.1. Llaves primarias en MongoDB

- 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 contienen 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.

4.5. Instalación de MongoDB Community Edition en Ubuntu/Mint

- 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 (daemon) `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

A. Instalación de dependencias

```
sudo apt-get install gnupg curl
```

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

```
curl -fsSL https://www.mongodb.org/static/pgp/server-7.0.asc | \
sudo gpg -o /usr/share/keyrings/mongodb-server-7.0.gpg --dearmor
```

Se deberá mostrar OK cuando se ejecute dicho comando.

C. Crear un archivo list en `/etc/apt/sources.list.d/mongodb-org-7.0.list` . En este archivo se guardará la configuración de los repositorios de código de dónde se obtendrán los paquetes.

```
echo "deb [ arch=amd64,arm64
signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg ]
https://repo.mongodb.org/apt/<ubuntu_codename>/mongodb-org/7.0 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-7.0.list
```

Sustituir el valor de `ubuntu_codename` por el correspondiente.

D. Actualizar la base de datos de paquetes.

```
sudo apt-get update
```

E. Instalar los paquetes

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

4.6. Iniciando 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.

A. Para iniciar la instancia, ejecutar el siguiente comando:

```
sudo systemctl start mongod
```

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

```
sudo systemctl status mongod
```

Se debe mostrar una salida similar a la siguiente:

```
● mongod.service - MongoDB Database Server
     Loaded: loaded (/lib/systemd/system/mongod.service; disabled; vendor
```

```
preset: enabled)
     Active: active (running) since Tue 2023-03-07 16:05:01 CST; 7min ago
       Docs: https://docs.mongodb.org/manual
   Main PID: 41041 (mongod)
     Memory: 130.9M
     CGroup: /system.slice/mongod.service
             └─41041 /usr/bin/mongod --config /etc/mongod.conf
```

C. Comando para detener la instancia:

```
sudo systemctl stop mongod
```

4.7. Características generales de MongoDB

MongoDB 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.

- 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 o 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.
En este artículo se presenta una guía inicial para comenzar a diseñar empleando agregaciones: Iniciar pensando o diseñando en términos de documentos.

4.7.1. Llaves primarias en MongoDB

- 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 contienen 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.

4.8. Shell de MongoDB

A. Ejecutar la siguiente instrucción para iniciar el Shell de MongoDB.

```
mongosh
```

4.9. Colecciones y bases de datos

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 accede a una base de datos llamada `test`.
- El comando `show dbs` muestra las bases de datos existentes en el servidor.
- Para eliminar una base de datos en mongodb:

Ejemplo:

```
use <databaseName>
db.dropDatabase()
```

Ejemplo:

Crear una base de datos llamada `tema4`. En caso de existir, eliminarla.

```
use tema4
```

4.10. Operaciones CRUD

4.10.1. Inserción de documentos.

1. Cambiarse a la base de datos `tema4`
2. Crear el siguiente documento

```
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.

```
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.

Ejercicio en clase

Realizar las actividades del siguiente ejercicio. La respuesta no se incluye en estas notas, se sugiere agregar la respuesta en sus apuntes.

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
- Número de asignaturas del plan: no debe ser mayor a 150, requerido

Insertar un documento que cumpla con el esquema anterior.

4.10.2. Eliminación de documentos.

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>})
```

4.11. Búsquedas y consultas de documentos

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

```
mongoimport --db tema4 --collection estudiantes --file estudiantes.json
```

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

4.11.1. 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()
```

4.11.2. 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.

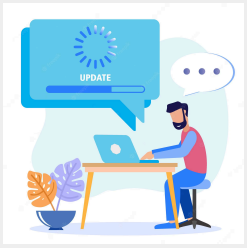
```
cur.sort({name:-1});null
```

4.12. 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 de 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.
```

- Eliminar elementos de un arreglo por valor.

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

1.1.1. 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}})
```

1.2. Búsquedas de documentos

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 4.
- 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.countDocuments()
```

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" })
```

1.2.1. 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 `text` 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}
)
```

1.3. Operaciones sobre agregaciones

Esta funcionalidad permite procesar múltiples documentos, aplicar algún cálculo o procesamiento para finalmente regresar un resultado. Operaciones típicas que se aplican a un conjunto de documentos son:

- Agrupar valores que provienen de diferentes documentos
- Realizar alguna operación sobre los datos agrupados, regresar un solo resultado
- Analizar cambios en los datos a lo largo del tiempo

El método recomendado para realizar este tipo de operaciones es el llamado ***Aggregation pipeline***

1.3.1. Aggregation pipelines

Un aggregation pipeline está formado por una serie de etapas (stages) encargadas de procesar documentos.

- En cada etapa se realiza una operación sobre los documentos de entrada. Por ejemplo, en una primera etapa, se aplica un filtro de documentos, posteriormente se agrupan para finalmente calcular algún valor.
- La salida de un stage corresponde a la entrada del siguiente.
- Para realizar operaciones de tipo aggregation pipeline se emplea el método `db.collection.aggregate()`.

Ejemplos

Crear la siguiente colección de datos

```
db.orders.insertMany( [
   { _id: 0, name: "Pepperoni", size: "small", price: 19,
     quantity: 10, date: ISODate( "2021-03-13T08:14:30Z" ) },
   { _id: 1, name: "Pepperoni", size: "medium", price: 20,
     quantity: 20, date : ISODate( "2021-03-13T09:13:24Z" ) },
   { _id: 2, name: "Pepperoni", size: "large", price: 21,
     quantity: 30, date : ISODate( "2021-03-17T09:22:12Z" ) },
   { _id: 3, name: "Cheese", size: "small", price: 12,
     quantity: 15, date : ISODate( "2021-03-13T11:21:39.736Z" ) },
   { _id: 4, name: "Cheese", size: "medium", price: 13,
     quantity:50, date : ISODate( "2022-01-12T21:23:13.331Z" ) },
   { _id: 5, name: "Cheese", size: "large", price: 14,
     quantity: 10, date : ISODate( "2022-01-12T05:08:13Z" ) },
   { _id: 6, name: "Vegan", size: "small", price: 17,
     quantity: 10, date : ISODate( "2021-01-13T05:08:13Z" ) },
   { _id: 7, name: "Vegan", size: "medium", price: 18,
     quantity: 10, date : ISODate( "2021-01-13T05:10:13Z" ) }
] )
```

A. Para cada una de las pizzas de tamaño mediano, mostrar el nombre y la cantidad de pizzas vendidas.

```
db.orders.aggregate( [

   // Stage 1: Filter pizza order documents by pizza size
```

```
   {
      $match: { size: "medium" }
   },

   // Stage 2: Group remaining documents by pizza name and calculate total quantity
   {
      $group: { _id: "$name", totalQuantity: { $sum: "$quantity" } }
   }

] )
```

B. ¿Qué realizará el siguiente ejemplo?

```
db.orders.aggregate( [
   // Stage 1: Filter pizza order documents by date range
   {
      $match:
      {
         "date": { $gte: new ISODate( "2020-01-30" ), $lt: new ISODate( "2022-01-30" ) }
      }
   },
   // Stage 2: Group remaining documents by date and calculate results
   {
      $group:
      {
         _id: { $dateToString: { format: "%Y-%m-%d", date: "$date" } },
         totalOrderValue: { $sum: { $multiply: [ "$price", "$quantity" ] } },
         averageOrderQuantity: { $avg: "$quantity" }
      }
   },
   // Stage 3: Sort documents by totalOrderValue in descending order
   {
      $sort: { totalOrderValue: -1 }
   }
 ] )
```

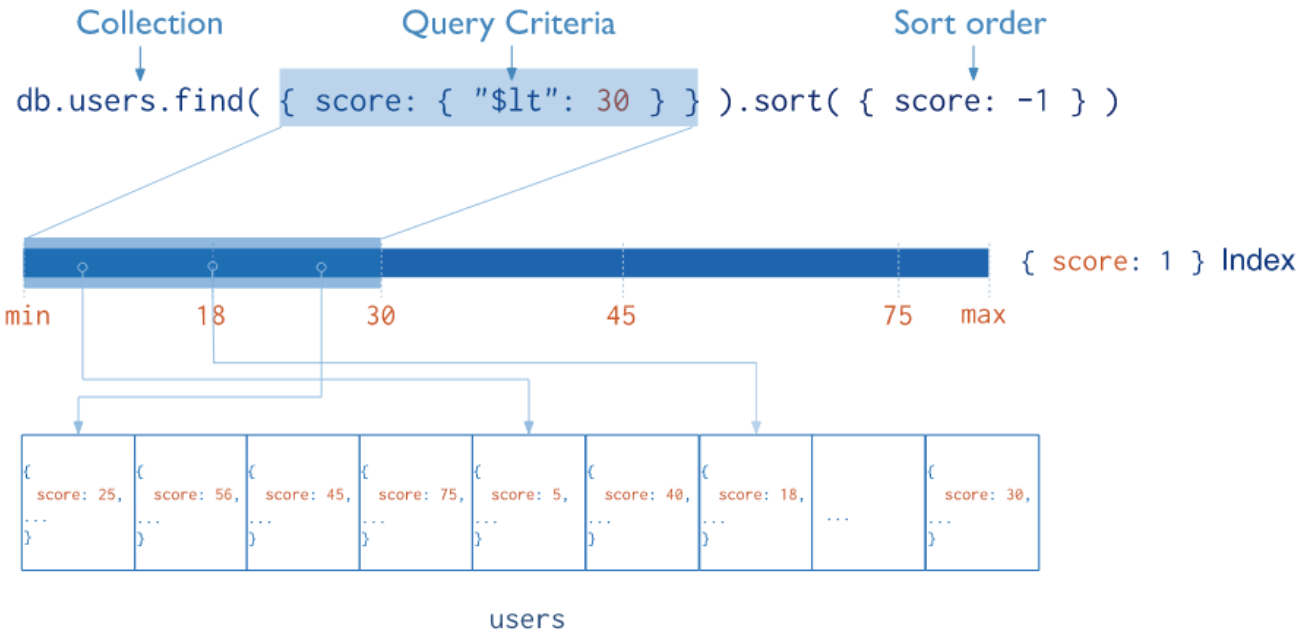
Lista de operadores para generar expresiones Click aquí.

1.4. Índices

De forma similar a una BD relacional, los índices permiten mejorar el desempeño en cuanto a la ejecución de consultas. De no existir, MongoDB realizará un ***collection scan*** en el que se deberá leer cada uno de los documentos de la colección para encontrar los documentos que coincidan con los criterios de búsqueda.

Un índice en MongoDB es una estructura de datos que almacena una porción pequeña de la colección. El índice almacena los valores de un atributo o de un conjunto de atributos en particular de forma ordenada.

Ejemplo



- Existe un índice de tipo ***unique*** creado de forma implícita para el campo `_id`.

Sintaxis para crear un índice

```
db.collection.createIndex( <key and index type specification>, <options> )
```

Ejemplo
Crear un índice descendente para un atributo llamado `name`.

```
db.collection.createIndex( { name: -1 } )
```

El nombre por default asignado al índice se crea con base al nombre de los atributos y a la dirección de ordenamiento. Por ejemplo, si la definición del índice es `{ item : 1, quantity: -1 },` el nombre del índice será: `item_1_quantity_-1`

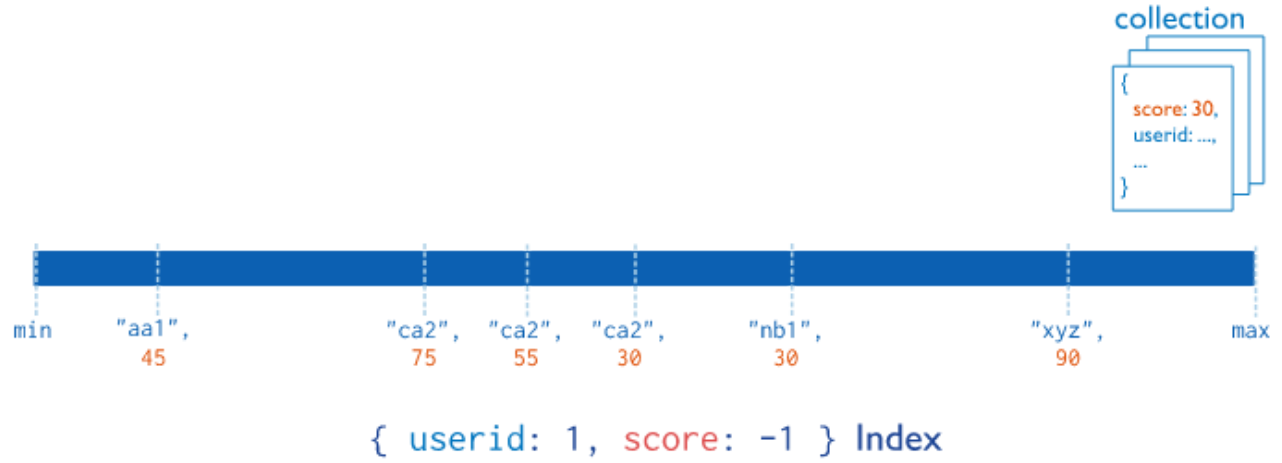
Ejemplo
Crear un índice con nombre personalizado

```
db.products.createIndex(
  { item: 1, quantity: -1 } ,
  { name: "query for inventory" }
)
```

1.4.1.Tipos de índices

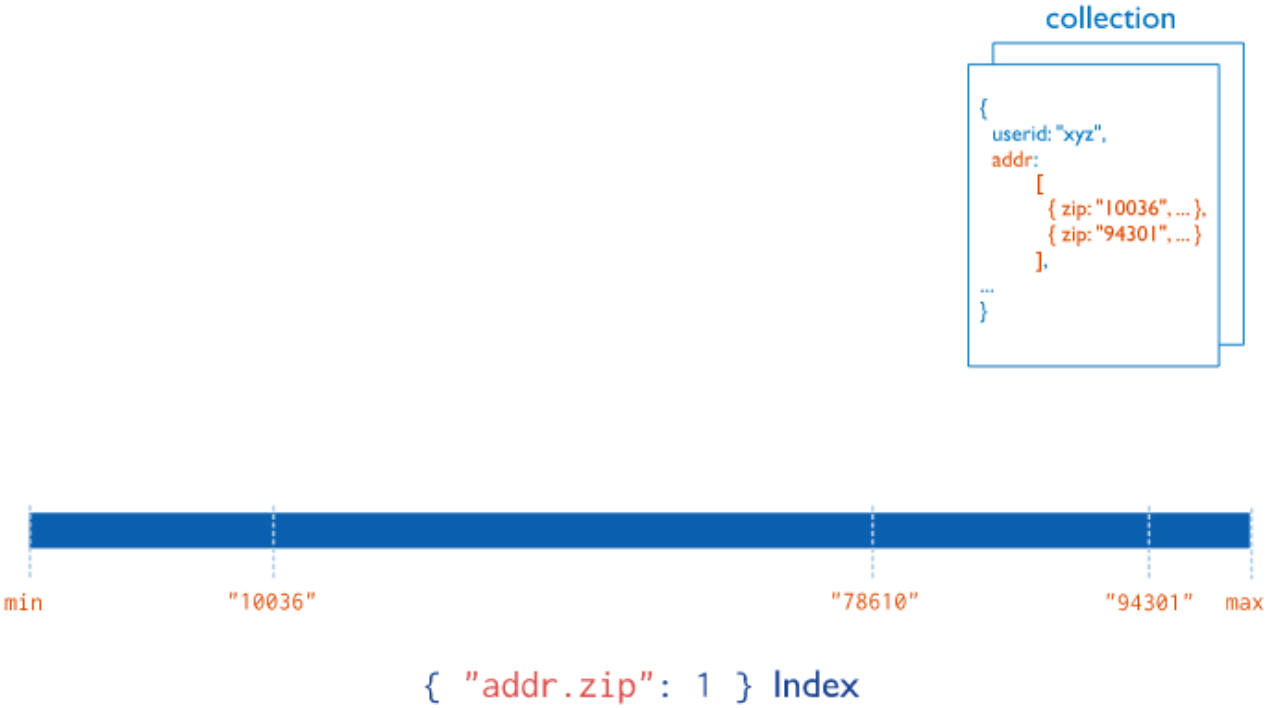
Índice para un solo atributo

Índice compuesto



Multi Key Index

Empleados para realizar el indexado de elementos contenidos en un arreglo. Al crear un índice sobre un atributo que representa a un arreglo, el índice contendrá una entrada por cada elemento del arreglo. Un multi key Index se crea sin la necesidad de especificar su tipo, basta con crear un índice para un atributo que representa a un arreglo.



Otros tipos de índices:

- Geospatial indexes
- Text search indexes
- Hashed Indexes
- Clustered indexes

Ejemplo

Indexar el atributo `name` para la colección de pizzas

```
db.orders.createIndex({name:1});
```

1.4.2. Explorar el uso de un índice

Para ilustrar el uso de un índice durante el procesamiento de una consulta, ejecutar la siguiente instrucciòn empleando la colección `tweets` vista anteriormente:

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

Observar el uso de explain con la cual se mostrará el detalle del uso del índice.

En la salida observar las siguientes secciones:

```
    winningPlan: {
      stage: 'PROJECTION_SIMPLE',
      transformBy: { created_at: true, text: true, _id: false },
      inputStage: {
        stage: 'TEXT_MATCH',
        indexPrefix: {},
        indexName: 'text_text_user.description_text',
  . . . .
```

Notar el uso del ìndice en la siguiente sección:

```
executionStats: {
    executionSuccess: true,
    nReturned: 5,
    executionTimeMillis: 1,
    totalKeysExamined: 5,
    totalDocsExamined: 5,
    executionStages: {
      . . .
      inputStage: {
        stage: 'TEXT_MATCH',
        nReturned: 5,
        executionTimeMillisEstimate: 0,
        . . .
        indexName: 'text_text_user.description_text',
        parsedTextQuery: {
          terms: [ 'appl', 'iphon' ],
          negatedTerms: [],
          phrases: [],
          negatedPhrases: []
        },
        textIndexVersion: 3,
```

- Winning plan indica la estrategia que se emplea para ejecutar la consulta.
- `totalKeysExamined` indica cuantos elementos en el índice fueron escaneados para procesar la consulta
- `total Docs Examined` indica el número total de documentos que fueron leídos de la colección para obtener el resultado (solo 5)!