

INTRODUCCIÓN

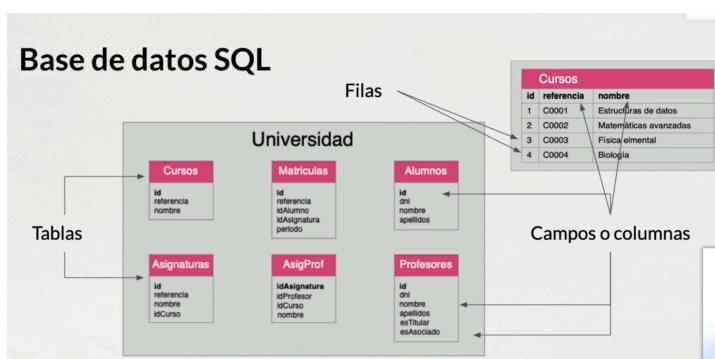
¿QUÉ SON LAS BASES DE DATOS SQL?

La información se organiza en entidades llamadas tablas, y cada una está formada por filas.
La información de las tablas está relacionada (por eso son bd relacionales - foreign keys y eso)

El lenguaje de consultas es SQL. Permiten el uso de transacciones aplicando los principios ACID:

Atomicidad - Consistencia - Aislamiento - Durabilidad

La separación de la información en tablas se hace mediante un proceso llamado normalización.
Siguen un esquema o conjunto de reglas que la información contenida en las tablas debe cumplir. Almacenan datos estructurados.



El esquema para Cursos podría ser:

Cursos		
id	referencia	nombre
1	C0001	Estructuras de datos
2	C0002	Matemáticas avanzadas
3	C0003	Física elemental
4	C0004	Biología

- Tiene 3 columnas:
- **id:**
 - int (entero)
 - debe ser positivo
 - no permite valores nulos
 - **referencia:**
 - string (texto)
 - permite valores nulos
 - **nombre:**
 - string (texto)
 - no admite valores nulos
- reglas

CARACTERÍSTICAS DE LAS BASES DE DATOS NOSQL

Surgen porque hay problemas con el enfoque relacional.

Son ideales para almacenar datos que no siguen una estructura fija, pero no garantizan los principios ACID.

Mejoran la flexibilidad y escalabilidad vs. SQL al no tener esquema.

Ahora las bases de datos nosql incorporan transacciones.

Las consultas se pueden hacer en SQL porque Mongo soporta ese lenguaje.

BD:

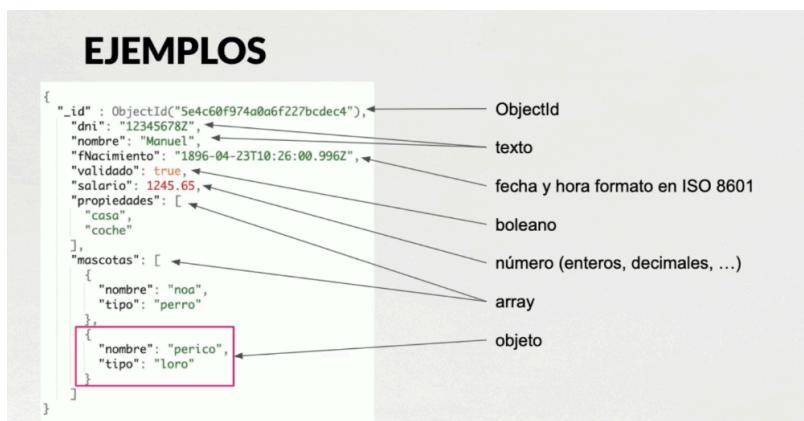
- Documentales - MongoDB
- Orientadas a grafos (estadística)
- Orientadas a clave-valor (índices - para mejorar rendimiento)

Están orientadas a documentos (JSON). Tiene una consola construida sobre Javascript por lo que se pueden ejecutar muchas de sus funciones.

Se utilizan sobretodo para internet y BigData

¿QUÉ ES EL FORMATO JSON?

JavaScript Object Notation > alternativa a XML > formato texto sencillo para intercambiar datos
Objeto JSON > formado por 1 o + pares de string:value



COMPARATIVA ENTRE BASES DE DATOS SQL Y NOSQL

Ventajas SQL:

- Madurez (clásico, muy utilizado)
- Principios ACID
- Estándares bien definidos (SQL)
- Sencillez en la escritura

Desventajas SQL:

- Mantenimiento difícil y costoso
- Poca flexibilidad frente a cambios en el diseño inicial
- Elección del proveedor más adecuado (según necesidad)
- Complejidad e instalación

Ventajas noSQL:

- Versatilidad / flexibilidad

- Crecimiento horizontal (es fácil crear nuevos nodos)
- Baja necesidad de recursos
- Optimización (algoritmo interno para reescribir las consultas)

Desventajas noSQL:

- No cuenta con los principios ACID
- La documentación del software no suele ser muy buena (salvo MongoDB)
- No hay un estándar consensuado
- Herramientas GUI (mongo se suele usar a través de la consola salvo en windows que tiene un cliente visual)

PRIMEROS PASOS CON MONGODB

INSTALACIÓN DE MONGODB

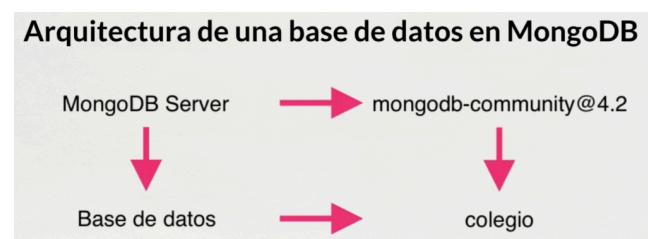
Lo hace con un mac de los cojones.

TRABAJANDO CON MONGODB DESDE LA CONSOLA

Por desgracia es la consola de mac:

- Para entrar a la consola → mongo
- Para salir de la consola → quit() / Ctrl + C
- Limpiar la consola → Ctrl + L
- Listar BD → show dbs
- Cambiarse de BD → use <dbname>
- Listar colecciones de una base de datos → show collections / show tables
- Mostrar el nombre de la base de datos → db.getName() / db
- Listar metadata de una BD → db.stats()
- Ayuda de comandos → db.help()
- Información sobre el servidor → db.hostInfo()
- Fecha y hora del sistema → Date()
- Dar formato JSON → db.<collectionName>.find().pretty()

CREACIÓN Y GESTIÓN DE BASES DE DATOS



OPERACIONES

- Creación / usar
- Eliminación

```
bash
> show dbs
admin 0.000GB
colegio 0.000GB
config 0.000GB
local 0.000GB
> use universidad
switched to db universidad
>
```

Si al utilizar `use <dbName>` no hay una base de datos con ese nombre, se crea una nueva.
No aparecerá listada en la BD (`show dbs`) hasta que yo no haga la primera inserción.

```
bash mongo bash mongo
> db.asignaturas.insertOne({ 'id': '881716', 'nombre': 'Matemáticas avanzadas' })
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e498d8e4b839543f78c14a4")
}
> show dbs
admin 0.000GB
colegio 0.000GB
config 0.000GB
local 0.000GB
universidad 0.000GB
>
```

Tabla concesionario

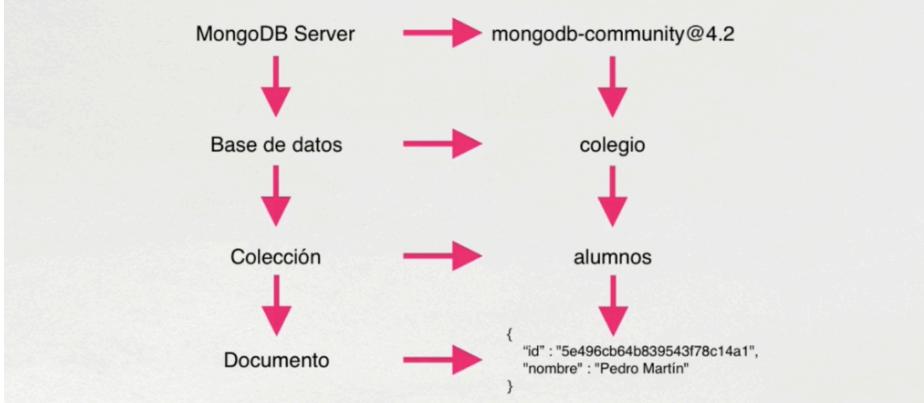
```
bash mongo bash mongo
> db.coches.insertOne({ matricula: "AAA5678" })
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e4fd2133825663c7b7cc3b5")
}
```

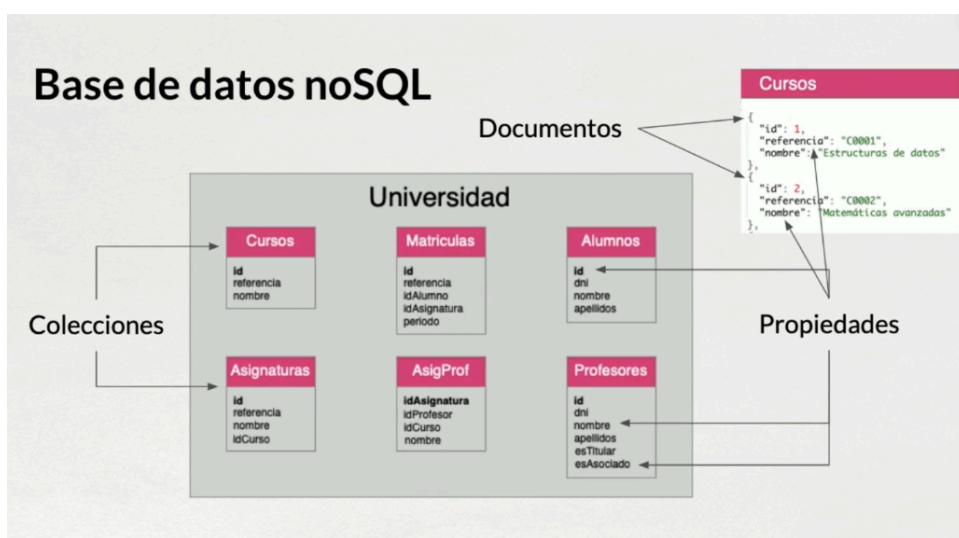
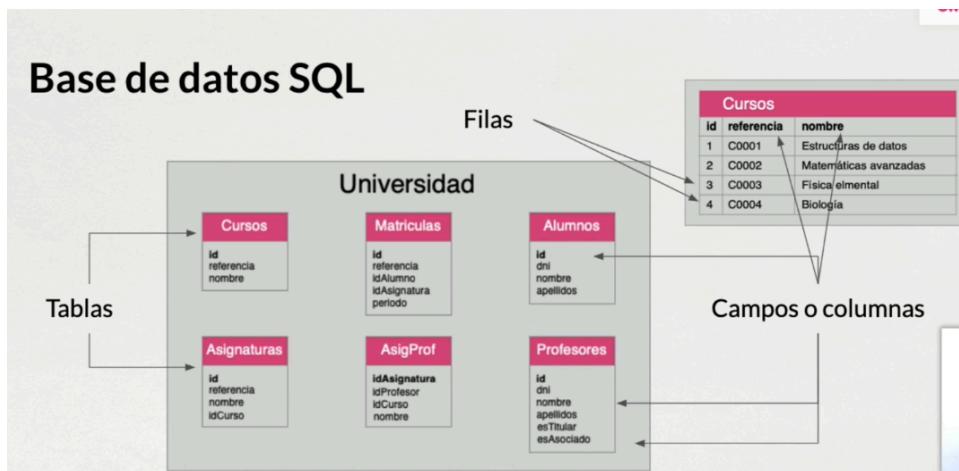
Si yo quiero insertar un elemento en coches, pero coches no existe, MongoDB lo creará.

```
• Eliminación
bash mongo bash mongo
> show dbs
colegio 0.000GB
config 0.000GB
local 0.000GB
universidad 0.000GB
> db.universidad
> db.dropDatabase()
{
  "dropped" : "universidad",
  "ok" : 1
}
> show dbs
```

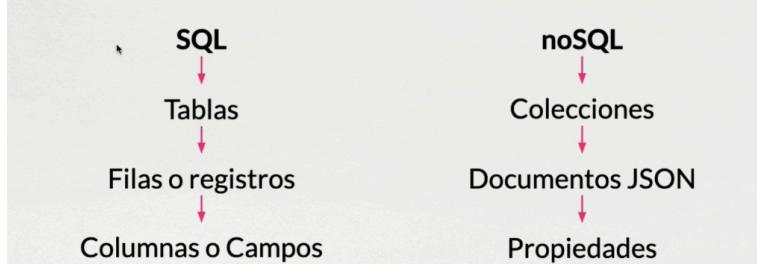
COLECCIONES Y DOCUMENTOS

Arquitectura de una base de datos en MongoDB





Jerarquía del almacenamiento de la información



TIPOS DE DATOS

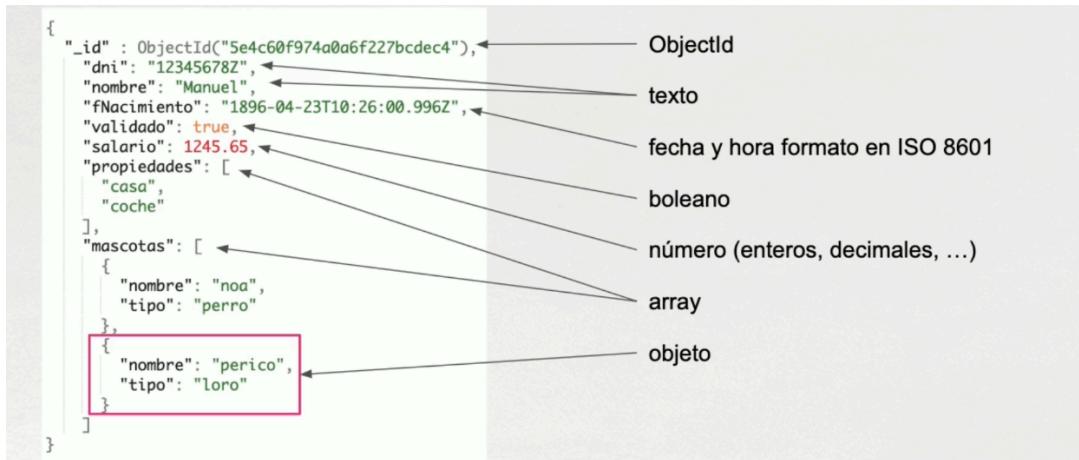
SIMPLES	COMPLEJOS
• números	• arrays
• cadenas de texto	• objeto
• fecha	• binary data
• hora	• objectId
• booleans	• Expresiones regulares

Binary data > guardar datos muy específicos como archivos multimedia.

ObjectId > cada vez que se crea un registro nuevo tú le puedes asignar un campo especial (un identificador) pero si no lo haces MongoDB lo hará por ti (le asignará un ObjectId con un valor)

Es como una PK.

Ejemplo con un documento:



PRÁCTICA: CREACIÓN DE UNA BASE DE DATOS

- Partiendo del escritorio de tu ordenador, ejecuta los pasos necesarios para poder usar la consola de mongodb (abrir un terminal o consola, iniciar/parar servicios, ejecutar la shell de mongodb)
- Crea una base de datos llamada **concesionario**. Recuerda insertar al menos un documento en una colección que se llame **coches**. Como propiedades del documento json, puedes usar **matrícula, marca, modelo, versiones** (sport, confort), **kms** y **fecha de matriculación**. (Aunque veremos cómo insertar documentos JSON en próximas clases, puedes usar el comando **db.insertOne(aquí va tu documento json)**)
- Visualiza el listado de todas las bases de datos disponibles.
- Elimina la base de datos creada y comprueba que ya no existe al pedir el listado.

OPERACIONES CON DATOS I (CRUD)

INSERTAR

Para insertar un solo documento → db.<collectionName>.insertOne(<json>)

Este cuadro muestra una captura de pantalla de la consola MongoDB. La parte superior muestra el comando `db.collectionName.insertOne(<json>);` seguido de un cursor que apunta a un cuadro resaltado que contiene el JSON de inserción. La parte inferior muestra la respuesta de MongoDB, que incluye el resultado de `show dbs`, `show collections`, el comando `db.cursos.insertOne(...)` y su resultado, y el resultado final de la inserción.

```
db.collectionName.insertOne(<json>);

{
  "referencia": "C0001",
  "nombre": "Estadística",
  "duración": 6,
  "activo": true,
  "fInicio": "2020-01-21T09:00:00.000Z"
}

> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
universidad 0.000GB
> use universidad
switched to db universidad
> show collections
> db.cursos.insertOne({
...   "referencia": "C0001",
...   "nombre": "Estadística",
...   "duración": 6,
...   "activo": true,
...   "fInicio": "2020-01-21T09:00:00.000Z"
... });
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e4e946177a92631239bd288")
}>
```

Para ver lo que acabamos de insertar → db.personas.find().pretty()

Para insertar varios → db.<collectionName>.insertMany(<json>)

Pasamos un array de documentos.

```
> db.collectionName.insertMany(<json>);  
[  
  {  
    "referencia": "C0002",  
    "nombre": "Pintura",  
    "duracion": 24,  
    "activo": false,  
    "fInicio": "2020-02-18T16:30:00.000Z"  
  },  
  {  
    "referencia": "C0003",  
    "nombre": "Fisica",  
    "duracion": 46,  
    "activo": true,  
    "fInicio": "2019-12-13T10:30:00.000Z"  
  }]  
  
> use universidad  
switched to db universidad  
db.cursos.insertMany([  
  {  
    "referencia": "C0002",  
    "nombre": "Pintura",  
    "duracion": 24,  
    "activo": false,  
    "fInicio": "2020-02-18T16:30:00.000Z"  
  },  
  {  
    "referencia": "C0003",  
    "nombre": "Fisica",  
    "duracion": 46,  
    "activo": true,  
    "fInicio": "2019-12-13T10:30:00.000Z"  
  }]);  
  
{"acknowledged": true,  
 "insertedIds": [  
   ObjectId("5e4e98be1cd78466ff804f1f"),  
   ObjectId("5e4e98be1cd78466ff804f20")  
 ]}
```

ELIMINAR

Eliminar un solo documento → db.<collectionName>.deleteOne(<json>)

```
> db.cursos.find().pretty();  
  
CRUD - Eliminar- Ejemplos  
  
db.collectionName.deleteOne(<json>);  
  


- donde <json> es un documento con condiciones para eliminar un documento concreto.
- Si más de un documento cumple con la condición, se eliminará el primero de ellos.

  
> db.cursos.deleteOne({activo: true});  
{"_id": ObjectId("5e4e15a1cd78466ff804f24"),  
 "referencia": "C0003",  
 "nombre": "estadistica",  
 "duracion": 30,  
 "activo": true,  
 "fInicio": "2020-01-21T09:00:00.000Z"}  
  
> db.cursos.deleteOne({activo: true});  
{"_id": ObjectId("5e4e15a1cd78466ff804f25"),  
 "referencia": "C0002",  
 "nombre": "Pintura",  
 "duracion": 24,  
 "activo": false,  
 "fInicio": "2020-02-18T16:30:00.000Z"}  
  
> db.cursos.deleteOne({activo: true});  
{"_id": ObjectId("5e4e15a1cd78466ff804f26"),  
 "referencia": "C0003",  
 "nombre": "Fisica",  
 "duracion": 46,  
 "activo": true,  
 "fInicio": "2019-12-13T10:30:00.000Z"}  
  
> db.cursos.deleteOne({activo: true});  
{"_id": ObjectId("5e4e15a1cd78466ff804f25"),  
 "referencia": "C0002",  
 "nombre": "Pintura",  
 "duracion": 24,  
 "activo": false,  
 "fInicio": "2020-02-18T16:30:00.000Z"}  
  
> db.cursos.deleteOne({activo: true});  
{"_id": ObjectId("5e4e15a1cd78466ff804f26"),  
 "referencia": "C0003",  
 "nombre": "Fisica",  
 "duracion": 46,  
 "activo": true,  
 "fInicio": "2019-12-13T10:30:00.000Z"}
```

Eliminar varios documentos → db.<collectionName>.deleteMany(<json>)

Lo mismo que lo anterior pero si más de un documento cumple con la condición, se eliminarán todos.

```
> db.personas.deleteMany({});
```

Si lo hacemos sin determinar condición, se eliminarán todos los objetos de una condición

ACTUALIZAR

- db.collection.updateOne(<filter>, <update>)
- db.collection.updateMany(<filter>, <update>)
- db.collection.replaceOne(<filter>, <update>)

UPDATE ONE

```
db.cursos.updateOne(  
  { referencia: "C0002"},  
  {  
    $set: {  
      nombre: "Fisica elemental",  
      duracion: 16  
    }  
};
```

En este ejemplo, la condición o filtro es que la referencia sea “C0002” y se tiene que actualizar el nombre y la duración (marcado por el \$set > un comando basado en javascript)

PRIMERO FILTRO Y LUEGO ACTUALIZACIÓN)

```
> db.profesores.updateOne({nombre: "Elena"}, { $set: { esTitular: false, esAsociado: true } })  
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }  
> |
```

UPDATE MANY

```
db.cursos.updateMany(  
  { duracion: { $gt: 10 } },  
  {  
    $set: {  
      nombre: "Fisica elemental",  
      duracion: 20  
    }  
};
```

El comando \$gt → greater than

REPLACE ONE

El filtro se encarga de encontrar el documento a reemplazar y en la parte de update pondrás el documento con el que será reemplazado.

```
db.cursos.replaceOne(  
  { "referencia": "C0003" },  
  {  
    "nombre": "Fisica",  
    "duracion": 46,  
    "matriculados": 15,  
    "listaEspera": 6  
});
```

PRÁCTICA: INSERTAR, ELIMINAR Y ACTUALIZAR

- Vamos a crear una base de datos llamada **tienda**.
- La base de datos tendrá una colección llamada **productos**

- Crea la base de datos.
- Inserta en la colección los siguientes documentos de uno en uno:

```
{  
  "referencia": "P0001",  
  "tipo": "camisa",  
  "paraHombre": true,  
  "talla": "XS",  
  "precio": 20.99  
}
```

```
{  
  "referencia": "P0002",  
  "tipo": "camisa",  
  "paraHombre": true,  
  "talla": "XL",  
  "precio": 30.25  
}
```

```
{  
  "referencia": "P0003",  
  "tipo": "pantalon",  
  "paraMujer": true,  
  "talla": "L",  
  "precio": 20.99  
}
```

- Elimínalos todos.
- Ahora créalos, pero todos a la vez.

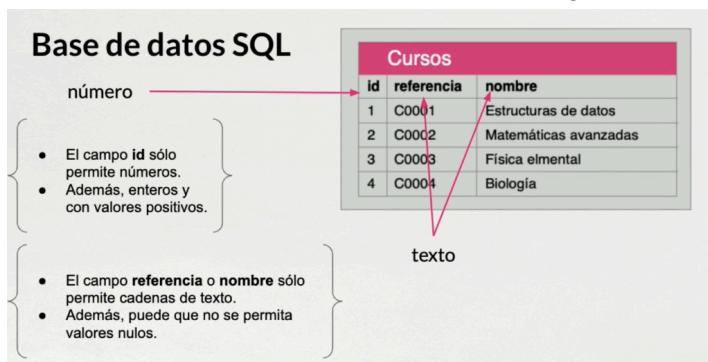
- Actualiza todos los documentos con un precio inferior a 25 para que tenga un precio un 10% más caro.
- Reemplaza cada documento que sea para hombre con la misma estructura pero añadiendo una propiedad nueva: **"paraMujer": false**.
- Reemplaza cada documento que sea para mujer con la misma estructura pero añadiendo una propiedad nueva: **"paraHombre": false**.
- Actualiza cada documento, filtrando por sus referencia. Las propiedades cambiando el tipo *camisa* por *chaqueta* y la talla debe ser igual a M.

DISEÑANDO EL MODELO DE DATOS

CONCEPTO DE SCHEMALESS

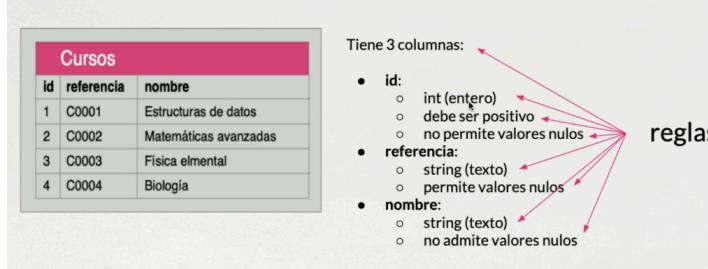
El concepto de *schemaless* significa:

- no tiene schema > las normas de organización de las BD SQL



Las reglas tienen que ver con el tipo de información que se almacena

El esquema para Cursos podría ser:



Tener esquema ofrece un mayor control pero una menor flexibilidad.

Una BD *schemaless* no sigue una estructura predefinida por lo que si hay datos en un documento, en otro puede no haberlos. Se usa menos gestión de memoria.

Colecciones → documentos JSON

En una colección se pueden almacenar documentos JSON con contenido distinto entre sí.

Aún así, se pueden guardar datos estructurados en una BD noSQL. (mientras estén en JSON)

Ejemplos

```
> db.cursos.find().pretty()
[{
    "_id" : ObjectId("5e4d9386deaedebe80a7c17b"),
    "id" : 1,
    "referencia" : "C0001",
    "nombre" : "Física"
},
{
    "_id" : ObjectId("5e4d93f5deaedebe80a7c17c"),
    "id" : 1,
    "referencia" : "C0001",
    "nombre" : "Física",
    "fInicio" : "12/01/2019",
    "esGratis" : true
}]
```

Gran flexibilidad

documentos JSON

DISEÑO I: DOCUMENTOS EMBEBIDOS

Embebido → anidado (documentos unos dentro de otros como valor)

Curso C0001

```
{
    "id": 1,
    "referencia": "C0001",
    "nombre": "Física",
    "fInicio": "12/01/2019",
    "esGratis": true
}
```

Asignaturas que se imparten en el curso C0001

```
[
    {
        "id": 2,
        "nombre": "Física elemental",
        "duracion": 5
    },
    {
        "id": 3,
        "nombre": "Termodinámica",
        "duracion": 4
    }
]
```

Ejemplo

Las asignaturas han sido **embebidas** (metidas dentro) del documento JSON que representa al curso C0001 asociandolos una nueva propiedad llamada "**asignaturas**".

Puede que estemos ante una forma de **relacionar** la información contenida en distintas colecciones?

Curso C0001

```
{
    "id": 1,
    "referencia": "C0001",
    "nombre": "Física",
    "fInicio": "12/01/2019",
    "esGratis": true,
    "asignaturas": [
        {
            "id": 2,
            "nombre": "Física elemental",
            "duracion": 5
        },
        {
            "id": 3,
            "nombre": "Termodinámica",
            "duracion": 4
        }
    ]
}
```

Esta es una manera de relacionar la información en 2 colecciones.

DISEÑO 2: DOCUMENTOS REFERENCIADOS

Normalmente en un documento JSON se guarda el valor de una de las propiedades de lo que se quiere referenciar, en lugar del documento completo. (valor identificador > como PK)

Ejemplo

Las asignaturas han sido **referenciadas**, mediante sus *ids*, en lugar de guardar el documento JSON con las asignaturas dentro del documento JSON que representa al curso C0001 .

Curso C0001

```
{  
  "id": 1,  
  "referencia": "C0001",  
  "nombre": "Física",  
  "fInicio": "12/01/2019",  
  "esGratuito": true,  
  "asignaturas": [2, 3]  
}
```

Documentos embebidos

Ventajas

- Al recuperar un curso, podemos traernos toda la información relacionada en otras colecciones (p.e. asignaturas)

Desventajas:

- Al hacer consultas sobre la colección externa (cursos), si uno de los criterios afecta a la información de los documentos embebidos, el tiempo para realizar dicha consulta se verá incrementando.
- Las actualizaciones serán más costosas si alguna de las propiedades a actualizar pertenece al documento embebido

Documentos referenciados

Ventajas

- Las consultas sobre la colección principal y las relacionadas se ejecutarán más rápido.
- Al actualizar información relacionada, se hace directamente en sus documentos sin tener que revisar la colección externa.

Desventajas:

- Al recuperar un curso, habrá que consultar los identificadores que relacionan a los documentos en otras colecciones y hacer consultas adicionales para obtener la información relacionada.

Como no es probable que los IDs de esos documentos cambien, se puede modificar todo sin armar mucho lío.

¿Qué diseño elegir? dependerá de:

- **Documentos embebidos**
- **Documentos referenciados**

- cómo se quiere almacenar la información.
- la naturaleza y el contexto de las aplicaciones que vayan a consumir la información.
- Las preferencias de roles como arquitectos de software y de bases de datos teniendo en cuenta factores futuros como la escalabilidad en cuanto a volumen de datos, usuarios / aplicaciones y sus formas de acceder a la información, etc.

A mí especialmente me viene bien la de documentos referenciados (al haber muchos accesos) porque querré que vaya rápido.

OPERACIONES CON DATOS II (CRUD)

CONSULTAS CON TIPOS DE DATOS SIMPLES

Para hacer consultas lo mejor es usar:

- db.collection.find() → devuelve todos los documentos de la colección
- db.collection.find(<filter>) → devuelve los documentos que cumplan con el filtro



```
[{"nombre": "Maria", "apellidos": "Su\u00e1rez Manrique", "especialidad": ["biolog\u00eda"], "esTitular": true, "esAsociado": false, "edad": 51}, {"nombre": "Jose Luis", "apellidos": "L\u00f3pez P\u00e9rez", "especialidad": ["matem\u00e1ticas", "f\u00f3sica"], "esTitular": false, "esAsociado": true, "edad": 39}, {"nombre": "Antonio", "apellidos": "Munguia Arteche", "especialidad": ["f\u00f3sica", "qu\u00famica"], "esTitular": true, "esAsociado": false, "edad": 54}]
```

- Obtener todos los profesores.
- Obtener todos los profesores asociados.
- Obtener todos los profesores titulares y mayores de 50.
- Obtener todos los profesores con edades entre 50 y 60.
- N\u00famero total de profesores.
- N\u00famero total de profesores asociados.
- Obtener solo los 2 primeros resultados.

1. Obtener todos los profesores:

- a. db.find()
- b. db.find({}) → lo mismo que lo de arriba pero “sin filtrar” expl\u00f3citamente
- c. db.find().pretty()

2. Obtener todos los profesores asociados:

- a. db.profesores.find({esAsociado:true}).pretty()

3. Obtener todos los profesores titulares y mayores de 50:

- a. db.profesores.find({esTitular:true, edad: {\$gt: 50}}).pretty()

4. Obtener todos los profesores con edades entre 50 y 60:

- a. db.profesores.find({edad: {\$gt: 50, \$lt: 60}}).pretty()

5. N\u00famero total de profesores:

- a. db.profesores.count() → el n\u00famero de documentos JSON en la colecci\u00f3n
- b. db.profesores.find().count() → en esta s\u00f3 puedes especificar un filtro

6. N\u00famero total de profesores asociados:

- a. db.profesores.find({esAsociado: true}).count()

7. Obtener solo los 2 primeros resultados:

- a. db.profesores.find().limit(2) → con esto podemos paginar los resultados.

CONSULTAS AVANZADAS: ARRAYS

1. Obtener todos los profesores que imparten ex\u00e1ctamente matem\u00e1ticas y f\u00f3sica:

- a. db.profesores.find({ especialidad: ["matem\u00e1ticas", "f\u00f3sica"] }).pretty()

2. Obtener todos los profesores que imparten matemáticas y física:
 - a. db.profesores.find({ especialidad: { \$all: ["física", "matemáticas"] } }).pretty()
3. Obtener todos los profesores que imparten química:
 - a. db.profesores.find({ especialidad: "química" }).pretty()
4. Número total de profesores que imparten biología:
 - a. db.profesores.find({ especialidad: "biología" }).count()
5. Obtener solo un profesor que imparta física:
 - a. db.profesores.find({ especialidad: "física" }).limit(1).pretty()

CONSULTAS AVANZADAS: DOCUMENTOS EMBEBIDOS

Trabajando con Profesores y asignaturas

```

[{"profesor": {
    "nombre": "María",
    "apellidos": "Suárez Manrique",
    "especialidad": [
        "biología"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 51,
    "asignatura": {
        "id": "A0001",
        "nombre": "Biología molecular",
        "creditos": 6
    }
},
{"profesor": {
    "nombre": "Jose Luis",
    "apellidos": "López Pérez",
    "especialidad": [
        "matemáticas",
        "física"
    ],
    "esTitular": false,
    "esAsociado": true,
    "edad": 39,
    "asignatura": {
        "id": "A0002",
        "nombre": "Termodinámica",
        "creditos": 9
    }
},
{"profesor": {
    "nombre": "Antonio",
    "apellidos": "Munguía Arteche",
    "especialidad": [
        "física, química"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 55,
    "asignatura": {
        "id": "A0002",
        "nombre": "Termodinámica",
        "creditos": 9
    }
}]

```

1. Obtener todos los profesores que imparten la asignatura A0002:
 - a. db.profesores.find({ "asignatura.id" : "A0002" }).pretty()
2. Obtener todos los profesores que imparten termodinámica:
 - a. db.profesores.find({ "asignatura.nombre" : "Termodinámica" }).pretty()
3. Obtener todos los profesores que imparten asignaturas de más de 6 créditos:
 - a. db.profesores.find({ "asignatura.creditos" : { \$gt: 6 } }).pretty()
4. Obtener todos los profesores que imparten asignaturas de más de 6 créditos mayores de 40:
 - a. db.profesores.find({ "asignatura.creditos" : { \$gt: 6 }, "edad": { \$gt: 40 } }).pretty()

PRÁCTICA: CONSULTAS AVANZADAS:

- Obtener todos los profesores que tienen un SUV.
- Obtener todos los profesores que tienen un moto y un uso de menos de 5 años.
- Obtener todos los profesores con vehículo de más de 150 caballos y que imparten física.
- Obtener el nº de profesores con más de 50 que tiene moto.
- Obtener el nº de profesores titulares que tienen un SUV.

MEJORANDO EL RENDIMIENTO

¿QUÉ SON LOS ÍNDICES?

Cuando tenemos una BD muy grande, su rendimiento se ve afectado. Para solucionarlo, usaremos índices.

Son estructuras de datos especiales gestionadas por MongoDB que almacenan una pequeña porción de los datos de una colección.

El índice almacena el valor de un campo o conjunto de campos específicos (simples o compuestos), ordenados por su valor.

Si tienes un índice, MongoDB, antes de hacer la consulta, va a consultar el índice para no tener que consultar TODA la colección.

A priori, ya hay creados algunos índices a nivel de colección.

Listar los índices de una colección → db.<collectionName>.getIndexes()

Siempre que tengamos el campo `_id` (que puede crearse solo si tú no asignas un campo clave), tendremos un índice asociado → este es un índice simple (1 solo campo) y único (tienes la certeza de que todos los valores del índice serán únicos - al ser un unique ID).

```
{  
    "_id" : ObjectId("5e50dbbb0bfaf3bc4800e040"),  
    "nombre" : "María"
```

ÍNDICES SIMPLES

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

EJEMPLO CREAR:

- db.profesores.createIndex({ "edad": -1 }) (-1 → ordenados descendientemente)

```
> db.profesores.createIndex({edad: -1});  
{  
    "createdCollectionAutomatically" : false,  
    "numIndexesBefore" : 1,  
    "numIndexesAfter" : 2,  
    "ok" : 1  
}
```

Si hacemos una consulta con la edad de profesores, lo primero que hará MongoDB es mirar si hay un índice y mirarlo.

EJEMPLO DESTRUIR:

- db.profesores.dropIndex('edad_-1') → borra solo el especificado

- db.profesores.dropIndexes() → borra todos los de una colección

ÍNDICES COMPUESTOS

```
db.profesores.createIndex({ matricula: 1, edad: 1 })
```

ÍNDICES ÚNICOS

Para crear este tipo de índices, debemos estar seguros de que los valores del campo elegido:

- No se pueden repetir
- No pueden ser nulos

Si se intenta hacer pero hay valores repetidos o nulos, no nos dejará.

```
db.profesores.createIndex({ nombre: 1 }, { unique: 1 })
```

PRÁCTICA: CREAR Y ELIMINAR ÍNDICES

- Descarga el fichero **coches_profesores.json** disponible en el material adjunto
- Modifica sus documentos añadiendo las siguientes propiedades:
 - **email**
 - **clave**
 - **salario**
- Crea un índice simple sobre el campo **salario**.
- Crea un índice compuesto sobre los campos **email** y **clave**. Elige el orden sobre cada campo como deseas.
- Elimina los indices (uno a uno por su nombre o todos de una vez)