

AMPLIACIÓN BBDD

DISEÑO, CREACIÓN Y USO DE MONGODB,

DOCKER Y CONEXIÓN JAVA



Nombre del estudiante: David Romero

Asignatura: Base de Datos

Profesor: Manuel Mauri Pajares

Título del trabajo: Diseño, creación y uso de MongoDB, Dockers y conexión Java

Documentación técnica: Instalación y puesta en marcha de una bddb Mongo

Documentación sobre la bddb: Configuración de la bddb Cine

Fecha de entrega: 16-06-2025

Índice:

- 1. Introducción**
 - 1.1. Breve descripción de la BBDD**
 - 1.2. Objetivos del proyecto**
 - 1.3. Uso de Mongo y Docker**
- 2. Requisitos y software utilizado**
 - 2.1. Requisitos**
 - 2.2. Software utilizado**
- 3. Instalación y despliegue de la BBDD Cine en el contenedor Mongo en Docker**
 - 3.1. Consideraciones Clave**
 - 3.2. Creación archivos de inicio e inserción datos JS**
 - 3.2.1. Validaciones**
- 4. Estructura de la BBDD Cine**
- 5. Consultas avanzadas en Mongo Compass**
 - 5.1. Conexión Docker-Mongo Compass**
 - 5.2. Agregaciones**
 - 5.3. Indexación de campos**
 - 5.4. Comandos buenos de conocer**
- 6. Conexión con Java**
- 7. Conclusión**

1. Introducción

Primero indicar lo siguiente, aunque es una ampliación conjunta de 2 asignaturas: Programación y Base de Datos, solo puedo optar a la subida de nota en Base de Datos, por lo que la entrega (aunque llevará la parte de Java, será solo a modo de prueba de que la conexión y operaciones CRUD se llevan a cabo correctamente) estará enfocada, en cuanto a la redacción de la documentación, a la parte de BBDD.

En este proyecto se crea desde cero, una pequeña *base de datos no relacional* para albergar información sobre películas y después conectarse con Java.

Para ello, se investigó (y después se peleó con la parte práctica) sobre una base de datos concreta, MongoDB así como su uso en contenedores Docker, además levantada con JavaScript automáticamente.

1.1 Breve descripción de la BBDD

La BBDD tiene por nombre Cine, y consta de 5 colecciones, llamadas:

- País
- Género
- Actor
- Director
- Películas

Aunque no es relacional, se establecen ciertas relaciones entre colecciones, en este caso se opta por referenciar el ID, frente a la otra opción disponible de embeber un documento dentro de otro. (La diferencia radica en que uno pasa el Id referenciado, mientras que otro contiene el documento completo en su interior, más adelante se ahondará en ese tema).

1.2 Objetivo del Proyecto

La finalidad de este proyecto es aprender el funcionamiento de Mongo DB una BBDD NoSQL, las herramientas que ofrece y cómo se usan, su integración en un contenedor Docker y realizar una pequeña BBDD llevando a la práctica la teoría vista a lo largo del proceso de investigación.

1.3 Uso de Mongo y Docker

En el desarrollo de este proyecto se ha utilizado MongoDB como SGBD, de forma obligatoria según los requisitos del enunciado. Su naturaleza orientada a documentos ofrece ventajas específicas que se han aprovechado en este trabajo:

- ☑ Flexibilidad en la estructura de los datos gracias al modelo BSON.
- ☑ Posibilidad de validar documentos mediante esquemas (\$jsonSchema).
- ☑ Soporte para relaciones mediante referencias o documentos embebidos.
- ☑ Herramientas gráficas como MongoDB Compass que facilitan la inspección, agregaciones y gestión de datos.
- ☑ Integración sencilla con aplicaciones escritas en JavaScript o Java.

Este entorno también ha permitido profundizar en conceptos como el modelado NoSQL, la indexación para mejorar rendimiento, y el uso de agregaciones para consultas complejas.

La elección de Docker se basa en las siguientes ventajas clave:

- ☑ Portabilidad y reproducibilidad: Al contenerizar MongoDB, se garantiza que la base de datos se comportará igual en cualquier entorno que tenga Docker instalado, evitando problemas de configuración local.
- ☑ Despliegue automático: Gracias a los archivos de inicialización .js, Docker permite automatizar la creación de colecciones, validaciones e inserción de datos sin intervención manual.

- ☑ Aislamiento: MongoDB corre en un entorno aislado del sistema anfitrión, reduciendo riesgos de conflictos entre versiones y facilitando la limpieza del entorno tras el desarrollo.
- ☑ Facilidad de integración futura: El entorno Docker se puede extender fácilmente para incluir servicios adicionales si el proyecto crece.

En conjunto, MongoDB y Docker forman una base sólida, moderna y mantenible para este tipo de aplicaciones.

2. Requisitos y Software utilizado

2.1 Requisitos

- ☐ Tener Docker instalado.
- ☐ Una versión disponible de la imagen del contenedor para Mongo.
- ☐ Creación de carpetas init y data en almacenamiento local, para volúmenes.
- ☐ Opcional: Mongo DB Compass y Docker Desktop.

2.2 Software utilizado

Software		Versión	
Sistema Operativo Windows 11 Pro		24H2	
Editor de código VS Code		1.100.3	
Docker		28.1.1	
Docker Desktop		4.41.2	
Mongo DB		8.0.9	
Mongo Compass		1.46.3	
Java	Eclipse	17 LTS	2024-12 (4.34.0)

3. Instalación y despliegue de la BBDD Cine en el contenedor Mongo en Docker

3.1 Consideraciones Clave

Para este proyecto se opta por montar la BBDD automáticamente mediante JavaScript con la creación de un contenedor en Docker. Durante la creación del contenedor con la imagen de mongo (previamente descargada), será necesario instalar 2 volúmenes, uno para la carpeta **init** (contiene los archivos .js del 01 al 06) y otro para la carpeta **data** (hay que asegurarse que la carpeta está vacía).

Además, por buenas prácticas se establece el nombre de la base de datos y autenticación, es decir, se añaden usuario root con nombre y contraseña para el contenedor. También, se especifica el puerto asignado para conectarse por parte del host local (en mi caso el 27018, el contenedor sigue en su puerto 27017).

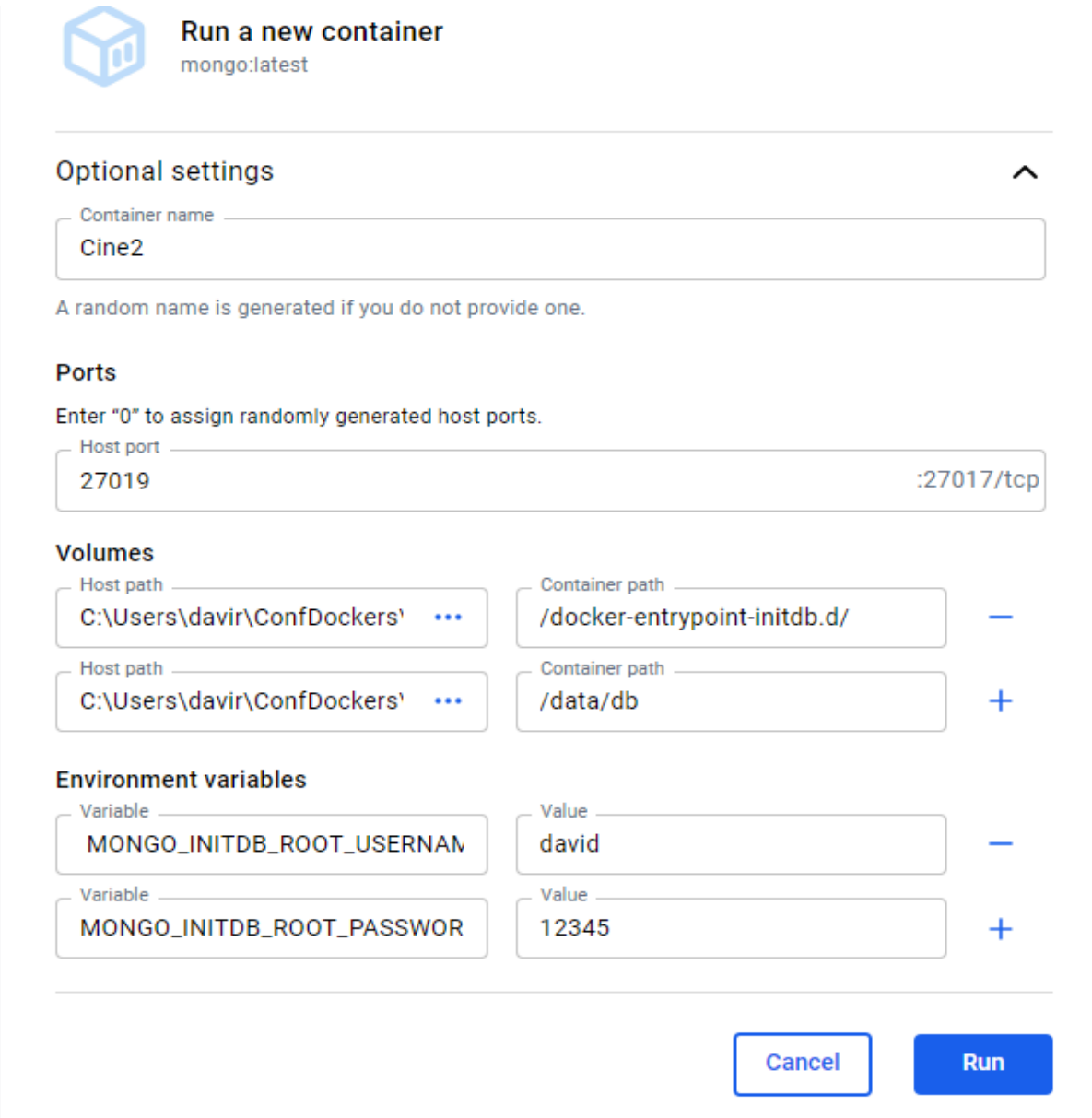
En cuanto a los volúmenes, es importante reseñar el path correspondiente en el contenedor: `/docker-entrypoint-initdb.d/` para la carpeta **init** (esta carpeta solo se lee y ejecuta los scripts de su interior una vez, al crear el contenedor pero debe estar en la ruta antes citada) y `/data/db` para la carpeta **data** (es donde guardan los datos de base en almacenamiento local, generando así persistencias de datos si el contenedor fuera eliminado).


Es importante que la carpeta esté vacía (principalmente hay que prestar especial consideración si ya se instaló un contenedor antes en ella, o la borras o creas una nueva ruta para el nuevo contenedor. Que esté vacía es lo que alerta a Docker para que ejecute lo que haya en `/docker-entrypoint-initdb.d/`, donde se colocó la carpeta **init**).

En el caso de la ruta local se crearon 2 carpetas nuevas para la nueva base de datos, dentro de la carpeta propia, ya existente de configuración de contenedores.

Tras toda esta configuración en Docker Desktop (tiene terminal integrado, alternando lo que sí permite la GUI → creación de contenedor; con los comandos que se recomienda hacer en consola (descarga imagen, acceder al contenedor desde docker con exec, etc..) podemos pulsa Run y se crea el contenedor Mongo y la BBDD con sus colecciones establecidas en los scripts de la carpeta init.

Todos los detalles descritos para la creación del contenedor anteriormente, pueden observarse en la siguiente captura de pantalla (Es a modo de ejemplo, no es el contenedor creado para el proyecto) :



 **Run a new container**
mongo:latest

Optional settings ^

Container name

A random name is generated if you do not provide one.

Ports

Enter "0" to assign randomly generated host ports.

Host port :

Volumes

Host path <input type="text" value="C:\Users\davir\ConfDockers\"/>	Container path <input type="text" value="/docker-entrypoint-initdb.d/"/>	—
Host path <input type="text" value="C:\Users\davir\ConfDockers\"/>	Container path <input type="text" value="/data/db"/>	+

Environment variables

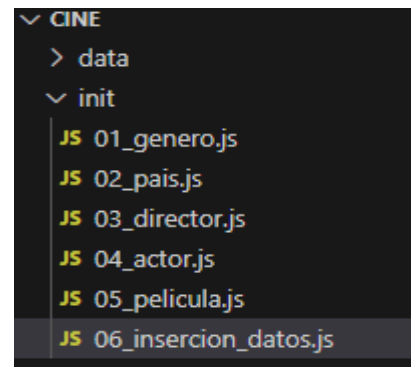
Variable <input type="text" value="MONGO_INITDB_ROOT_USERNAME"/>	Value <input type="text" value="david"/>	—
Variable <input type="text" value="MONGO_INITDB_ROOT_PASSWORD"/>	Value <input type="text" value="12345"/>	+

3.2 Creación archivos de inicio con JS

La carpeta init está compartida por volumen con el contenedor, este la ejecuta en el momento de su creación. En lugar de todo en un mismo archivo.js se ha establecido un archivo para cada colección, y ya que el orden en el que estén, es el orden en que se ejecutan, se han de tener en cuenta ciertas consideraciones:

☑ Por defecto, Docker ordena los archivos alfabéticamente para establecer el orden, entonces se emplea una numeración al nombrar los archivos para aprovechar ese orden por defecto. En este caso, no superará los 2 dígitos el nº de scripts, por lo que serán 01.....,02....., etc.

☑ Elección del orden, aunque no es una BBDD relacional, se establece cierta relación entre documentos por lo que hay que tener en cuenta que requiere que esté ya creado antes. A continuación se argumenta el orden establecido para del proyecto:



1. Las colecciones género y país, no contienen datos referentes a otras colecciones, por lo que se pueden crear, indistintamente el orden, primera y segunda.
2. Las colecciones actor y director, contienen los ids respectivos de la colección País por lo que deben ir después, ya que necesitan un país (requerido en la validación del Schema), Además, un array con los Id respectivos de las películas que estos han trabajado (están referenciadas) pero ese array puede estar vacío, sin películas referenciadas aún (no es requerido). Por tanto, ambas colecciones se crean tercera y cuarta, sin importar el orden entre ellas.
3. La colección película va en quinto lugar, ya que requiere de referencias(ya deben existir esos documentos) para director,

género y país; no así para actor, ya que contiene un array que puede estar vacío, aún así actor se coloca antes.

4. Por último, aunque no necesario para la creación, se realiza un script para insertar 5 documentos en cada colección. Creo que esta claro por que debe ir el último, las colecciones deben estar creadas para recibir los datos.

3.2.1 Validaciones

Son usadas en Mongo, para dar forma a un documento en una colección de manera que filtra o verifica que los datos guardados son los correctos para dicha colección.

Son unas segunda reglas de control extra de los datos de la app, en la parte de la BBDD, que suplementa las que apliquen en el código utilizado para programar (Java) en la parte del backend. Es recomendable hacer el control en ambos sitios.

Además son utilizadas para: entornos de desarrollo y testing, aplicar seguridad, mantener estándares estructurales y optimizar el análisis.

A continuación detallo su estructura y explico los comandos usados mediante un ejemplo de código para la validación de la colección.

```
db.createCollection( → Comando de Mongo para crear una colección.
  "genero",{          → Nombre de la colección.
    validator:{        → Comando que contiene la validación.
      $jsonSchema:{ → $ indica que se va realizar una operación especial y
                    jsonSchema es un formato estándar para definir
                    estructuras.
      bsonType:"Object", → Indica el tipo esperado: objeto, cadena, entero,etc
```

```

required: [""], → Aquí se establece que campos (atributos) son
                obligatorios y cuáles no para una correcta inserción de
                documentos. Es un array.
properties: { → Es donde se establecen las características de los campos
    nombre: { → La denominación del campo
        bsonType: "string", → De tipo String
    enum: [ "", "", "" ], → Array creado para los nombres de los géneros, es
        decir, si el género especificado no está en el Enum, dará
        error.
    description: "" → Breve descripción de qué se espera de los
        datos, útil durante el desarrollo y posteriores
        actualizaciones.
    },
    subgeneros: { → Nuevo campo, de nombre subgéneros
        bsonType: "array", → De tipo Array
        items: { → Para descomponer el array, incluye su propia
            estructura interna definida.
            bsonType: "string",
            pattern: "^.{3,}$", → se usa para reglas de filtrado, con Regex
            description: "Debe ser una cadena con 3 o más caracteres"
        },
        description: "Lista de subgeneros como strings"
    }
}
}
}
},
validationLevel: "strict", → Este comando, es importante, ya que indica el
nivel y rango con el que se aplican las validaciones.
Hay 2 opciones:

```

La recomendada es “strict”, validando todos los documentos , tanto nuevos como actualizados, esto entre otras, permite aplicar dichas reglas a documentos existentes previos a la validación.

“moderate” solo valida los documentos nuevos y los actualizados con campos afectados. Útil cuando no quieres que los datos preexistentes sean modificados por la nueva validación.

validationAction: “error” → Este comando, indica que ocurre cuando un documento no cumple las reglas de validación, es decir, si se almacena o se rechaza.

Hay 2 opciones:

La recomendada es “error”, donde mongo rechaza la operación e impide que el documento se inserte o actualice.

“warn” es la otra opción, donde se permite la inserción y se genera un log de aviso. Útil para migrar datos o supervisar errores sin bloquear las acciones.

})

Anotación extra, aunque en este ejemplo no aparecen, existen las siguientes opciones (sustituye al pattern en los números):

minimum: 18 → establece el mínimo para un dato tipo numérico, en este caso, sólo serán válidos los casos mayores de 17.

maximun: 25 → establece el máximo para un dato tipo numérico, en este caso, sólo serán válidos los casos que sean menores que 26 .

4. Estructura de la BBDD Cine

La bbdd tiene la siguiente estructura, donde alberga 5 colecciones llamadas: País, Género, Actor, Director y Película. Aunque la clase Actor y Director, si se observan, son iguales en cuanto a contenido estructural, cabe reseñar la separación en colecciones distintas para una mejor organización de la app (esto es un espejo con la parte de Java en cuanto a almacenar información).

Además algunas de las colecciones mantienen una relación al referenciar o ser referenciadas, con su Id(en Mongo es tratado como un objeto por defecto, o como alternativa válida tratado como String si se configura, por otras colecciones.

Para todas las relaciones se eligió referenciar por `_id` frente a embeber documentos, ya que se estima oportuno que solo un dato, por ejemplo el campo que les de nombre, sea el que se obtenga como información de ese documento embebido para no saturar la información que le llega al usuario.

Por tanto, se considera que si se quiere ver datos totales de los documentos embebidos sea a través de una consulta propia. Estas relaciones son entre:

→País con Actor/Director/Película donde país es la referenciada.

→Género con Película donde género es el referenciado.

→Actor con País/Película donde actor es referenciado. Y con Película donde actor hace referencia a un array de películas.

→Director con País/Película, donde director es referenciado. Y con Película donde director hace referencia a un array de películas.

→Película con País/Género/Actor/Director donde película hace referencia a los respectivos Ids. Y con Actor/Director donde película es referencia en un array.

Por último, en este punto 4, paso a detallar el contenido de las colecciones una a una, pero antes comentar sobre el `_id`, ya que es común para todas las colecciones, que se genera de forma automática un nº en Hexadecimal para `_id` y es del tipo `ObjectId`, clase especial en Mongo para tratar los Ids. Por tanto, todas las colecciones tienen su `_id`. Hay que resaltar el **guión bajo de id**. Dicho esto, seguimos:

- ★ País. Está colección alberga los campos nombre e idioma oficial, ambos tratados como String y reglas de validación para impedir inserciones

vacías (mínimo 3 Caracteres) donde el campo nombre es el único requerido.

- ★ Género. Esta colección alberga los campos nombre y subgéneros, donde nombre es del tipo Enum, que establece 6 nombres posibles : ["ACCION", "COMEDIA", "DRAMA", "TERROR", "CIENCIA FICCIÓN", "FANTASÍA"]. Y el subgénero es un array de String. Ambas están validadas, el campo nombre el enum y los string del array para impedir inserciones vacías (mínimo 3 Caracteres) donde el campo nombre es el único requerido.
- ★ Actor. Esta colección alberga los campos nombre, país y películas, donde nombre es del tipo String y reglas de validación para definir su estructura e impedir inserciones vacías (mínimo 1 Caracter), país es referenciada por su _id y películas es referenciada con un array de sus correspondientes _ids.
- ★ Director. Esta colección alberga los campos nombre, país y películas, donde nombre es del tipo String y reglas de validación para definir su estructura e impedir inserciones vacías (mínimo 1 Caracter), país es referenciada por su _id y películas es referenciada con un array de sus correspondientes _ids.
- ★ Película. Esta colección alberga los campos título, director, elenco_actores, anio (año, porque la ñ no es recomendada nos guste o no), sinopsis, duración, género y país.

Vamos por parte y por grupos, primero los campos que son tipo de dato String:

→ título con sus reglas de validación para definir su estructura e impedir inserciones vacías (mínimo 1 Carácter).

→ sinopsis con sus reglas de validación para definir su estructura e impedir inserciones vacías (mínimo 10 Carácter).

Segundo los campos que son del tipo int:

→anio con sus reglas de validación para definir su estructura e impedir inserciones no válidas o lógicas, para este caso se toma como referencia la invención del cine, aunque es muy difícil conseguir ver una película de sus primeros años, lo imposible es poder ver una antes de su invención (año estimado del primer film público 1900).

→duración con sus reglas de validación para definir su estructura e impedir inserciones no válidas (duración mínima establecida para que se considere una película de 80 minutos).

Por último tenemos el grupo de campos que referencian a otras colecciones por su _id como director, país y género directamente y actor en un array.

5. Consultas avanzadas en Mongo Compass

Una vez pasado todo el proceso de creación del contenedor con la bbdd, llega el momento de manejar esos datos y para ello use Mongo Compass la GUI Oficial de Mongo, no la única existe la versión para trabajar en la nube.

Como todo este punto de la conexión, consultas, agregaciones(aunque de las agregaciones si se adjunta un script) e índices no se entregan con el proyecto, el contenedor no se envía, se basará en capturas de pantalla realizadas como demostración gráfica.

5.1 Conexión Docker-Mongo Compass

Procedemos a conectar el contenedor mongo creado en docker con Compass, para ello primero debemos arrancar el contenedor en Docker Desktop (el proceso de instalación del contenedor ya debe estar hecho y la base de datos

con sus colecciones creadas), lo localizamos y pulsamos run (símbolo play). Después abrimos Compass y pulsamos en nueva conexión (símbolo +, en conexiones). Llegamos a la siguiente pantalla que veremos:

New Connection
Manage your connection settings

URI 📘 Edit Connection String 🔵

mongodb://david:*****@localhost:27018/?authMechanism=SCRAM-SHA-256&authSource=admin

Name **Color** Red

☐ **Favorite this connection**
Favoriting a connection will pin it to the top of your list of connections

Advanced Connection Options

General **Authentication** TLS/SSL Proxy/SSH In-Use Encryption Advanced

Authentication Method

Username/Password OIDC X.509 Kerberos LDAP AWS IAM

Username
 Optional

Password
 Optional

Authentication Database 📘
 Optional

Authentication Mechanism

Default SCRAM-SHA-1 SCRAM-SHA-256

Cancel Save Connect Save & Connect

How do I find my connection string in Atlas?
If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.
[See example](#)

string in Atlas?
If you have an Atlas cluster, go to the Cluster view. Click the 'Connect' button for the cluster to which you wish to connect.
[See example](#)

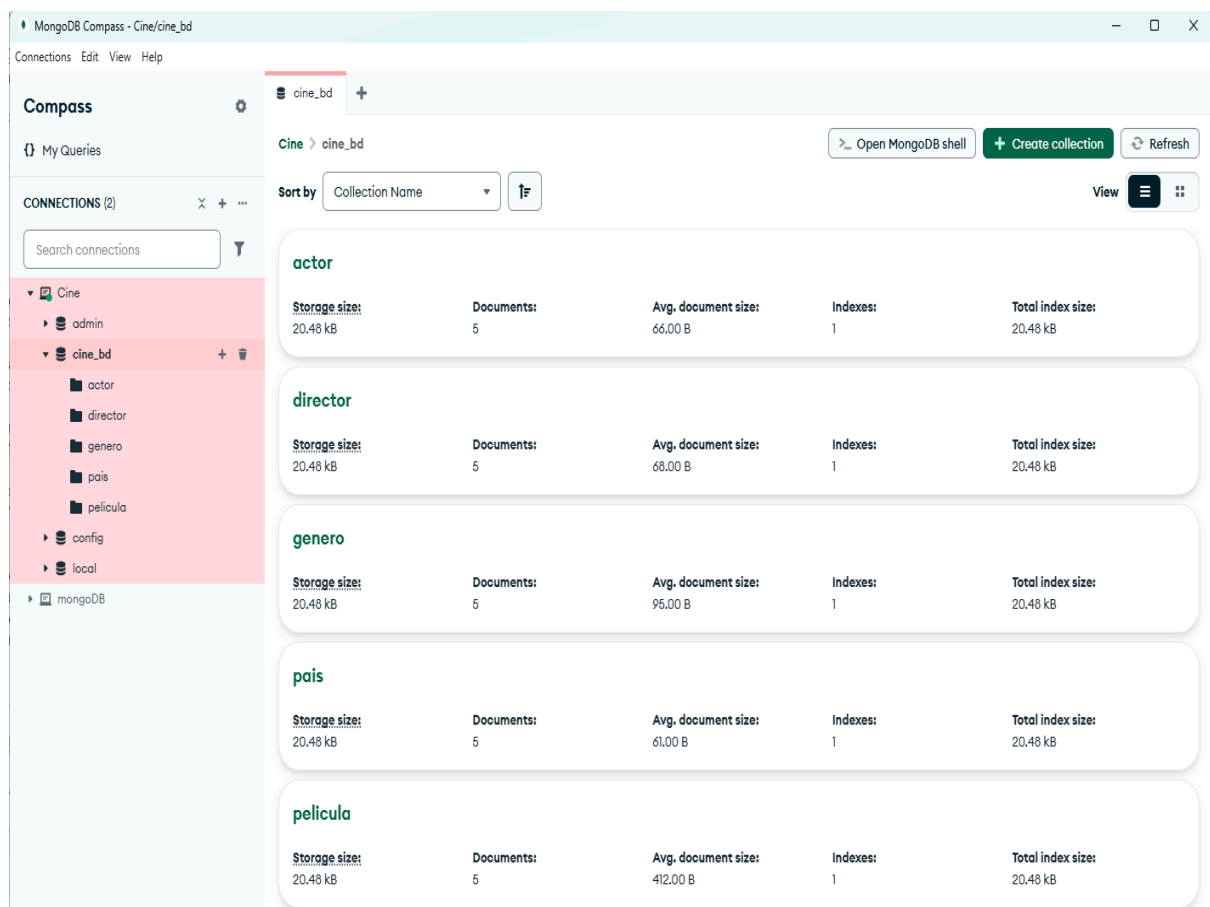
How do I format my connection string?
[See example](#)

Pasos y campos importantes a rellenar para conectar:

1º Por buenas prácticas Nombre de la base de datos: Cine

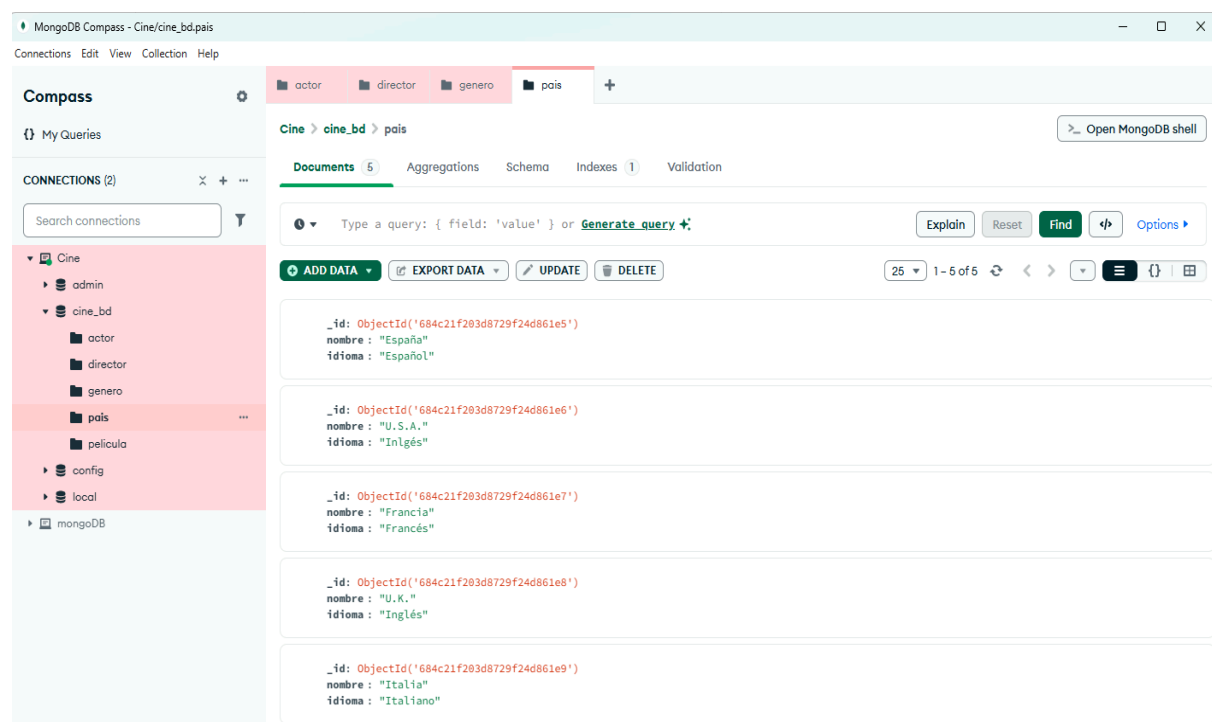
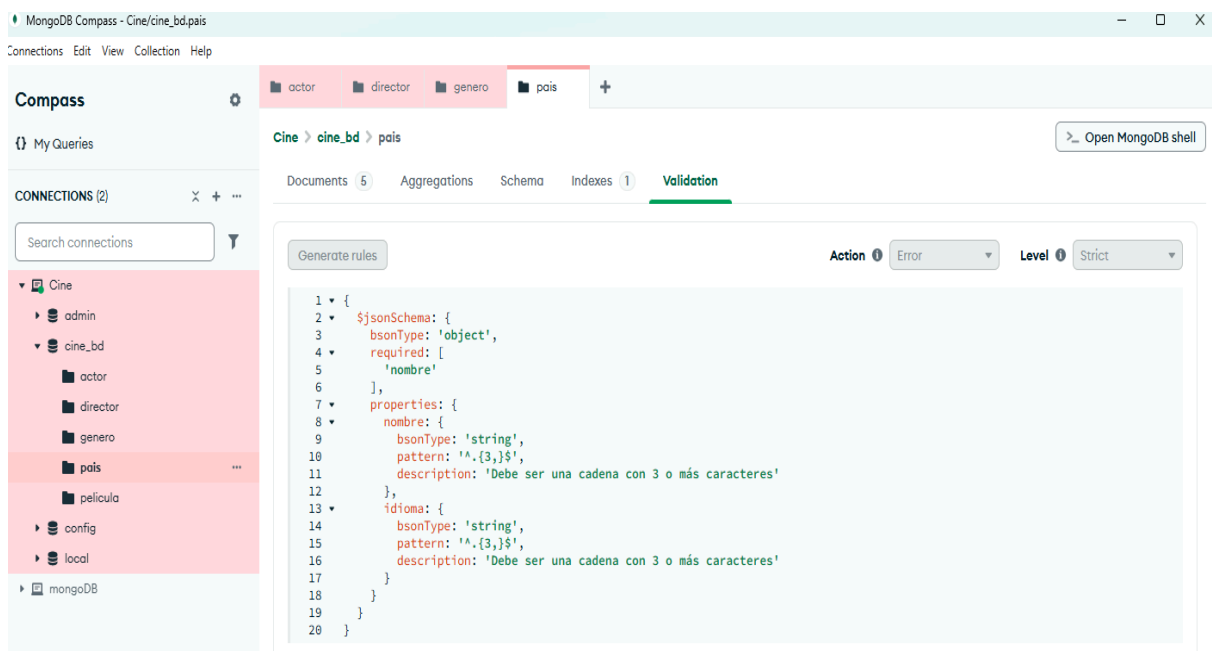
2º Pulsamos Advanced Connection Options para que se nos despliegue el panel visto en la foto. En Authentication Method, elegimos Username/Password (aquí indicaremos el nombre de usuario y contraseña root que se dieron al contenedor en sus respectivos campos). Authentication DataBase (aquí se especifica el nivel del usuario, en este caso es admin, es decir permiso de administrador o root) y se marca como Authentication Mechanism SCRAM-SHA-256. Si se observa se verá como la parte de URI cambió automáticamente quedando lista.

3º Una vez tenemos todos los campos rellenos, le damos a guardar. Se cerrará la ventana y ya aparecerá la base de datos lista para pulsar conectar (justo a la derecha del nombre de la bbdd) y conectarse a ella. Además como se observa en la siguiente imagen, las colecciones están creadas correctamente con el script de inicio.



Así se ven las colecciones y sus validaciones en Compass (aquí primero se creó el código en js para su instalación de inicio, a diferencias de las agregaciones que se crearán en Compass y se podrá obtener su código para JS o export a otros idiomas de programación como Java):

Colección y Validación País



Colección y Validación Género

MongoDB Compass - Cine/cine_bd.genero

Connections Edit View Collection Help

Compass

{ } My Queries

CONNECTIONS (2)

Search connections

Cine

- admin
- cine_bd
 - actor
 - director
 - genero
 - pais
 - pelicula
- config
- local

mongodb

Cine > cine_bd > genero

Documents 5 Aggregations Schema Indexes 1 Validation

Generate rules

Action Error Level Strict

```
1 {
2   $jsonSchema: {
3     bsonType: 'object',
4     required: [
5       'nombre'
6     ],
7     properties: {
8       nombre: {
9         bsonType: 'string',
10        enum: [
11          'ACCION',
12          'COMEDIA',
13          'DRAMA',
14          'TERROR',
15          'CIENCIA_FICCION',
16          'FANTASIA'
17        ],
18        description: 'Debe ser un valor del enum GeneroEnum en Java'
19      },
20      subgeneros: {
21        bsonType: 'array',
22        items: {
23          bsonType: 'string',
24          pattern: '^[3,]$',
25          description: 'Debe ser una cadena con 3 o más caracteres'
26        },
27        description: 'Lista de subgeneros como strings'
28      }
29    }
30  }
31 }
```

MongoDB Compass - Cine/cine_bd.genero

Connections Edit View Collection Help

Compass

{ } My Queries

CONNECTIONS (2)

Search connections

Cine

- admin
- cine_bd
 - actor
 - director
 - genero
 - pais
 - pelicula
- config
- local

mongodb

Cine > cine_bd > genero

Documents 5 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1 - 5 of 5

_id: ObjectId('684c21f283d8729f24d861e0')

nombre: "COMEDIA"

subgeneros: Array (2)

_id: ObjectId('684c21f283d8729f24d861e1')

nombre: "DRAMA"

subgeneros: Array (1)

_id: ObjectId('684c21f283d8729f24d861e2')

nombre: "ACCION"

subgeneros: Array (3)

_id: ObjectId('684c21f283d8729f24d861e3')

nombre: "TERROR"

subgeneros: Array (2)

_id: ObjectId('684c21f283d8729f24d861e4')

nombre: "CIENCIA_FICCION"

subgeneros: Array (3)

Colección y Validación Actor

MongoDB Compass - Cine/cine_bd.actor

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (2)

Search connections

Cine

admin

cine_bd

actor

director

genero

pais

pelicula

config

local

mongodb

actor

Cine > cine_bd > actor

Documents 5 Aggregations Schema Indexes 1 Validation

Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1 - 5 of 5

_id: ObjectId('684c21f203d8729f24d861ef')

nombre: "Samuel L. Jackson"

pais: ObjectId('684c21f203d8729f24d861e6')

_id: ObjectId('684c21f203d8729f24d861f0')

nombre: "Aubrey Plaza"

pais: ObjectId('684c21f203d8729f24d861e6')

_id: ObjectId('684c21f203d8729f24d861f1')

nombre: "Audrey Tautou"

pais: ObjectId('684c21f203d8729f24d861e7')

_id: ObjectId('684c21f203d8729f24d861f2')

nombre: "Nicoleta Braschi"

pais: ObjectId('684c21f203d8729f24d861e9')

_id: ObjectId('684c21f203d8729f24d861f3')

nombre: "Alex Angulo"

pais: ObjectId('684c21f203d8729f24d861e5')

MongoDB Compass - Cine/cine_bd.actor

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (2)

Search connections

Cine

admin

cine_bd

actor

director

genero

pais

pelicula

config

local

mongodb

actor

Cine > cine_bd > actor

Documents 5 Aggregations Schema Indexes 1 Validation

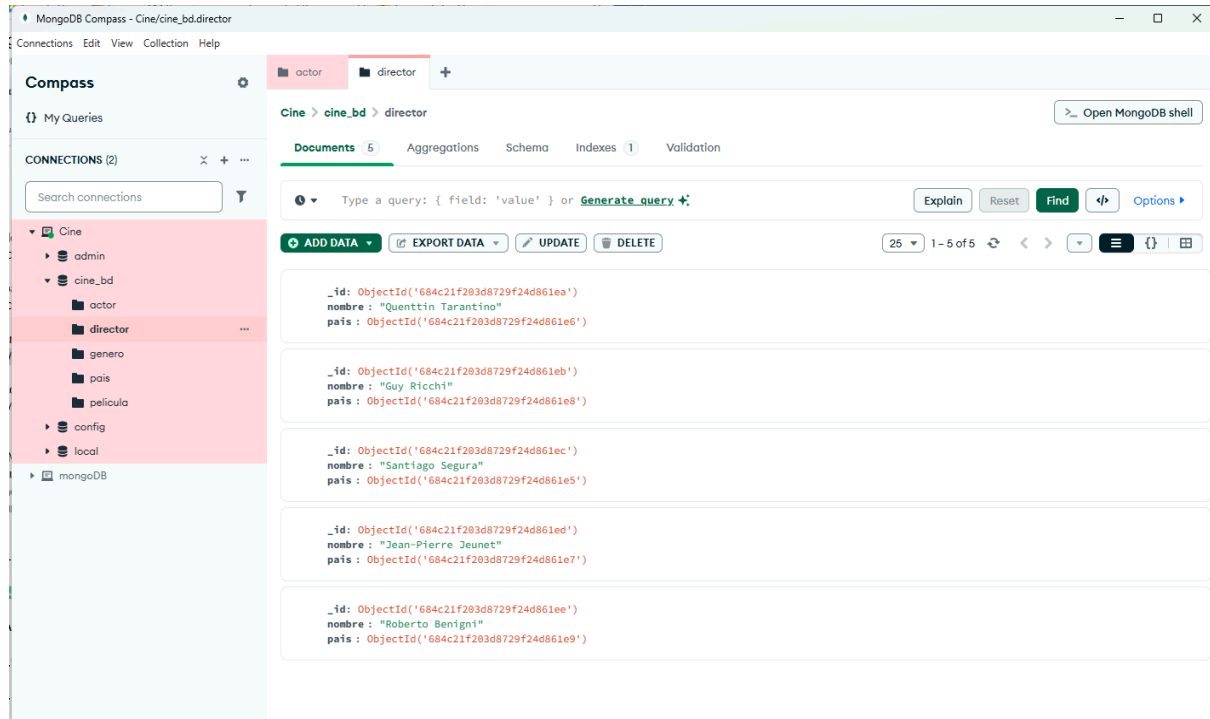
Generate rules

Action Error Level Strict

```
1 {
2   $jsonSchema: {
3     bsonType: 'object',
4     required: [
5       'nombre',
6       'pais'
7     ],
8     properties: {
9       nombre: {
10        bsonType: 'string',
11        pattern: '^[1,]*$',
12        description: 'Debe ser una cadena caracteres con al menos 1 caracter'
13      },
14       pais: {
15        bsonType: 'objectId',
16        description: 'Id de la Clase Pais'
17      },
18       peliculas: {
19        bsonType: 'array',
20        items: {
21          bsonType: 'objectId'
22        },
23        description: 'Filmografia del actor'
24      }
25    }
26  }
27 }
```

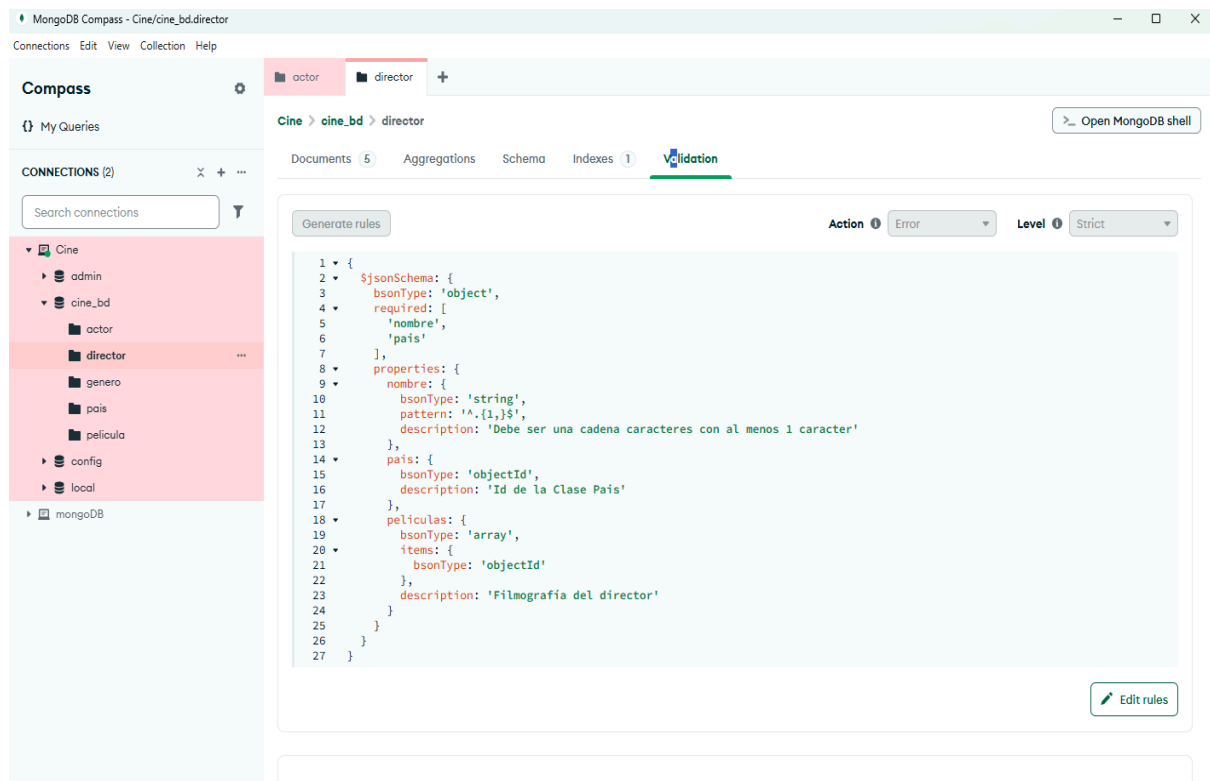
Edit rules

Colección y Validación Director



The screenshot shows the MongoDB Compass interface for the 'cine_bd' database, specifically the 'director' collection. The left sidebar shows the database structure with 'cine_bd' expanded, showing collections like 'actor', 'director', 'genero', 'pais', and 'pelicula'. The main panel displays the 'Documents' tab for the 'director' collection. It shows a list of 5 documents, each with fields: '_id' (ObjectId), 'nombre' (string), and 'pais' (ObjectId). The documents are for directors: Quentin Tarantino, Guy Ricchi, Santiago Segura, Jean-Pierre Jeunet, and Roberto Benigni.

_id	nombre	pais
ObjectId('684c21f283d8729f24d861ea')	Quentin Tarantino	ObjectId('684c21f283d8729f24d861e6')
ObjectId('684c21f283d8729f24d861eb')	Guy Ricchi	ObjectId('684c21f283d8729f24d861e8')
ObjectId('684c21f283d8729f24d861ec')	Santiago Segura	ObjectId('684c21f283d8729f24d861e5')
ObjectId('684c21f283d8729f24d861ed')	Jean-Pierre Jeunet	ObjectId('684c21f283d8729f24d861e7')
ObjectId('684c21f283d8729f24d861ee')	Roberto Benigni	ObjectId('684c21f283d8729f24d861e9')



The screenshot shows the MongoDB Compass interface for the 'cine_bd' database, specifically the 'director' collection, with the 'Validation' tab selected. The 'Generate rules' button is visible. The JSON Schema for the collection is displayed, defining the structure and validation rules for the documents.

```
1 {
2   $jsonSchema: {
3     bsonType: 'object',
4     required: [
5       'nombre',
6       'pais'
7     ],
8     properties: {
9       nombre: {
10        bsonType: 'string',
11        pattern: '^[1,]{5}$',
12        description: 'Debe ser una cadena caracteres con al menos 1 caracter'
13      },
14       pais: {
15        bsonType: 'objectId',
16        description: 'Id de la Clase Pais'
17      },
18       peliculas: {
19        bsonType: 'array',
20        items: {
21          bsonType: 'objectId'
22        },
23        description: 'Filmografía del director'
24      }
25    }
26  }
27 }
```

Colección y Validación Película

MongoDB Compass - Cine/cine_bd.película

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (2)

Search connections

Cine

- admin
- cine_bd
 - actor
 - director
 - genero
 - pais
 - película**
- config
- local

mongoDB

Cine > cine_bd > película

Documents (5) Aggregations Schema Indexes (1) Validation

Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find </> Options

ADD DATA EXPORT DATA UPDATE DELETE

25 1-5 of 5

```
{
  "_id": ObjectId("684c21f283d8729f24d861f4"),
  "titulo": "La vida es bella",
  "director": ObjectId("684c21f283d8729f24d861ee"),
  "elenco_actores": Array (1)
  "año": 1997
  "sinopsis": "Relata la historia de Guido Orefice, un hombre judío italiano, que usa..."
  "duración": 116
  "género": ObjectId("684c21f283d8729f24d861e1")
  "país": ObjectId("684c21f283d8729f24d861e9")
}
```

```
{
  "_id": ObjectId("684c21f283d8729f24d861f5"),
  "titulo": "Operación Fortune",
  "director": ObjectId("684c21f283d8729f24d861eb"),
  "elenco_actores": Array (1)
  "año": 2023
  "sinopsis": "El espía de élite Orson Fortune debe localizar y detener la venta de u..."
  "duración": 114
  "género": ObjectId("684c21f283d8729f24d861e2")
  "país": ObjectId("684c21f283d8729f24d861e8")
}
```

```
{
  "_id": ObjectId("684c21f283d8729f24d861f6"),
  "titulo": "El día de la bestia",
  "director": ObjectId("684c21f283d8729f24d861ec"),
  "elenco_actores": Array (1)
  "año": 1995
  "sinopsis": "Un sacerdote cree haber descifrado el mensaje secreto del Apocalipsis:..."
  "duración": 103
}
```

MongoDB Compass - Cine/cine_bd.película

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (2)

Search connections

Cine

- admin
- cine_bd
 - actor
 - director
 - genero
 - pais
 - película**
- config
- local

mongoDB

Cine > cine_bd > película

Documents (5) Aggregations Schema Indexes (1) **Validation**

Open MongoDB shell

```
1 {
2   "$schema": "http://json-schema.org/draft-07/schema#",
3   "bsonType": "object",
4   "required": [
5     "titulo",
6     "director",
7     "elenco_actores",
8     "año",
9     "sinopsis",
10    "duración",
11    "género",
12    "país"
13  ],
14  "properties": {
15    "titulo": {
16      "bsonType": "string",
17      "pattern": "^[a-zA-Z ]+$",
18      "description": "Debe ser una cadena con al menos un caracter"
19    },
20    "director": {
21      "bsonType": "objectId",
22      "description": "Id de la Clase Director"
23    },
24    "elenco_actores": {
25      "bsonType": "array",
26      "items": {
27        "bsonType": "objectId"
28      },
29      "description": "Elenco de actores que participan en la película"
30    },
31    "año": {
32      "bsonType": "int",
33      "minimum": 1900,
34      "description": "Debe ser un año posterior a la invención del cine"
35    },
36    "sinopsis": {
37      "bsonType": "string",
38      "pattern": "^[a-zA-Z ]+$",
39      "description": "Debe ser una cadena de 10 o más caracteres"
40    },
41    "duración": {
42      "bsonType": "int",
43      "minimum": 80,
44      "description": "Debe ser de 80 minutos o más para considerarse película"
45    },
46    "género": {
47      "bsonType": "objectId",
48      "description": "Id de la Clase Genero"
49    },
50    "país": {
51      "bsonType": "objectId",
52      "description": "Id de la Clase País"
53    }
54  }
55 }
```

5.2 Agregaciones

Las agregaciones en Mongo se utilizan para realizar búsquedas complejas o realizar ciertas operaciones sobre los datos. Básicamente funcionan con el uso de pipelines (tuberías para la conexión). Se componen de esquemas que establecen el orden de ejecución y como veremos en las siguientes imágenes Compass ofrece una GUI, fácil y eficaz, donde escribir y ordenar estos esquemas y las funciones que se usan. Veamos con imágenes cómo se crea por ejemplo que nos muestre una consulta con todos los datos. Compass nos permite exportar (con explain) el código a otros lenguajes de programación. Así como exportar a archivos tipo json o csv, o copiar para usar en JS.

Export

Aggregation on cine_bd.pelicula

Export results from the aggregation below

```
db.getCollection('pelicula').aggregate([
  {
    $lookup: {
      from: 'director',
      localField: 'director',
      foreignField: '_id',
      as: 'director'
    }
  }
],
{ maxTimeMS: 60000, allowDiskUse: true }
);
```

Export File Type

JSON

CSV

> Advanced JSON Format

Cancel

Export...

Al realizar la agregación se está en la colección Película.

Añadir stages (etapas del esquema) En este ejemplo se añade primero un \$lookup (que es un join, por comparar con algo conocido como es MySQL, para unir dos colecciones) En este caso se unen Película con Director, siendo Director la “huésped” y sobre la que se aportan los datos. Segundo \$unwind (que se usa para desfragmentar un array de objetos y devuelve como objeto y así poder acceder a sus campos) . Como se puede observar al no haber realizado ningún filtro primero nos muestra la agregación aplicada a todas las colecciones. Al igual que en sql para mejorar el rendimiento primero se filtra, y en mongo luego se opera y por último se muestra. También indicar como a la izquierda está la zona de codificación y a la derecha se obtiene la vista de ese resultado. Si no se codificó correctamente, aparece un mensaje de aviso. Los campos requeridos son from: indica la colección con la que se hará la unión, localField: indica el nombre del campo que alberga el _id de la colección, foreignField: indica el nombre del campo en el que debe coincidir, as: indica el nombre que del campo que se muestra. Como se puede observar en las colecciones de la derecha, director tras la unión con lookup aparece como array (de ObjectId) y tras el unwind se puede ver que aparecen como objetos y ya se pueden acceder a sus campos.

The screenshot displays the MongoDB Aggregation Framework interface with two stages defined.

Stage 1: \$lookup

The pipeline stage is defined as follows:

```
1 // **
2 // from: The target collection.
3 // localField: The local join field.
4 // foreignField: The target join field.
5 // as: The name for the results.
6 // pipeline: Optional pipeline to run on the
7 // let: Optional variables to use in the pipeline
8 //
9 {
10   from: "director",
11   localField: "director",
12   foreignField: "_id",
13   as: "director"
14 }
```

The output after the \$lookup stage (Sample of 5 documents) shows three documents where the 'director' field is an array of ObjectId values:

- Document 1: titulo: "La vida es bella", director: Array (1), elenco_actores: Array (1), año: 1997, sinopsis: "Relata la historia de Guido Orefice, un hombre judío italiano, que usa...", duración: 116, genero: ObjectId('684c21f203d8729f24d861e2...'), país: ObjectId('684c21f203d8729f24d861e9').
- Document 2: titulo: "Operación Fortune", director: Array (1), elenco_actores: Array (1), año: 2023, sinopsis: "El espía de élite Orson Fortune debe localizar y detener la venta de u...", duración: 114, genero: ObjectId('684c21f203d8729f24d861e2...'), país: ObjectId('684c21f203d8729f24d861e9').
- Document 3: _id: ObjectId('684c21f203d8729f24d861f6'), titulo: "El día de la bestia", director: Array (1), elenco_actores: Array (1), año: 1995, sinopsis: "Un sacerdote cree haber descifrado el mensaje secreto del Apocalipsis...", duración: 103, genero: ObjectId('684c21f203d8729f24d861e9').

Stage 2: \$unwind

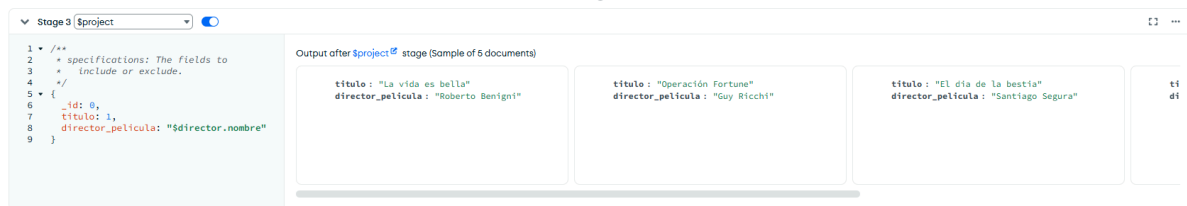
The pipeline stage is defined as follows:

```
1 // **
2 // path: Path to the array field.
3 // includeArrayIndex: Optional name for the
4 // preserveNullAndEmptyArrays: Optional
5 // toggle to unwind null and empty value
6 //
7 {
8   path: "$director",
9   includeArrayIndex: 'string',
10   preserveNullAndEmptyArrays: true
11 }
```

The output after the \$unwind stage (Sample of 5 documents) shows the same three documents, but the 'director' field is now an object (ObjectId) instead of an array:

- Document 1: _id: ObjectId('684c21f203d8729f24d861f4'), titulo: "La vida es bella", director: ObjectId('684c21f203d8729f24d861e2...'), elenco_actores: Array (1), año: 1997, sinopsis: "Relata la historia de Guido Orefice, un hombre judío italiano, que usa...", duración: 116.
- Document 2: _id: ObjectId('684c21f203d8729f24d861f5'), titulo: "Operación Fortune", director: ObjectId('684c21f203d8729f24d861e2...'), elenco_actores: Array (1), año: 2023, sinopsis: "El espía de élite Orson Fortune debe localizar y detener la venta de u...", duración: 114.
- Document 3: _id: ObjectId('684c21f203d8729f24d861f6'), titulo: "El día de la bestia", director: ObjectId('684c21f203d8729f24d861e9'), elenco_actores: Array (1), año: 1995, sinopsis: "Un sacerdote cree haber descifrado el mensaje secreto del Apocalipsis...", duración: 103.

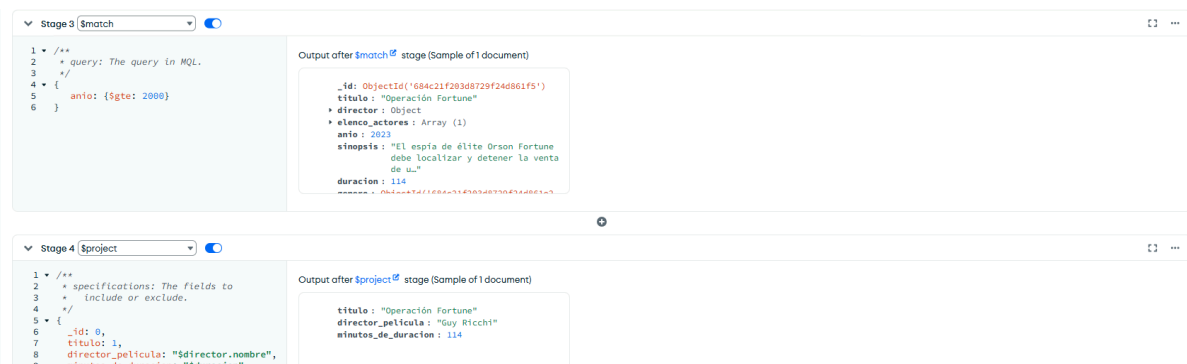
Tras realizar el lookup y el respectivo unwind para descomponer array de objetos, se añade un nuevo stage \$project, que es usado para mostrar el resultado de todo lo anterior, se indica el nombre del campo y 1 para que se muestre y 0 para que no. El _id se muestra por defecto, sino quiere verse se le da valor 0. En este caso se muestra el título y el nombre del director de la película.



Otro ejemplo con project, en este caso se muestra además la duración



Filtrar, como comente anteriormente, es buenas prácticas y mejora el rendimiento. Filtrar antes de seguir operando, en este caso creo antes del project el filtro con \$match, para que muestre solo las películas posteriores al año 2000. Como se puede ver, entre las stages hay un símbolo + para poder agregar en cualquier orden de inserción, antes o después. Además, admite seleccionar, arrastrar y soltar para organizar los stage del esquema.



Agregación para mostrar todos los datos legibles al usuario de una película. Si se observa, se puede ver que estamos en las agregaciones de la colección película, y justo debajo aparece el orden de los stages del esquema. Para cada lookup hay que aplicar un unwind. Y además se ve el resultado obtenido tras el último stage \$project.

Cine > cine_bd > película

Documents 5 Aggregations Schema Indexes 1 Validation

Match Lookup Unwind Lookup Unwind Lookup Unwind Lookup Unwind \$project

Untitled - modified SAVE + CREATE NEW EXPORT TO LANGUAGE

```
1 /**
2  * path: Path to the array field.
3  * includeArrayIndex: Optional name for in
4  * preserveNullAndEmptyArrays: Optional
5  * toggle to unwind null and empty value
6  */
7 {
8   path: "$pais",
9   includeArrayIndex: 'string',
10  preserveNullAndEmptyArrays: true
11 }
```

Output after \$unwind stage (Sample of 1 document)

```
{
  _id: ObjectId('684c21f203d8729f24d861f5'),
  titulo: "Operación Fortune",
  director: Object,
  elenco_actores: Object,
  anio: 2023,
  sinopsis: "El espía de élite Orson Fortune debe localizar y detener la venta de u...",
  duracion: 114,
  genero: Object
}
```

Stage 10 \$project

```
1 /**
2  * specifications: The fields to
3  * include or exclude.
4  */
5 {
6   _id: 0,
7   titulo: 1,
8   director_pelicula: "$director.nombre",
9   elenco_actores: "$elenco_actores.nombre",
10  anio: 1,
11  sinopsis: 1,
12  duracion: 1,
13  genero: "$genero.nombre",
14  pais: "$pais.nombre"
15 }
```

Output after \$project stage (Sample of 1 document)

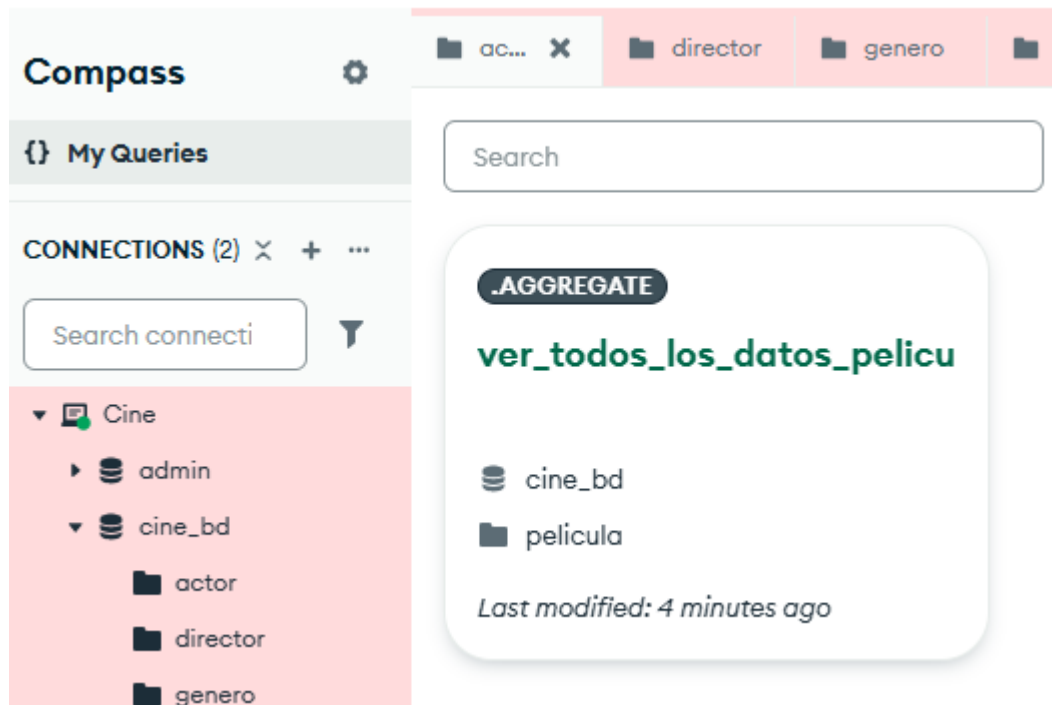
```
{
  titulo: "Operación Fortune",
  anio: 2023,
  sinopsis: "El espía de élite Orson Fortune debe localizar y detener la venta de u...",
  duracion: 114,
  director_pelicula: "Guy Ricchi",
  elenco_actores: "Aubrey Plaza",
  genero: "ACCION",
  pais: "U.K."
}
```

+ Add Stage

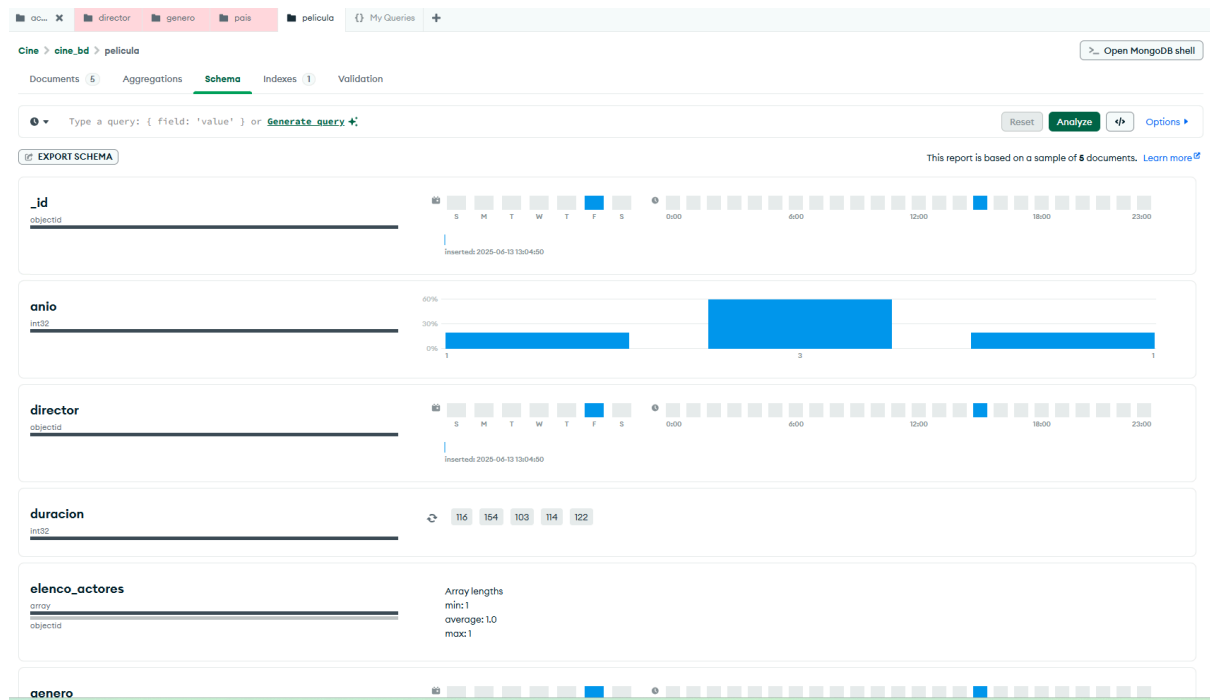
[Learn more about aggregation pipeline stages](#)

Una vez creado, te ofrecen 3 opción de guardado: guardar , guardar como para dar un nombre y crear vista(se crea en memoria un Documento explícito con los resultados de la agregación).

En la sección de { } My Queries se encuentra las agregaciones que vayamos creando, que después podrán ser usadas por el lenguaje de programación e incluso te prepara el código para ese otro lenguaje como Java.



Se puede consultar el schemes donde te da los datos en una forma más visual.



Insertar un nuevo documento en la colección, en versión actual la referencia al ObjectId se realiza con \$soid en lugar de con objectId(.....)

Insert Document

To collection cine_bd.actor

VIEW  

```
1  /**
2  * Paste one or more documents here
3  */
4  {
5    "_id": {
6      "$soid": "684c564555d855ac0cdec289"
7    },
8    "nombre": "Jason Statham",
9    "pais": {
10     "$soid": "684c21f203d8729f24d861e8"
11   }
12 }
```

Cancel

Insert

5.3 Indexación de Campos

Aunque Mongo ya genera un _id para documento, da la posibilidad de indexar por otros campos, con el único fin de mejorar el rendimiento en las búsquedas de manera interna. Se suele hacer sobre el campo que más veces sea consultado, por ejemplo en este proyecto, para buscar un país o un actor se utiliza su campo nombre, entonces nombre será el segundo índice (tras el _id del documento) con el fin de agilizar la búsquedas en esa colección. Veamos la ventana de creación que nos muestra Compass y las opciones marcadas como el campo que se indexará, el 1 para indicar ascendente, que sean único y use custom locale para indicar que las normas ortográficas es en español y que ignore las diferencias entre mayúsculas, minúsculas y acentos(strength:1):

Create Index

cine_bd.actor

Index fields

nombre

▼

1 (asc)

▼

+

▼ Options

☒ **Create unique index**

A unique index ensures that the indexed fields do not store duplicate values; i.e. enforces uniqueness for the indexed fields.

☐ **Index name**

Enter the name of the index to create, or leave blank to have MongoDB create a default name for the index.

☐ **Create TTL**

TTL indexes are special single-field indexes that MongoDB can use to automatically remove documents from a collection after a certain amount of time or at a specific clock time.

☐ **Partial Filter Expression**

Partial indexes only index the documents in a collection that meet a specified filter expression.

☐ **Wildcard Projection**

Wildcard indexes support queries against unknown or arbitrary fields.

☒ **Use Custom Collation**

Collation allows users to specify language-specific rules for string comparison, such as rules for lettercase and accent marks.

```
1 ▼ {
2     "locale": "es",
3     "strength": 1,
4
5 }
```



☐ **Create sparse index**

Sparse indexes only contain entries for documents that have the indexed field, even if the index field contains a null value. The index skips over any document that is missing the indexed field.

Cancel

Create Index

5.4 Comandos buenos de conocer

Aunque una vez creada la configuración para el inicio del contenedor, todo se realizó por Mongo Compass, es bueno conocer esos comandos ya sea por la terminal o Compass.

Para consultas: búsqueda, filtros y proyecciones →

`.find()` para búsquedas → // devuelve todas

`db.pelicula.find();`

`$eq` igual → //Devuelve las igual a 180

`db.pelicula.find({ duracion: { $eq: 180 } })`

`$gt` mayor que → //Devuelve las mayor a 120

`db.pelicula.find({ duracion: { $gt: 120 } });`

`$lt` menor que → //Devuelve las anteriores a 2010

`db.pelicula.find({ anio: { $lt: 2010 } });`

`$in` en una lista → //Muestra los documentos en la colección Pais con los nombre de España y Francia.

`db.pais.find({nombre: { $in: ["España", "Francia"]}})`

//Muestra los documentos en la colección Pais con los nombre dados.

`$and` / `$or` combinaciones → //Muestra los documentos en la colección película que su año sea 2018 o su país estados unidos

`db.pelicula.find({ $or: [{ anio: { $lt: 2018 } }, { pais: "U.S.A." }]});`

Proyección, se marca con 0 para que no se muestre el campo o 1 para que se muestre → //Muestra solo el campo título.

`db.pelicula.find({ _id: 0, titulo: 1});`

Inserciones y modificaciones →

`.insertOne()` / `.insertMany()` → Se inserta uno o varios documentos.

`db.pais.insertOne({ nombre: "Peru", idioma: "Español" });`

`db.pais.insertMany({....., });`

`.updateOne()` / `.updateMany()` → Cambia campos existentes o añade nuevos. Usa los siguiente comandos:

\$set → Añade o actualiza campo.

\$unset → Elimina campo.

\$inc → Incrementa en un número.

\$rename → Cambia el nombre al campo.

`db.pais.updateOne({nombre: "España"}, {$set :{idioma: "Castellano"}})`

`.deleteOne()` / `.deleteMany()` → Borra uno o todos los documentos

`db.pais.deleteOne({nombre: "Italia"});`

Pipelines de agregación →

El actual Compass ofrece un generador visual que te permite añadir etapas secuenciales (stages) y ver resultados en tiempo real:

\$match: Filtra documentos según criterios específicos.

\$group: Agrupa documentos y permite realizar cálculos como sumas y promedios.

\$sort: Ordena los documentos según un campo determinado.

\$project: Selecciona y transforma los campos de salida.

\$limit: Restringe la cantidad de documentos devueltos.

\$lookup: Realiza uniones entre colecciones.

\$unwind: Descompone arrays en documentos individuales.

6. Conexión con Java

Para conectar la BBDD MongoDB con Java uso la librería oficial de Mongo, Mongo Java Driver. La cual ofrece Clases muy útiles para trabajar con Mongo en Java. Para levantar el proyecto de Java utilice Maven, por lo que solo tuve que añadir las [dependencias de MongoDB](#) en el archivo pom.xml. Como mayor consideración, se tuvo que elegir la versión sync, más acorde para este trabajo y empezar a conocer MongoDB. Estas son:

MongoClient →

Es el punto de entrada para conectar MongoDB desde Java. Se crea con MongoClient.create(...), permite acceder a la bbdd y gestionar la conexión con el servidor. Código de ejemplo:

```
private static final String URI =  
"mongodb://david:12345@localhost:27018/?authSource=admin&authMechanism=SCRAM-SHA-256";  
private MongoClient conexionMongo;  
conexionMongo = MongoClient.create(URI);
```

MongoDataBase →

Representa una base de datos específica dentro del servidor. Se obtiene a través de MongoClient y permite acceder a las colecciones. Código de ejemplo:

```
private static final String DATABASE_NAME = "cine_bd";  
baseDatosMongo = conexionMongo.getDatabase(DATABASE_NAME);
```

MongoCollection<T> →

Representa una colección de documentos (una tabla, en términos relacionales). Se obtiene desde MongoDatabase y permite realizar operaciones CRUD sobre los documentos. Código de ejemplo:

```
private final MongoCollection<Document> coleccion;  
public PeliculaDao(MongoDatabase db) {  
    this.coleccion = db.getCollection("pelicula");  
}  
public boolean insertar (Pelicula peli) {  
    InsertOneResult resultado = coleccion.insertOne (peli.convertirEnDocumento ());  
    return resultado.wasAcknowledged(); }  
}
```

Document →

Es una clase genérica de MongoDB que representa un documento BSON como si fuera un Map<String, Object>. Es muy versátil, usada tanto para insertar, actualizar o consultar documentos. Código de ejemplo:

```
public Document convertirEnDocumento() {  
  
    Document doc = new Document();  
    doc.append("_id", _id);  
    doc.append("nombre", nombre);  
    doc.append("idioma", idioma);  
  
    return doc;  
}
```

Filters →

Es una clase utilitaria para crear filtros de consulta (WHERE en SQL). Usados en métodos como .find() para búsquedas, .updateOne() para modificar uno, .deleteOne () para eliminar uno, etc..

Y tiene métodos estáticos como eq (igual que), gt (mayor que), lt (menor que), and, or, etc... Código de ejemplo:

```
public Pelicula buscarPorId(ObjectId id) {  
    Document doc = coleccion.find(eq("_id", id)).first();  
    if (doc == null) return null;  
    return Pelicula.convertirDesdeDocumento(doc);  
}
```

Updates →

Clase utilitaria para crear operaciones de actualización como \$set para modificar o crear un campo, \$push para agregar un valor a un array, \$pull para eliminar un valor del array. Código de ejemplo:

```
public boolean modificarIdiomaPelicula(ObjectId id, String idioma) {  
    UpdateResult resultado = coleccion.updateOne(  
        eq("_id", id),  
        set("idioma", idioma)  
    );  
    return resultado.getModifiedCount() > 0;  
}
```


UpdateResult, InsertOneResult y DeleteResult →

Representan el resultado de operaciones de escritura (inserción, actualización o eliminación). Código de ejemplo:

```
public boolean eliminarPelicula(ObjectId id) {  
    DeleteResult resultado = coleccion.deleteOne(eq("_id", id));  
  
    return resultado.getDeletedCount() > 0;  
}
```

ObjectId →

Representa los identificadores únicos (_id) generados automáticamente por MongoDB. Cabe resaltar que son Objetos, no strings ni enteros.

Además, Java cuando lo creas, lo asigna automáticamente, es decir, con new ObjectId() sin pasar ningún parámetro se crea.

No existen conflictos entre Java y MongoDB a pesar de que se pueda crear en ambos sitios indistintamente. Solo tener la consideración a la hora de hacer los constructores de la clase: uno con _id(y resto de parámetros necesarios) para pasar la información del objeto a la BBDD y otro sin _id(y restos de parámetros necesarios) para cuando se reciben los datos del documento creado, esta vez, en la BBDD.

FindIterable<T> / MongoClient<T> →

Permiten iterar sobre los resultados de una consulta. El comando .find() devuelve un FindIterable, pudiendo convertirlo a lista o recorrerlo con cursor.

Aunque estas clase no llegué a usarla en este proyecto, parecen bastantes utiles.

Bson →

Interfaz común para los filtros, actualizaciones, proyecciones, etc...

7. Conclusión

Este proyecto ha sido una experiencia valiosa para aprender a diseñar, construir y gestionar una base de datos MongoDB dentro de un contenedor Docker, comprendiendo además la importancia de automatizar su creación mediante JavaScript. La combinación de estas tecnologías no solo ha facilitado la estructuración y despliegue de la BBDD, sino que también ha servido como base para futuras implementaciones más complejas.

Para principiantes, enfrentarse a la diversidad sintáctica entre lenguajes puede ser un desafío adicional. Diferencias en la escritura de estructuras como corchetes, llaves, comillas, y estilos de nomenclatura como *camelCase* o *snake_case* pueden generar errores leves pero recurrentes, afectando tanto la corrección del código como la adopción de buenas prácticas de desarrollo. Sin embargo, superar estas dificultades es clave para consolidar una base sólida en programación y optimizar la eficiencia en proyectos multidisciplinarios.

Uno de los aspectos que más he valorado en este proceso es la facilidad y agilidad que ofrece MongoDB Compass como herramienta visual para interactuar con la BBDD. Su interfaz intuitiva hace que las consultas, agregaciones y administración de datos sean accesibles, lo que resulta especialmente útil durante el aprendizaje. Aunque en entornos profesionales la preferencia suele inclinarse hacia herramientas más especializadas o directamente la línea de comandos, MongoDB Compass sigue siendo una opción poderosa para visualizar datos y construir consultas de manera eficiente.

Aunque hay frameworks que te ayudan (cuando ya los dominas) en muchas cosas (generar y manipular Clases DAO, DTO, conexiones, etc...), las clases que ofrece la librería de Mongo Java Driver (frente a MongoDB SpringBoot Starter) también me resultaron interesantes y útiles, incluso más que las SQL con Connection, PreparedStatement y ResultSet que pueden resultar más laboriosas para el desarrollador inexperto.