

INSTITUTO TECNOLÓGICO DEL PUTUMAYO

Taller Crud Mongo Db

Base de datos y almacenamiento masivo

Ingry Nathaly Silva

Octubre 2024

TABLA DE CONTENIDOS

Taller Crud Mongo Db.....	1
Resumen Ejecutivo.....	3
Introducción.....	4
<i>Contexto y Motivación</i>	4
<i>Alcance del Informe.....</i>	4
<i>Objetivos.....</i>	4
Metodología.....	5
<i>Herramientas Utilizadas</i>	5
<i>Procedimientos.....</i>	5
Desarrollo del Informe.....	21
<i>Descripción de la Base de Datos de Datos Personal:</i>	21
<i>Diseño de la base de datos:</i>	21
<i>Métodos de captura:.....</i>	23
<i>Consultas:</i>	24
Análisis y Discusión.....	62
Conclusiones	63
Recomendaciones	63
Referencias	64

Resumen Ejecutivo

El presente informe documenta el desarrollo de un taller sobre operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en MongoDB, una base de datos NoSQL orientada a documentos. Durante el taller, se implementaron diversas operaciones utilizando herramientas gráficas como MongoDB Compass, JSONGrid y Studio 3T, facilitando la gestión de datos estructurados en formato JSON. Se trabajó con bases de datos como "Episodios" y "School", aplicando consultas avanzadas y actualizaciones a través de operadores específicos de MongoDB.

El informe abarca la creación de bases de datos y colecciones, la carga de datos mediante archivos JSON, y la ejecución de consultas para extraer, modificar y gestionar información almacenada. Además, se analizó la estructura de los datos y se propuso el uso de embebidos (datos anidados) para optimizar el acceso a datos relacionados. Se discutieron las ventajas y desafíos de este enfoque, particularmente en términos de redundancia de datos y eficiencia en las consultas.

MongoDB se presentó como una herramienta versátil para manejar grandes volúmenes de datos no estructurados, ofreciendo flexibilidad y escalabilidad. Se recomendaron buenas prácticas, como la optimización de índices y la evitación de la duplicación de datos, para maximizar el rendimiento del sistema. Este taller proporciona una base sólida para el manejo de bases de datos NoSQL y su aplicación en entornos de datos complejos.

Introducción

El documento es un informe sobre el taller de CRUD en MongoDB, una base de datos NoSQL orientada a documentos. En este taller, se trabajó con varias herramientas para gestionar bases de datos y colecciones dentro de MongoDB, incluyendo MongoDB Compass, JSONGrid, y Studio 3T, enfocándose en operaciones CRUD (Crear, Leer, Actualizar, Eliminar) con datos en formato JSON.

Contexto y Motivación

El taller tiene como objetivo la comprensión y aplicación de las operaciones CRUD en una base de datos MongoDB, esencial para la gestión eficiente de datos en sistemas modernos. MongoDB permite manejar grandes volúmenes de información de manera flexible y escalable, lo que lo convierte en una herramienta popular para desarrolladores y administradores de bases de datos.

Alcance del Informe

Este informe cubre desde la creación de bases de datos y colecciones, hasta la inserción, consulta y actualización de datos en MongoDB. Además, se incluye la discusión de las mejores prácticas y análisis de rendimiento en operaciones específicas.

Objetivos

- Aplicar operaciones CRUD en una base de datos MongoDB.
- Utilizar herramientas gráficas para administrar colecciones y consultar datos.
- Analizar y optimizar la estructura de datos mediante el uso de relaciones embebidas en MongoDB.

Metodología

Herramientas Utilizadas

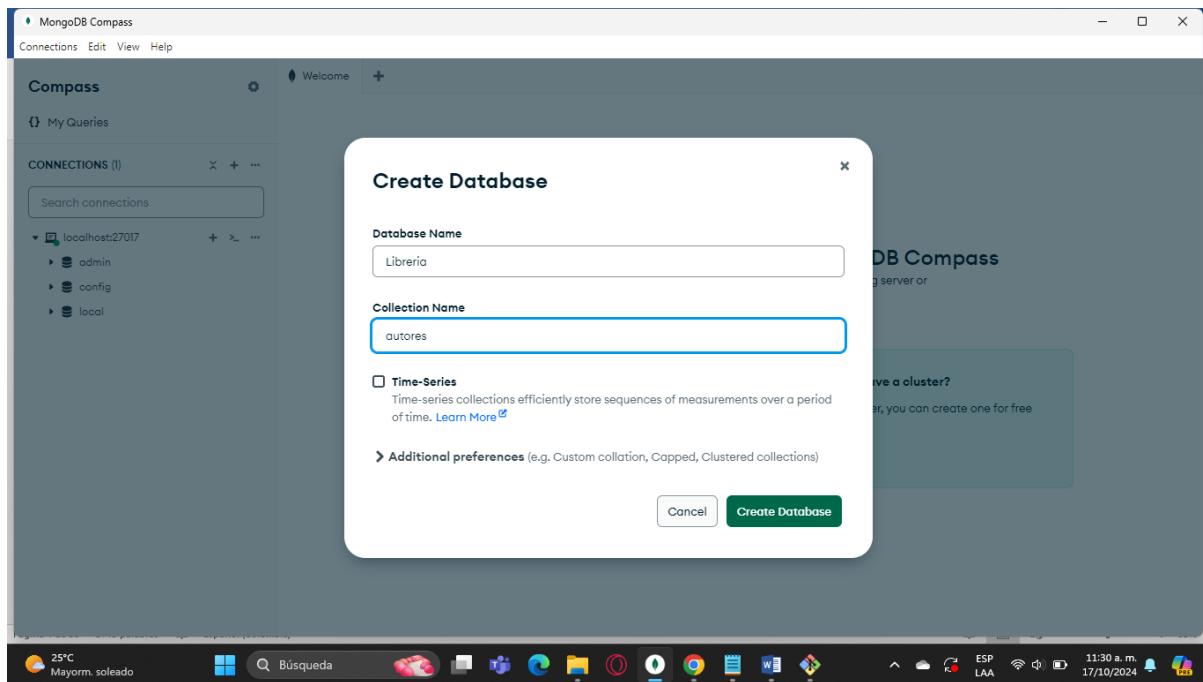
- MongoDB: Base de datos NoSQL utilizada para almacenar y gestionar la información del sistema de reservas de vuelos
- MongoDBCompass: Es una herramienta gráfica oficial de MongoDB diseñada para facilitar la visualización, consulta, y administración de bases de datos MongoDB sin necesidad de usar la línea de comandos. A través de su interfaz de usuario, permite explorar el contenido de las colecciones, ejecutar consultas, analizar índices y realizar tareas administrativas de manera intuitiva.
- JSONGrid: Es una herramienta o componente de visualización que permite trabajar con datos en formato JSON de una manera tabular y fácil de gestionar. A menudo es utilizada en aplicaciones web o entornos donde se manejan grandes cantidades de datos estructurados en JSON, proporcionando una forma más intuitiva de visualización y edición.
- Studio 3T: Es una herramienta GUI avanzada para MongoDB que facilita la interacción con bases de datos MongoDB mediante una interfaz gráfica amigable y funcionalidades de desarrollo integradas. Está diseñada tanto para desarrolladores como para administradores de bases de datos, ofreciendo características avanzadas que permiten optimizar y simplificar el trabajo con MongoDB.

Procedimientos

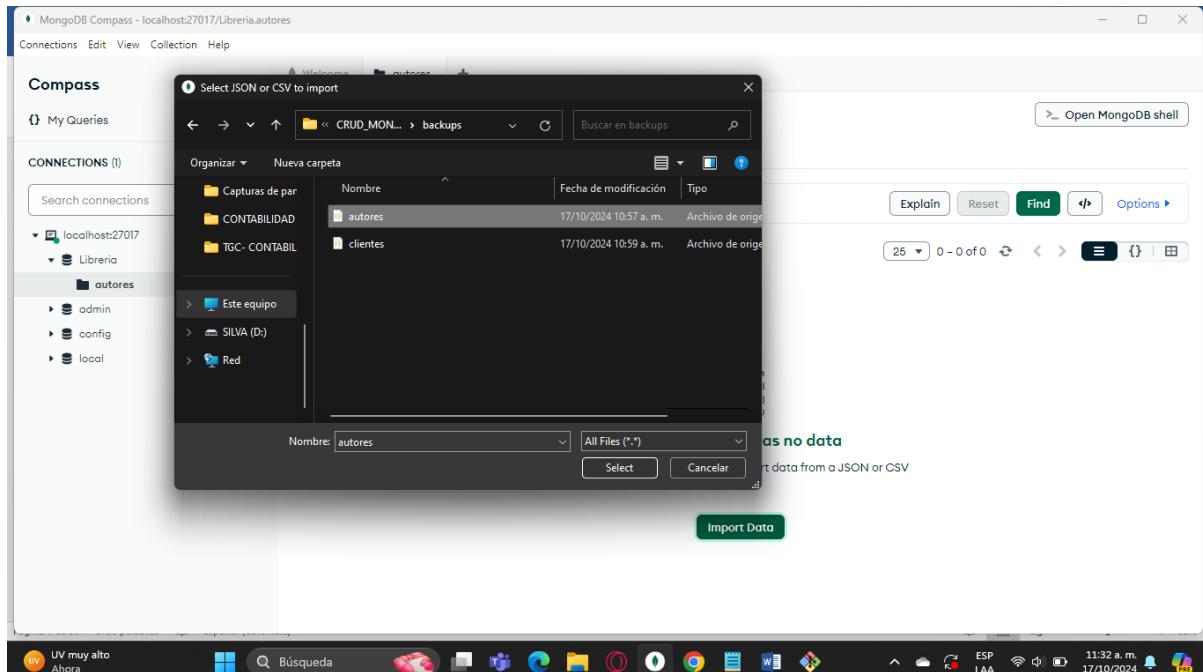
1. Métodos y pasos seguidos para llevar a cabo el análisis o el trabajo

- **Creación de bases de datos y colecciones:** Se crearon varias bases de datos como "Episodios" y "School", con colecciones como "Narcos", "Westworld", "grades", entre otras.
- **Importación de datos:** Utilización de archivos JSON para la carga de datos en MongoDB a través de herramientas como MongoDB Compass.
- **Consultas:** Se realizaron diversas consultas para obtener, filtrar y ordenar datos de las colecciones, utilizando comandos como find y sort.
- **Operaciones de actualización:** Se usaron operadores como \$set y \$push para modificar documentos existentes y agregar nueva información.

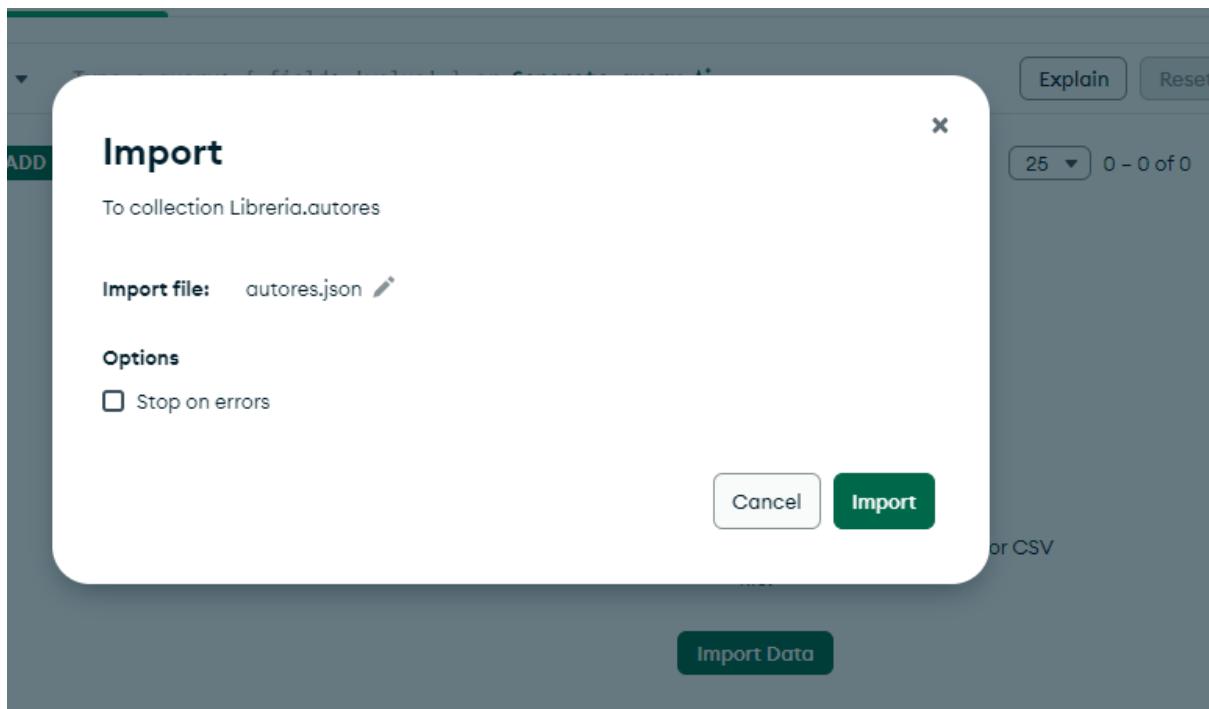
1.1. Creación base de datos personal



Creacion de la base de datos libreria con la collection autores.



Importacion del JSON a la collection.



MongoDB Compass - localhost:27017/Libreria.autores

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS ()

localhost:27017

Libreria

autores

Welcome autores

localhost:27017 > Libreria > autores

Documents 2 Aggregations Schema Indexes 1 Validation

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

Explain Reset Find Options

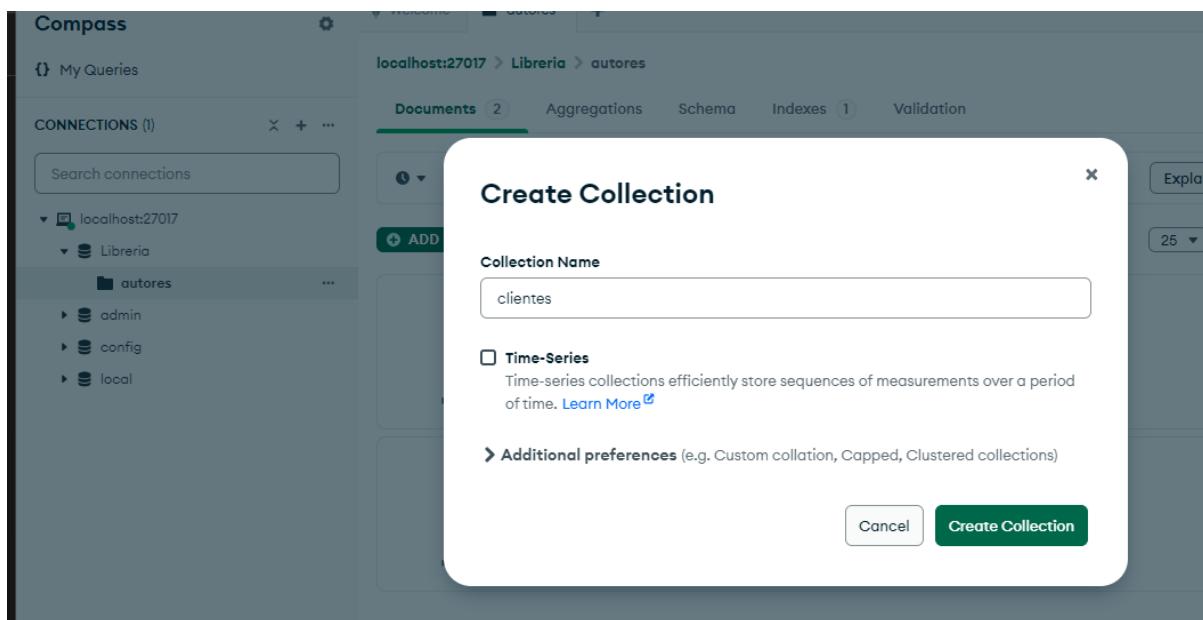
ADD DATA EXPORT DATA UPDATE DELETE

_id: ObjectId('67113d299b74dfcd79f90f48')
autor_id: 1
nombre: "Gabriel García Márquez"
nacionalidad: "Colombiana"
fecha_nacimiento: "1927-03-06"
libros: Array (1)

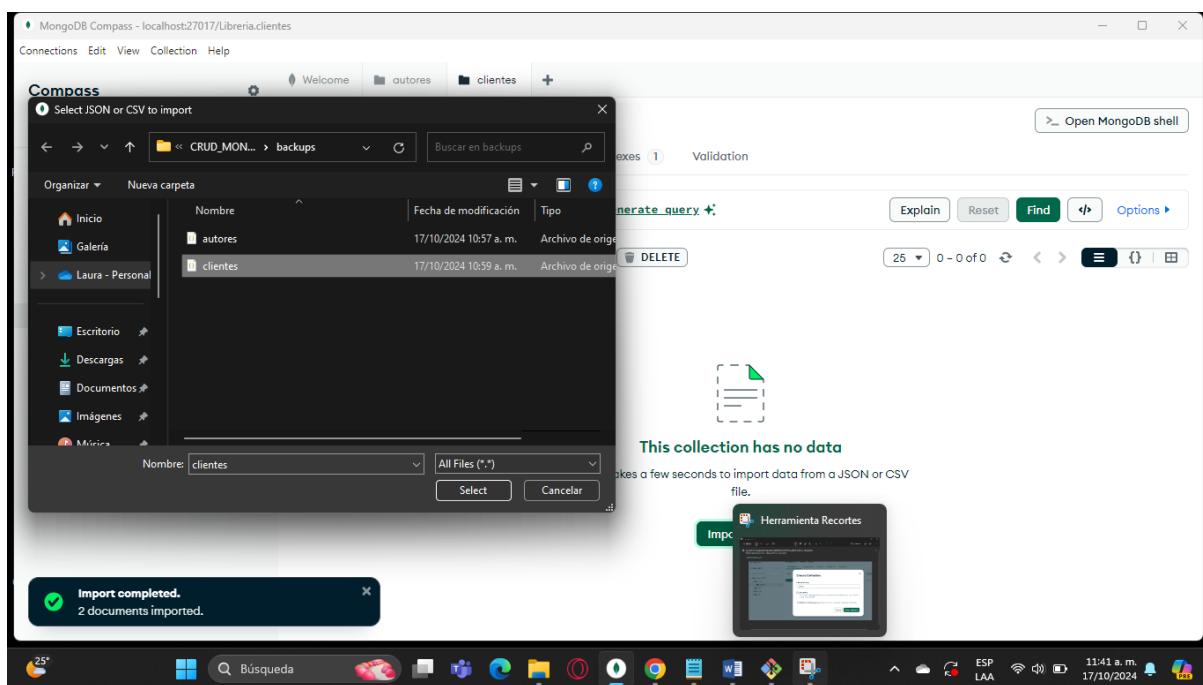
_id: ObjectId('67113d299b74dfcd79f90f49')
autor_id: 2
nombre: "J.K. Rowling"
nacionalidad: "Británica"
fecha_nacimiento: "1965-07-31"
libros: Array (1)

Import completed.
2 documents imported.

Importación correcta del JSON en la collection autores.



Creación de la collection clientes.



Importación del JSON a la collection clients.

MongoDB Compass - localhost:27017/Libreria.clientes

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (1)

localhost:27017 Libreria clientes

localhost:27017 > Libreria > clientes

Documents 2 Aggregations Schema Indexes 1 Validation

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

[EXPLAIN](#) [RESET](#) [Find](#) [Options](#)

[ADD DATA](#) [EXPORT DATA](#) [UPDATE](#) [DELETE](#)

1 - 2 of 2

```
_id: ObjectId('67113ecf9b74dfcd79f90f4b')
cliente_id: 1
nombre: "Juan Pérez"
email: "juanperez@example.com"
telefono: "3001234567"
dirección: "Calle 123, Bogotá"
prestamos: Array (1)

_id: ObjectId('67113ecf9b74dfcd79f90f4c')
cliente_id: 2
nombre: "Ana Gómez"
email: "anagomez@example.com"
telefono: "3107654321"
dirección: "Avenida 456, Medellín"
prestamos: Array (1)
```

Búsquedas 11:44 a.m. 17/10/2024

Correcta importación del JSON en la collection clientes.

MongoDB Compass - localhost:27017/Libreria.clientes

Connections Edit View Collection Help

Compass

My Queries

CONNECTIONS (1)

localhost:27017 Libreria clientes

localhost:27017 > Libreria > clientes

Documents 2 Aggregations Schema Indexes 1 Validation

[EXPLAIN](#) [RESET](#) [Find](#) [Options](#)

[CREATE](#)

Create Collection

Collection Name: libros

Time-Series

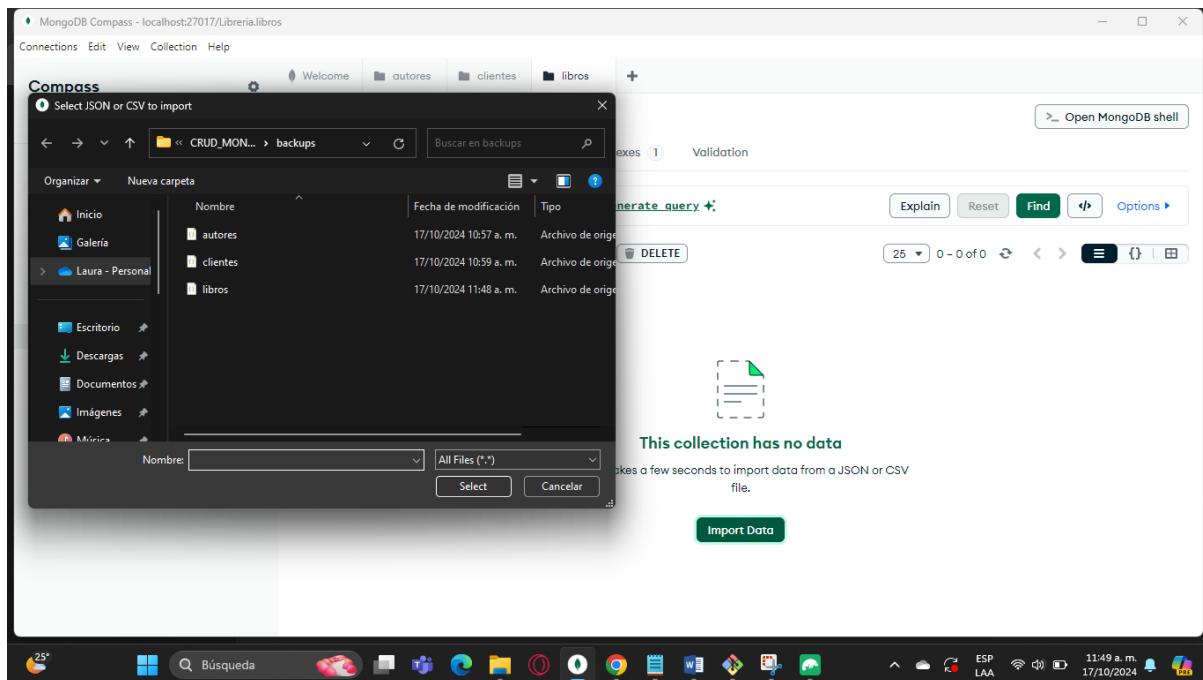
Time-series collections efficiently store sequences of measurements over a period of time. [Learn More](#)

[Additional preferences](#) (e.g. Custom collation, Capped, Clustered collections)

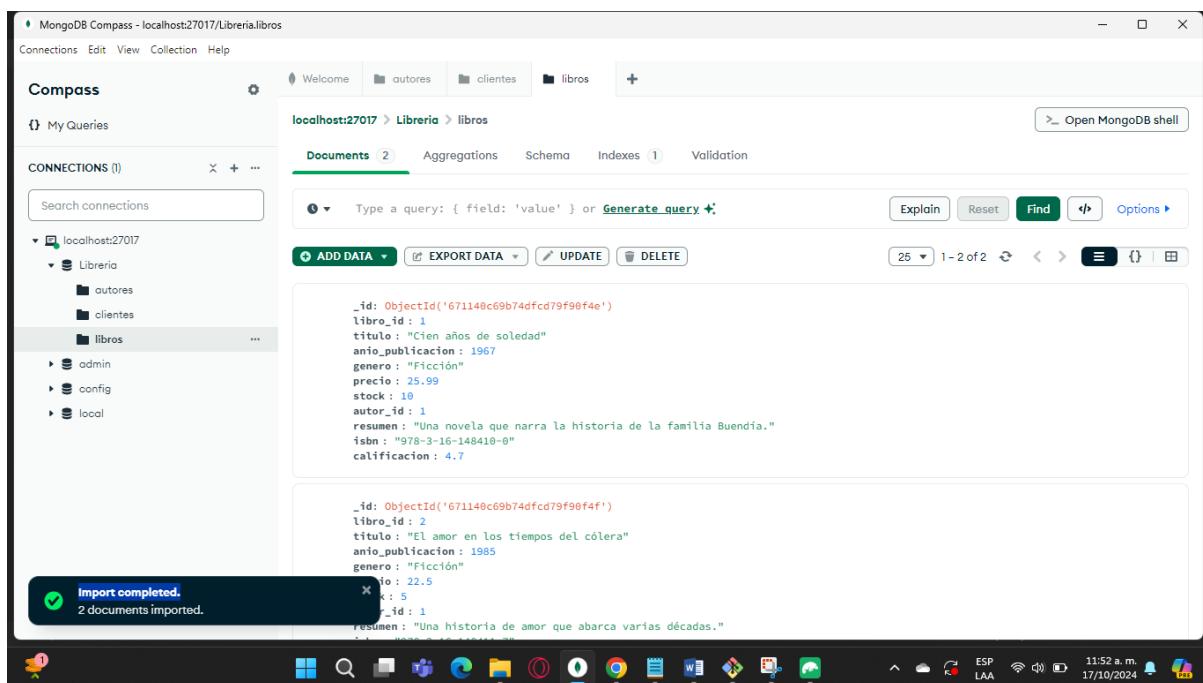
[Cancel](#) [Create Collection](#)

Búsquedas 11:49 a.m. 17/10/2024

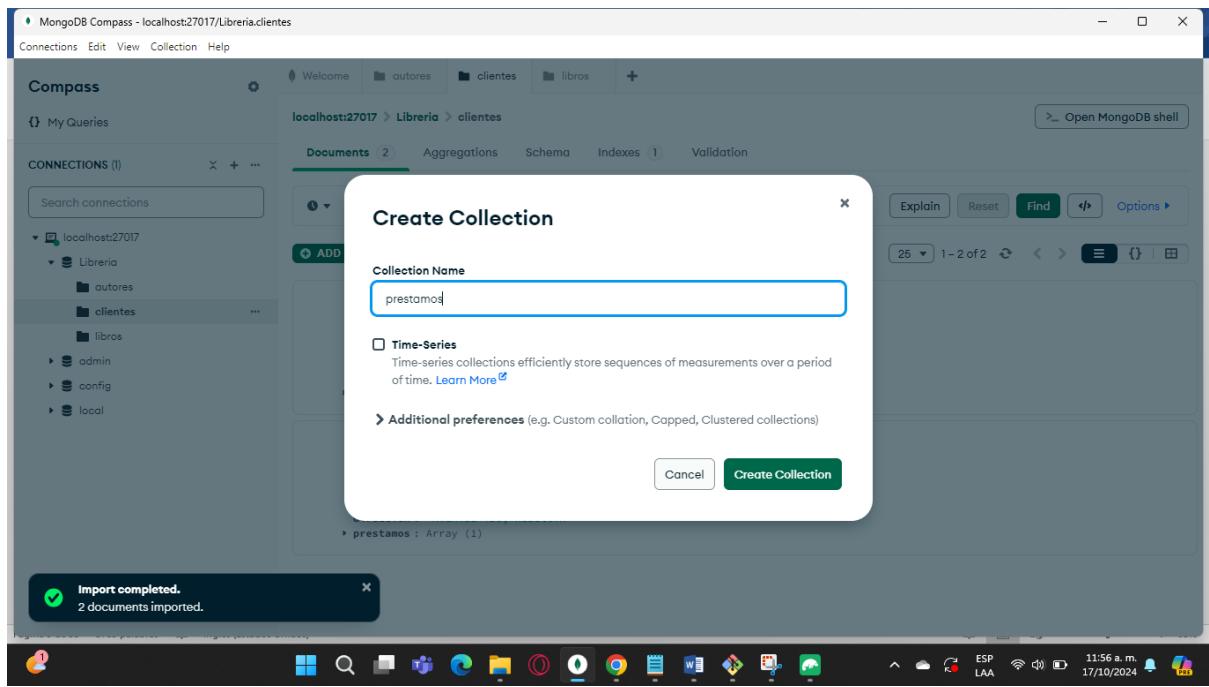
Creación de la collection libros.



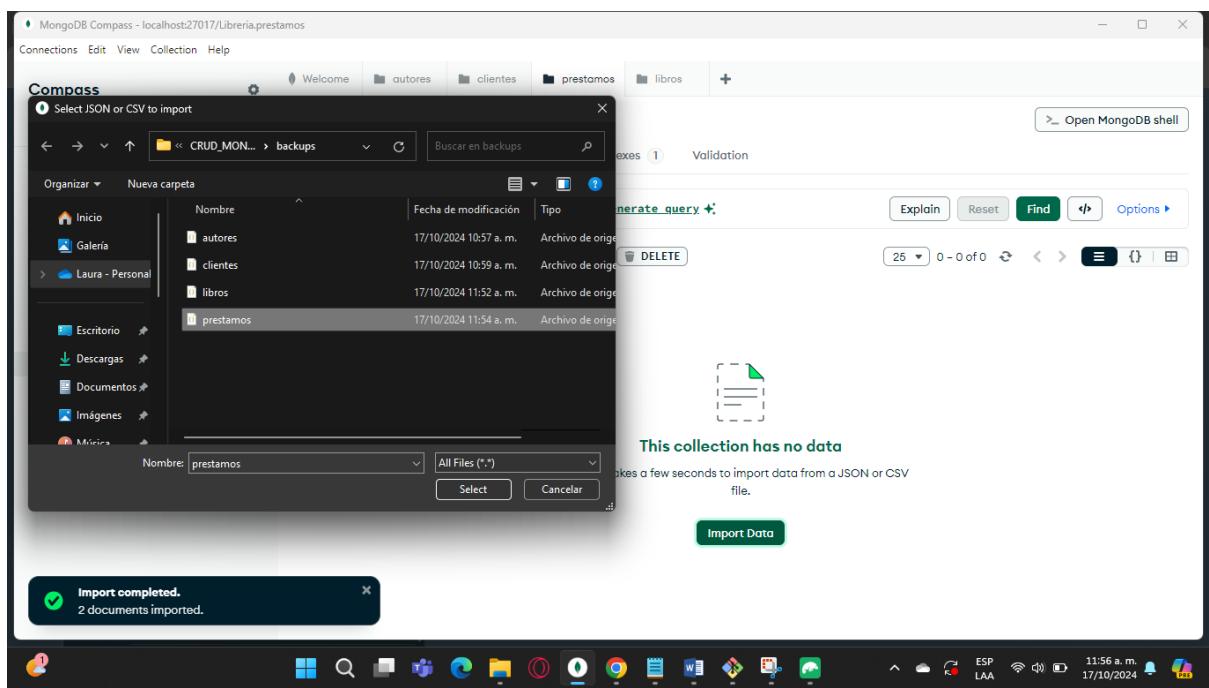
Importación del JSON en la collection libros.



Correcta importación del JSON en la collection libros.



Creación de la collection prestamos.



Importación del JSON en la collection prestamos.

Diagramas de las collectiones de la base de datos personal:

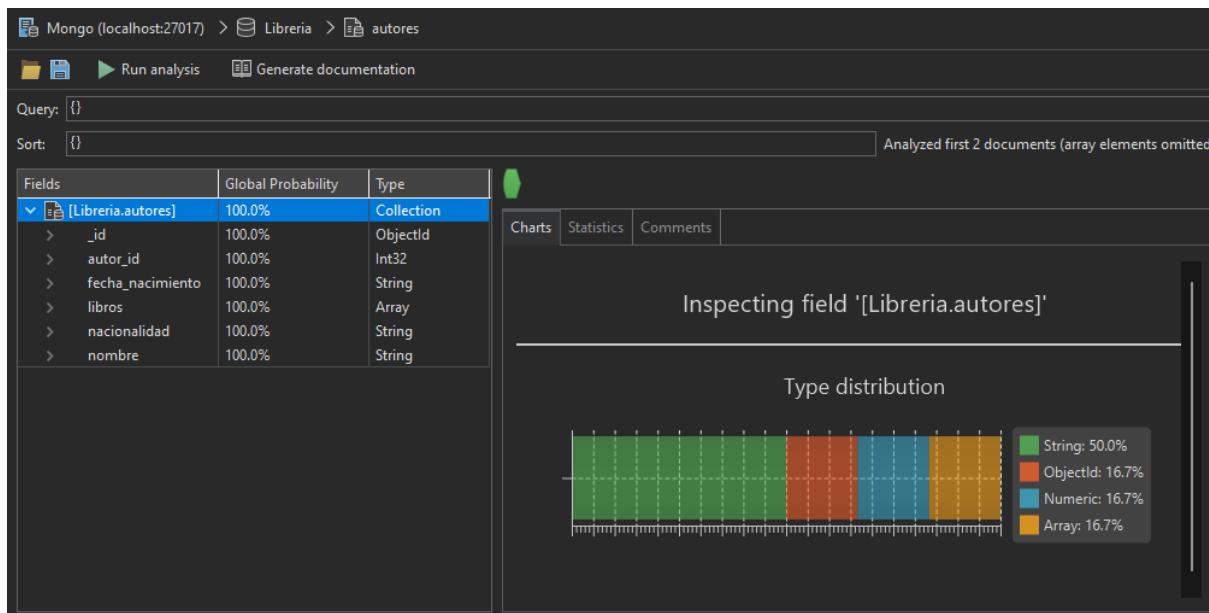


Diagrama de la collection autores.

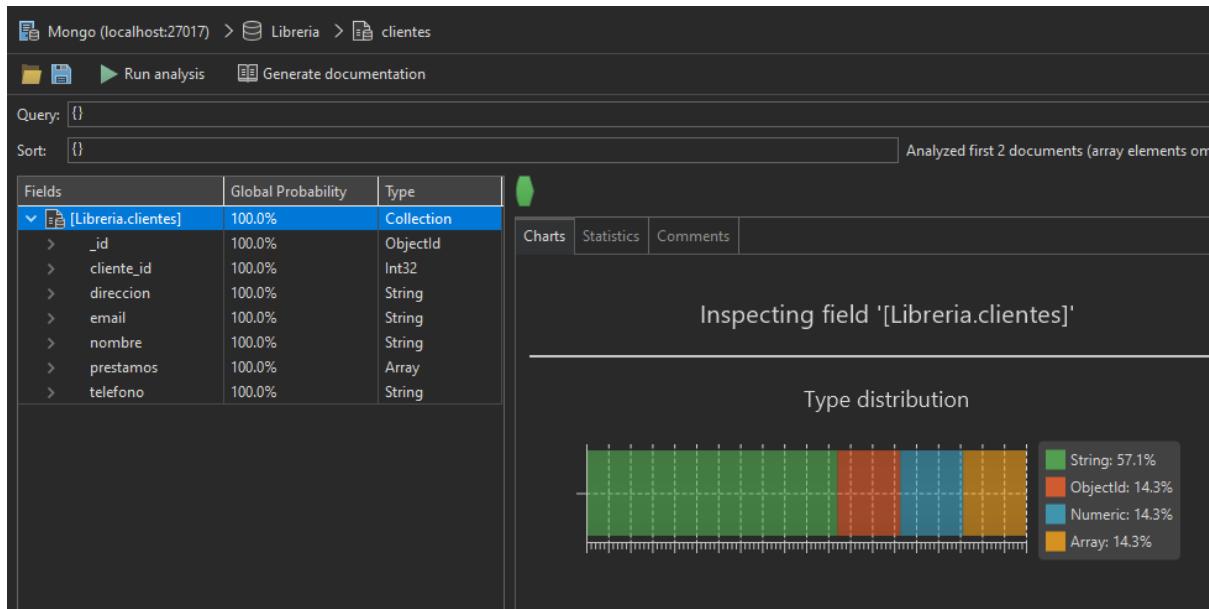


Diagrama de la collection clientes.

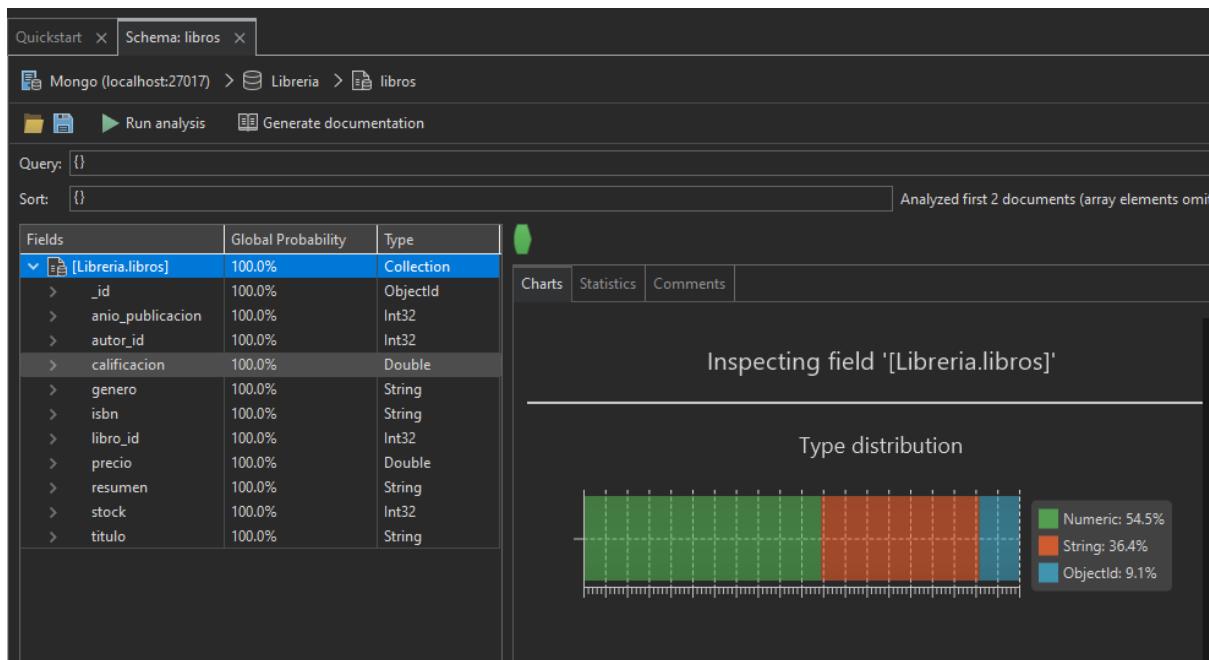


Diagrama de la collection libros.

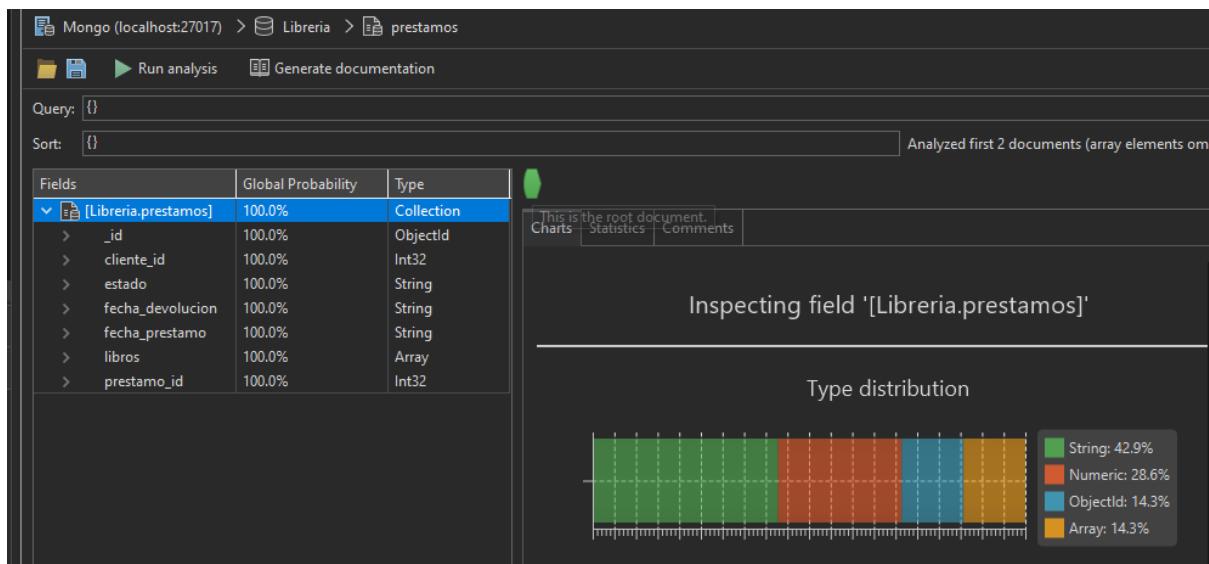
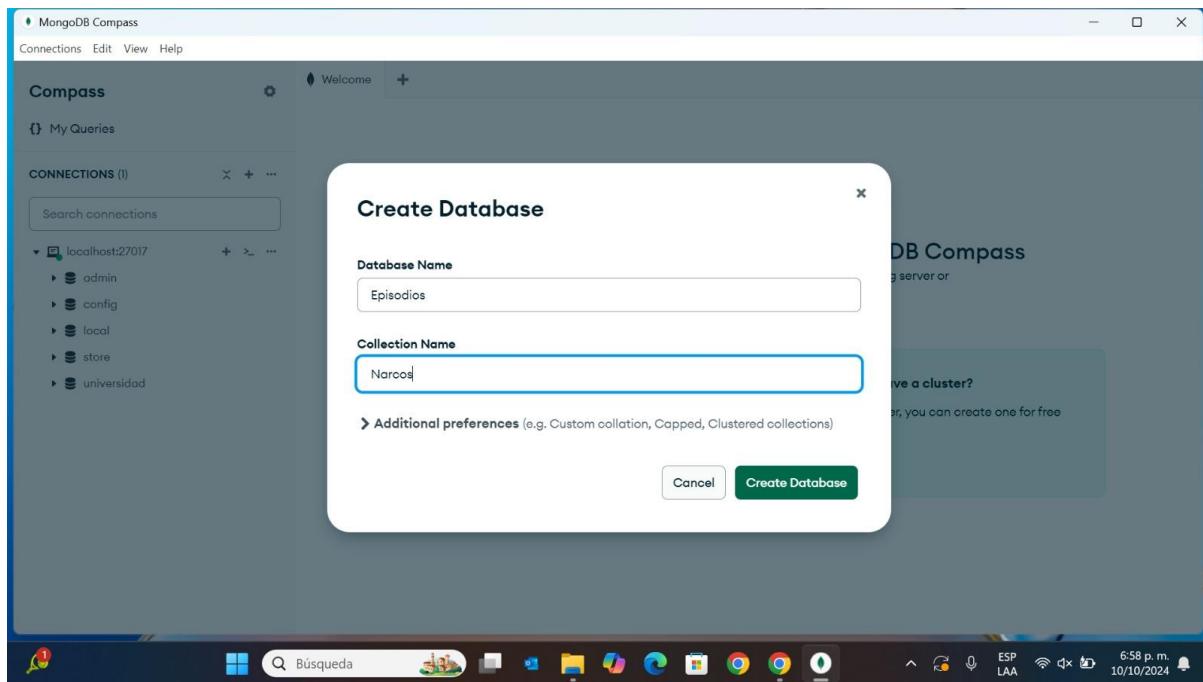


Diagrama de la collection prestamos.

1.1.2 Realización taller

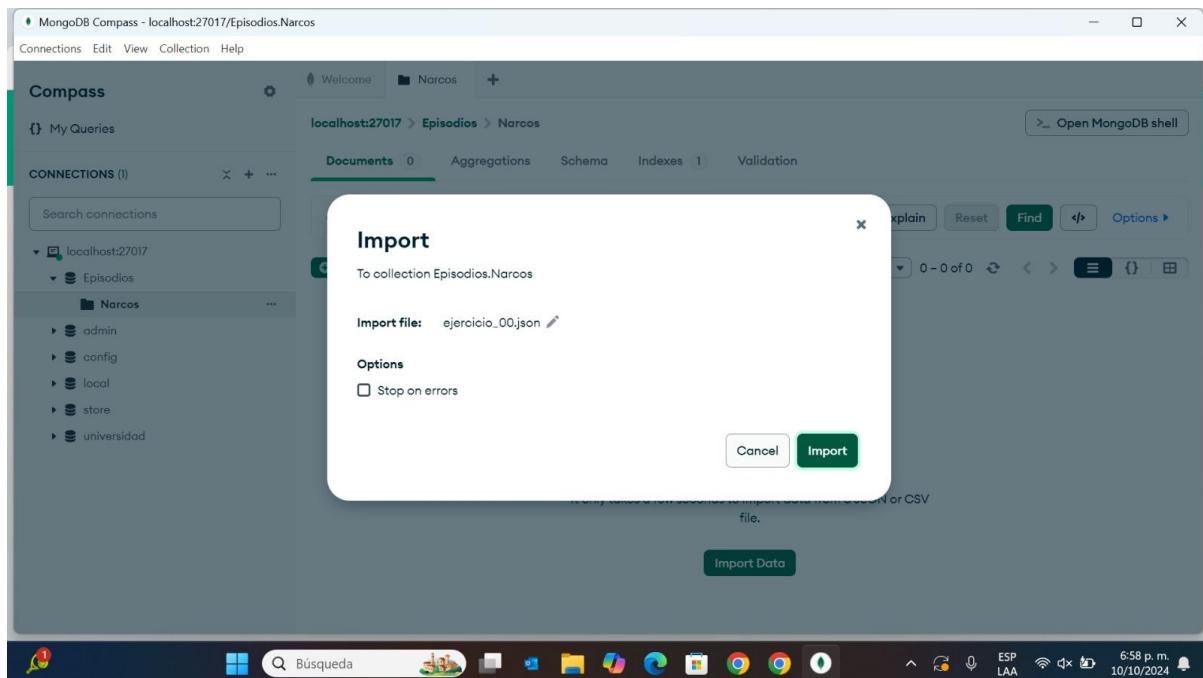
- 1) Carga los datos del fichero “ejercicio_00.json” en la base de datos llamada Episodios, sobre la colección Narcos.



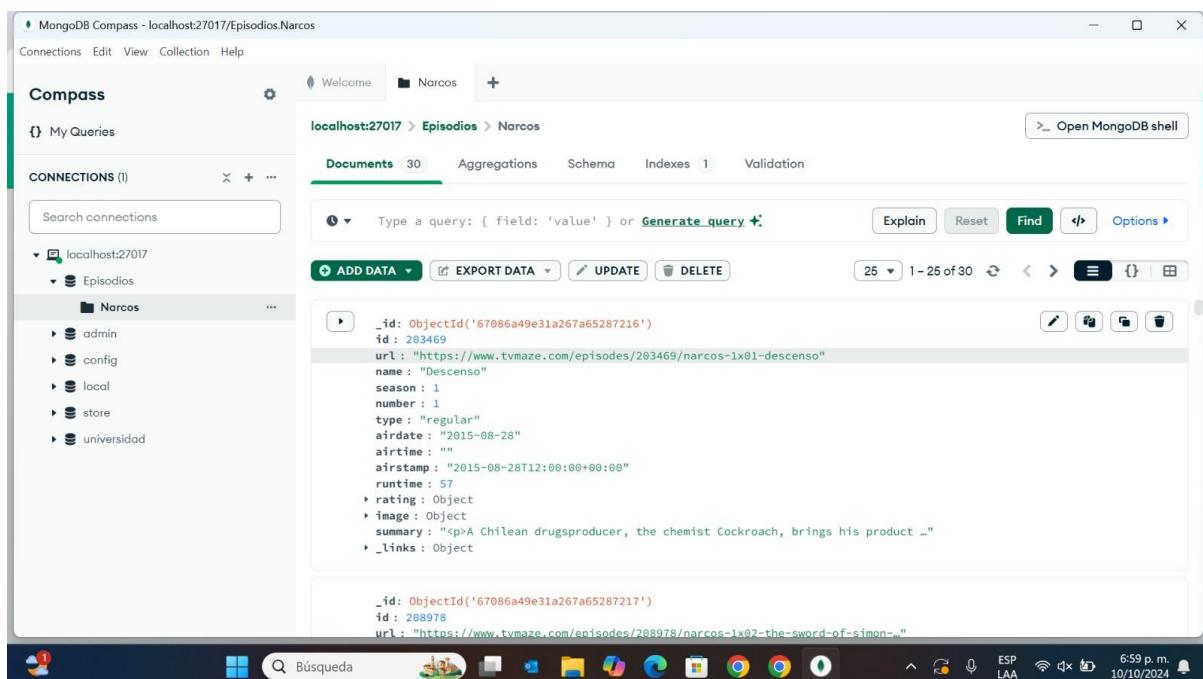
Para iniciar creamos la base de datos llamada Episodios con la collection Narcos.

The screenshot shows the MongoDB Compass interface with a file selection dialog open. The dialog is titled 'Select JSON or CSV to import' and shows a list of files from 'OneDrive - Perso'. One file, 'ejercicio_00', is selected. The main Compass window shows the 'Narcos' collection under the 'Episodios' database. It displays 1 index and 0 documents. A message at the bottom states 'This collection has no data'.

Añadimos el fichero sobre la collection.



Nos aparecerá esta ventana después de seleccionar el fichero y damos import.



Finalmente después de haber cargado el fichero correctamente, obtendremos esto.

- 2) Visualiza el contenido del fichero “ejercicio_01.json” con la herramienta online <https://jsongrid.com/json-viewer>. Observa la estructura del documento y recupera solo la lista de episodios que contienen dicho fichero.

The screenshot shows the JSONGrid.com/json-viewer interface. On the left, there's a code editor with line numbers (51 to 81) showing a portion of a JSON document. On the right, there's a grid view with columns like 'GRID' and 'JSON'. The JSON grid shows fields such as 'ended', 'officialSite', 'schedule', 'rating', 'weight', 'network', 'webChannel', 'dvdCountry', 'externals', 'image', 'summary', 'updated', '_links', and '_embedded'. The '_embedded' row contains a sub-grid for 'episodes'. One specific episode object is highlighted with a yellow background in both the code editor and the grid view.

Con la lista de episodios crea otro fichero JSON llamado “episodios_westworld.json”.

episodios_Westword 10/10/2024 7:20 p. m. Archivo de origen JS... 33 KB

Usando MongoDB Compass carga dicho fichero en la colección “Westworld” dentro de la base de datos Episodios.

The screenshot shows the MongoDB Compass interface. It's connected to a database named 'Westworld' and a collection named 'Episodios'. The document count is 36. At the top, there are buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. Below the table, there are buttons for 'Find', 'Options', and navigation arrows. The table itself shows a list of documents, with one document expanded to show its detailed fields: _id, id, url, name, season, number, type, airdate, airtime, airstamp, runtime, rating, image, summary, and links.

3) Repite el paso 2 con el fichero “ejercicio_02.json” pero esta vez la colección debe llamarse “RickiLake”.

JSONGRID All-in-One Json Grid Json Validator Json Parser Request Feature Contact Us Save & Share Sign up Search Expand All Collapse All

GRID

ended	2004-05-26
officialSite	null
schedule	[+] schedule {}
rating	[+] rating {}
weight	66
network	[+] network {}
webChannel	null
dvdCountry	null
externals	[+] externals {}
image	[+] image {}
summary	<p>Ricki Lake</p> is a daytime tabloid talk show hosted by American actress Ricki Lake.
updated	1482024492
_links	[+] _links {}
_embedded	[-] _embedded {}
episodes	[+] episodes[2]

10 episodios_RickiLake 10/10/2024 8:14 p. m. Archivo de origen JS... 2 KB

RickiLake +

localhost:27017 > Episodios > RickiLake > Open MongoDB shell

Documents 2 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 - 2 of 2

```
_id: ObjectId('67087c0de31a267a6528725c')
id: 363093
url: "https://www.tvmaze.com/episodes/363093/ricki-lake-1x01-pilot"
name: "Pilot"
season: 1
number: 1
type: "regular"
airdate: "1993-09-13"
airtime: "17:00"
airstamp: "1993-09-13T21:00:00+00:00"
runtime: 60
rating: Object
image: null
summary: ""
_links: Object

_id: ObjectId('67087c0de31a267a6528725d')
id: 363094
url: "https://www.tvmaze.com/episodes/363094/ricki-lake-12x73-season-12-epis..."
```

4) Repite el paso 2 con el fichero “ejercicio_03.json” pero esta vez la colección debe llamarse “Homeland”.

Screenshot showing the JSONGRID interface for viewing and editing JSON data.

JSONGRID Tools: All-in-One, Grid, Json Formatter, Validator, Json Viewer, Json Parser, Request Feature, Feedback, Contact Us, Save & Share, Sign up.

GRID View:

Field	Value
ended	2020-04-26
officialSite	http://www.sho.com/sho/homeland/home
schedule	[+] schedule {}
rating	[+] rating {}
weight	96
network	[+] network {}
webChannel	null
dvdCountry	null
externals	[+] externals {}
image	[+] image {}
summary	<p>The winner of 6 Emmy Awards including Outstanding Drama Series, Homeland is an edg...
updated	1669208031
_links	[+] _links {}
_embedded	[+] _embedded {}
episodes	[+] episodes[96]

File Details: episodios_Homeland, 10/10/2024 8:23 p. m., Archivo de origen JS..., 87 KB

MongoDB Shell: localhost:27017 > Episodios > Homeland > Open MongoDB shell

Collection View: Documents (96), Aggregations, Schema, Indexes (1), Validation.

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

Action Buttons: ADD DATA, EXPORT DATA, UPDATE, DELETE, Explain, Reset, Find, Options.

Document Preview:

```

_id: ObjectId('67087e3ee31a267a65287263')
id: 189
url: "https://www.tvmaze.com/episodes/189/homeland-1x01-pilot"
name: "Pilot"
season: 1
number: 1
type: "regular"
airdate: "2011-10-02"
airtime: "22:00"
airstamp: "2011-10-03T02:00:00+00:00"
runtime: 60
rating: Object
image: Object
summary: "<p>In the opener of this "Manchurian Candidate"-like political thriller, a former CIA agent returns home to find his wife has been killed and his son is missing. As he begins to unravel the mystery, he uncovers a plot that reaches all the way to the highest levels of power in Washington D.C.</p>"
_links: Object

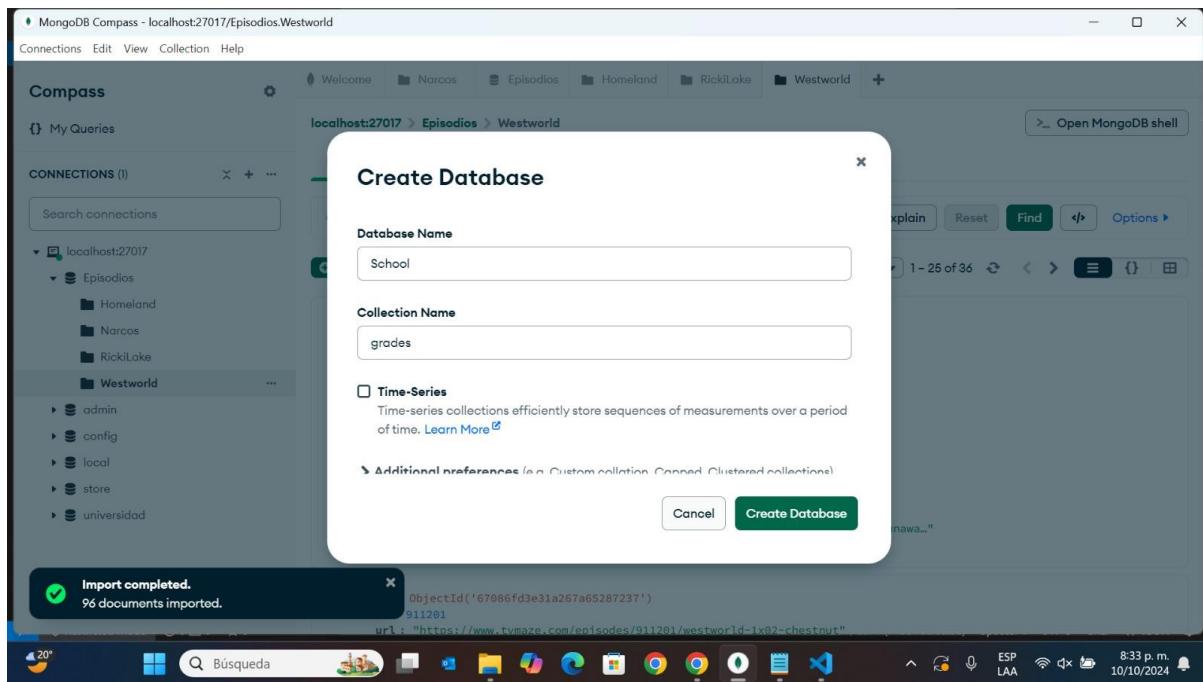
```

```

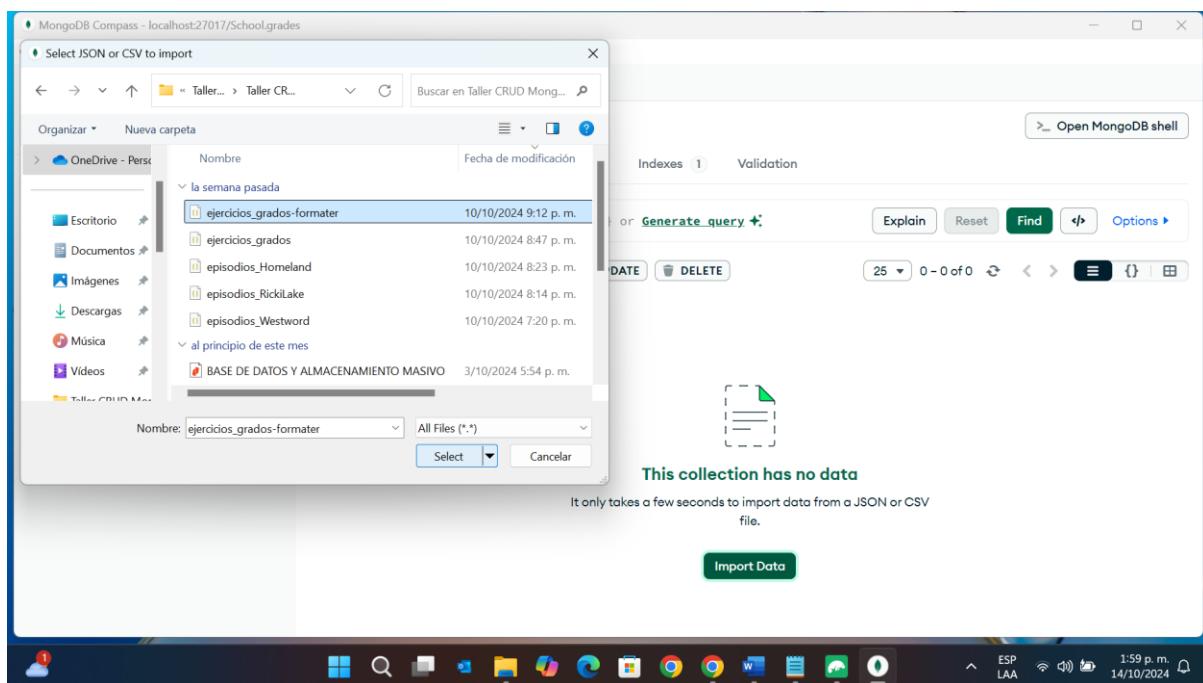
_id: ObjectId('67087e3ee31a267a65287264')
id: 190
url: "https://www.tvmaze.com/episodes/190/homeland-1x02-the-ghost"
name: "The Ghost"
season: 1
number: 2
type: "regular"
airdate: "2011-10-09"
airtime: "22:00"
airstamp: "2011-10-10T02:00:00+00:00"
runtime: 60
rating: Object
image: Object
summary: "<p>A mysterious figure known only as "The Ghost" appears to the Homeland team, revealing information about their target's past. The team must navigate through a web of lies and truths to uncover the truth behind his identity and intentions.</p>"
_links: Object

```

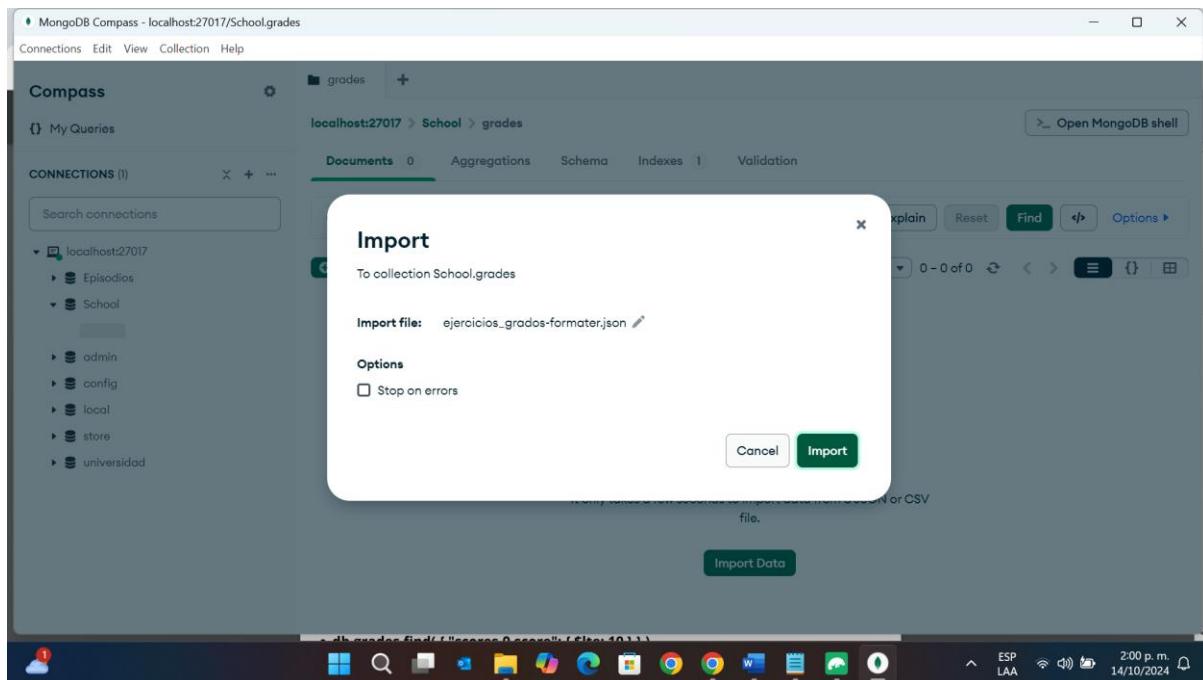
- 5) Carga los datos del fichero “ejercicio_grados.json” en la base de datos School sobre la colección grades.



Inicialmente creamos la base datos School con la collection grades.



Buscamos el fichero para cargarlo y lo seleccionamos.



En esta ventana solo damos click en Import.

The screenshot shows the 'grades' collection in MongoDB Compass after the import process. The document count is now 280. The interface includes a search bar, a query builder, and buttons for 'ADD DATA', 'EXPORT DATA', 'UPDATE', and 'DELETE'. The main pane displays several documents, each with fields like '_id', 'student_id', 'class_id', and 'scores'. The 'scores' field is shown as an array of values.

Finalmente despues de cargar el fichero, obtenemos lo siguiente.

Desarrollo del Informe

Descripción de la Base de Datos de Datos Personal:

La base de datos para la librería está diseñada para gestionar la información de **autores**, **libros**, **clientes** y **préstamos**. Cada colección se organiza para facilitar la consulta y el mantenimiento de la información relacionada, permitiendo un acceso eficiente a los datos y garantizando la integridad de las relaciones entre las entidades.

Diseño de la base de datos:

Modelo de datos: Normalización y cardinalidad

Autores:

- **Campos:** autor_id, nombre, nacionalidad, fecha_nacimiento, libros.
- **Cardinalidad:** Un autor puede tener **uno o más libros** (relación 1 a muchos) embebidos dentro de su documento.

Libros:

- **Campos:** libro_id, titulo, anio_publicacion, genero, precio, stock, resumen, isbn, calificacion.
- **Cardinalidad:** Un libro puede ser escrito por **un solo autor** (relación 1 a 1).

Clientes:

- **Campos:** cliente_id, nombre, email, telefono, direccion, prestamos.
- **Cardinalidad:** Un cliente puede tener **uno o más préstamos** (relación 1 a muchos) embebidos dentro de su documento.

Prestamos:

- **Campos:** prestamo_id, fecha_prestamo, fecha_devolucion, libros (embebiendo la información de los libros prestados).
- **Cardinalidad:** Un préstamo puede incluir **uno o más libros** (relación muchos a muchos, ya que un libro puede ser prestado a diferentes clientes en diferentes momentos).

Relaciones entre Colecciones

Autores y Libros:

- Relación 1 a muchos (1): Un autor puede tener varios libros, por lo que los libros se embeben dentro de cada documento de autor.

Ejemplo: Gabriel García Márquez puede tener varios libros como "Cien años de soledad" y "El amor en los tiempos del cólera".

Clients y Prestamos:

- Relación 1 a muchos (1): Un cliente puede tener múltiples préstamos a lo largo del tiempo, lo que se refleja al embeber los préstamos dentro de cada documento de cliente.

Ejemplo: Juan Pérez puede tener varios préstamos como "Cien años de soledad" y "Harry Potter y la piedra filosofal".

Préstamos y Libros:

- Relación muchos a muchos (M): Un préstamo puede incluir varios libros, y un libro puede ser prestado a diferentes clientes en diferentes momentos. Para representar esto, los libros se embeben dentro de cada objeto de préstamo.

Consideraciones de Diseño

- **Normalización:** Aunque los libros están embebidos en las colecciones de autores y préstamos, se mantiene un esquema de normalización donde cada libro se puede identificar de forma única por libro_id, lo que evita duplicaciones innecesarias.
- **Embebido:** La decisión de embeber libros en préstamos y autores en libros ayuda a reducir la cantidad de consultas necesarias y mejora la eficiencia al acceder a datos relacionados.

Colección de Clientes con Préstamos Embebidos

En esta colección, cada cliente tiene un campo llamado prestamos, que es un arreglo de objetos. Cada objeto representa un préstamo realizado por el cliente.

Explicación:

- En el documento del cliente Juan Pérez, se embeben los detalles de sus préstamos en el campo prestamos.
- Cada préstamo tiene un identificador único (prestamo_id), fechas y un arreglo de libros prestados, lo que permite acceder a toda la información relacionada con un solo cliente en una sola consulta.

Colección de Autores con Libros Embebidos

De manera similar, en la colección de autores, los libros se embeben dentro del documento del autor:

Explicación:

- En el documento de Gabriel García Márquez, se embeben los libros que ha escrito en el campo libros.
- Cada libro contiene su propio identificador (libro_id), título, año de publicación y género.

Campos Comunes: Se han definido campos comunes que permiten un acceso fácil y efectivo a la información. Por ejemplo, los libros tienen campos como titulo, genero, precio, que son relevantes tanto para la colección de autores como para la de préstamos.

Métodos de captura:

Creación de Archivos JSON

Los archivos JSON son utilizados para almacenar datos de manera estructurada. En el caso de la base de datos de la librería, necesitarás crear varios archivos JSON para las diferentes colecciones: autores, libros, clientes y préstamos.

Nombre	Fecha de modificación	Tipo	Tamaño
autores	17/10/2024 12:03 p. m.	Archivo de origen ...	2 KB
clientes	17/10/2024 12:07 p. m.	Archivo de origen ...	2 KB
libros	17/10/2024 11:52 a. m.	Archivo de origen ...	1 KB
prestamos	17/10/2024 11:54 a. m.	Archivo de origen ...	1 KB

Cargar los Archivos JSON en MongoDB

Para cargar los datos en MongoDB, se utiliza la consola de MongoDB o un script que permita la ejecución de comandos JavaScript. Aquí se detalla cómo hacerlo:

```
// Conectar a la base de datos
use libreria;

// Crear colecciones e insertar documentos desde archivos JSON
db.createCollection("autores");
db.autores.insertMany(JSON.parse(cat("ruta/a/autores.json")));

db.createCollection("clientes");
db.clientes.insertMany(JSON.parse(cat("ruta/a/clientes.json")));
```

Validación de la Inserción de Datos

Después de ejecutar los comandos, se puede validar que los documentos se hayan insertado correctamente con consultas simples:

```
// Validar los autores
db.autores.find().pretty();

// Validar los clientes
db.clientes.find().pretty();
```

Consultas:

Trabaja con READ

1. Visualiza todas las bases de datos existentes: show dbs
2. Localiza la base de datos School, ejecuta las siguientes consultas y describe qué hace cada una de ellas:

db.grades.find({ student_id: 2 })

The screenshot shows the MongoDB Compass interface. In the top-left corner, it says "localhost:27017 > School". Below that is a toolbar with icons for New, Load script, Save script, Script history, and Enable. The script editor contains the following code:

```

1 use School;
2
3 db.grades.find( { student_id: 2 } )

```

Below the editor is a "Raw shell output" tab and a "Find Query (line 3)" tab. The "Find Query" tab shows the results of the query. The results table has the following columns: _id, student_id, class_id, and scores. There are three documents listed, each with a different _id but the same student_id (2) and class_id (either 25, 27, or 24). Each document's scores array has either 5, 4, or 4 elements respectively.

_id	student_id	class_id	scores
<code>50b59cd75bed7...</code>	2	25	[5 elements]
<code>50b59cd75bed7...</code>	2	27	[4 elements]
<code>50b59cd75bed7...</code>	2	24	[4 elements]

Explicación:

La consulta busca todos los documentos en la colección grades donde el campo student_id sea igual a 2. Si hay registros de ese estudiante en la base de datos, MongoDB devolverá todos esos documentos. Si no hay coincidencias, devolverá un conjunto vacío.

Resultado:

Esta consulta retornará una lista de documentos que contienen las calificaciones (como exam, quiz, homework) de todos los cursos en los que el estudiante con student_id = 2 está inscrito.

```
db.grades.find( { "scores.0.score": { $lte: 10 } } )
```

The screenshot shows the MongoDB Compass interface. At the top, the address bar says "localhost:27017 > School". Below it is a toolbar with icons for running scripts, creating new ones, loading scripts, saving scripts, and viewing history. A button for "Enable Query Profiling" is also present. The main area contains a code editor with the following MongoDB query:

```

1 use School;
2
3 db.grades.find( { "scores.0.score": { $lte: 10 } } )
4

```

Below the code editor is a "Raw shell output" tab and a "Find Query (line 3)" tab. The "Find Query" tab shows the results of the query. The results table has columns: _id, student_id, class_id, and scores. The data is as follows:

_id	student_id	class_id	scores
_id 50b59cd75bed7...	student_id 0	class_id 24	[scores [4 elements]]
_id 50b59cd75bed7...	student_id 2	class_id 25	[scores [5 elements]]
_id 50b59cd75bed7...	student_id 3	class_id 13	[scores [6 elements]]
_id 50b59cd75bed7...	student_id 3	class_id 16	[scores [3 elements]]
_id 50b59cd75bed7...	student_id 4	class_id 5	[scores [5 elements]]
_id 50b59cd75bed7...	student_id 4	class_id 12	[scores [4 elements]]
_id 50b59cd75bed7...	student_id 6	class_id 22	[scores [4 elements]]
_id 50b59cd75bed7...	student_id 8	class_id 29	[scores [3 elements]]

Explicación:

Esta consulta devuelve todos los documentos de la colección grades donde el primer score dentro del arreglo scores (es decir, el primer puntaje registrado, generalmente de un examen o quiz) es menor o igual a 10.

Resultado:

Si algún estudiante tiene un puntaje en el primer elemento de su arreglo scores que sea menor o igual a 10, esos documentos serán devueltos. Si no existe ningún puntaje que cumpla con esa condición, la consulta no devolverá nada.

db.grades.find({ "scores.4.score": { \$lte: 10 } })

The screenshot shows the IntelliShell interface for MongoDB. The top bar indicates the connection is to 'localhost:27017' and the database is 'School'. The script editor contains the following MongoDB query:

```

1 use School;
2
3 db.grades.find( { "scores.4.score": { $lte: 10 } } )
4

```

The result pane, titled 'grades > student_id', displays a table of documents from the 'grades' collection. The table has columns: '_id', 'student_id', 'class_id', and 'scores'. The 'scores' column shows arrays of scores for each student. The document where the fifth score is 10 is highlighted.

_id	student_id	class_id	scores
<code>50b59cd75bed7...</code>	0	5	[6 elements]
<code>50b59cd75bed7...</code>	13	22	[6 elements]
<code>50b59cd75bed7...</code>	15	21	[5 elements]
<code>50b59cd75bed7...</code>	15	22	[5 elements]
<code>50b59cd75bed7...</code>	17	19	[5 elements]
<code>50b59cd75bed7...</code>	19	22	[5 elements]
<code>50b59cd75bed7...</code>	24	22	[6 elements]
<code>50b59cd75bed7...</code>	30	0	[6 elements]
<code>50b59cd75bed7...</code>	17	11	[5 elements]

Explicación:

Esta consulta devuelve todos los documentos en la colección grades donde la calificación del quinto puntaje registrado en el arreglo scores es menor o igual a 10.

Resultado:

Si un estudiante tiene al menos 5 calificaciones (es decir, tiene un quinto puntaje dentro del arreglo scores), la consulta buscará si esa calificación es menor o igual a 10.

Si el quinto puntaje es menor o igual a 10, el documento será devuelto.

`db.grades.find({ "scores.5.score": { $lte: 10 } })`

The screenshot shows the IntelliJShell interface for MongoDB. The top bar displays "IntelliShell: Base de datos y almacenamiento masivo*". Below it, the connection path is "localhost:27017 > School". The toolbar includes buttons for running scripts, loading scripts, saving scripts, and enabling/disabling the shell.

```

1 use School;
2
3 db.grades.find( { "scores.5.score": { $lte: 10 } } )
4

```

The "Raw shell output" panel shows the results of the query:

_id	student_id	class_id	scores
<code>id 50b59cd75bed7...</code>	6	8	[6 elements]
<code>id 50b59cd75bed7...</code>	12	4	[6 elements]
<code>id 50b59cd75bed7...</code>	35	18	[6 elements]
<code>id 50b59cd75bed7...</code>	36	23	[6 elements]
<code>id 50b59cd75bed7...</code>	41	18	[6 elements]

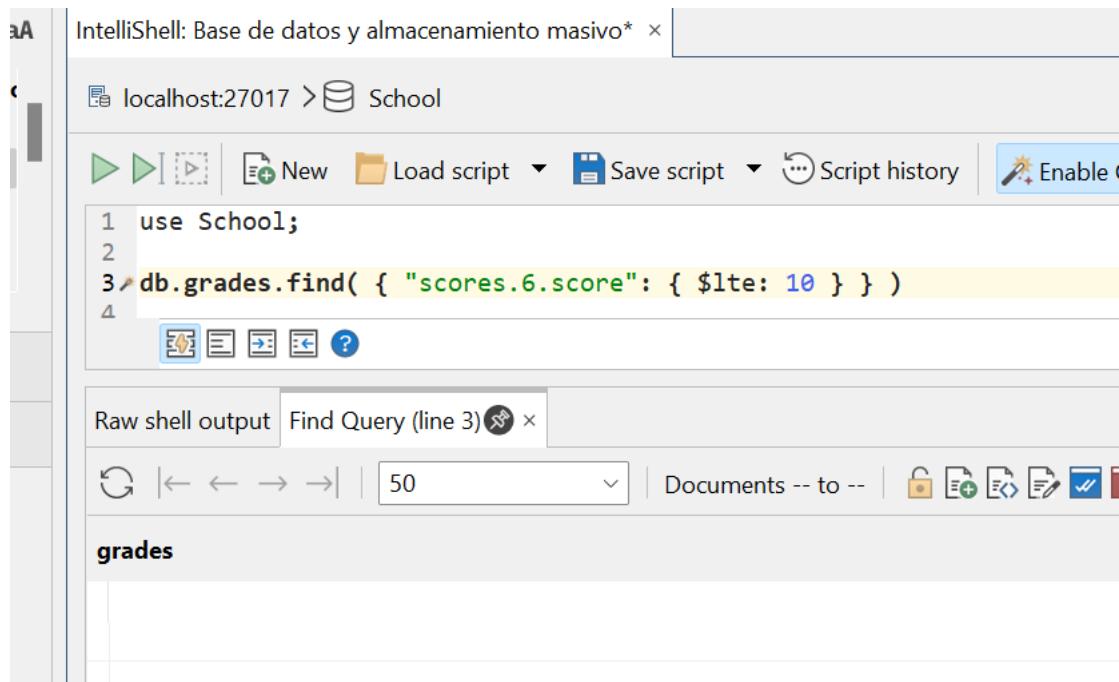
Explicación:

Esta consulta busca todos los documentos en la colección grades donde el sexto puntaje dentro del arreglo scores sea menor o igual a 10.

Resultado:

Si un estudiante tiene al menos 6 calificaciones (es decir, tiene un sexto puntaje en el arreglo scores), MongoDB evaluará si ese puntaje es menor o igual a 10. Si el puntaje en la sexta posición es menor o igual a 10, el documento será devuelto. Si un estudiante no tiene al menos 6 elementos en el arreglo scores (es decir, si no tiene un sexto puntaje), el documento no será devuelto.

`db.grades.find({ "scores.6.score": { $lte: 10 } })`



The screenshot shows the IntelliShell interface for MongoDB. The title bar says "IntelliShell: Base de datos y almacenamiento masivo*". The connection is set to "localhost:27017 > School". The script editor contains the following code:

```

1 use School;
2
3 db.grades.find( { "scores.6.score": { $lte: 10 } } )
4

```

The third line, `db.grades.find({ "scores.6.score": { \$lte: 10 } })`, is highlighted in yellow. Below the editor is a toolbar with icons for file operations like New, Load script, Save script, and Script history. The "Enable" button is also visible. The results pane at the bottom shows the collection name "grades".

Explicación:

Esta consulta busca todos los documentos en la colección grades donde el séptimo puntaje dentro del arreglo scores sea menor o igual a 10.

Resultado:

Si un estudiante tiene al menos 7 calificaciones (es decir, un séptimo puntaje en el arreglo scores), la consulta buscará si ese puntaje es menor o igual a 10. Si el séptimo puntaje es menor o igual a 10, el documento será devuelto. Si el estudiante no tiene un séptimo puntaje (menos de 7 elementos en el arreglo scores), el documento no será devuelto.

`db.grades.find({ "scores.0.score": { $gte: 60, $lte: 61 } })`

The screenshot shows the IntelliShell interface for MongoDB. The top bar displays "IntelliShell: Base de datos y almacenamiento masivo*". Below it, the connection path is "localhost:27017 > School". The toolbar includes standard shell controls like play, stop, and step, along with "New", "Load script", "Save script", "Script history", and "Enable" buttons.

```

1 use School;
2
3 db.grades.find( { "scores.0.score": { $gte: 60, $lte: 61 } } )
4

```

The results pane shows the output of the query. It has tabs for "Raw shell output" and "Find Query (line 3)". The "Find Query" tab displays the results of the find operation. The results table has columns: _id, student_id, class_id, and scores. The first row is selected, showing student_id 0, class_id 27, and a scores array with 3 elements. The other three rows show student_ids 4, 25, and 30 respectively, with their corresponding class_ids and score arrays.

_id	student_id	class_id	scores
<code>id 50b59cd75bed7...</code>	0	27	[3 elements]
<code>id 50b59cd75bed7...</code>	4	8	[5 elements]
<code>id 50b59cd75bed7...</code>	25	0	[4 elements]
<code>id 50b59cd75bed7...</code>	30	29	[4 elements]

Explicación:

La consulta busca todos los documentos de la colección grades donde el primer puntaje dentro del arreglo scores esté entre 60 y 61 (incluyendo ambos valores).

Resultado:

Si un estudiante tiene un puntaje en el primer elemento de su arreglo scores que esté entre 60 y 61 (inclusive), entonces el documento será devuelto. Si el primer puntaje en el arreglo scores no está dentro de este rango, el documento no será devuelto.

```
db.grades.find( { "scores.0.score": { $gte: 60, $lte: 61 } } ).sort( { student_id: 1 } )
```

The screenshot shows the IntelliShell interface for MongoDB. The top bar displays the title "IntelliShell: Base de datos y almacenamiento masivo*" and the connection information "localhost:27017 > School". Below the title is a toolbar with various icons for navigation and scripting. The main area contains a code editor with the following MongoDB query:

```

1 use School;
2
3 db.grades.find( { "scores.0.score": { $gte: 60, $lte: 61 } } ).sort( { student_id: 1 } )
4

```

The code editor has a yellow highlight on the third line of the query. Below the code editor is a "Raw shell output" tab with the heading "grades > student_id". The output table has four columns: "_id", "student_id", "class_id", and "scores". There are four rows of data:

_id	student_id	class_id	scores
50b59cd75bed7...	0	27	[3 elements]
50b59cd75bed7...	4	8	[5 elements]
50b59cd75bed7...	25	0	[4 elements]
50b59cd75bed7...	30	29	[4 elements]

Explicación:

Primero, busca todos los documentos en la colección grades donde el primer puntaje dentro del arreglo scores esté entre 60 y 61. Luego, los resultados obtenidos son ordenados de forma ascendente por el campo student_id, para que los registros se muestren en orden creciente de student_id.

Resultado:

Los documentos devueltos serán aquellos que cumplan con el filtro del puntaje en el rango de 60 a 61, y estarán ordenados por student_id de menor a mayor.

`db.grades.find({ student_id: 2, class_id: 24 })`

The screenshot shows the IntelliShell interface for MongoDB. The title bar says "IntelliShell: Base de datos y almacenamiento masivo*". The connection is set to "localhost:27017" and the database is "School". The script history shows the following command:

```
1 use School;
2
3 db.grades.find( { student_id: 2, class_id: 24 } )
```

Below the command line, there are several icons for file operations (New, Load script, Save script, Script history, Enable) and document viewing (View, Edit, Find, Help). The "Raw shell output" tab is selected, showing the results of the query:

grades > student_id

_id	student_id	class_id	scores
50b59cd75bed7...	2	24	[4 elements]

Explicación:

Esta consulta busca todos los documentos en la colección grades donde: student_id sea igual a 2, y class_id sea igual a 24.

Resultado:

La consulta devolverá todos los documentos de las calificaciones del estudiante con student_id: 2 en la clase con class_id: 24. Si no hay ningún documento que coincida con ambos criterios, no se devolverá ningún resultado.

```
db.grades.find( { class_id: 20, $and: [ {"scores.0.score": { $gte: 15 } }, {"scores.0.score": { $lte: 30 } } ] })
```

The screenshot shows the IntelliShell interface for MongoDB. The title bar says "IntelliShell: Base de datos y almacenamiento masivo*". The connection path is "localhost:27017 > School". The toolbar includes buttons for running, saving, and enabling Query Assist. The code pane contains the following MongoDB query:

```

1 use School;
2
3 db.grades.find( { class_id: 20, $and: [ { "scores.0.score": { $gte: 15 } }, {
4   "scores.0.score": { $lte: 30 } } ] } )
5

```

The results pane shows the output of the query:

```

grades > student_id
_id      student_id    class_id   scores
50b59cd75bed7... 46          20        [ 5 elements ]

```

Explicación:

Esta consulta busca todos los documentos en la colección grades donde: El class_id es igual a 20, y el primer puntaje en el arreglo scores (índice 0) es mayor o igual a 15 y menor o igual a 30.

Resultado:

La consulta devolverá todos los documentos de la clase con class_id: 20 donde el primer puntaje en el arreglo scores esté dentro del rango 15 a 30. Si no hay documentos que cumplan ambos criterios, no se devolverá ningún resultado.

`db.grades.find({ scores: { $elemMatch: { type: 'quiz', score: { $gte: 50 } } } })`

The screenshot shows the IntelliShell interface for MongoDB. The top bar displays the title "IntelliShell: Base de datos y almacenamiento masivo*" and the connection information "localhost:27017 > School". Below the title is a toolbar with various icons for navigation and scripting. The main area contains a code editor with the following query:

```

1 use School;
2
3 db.grades.find( { scores: { $elemMatch: { type: 'quiz', score: { $gte: 50 } } } } )
4
5

```

The result pane below the code editor shows the output of the query. It has tabs for "Raw shell output" and "Find Query (line 3)". The "Find Query" tab is selected, showing the results of the third line of the query. The results are displayed in a table:

_id	student_id	class_id	scores
<code>50b59cd75bed7...</code>	0	28	[6 elements]
<code>50b59cd75bed7...</code>	0	5	[6 elements]
<code>50b59cd75bed7...</code>	0	7	[5 elements]
<code>50b59cd75bed7...</code>	0	27	[3 elements]
<code>50b59cd75bed7...</code>	0	10	[4 elements]
<code>50b59cd75bed7...</code>	1	18	[3 elements]
<code>50b59cd75bed7...</code>	1	28	[5 elements]
<code>50b59cd75bed7...</code>	1	13	[4 elements]

Explicación:

Esta consulta busca todos los documentos en la colección grades donde el arreglo scores contiene al menos un elemento que cumpla con ambas condiciones:

type debe ser igual a 'quiz'.

score debe ser mayor o igual a 50.

Resultado:

La consulta devolverá todos los documentos que tengan al menos un puntaje de quiz donde el score sea mayor o igual a 50. Si no hay ningún puntaje de quiz con un score mayor o igual a 50, el documento no será devuelto.

`db.grades.find({ scores: { $elemMatch: { type: 'exam', score: { $gte: 50 } } } })`

The screenshot shows the MongoDB Intellishell interface. The top bar displays "IntelliShell: Base de datos y almacenamiento masivo* x" and the connection information "localhost:27017 > School". Below the top bar is a toolbar with various icons for navigation and scripting. The main area contains a code editor with the following query:

```

1 use School;
2
3 db.grades.find( { scores: { $elemMatch: { type: 'exam', score: { $gte: 50 } } } } )
4
5

```

The code editor has a yellow background for the third line of the query. Below the code editor is a "Raw shell output" tab and a search bar. The main result area is titled "grades > student_id" and displays a table with the following data:

_id	student_id	class_id	scores
<code>id 50b59cd75bed7...</code>	<code> 0</code>	<code> 2</code>	<code>[5 elements]</code>
<code>id 50b59cd75bed7...</code>	<code> 0</code>	<code> 5</code>	<code>[6 elements]</code>
<code>id 50b59cd75bed7...</code>	<code> 0</code>	<code> 16</code>	<code>[5 elements]</code>
<code>id 50b59cd75bed7...</code>	<code> 0</code>	<code> 6</code>	<code>[6 elements]</code>
<code>id 50b59cd75bed7...</code>	<code> 0</code>	<code> 27</code>	<code>[3 elements]</code>
<code>id 50b59cd75bed7...</code>	<code> 0</code>	<code> 11</code>	<code>[4 elements]</code>
<code>id 50b59cd75bed7...</code>	<code> 1</code>	<code> 18</code>	<code>[3 elements]</code>
<code>id 50b59cd75bed7...</code>	<code> 1</code>	<code> 16</code>	<code>[5 elements]</code>

Explicación:

Esta consulta busca todos los documentos en la colección grades donde el arreglo scores contenga al menos un elemento con:

type igual a 'exam', y
score mayor o igual a 50.

Resultado:

La consulta devolverá todos los documentos que tengan un puntaje de exam con una calificación mayor o igual a 50. Si no existe ningún exam con una calificación mayor o igual a 50, el documento no será devuelto.

3. Busca la colección Narcos, ejecuta las siguientes consultas y describe qué hace cada una de ellas:

`db.Narcos.find({ runtime: { $gte: 55 } }, { _id:0, name:1, season:1, number:1 })`

The screenshot shows the MongoDB shell interface. At the top, there's a toolbar with various icons for file operations like 'New', 'Load script', 'Save script', 'Script history', 'Enable Query Assist', and 'Change'. Below the toolbar, a code editor displays the following MongoDB query:

```

1 use Episodios;
2
3 db.Narcos.find( { runtime: { $gte: 55 } }, { _id:0, name:1, season:1, number:1 } )
4
5

```

Below the code editor is a navigation bar with buttons for 'Raw shell output' and 'Find Query (line 3)'. It also includes a search bar with the value '50' and a dropdown menu for 'Documents 1 to 10'. To the right of the search bar are several small icons for document manipulation.

The main area shows the results of the query. The title 'Narcos > season' is displayed above a table. The table has three columns: 'name', 'season', and 'number'. The data is as follows:

name	season	number
Descenso	1	1
There Will Be a F...	1	5
The Good, the B...	2	4
Deutschland 93	2	7
Exit El Patrón	2	8
Nuestra Finca	2	9
The Kingpin Strat...	3	1
Follow the Money	3	2

Explicación:

La consulta devolverá una lista de episodios donde el runtime es igual o superior a 55 minutos. El resultado incluirá:

El nombre del episodio (name).

La temporada a la que pertenece (season).

El número del episodio en esa temporada (number).

```
db.Narcos.find( { runtime: { $gte: 15 } }, { _id:0, name:1, season:1, number:1 } ).sort( {season:1, number:-1} )
```

Explicación:

La consulta devolverá una lista de episodios donde el runtime es igual o superior a 15 minutos. El resultado incluirá:

La temporada a la que pertenece (season).

El número del episodio en esa temporada (number).

Los episodios estarán ordenados por temporada de menor a mayor, y dentro de cada temporada, los episodios estarán ordenados de mayor a menor número.

```
db.Narcos.find( { season: { $type: 'number' } } )
```

The screenshot shows the IntelliShell interface with the following details:

- Query:**

```
1 use Episodios;
2
3 db.Narcos.find( { season: { $type: 'number' } } )
4
5
```
- Output:**

Lin 4, Col 1 No errors

_id	id	url	name	season	number	type	airdate
67086a49e31a2...	203469	https://www.tv...	Descenso	1	1	regular	20
67086a49e31a2...	208978	https://www.tv...	The Sword of Si...	1	2	regular	20
67086a49e31a2...	208979	https://www.tv...	The Men of Alw...	1	3	regular	20
67086a49e31a2...	208980	https://www.tv...	The Palace in Fla...	1	4	regular	20
67086a49e31a2...	208981	https://www.tv...	There Will Be a F...	1	5	regular	20
67086a49e31a2...	208982	https://www.tv...	Explosivos	1	6	regular	20
67086a49e31a2...	208983	https://www.tv...	You Will Crv Tea...	1	7	regular	20

Explicación:

La consulta devolverá todos los episodios que tengan un campo season que sea un número. Esto incluirá episodios de cualquier temporada, siempre y cuando el campo season tenga un valor numérico.

`db.Narcos.find({ rating: { $exists: 1 } })`

The screenshot shows the IntelliShell interface with the following details:

- Query:**

```
1 use Episodios;
2
3 db.Narcos.find( { rating: { $exists: 1 } } )
4
5
```
- Output:**

Lin 4, Col 1 No errors

_id	id	url	name	season	number	type	airdate
67086a49e31a2...	203469	https://www.tv...	Descenso	1	1	regular	20
67086a49e31a2...	208978	https://www.tv...	The Sword of Si...	1	2	regular	20
67086a49e31a2...	208979	https://www.tv...	The Men of Alw...	1	3	regular	20
67086a49e31a2...	208980	https://www.tv...	The Palace in Fla...	1	4	regular	20
67086a49e31a2...	208981	https://www.tv...	There Will Be a F...	1	5	regular	20
67086a49e31a2...	208982	https://www.tv...	Explosivos	1	6	regular	20
67086a49e31a2...	208983	https://www.tv...	You Will Crv Tea...	1	7	regular	20

Explicación:

La consulta devolverá todos los episodios que tienen el campo rating. Esto incluirá información sobre la calificación promedio de cada episodio.

```
db.Narcos.find( { rating: { $exists: 1 }, rating: { $type: "string" } } )
```

The screenshot shows the IntelliShell interface for MongoDB. The title bar says "IntelliShell: Base de datos y almacenamiento masivo*". The connection is set to "localhost:27017 > Episodios". The toolbar includes "New", "Load script", "Save script", "Script history", and "Enable Query Assist". The query editor contains the following code:

```
1 use Episodios;
2
3 db.Narcos.find( { rating: { $exists: 1 }, rating: { $type: "string" } } )
4
5
```

Below the editor is a toolbar with icons for copy, paste, run, and help. The "Raw shell output" tab is selected, showing the result of the query:

```
Narcos
```

Explicación:

La consulta intenta recuperar documentos de la colección "Narcos" donde el campo rating: Exista (es decir, el campo rating está presente en el documento). Sea de tipo cadena (string). En este caso, la consulta está tratando de aplicar ambas condiciones al campo rating, lo que no es válido. Si se desea realizar ambas verificaciones, se debe usar el operador \$and para combinarlas adecuadamente.

4. Prueba las siguientes consultas:

```
db.grades.distinct("student_id")
```

The screenshot shows the IntelliShell interface for MongoDB. The top bar indicates the connection is to 'localhost:27017' and the database is 'School'. The script pane contains the following code:

```

1 use School;
2
3 db.grades.distinct("student_id")
4
5

```

The third line, 'db.grades.distinct("student_id")', is highlighted in yellow. Below the script pane is a toolbar with various icons. The bottom pane, titled 'Raw shell output' and 'Shell Output (Array)', displays the results of the query. It shows a table titled 'Array' with 8 rows, indexed from 0 to 7. The values in the 'scores' column are: 0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, and 7.0. The first row (index 0) has a blue background, indicating it is the current selected item.

[Index]	scores
0	0.0
1	1.0
2	2.0
3	3.0
4	4.0
5	5.0
6	6.0
7	7.0

Explicación:

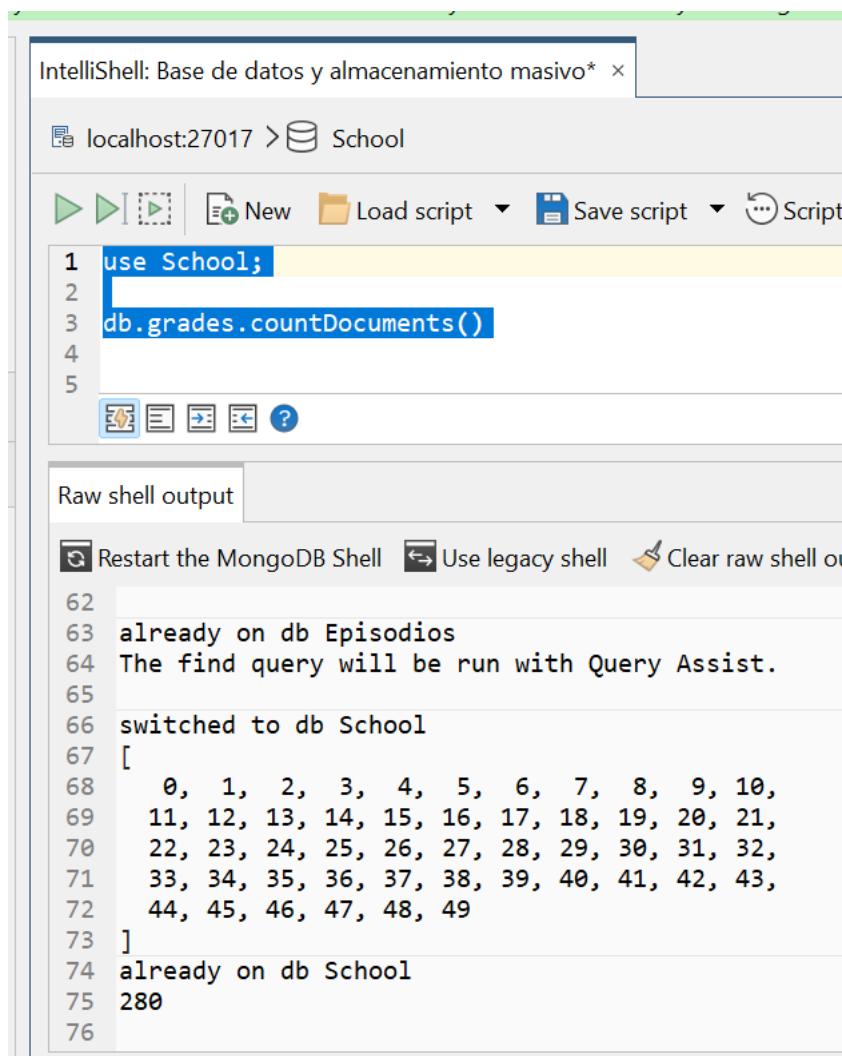
Esta consulta busca todos los documentos en la colección grades donde el arreglo scores contenga al menos un elemento con:

type igual a 'exam', y
score mayor o igual a 50.

Resultado:

La consulta devolverá todos los documentos que tengan un puntaje de exam con una calificación mayor o igual a 50. Si no existe ningún exam con una calificación mayor o igual a 50, el documento no será devuelto.

db.grades.countDocuments()



The screenshot shows the IntelliShell interface for MongoDB. The title bar says "IntelliShell: Base de datos y almacenamiento masivo*". The top menu bar includes "localhost:27017 > School". Below the menu are buttons for "New", "Load script", "Save script", and "Script". The main code editor area contains the following MongoDB script:

```

1 use School;
2
3 db.grades.countDocuments()
4
5

```

Below the code editor is a "Raw shell output" panel. It shows the results of the executed command:

```

62
63 already on db Episodios
64 The find query will be run with Query Assist.
65
66 switched to db School
67 [
68   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
69   11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
70   22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
71   33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
72   44, 45, 46, 47, 48, 49
73 ]
74 already on db School
75 280
76

```

Explicación:

La consulta devuelve el número total de documentos en la colección grades.

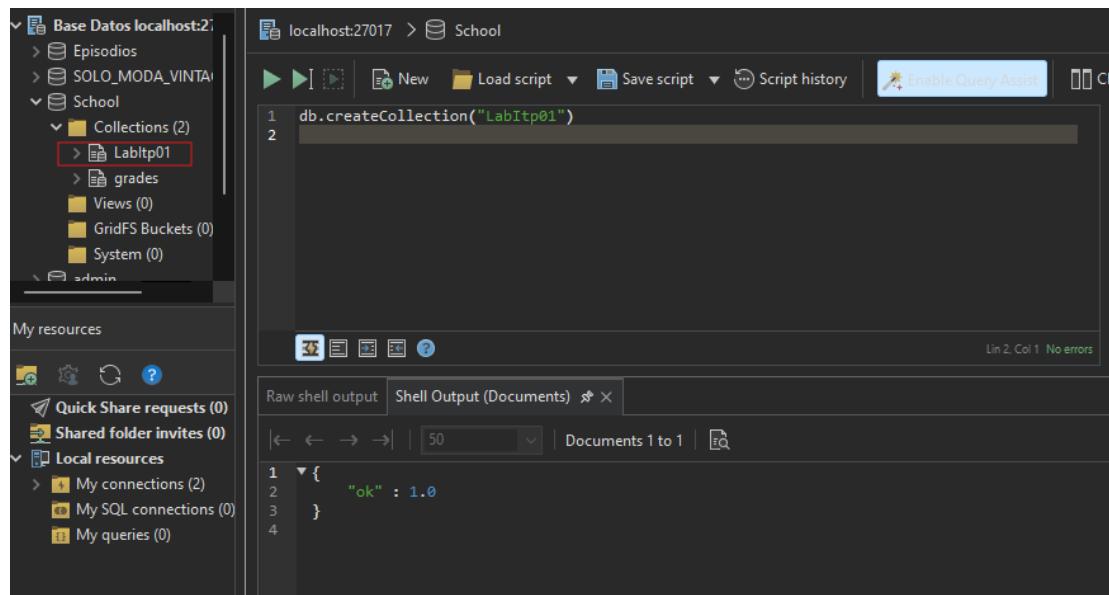
Resultado:

El resultado de esta consulta será un número entero que indica cuántos documentos hay en la colección grades.

Trabaja con CREATE

1. En la BD School crea la colección LabItp:

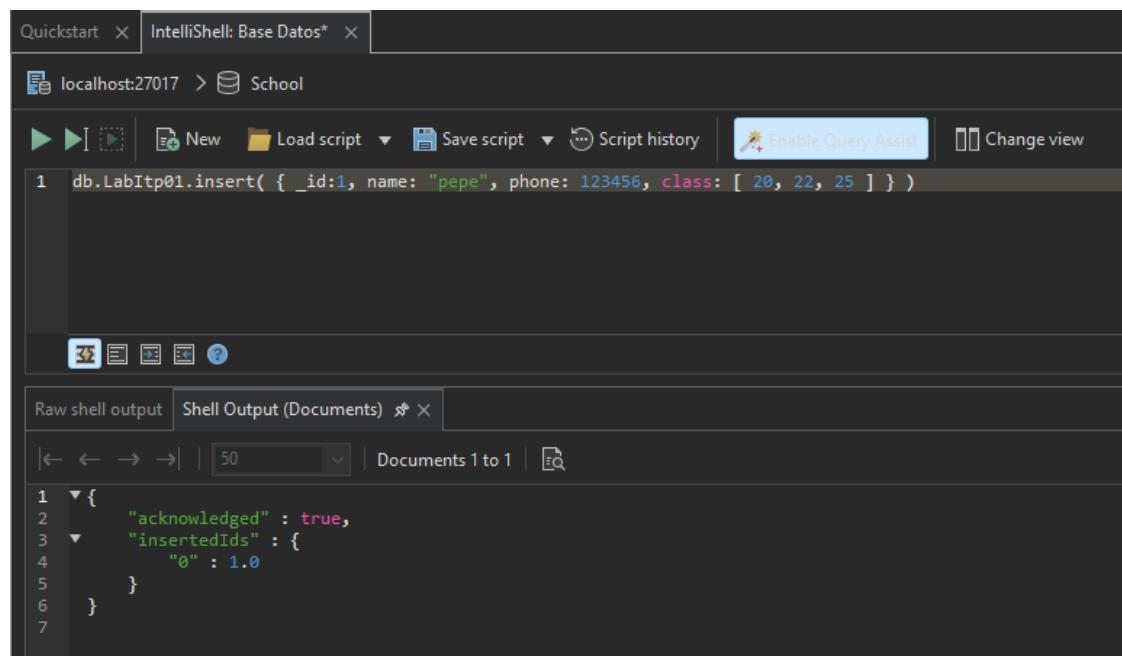
```
db.createCollection("LabItp01")
```



Explicación:

Crea una nueva colección llamada LabItp01 en la base de datos actual. En MongoDB, las colecciones no necesitan ser creadas explícitamente, ya que se crean automáticamente cuando se inserta el primer documento, pero este comando permite crearla manualmente.

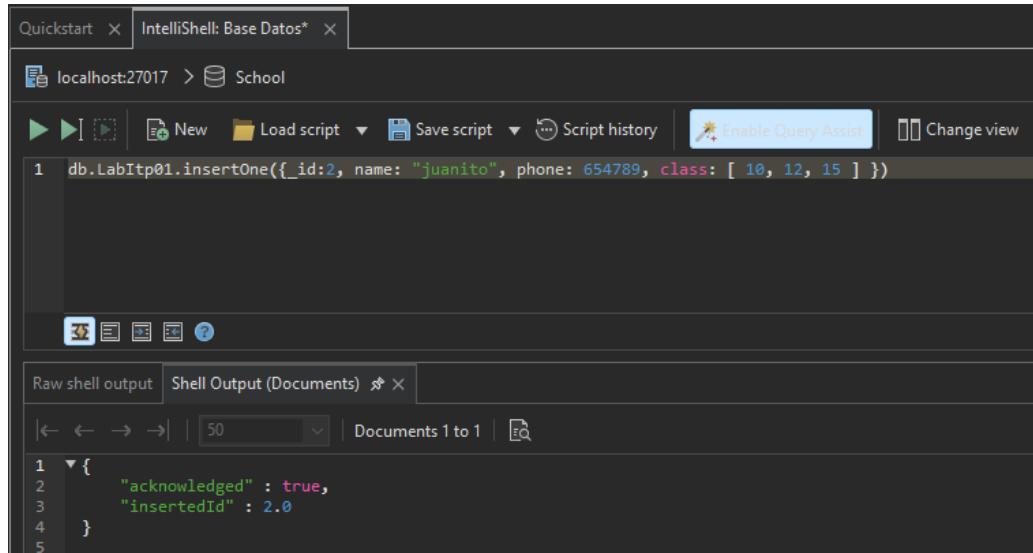
```
db.LabItp01.insert({ _id:1, name: "pepe", phone: 123456, class: [ 20, 22, 25 ] })
```



Explicación:

Inserta un documento en la colección LabItp01 con los campos _id, name, phone, y class (que es un arreglo de números). Usa insert, que es una forma antigua de insertar documentos en MongoDB.

```
db.LabItp01.insertOne({_id:2, name: "juanito", phone: 654789, class: [ 10, 12, 15 ] })
```



The screenshot shows the MongoDB shell interface. The top bar has tabs for 'Quickstart' and 'IntelliShell: Base Datos*'. Below that is a toolbar with icons for file operations like New, Load script, Save script, and Script history. A button for 'Enable Query Assist' is also present. The main area contains a command line and a results pane. The command entered is:

```
1 db.LabItp01.insertOne({_id:2, name: "juanito", phone: 654789, class: [ 10, 12, 15 ] })
```

The results pane shows the response from the database:

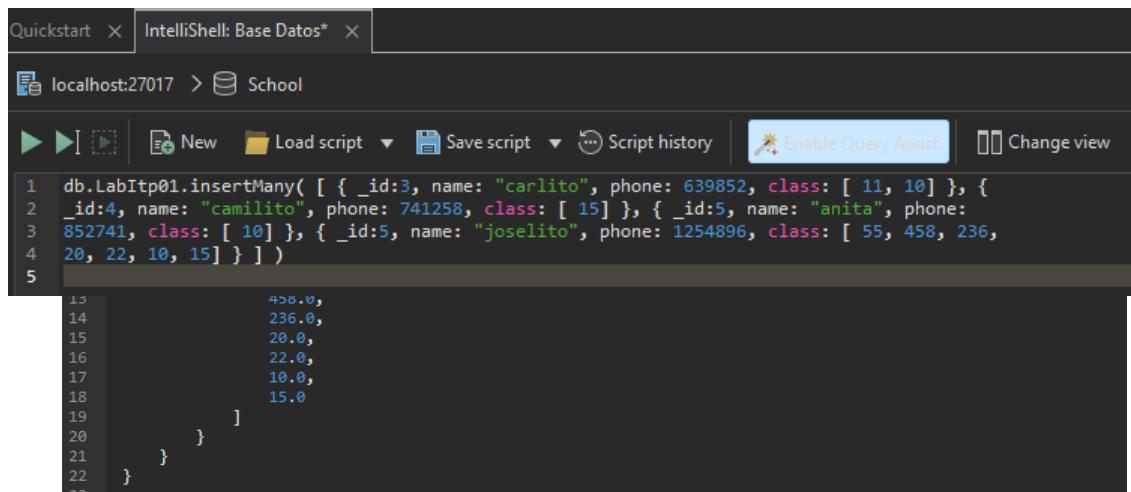
```
1 { "_id": 2, "name": "juanito", "phone": 654789, "class": [10, 12, 15] }
```

Below the results pane, there's a 'Raw shell output' tab and a 'Shell Output (Documents)' tab. The bottom of the interface has navigation buttons (left, right, first, last) and a document count indicator 'Documents 1 to 1'.

Explicación:

Inserta un único documento en la colección LabItp01 usando el método insertOne. Este método es preferido en lugar de insert para insertar un solo documento.

```
db.LabItp01.insertMany( [ { _id:3, name: "carlito", phone: 639852, class: [ 11, 10] }, { _id:4, name: "camilito", phone: 741258, class: [ 15] }, { _id:5, name: "anita", phone:852741, class: [ 10] }, { _id:5, name: "joselito", phone: 1254896, class: [ 55, 458, 236,20, 22, 10, 15] } ] )
```



The screenshot shows the MongoDB shell interface. The top bar has tabs for 'Quickstart' and 'IntelliShell: Base Datos*'. Below that is a toolbar with icons for file operations like New, Load script, Save script, and Script history. A button for 'Enable Query Assist' is also present. The main area contains a command line and a results pane. The command entered is:

```
1 db.LabItp01.insertMany( [ { _id:3, name: "carlito", phone: 639852, class: [ 11, 10] }, { _id:4, name: "camilito", phone: 741258, class: [ 15] }, { _id:5, name: "anita", phone:852741, class: [ 10] }, { _id:5, name: "joselito", phone: 1254896, class: [ 55, 458, 236, 20, 22, 10, 15] } ] )
```

The results pane shows the response from the database, which includes the inserted documents:

```
1 { "_id": 3, "name": "carlito", "phone": 639852, "class": [11, 10] }
2 { "_id": 4, "name": "camilito", "phone": 741258, "class": [15] }
3 { "_id": 5, "name": "anita", "phone": 852741, "class": [10] }
4 { "_id": 5, "name": "joselito", "phone": 1254896, "class": [55, 458, 236, 20, 22, 10, 15] }
```

Below the results pane, there's a 'Raw shell output' tab and a 'Shell Output (Documents)' tab. The bottom of the interface has navigation buttons (left, right, first, last) and a document count indicator 'Documents 1 to 1'.

Explicación:

Inserta múltiples documentos en la colección LabItp01 usando el método insertMany. Todos los documentos tienen diferentes _id, excepto el último, que tiene un _id duplicado (lo que generará un error si el índice _id es único).

db.LabItp01.find({ class: 10 })

```

1 db.LabItp01.find({"class": 10})
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24

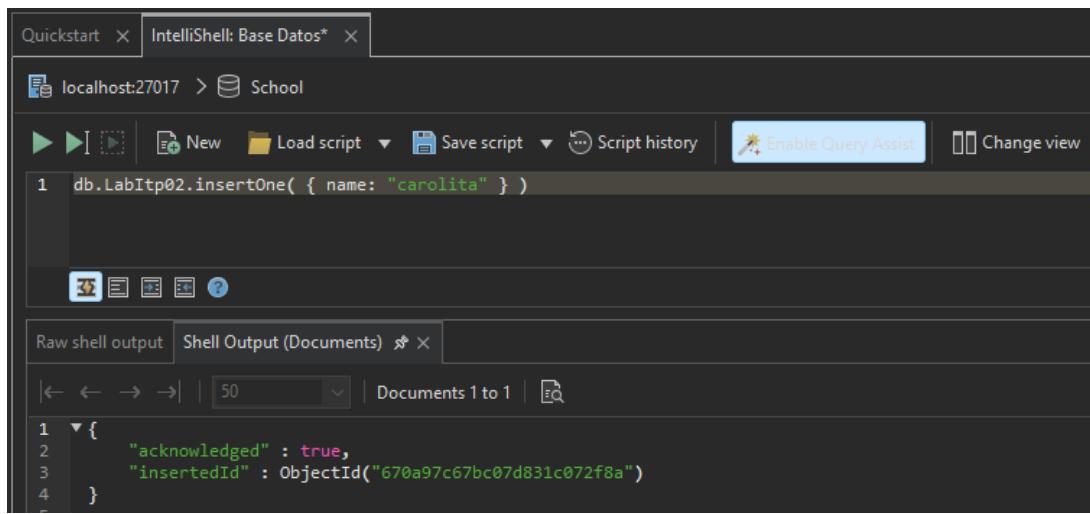
```

1 document selected

Explicación:

Busca en la colección LabItp01 todos los documentos donde el campo class contiene el valor 10. Como class es un arreglo, MongoDB busca si el valor está presente en cualquiera de sus posiciones.

db.LabItp02.insertOne({ name: "carolita" })



The screenshot shows the MongoDB Compass interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". Below that is a connection bar for "localhost:27017" and a database selector for "School". The toolbar includes buttons for "New", "Load script", "Save script", "Script history", "Enable Query Assist" (which is highlighted in blue), and "Change view". The main area contains a query editor with the following code:

```
1 db.LabItp02.insertOne( { name: "carolita" } )
```

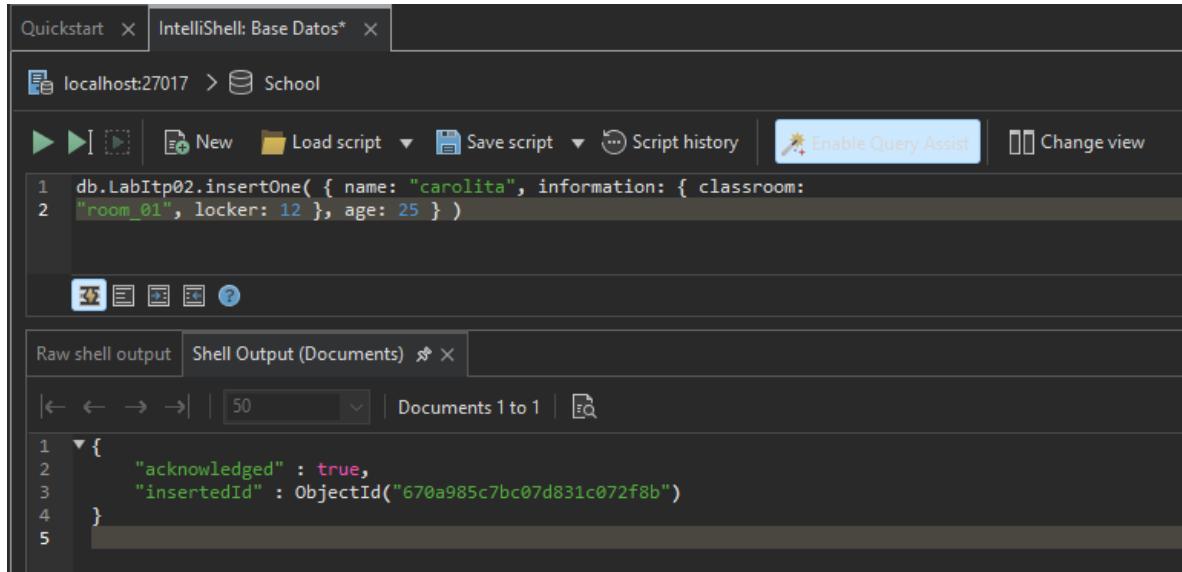
Below the code is a results pane showing the inserted document:

```
1 ▼ {
  2   "acknowledged" : true,
  3   "insertedId" : ObjectId("670a97c67bc07d831c072f8a")
  4 }
```

Explicación:

Inserta un documento con el campo name en la colección LabItp02. Si la colección LabItp02 no existía previamente, esta consulta la crea automáticamente.

db.LabItp02.insertOne({ name: "carolita", information: { classroom:"room_01", locker: 12 }, age: 25 })



The screenshot shows the MongoDB Compass interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". Below that is a connection bar for "localhost:27017" and a database selector for "School". The toolbar includes buttons for "New", "Load script", "Save script", "Script history", "Enable Query Assist" (highlighted in blue), and "Change view". The main area contains a query editor with the following code:

```
1 db.LabItp02.insertOne( { name: "carolita", information: { classroom:
2   "room_01", locker: 12 }, age: 25 } )
```

Below the code is a results pane showing the inserted document:

```
1 ▼ {
  2   "acknowledged" : true,
  3   "insertedId" : ObjectId("670a985c7bc07d831c072f8b")
  4 }
```

Explicación:

Inserta un documento en LabItp02 con los campos name, information (que es un objeto anidado con classroom y locker), y age.

db.LabItp02.find()

The screenshot shows the MongoDB Intellishell interface. The title bar says "IntelliShell: Base Datos*". Below it, the connection status is "localhost:27017 > School". The toolbar includes "New", "Load script", "Save script", "Script history", "Enable Query Assist", and "Change view". A query window displays the command "1 ↳ db.LabItp02.find()", followed by the results:

```

1 {
  "_id" : ObjectId("670a97c67bc07d831c072f8a"),
  "name" : "carolita"
}
2 {
  "_id" : ObjectId("670a985c7bc07d831c072f8b"),
  "name" : "carolita",
  "information" : {
    "classroom" : "room_01",
    "locker" : NumberInt(12)
  },
  "age" : NumberInt(25)
}

```

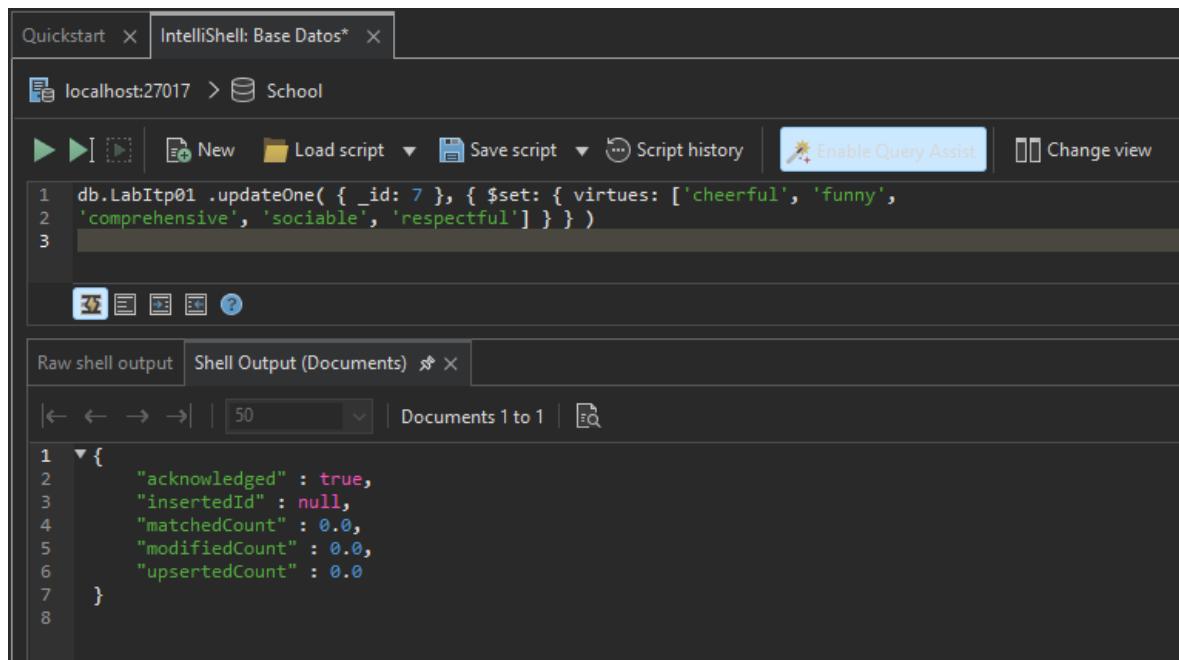
Explicación:

Busca y devuelve todos los documentos de la colección LabItp02.

Trabaja con UPDATE

En la colección LabItp01 realiza las siguientes actualizaciones

```
db.LabItp01 .updateOne( { _id: 7 }, { $set: { virtues: ['cheerful', 'funny', 'comprehensive', 'sociable', 'respectful'] } } )
```



The screenshot shows the MongoDB Compass interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". The main area shows a connection to "localhost:27017" and a database named "School". Below the connection info are buttons for "New", "Load script", "Save script", "Script history", "Enable Query Assist" (which is highlighted in blue), and "Change view". The query editor contains the following code:

```
1 db.LabItp01 .updateOne( { _id: 7 }, { $set: { virtues: [ 'cheerful', 'funny', 'comprehensive', 'sociable', 'respectful' ] } } )
```

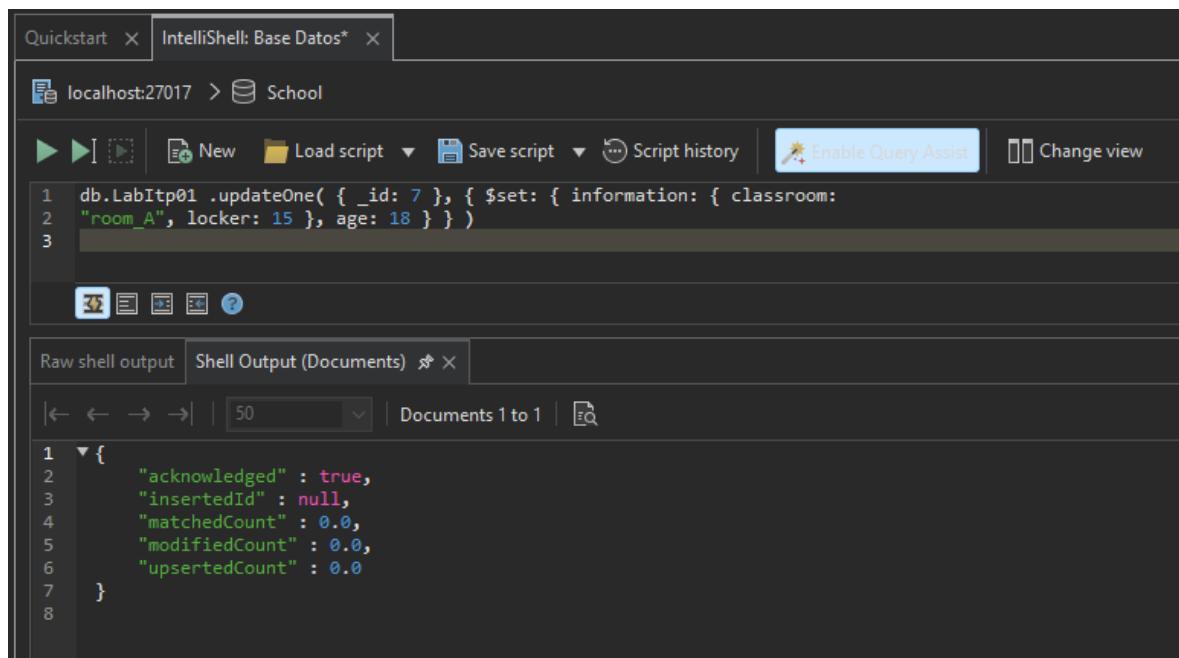
Below the query editor is a toolbar with icons for search, refresh, and other operations. The results pane is titled "Raw shell output" and "Shell Output (Documents)". It shows the response from the database:

```
1 {  
2   "acknowledged" : true,  
3   "insertedId" : null,  
4   "matchedCount" : 0.0,  
5   "modifiedCount" : 0.0,  
6   "upsertedCount" : 0.0  
7 }  
8
```

Explicación:

Esta consulta busca el documento con `_id: 7` y establece el campo `virtues` con el array de valores `['cheerful', 'funny', 'comprehensive', 'sociable', 'respectful']`. Si el campo `virtues` no existe, se crea.

`db.LabItp01 .updateOne({ _id: 7 }, { $set: { information: { classroom: "room_A", locker: 15 }, age: 18 } })`



The screenshot shows the MongoDB Compass interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". The main area shows a connection to "localhost:27017" and a database named "School". Below the connection info are buttons for "New", "Load script", "Save script", "Script history", "Enable Query Assist" (highlighted in blue), and "Change view". The query editor contains the following code:

```
1 db.LabItp01 .updateOne( { _id: 7 }, { $set: { information: { classroom: "room_A", locker: 15 }, age: 18 } } )
```

Below the query editor is a toolbar with icons for search, refresh, and other operations. The results pane is titled "Raw shell output" and "Shell Output (Documents)". It shows the response from the database:

```
1 {  
2   "acknowledged" : true,  
3   "insertedId" : null,  
4   "matchedCount" : 0.0,  
5   "modifiedCount" : 0.0,  
6   "upsertedCount" : 0.0  
7 }  
8
```

Explicación:

Esta consulta busca el documento con `_id: 7` y actualiza dos campos: `information`, que es un objeto anidado con los campos `classroom` y `locker`, y el campo `age` con el valor 18.

```
db.LabItp01 .updateOne( { _id: 7 }, { $set: { virtues: ['cheerful', 'funny', 'comprehensive', 'sociable', 'respectful'] }, $currentDate: { lastModified: true } } )
```

The screenshot shows the MongoDB Compass interface. The top navigation bar includes 'Quickstart' and 'IntelliShell: Base Datos*'. Below the bar, it says 'localhost:27017 > School'. The toolbar has buttons for 'New', 'Load script', 'Save script', 'Script history', 'Enable Query Assist' (which is highlighted), and 'Change view'. The main area contains the following code:

```
1 db.LabItp01 .updateOne( { _id: 7 }, { $set: { virtues: ['cheerful', 'funny', 'comprehensive', 'sociable', 'respectful'] }, $currentDate: { lastModified: true } } )
```

Below the code, the 'Raw shell output' tab is selected, showing the response:

```
1 {  
2   "acknowledged" : true,  
3   "insertedId" : null,  
4   "matchedCount" : 0.0,  
5   "modifiedCount" : 0.0,  
6   "upsertedCount" : 0.0  
7 }
```

Explicación:

Esta consulta actualiza el campo `virtues` con el array especificado y también actualiza el campo `lastModified` con la fecha y hora actual usando `$currentDate`.

```
db.LabItp01 .updateOne( { _id: 7 }, { $set: { information: { classroom: "room_A", locker: 15 }, age: 18 }, $currentDate: { lastModified: true } } )
```

The screenshot shows the IntelliJ IDEA interface with the MongoDB plugin. The top bar has tabs for 'Quickstart' and 'IntelliShell: Base Datos*'. The main area shows a connection to 'localhost:27017' and a database 'School'. A script editor window contains the following MongoDB update query:

```
1 db.LabItp01 .updateOne( { _id: 7 }, { $set: { information: { classroom: "room_A", locker: 15 }, age: 18 }, $currentDate: { lastModified: true } } )
```

The results pane shows the response document:

```
1 {
2     "acknowledged" : true,
3     "insertedId" : null,
4     "matchedCount" : 0.0,
5     "modifiedCount" : 0.0,
6     "upsertedCount" : 0.0
7 }
```

Explicación:

Similar a la anterior, pero además de actualizar la fecha en lastModified, también actualiza los campos information y age.

```
db.LabItp01 .updateOne( { _id: 10 }, { $set: { name: "Joan", age: 19, virtues: [], information: {} }, $currentDate: { lastModified: true } }, { upsert: true } )
```

The screenshot shows the IntelliJ IDEA interface with the MongoDB plugin. The top bar has tabs for 'Quickstart' and 'IntelliShell: Base Datos*'. The main area shows a connection to 'localhost:27017' and a database 'School'. A script editor window contains the following MongoDB update query:

```
1 db.LabItp01 .updateOne( { _id: 10 }, { $set: { name: "Joan", age: 19, virtues: [], information: {} }, $currentDate: { lastModified: true } }, { upsert: true } )
```

The results pane shows the response document:

```
1 {
2     "acknowledged" : true,
3     "insertedId" : 10.0,
4     "matchedCount" : 0.0,
5     "modifiedCount" : 0.0,
6     "upsertedCount" : 1.0
7 }
```

Explicación:

Busca el documento con `_id: 10` y actualiza los campos `name`, `age`, `virtues` (un array vacío), e `information` (un objeto vacío). Si el documento no existe, lo crea (upsert). Además, actualiza el campo `lastModified` con la fecha y hora actual.

```
db.LabItp01.updateMany( { _id: { $in: [1, 2, 3, 4, 5, 6] } }, { $set: { virtues: ['cheerful'], //  
Puedes elegir otro valor de la lista si prefieres status: 'A' } } );
```

The screenshot shows the MongoDB shell interface. The top bar indicates the connection is to 'localhost:27017' and the database is 'School'. The script tab contains the following code:

```
1 db.LabItp01.updateMany(  
2   { _id: { $in: [1, 2, 3, 4, 5, 6] } },  
3   {  
4     $set: {  
5       virtues: ['cheerful'],  
6       status: 'A'  
7     }  
8   }  
9 );  
10
```

The bottom pane shows the 'Raw shell output' tab with the following JSON response:

```
1 {  
2   "acknowledged" : true,  
3   "insertedId" : null,  
4   "matchedCount" : 5.0,  
5   "modifiedCount" : 5.0,  
6   "upsertedCount" : 0.0  
7 }  
8
```

Explicación:

Actualiza los documentos con `_id 1 – 6` y agrega el campo `virtues` con un array que contenga un único valor, el que decidas de la lista siguiente: ['cheerful', 'funny', 'comprehensive', 'sociable', 'respectful'].

```
db.LabItp01.updateMany( {}, { $set: { status: 'A' } } );
```

The screenshot shows the IntelliJ Shell interface with the title bar "IntelliShell: Base Datos*". Below it, the connection information "localhost:27017 > School" is displayed. The main area contains a code editor with the following MongoDB script:

```
1 db.LabItp01.updateMany(  
2     {},  
3     { $set: { status: 'A' } }  
4 );  
5
```

Below the code editor is a "Raw shell output" tab showing the results of the execution:

```
1 {  
2     "acknowledged" : true,  
3     "insertedId" : null,  
4     "matchedCount" : 6.0,  
5     "modifiedCount" : 1.0,  
6     "upsertedCount" : 0.0  
7 }  
8
```

Explicación:

Actualiza todos los documentos con una única instrucción y agrega el siguiente campo: status: 'A'.

```
db.LabItp01.updateMany( { name: { $in: ['pepe', 'camilito'] } }, { $set: { role: 'student' } } );
```

The screenshot shows the IntelliJ Shell interface with the title bar "IntelliShell: Base Datos*". Below the title bar, it says "localhost:27017 > School". The main area contains a code editor with the following MongoDB update script:

```

1 db.LabItp01.updateMany(
2   { name: { $in: ['pepe', 'camilito'] } },
3   {
4     $set: { role: 'student' }
5   }
6 );
7

```

Below the code editor is a "Raw shell output" tab showing the response from the database:

```

1 {
2   "acknowledged" : true,
3   "insertedId" : null,
4   "matchedCount" : 2.0,
5   "modifiedCount" : 2.0,
6   "upsertedCount" : 0.0
7 }
8

```

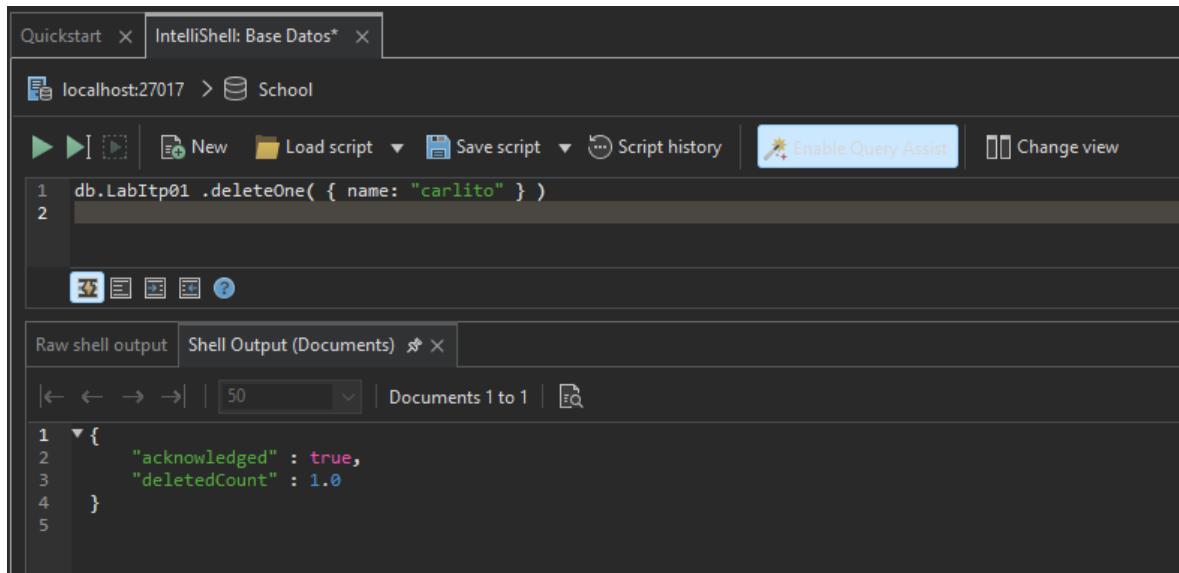
Explicación:

Actualiza los documentos de “pepe” y “camilito” y agrega el siguiente campo: role: 'student'

Trabaja con DELETE

En la colección LabItp01 realiza las siguientes actualizaciones:

db.LabItp01 .deleteOne({ name: "carlito" })



The screenshot shows the MongoDB Compass interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". Below that, it says "localhost:27017 > School". The toolbar includes "New", "Load script", "Save script", "Script history", "Enable Query Assist" (which is highlighted), and "Change view". The main area contains a code editor with the following command:

```
1 db.LabItp01.deleteOne( { name: "carlito" } )
```

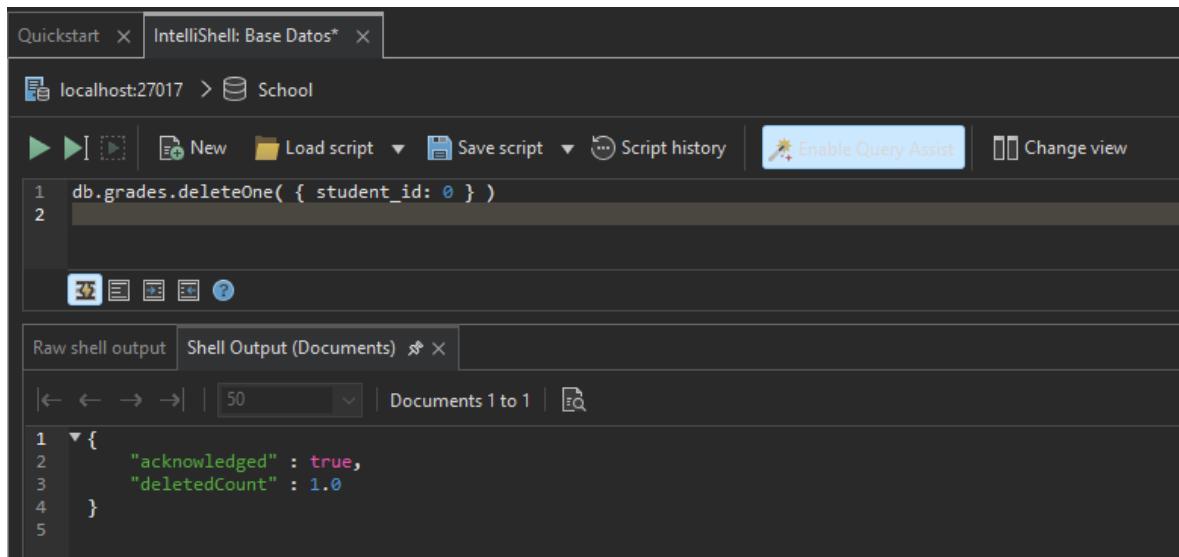
Below the code editor is a "Raw shell output" panel showing the response:

```
1 {  
2   "acknowledged" : true,  
3   "deletedCount" : 1.0  
4 }
```

Explicación:

Esta consulta elimina un solo documento en la colección LabItp01 donde el campo name es "carlito". Si hay más de uno, solo elimina el primero que encuentra.

db.grades.deleteOne({ student_id: 0 })



The screenshot shows the MongoDB Compass interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". Below that, it says "localhost:27017 > School". The toolbar includes "New", "Load script", "Save script", "Script history", "Enable Query Assist" (which is highlighted), and "Change view". The main area contains a code editor with the following command:

```
1 db.grades.deleteOne( { student_id: 0 } )
```

Below the code editor is a "Raw shell output" panel showing the response:

```
1 {  
2   "acknowledged" : true,  
3   "deletedCount" : 1.0  
4 }
```

Explicación:

Elimina un solo documento en la colección grades donde el campo student_id es igual a 0. Similar a la primera, solo elimina el primer documento que coincide.

db.grades.deleteMany({ student_id: 0 })

The screenshot shows the MongoDB Intellishell interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". The main area shows a connection to "localhost:27017" and a database named "School". Below the connection info are buttons for "New", "Load script", "Save script", "Script history", "Enable Query Assist", and "Change view". The script editor contains the following code:

```
1 db.grades.deleteMany( { student_id: 0 } )
```

The output pane shows the results of the command:

```
1 {  
2   "acknowledged" : true,  
3   "deletedCount" : 10.0  
4 }  
5
```

Explicación:

Esta consulta elimina todos los documentos en la colección grades donde student_id sea igual a 0, es decir, puede borrar más de uno si existen múltiples coincidencias.

db.grades.remove({ student_id: 1 }, {justOne: true})

The screenshot shows the MongoDB Intellishell interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". The main area shows a connection to "localhost:27017" and a database named "School". Below the connection info are buttons for "New", "Load script", "Save script", "Script history", "Enable Query Assist", and "Change view". The script editor contains the following code:

```
1 db.grades.remove( { student_id: 1 }, {justOne: true} )
```

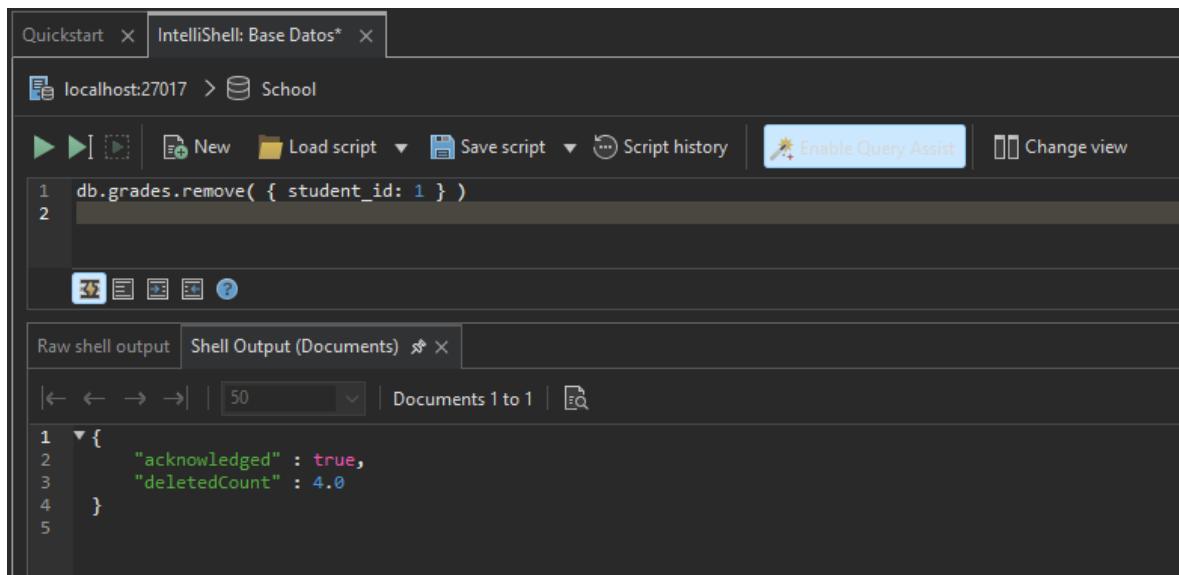
The output pane shows the results of the command:

```
1 {  
2   "acknowledged" : true,  
3   "deletedCount" : 1.0  
4 }  
5
```

Explicación:

Similar a deleteOne, pero usando el comando más antiguo remove. Elimina un solo documento en grades donde student_id es igual a 1. El parámetro {justOne: true} asegura que solo se elimine un documento.

db.grades.remove({ student_id: 1 })



The screenshot shows the MongoDB Intellishell interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". The main area shows a connection to "localhost:27017" and a database named "School". Below the connection info are buttons for "New", "Load script", "Save script", "Script history", "Enable Query Assist" (which is turned on), and "Change view". The script editor contains the following code:

```
1 db.grades.remove( { student_id: 1 } )
```

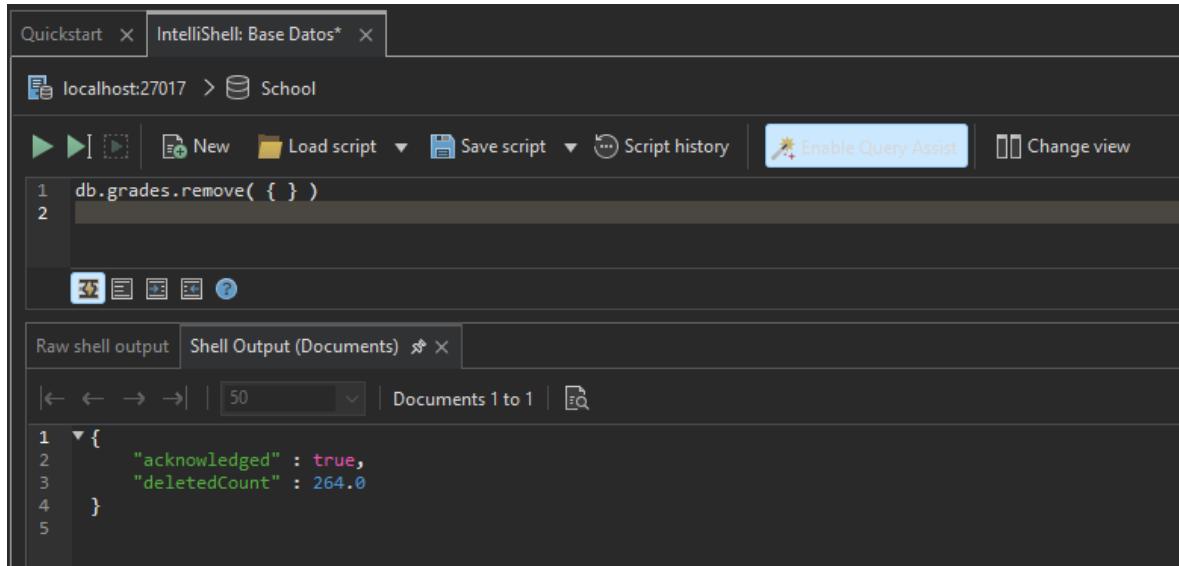
The output pane below shows the results of the command:

```
1 {  
2   "acknowledged" : true,  
3   "deletedCount" : 4.0  
4 }  
5
```

Explicación:

Elimina todos los documentos en grades donde student_id es igual a 1. Esta consulta podría eliminar múltiples documentos si hay más de uno con ese student_id.

db.grades.remove({})



The screenshot shows the MongoDB Intellishell interface. The top bar has tabs for "Quickstart" and "IntelliShell: Base Datos*". The main area shows a connection to "localhost:27017" and a database named "School". Below the connection info are buttons for "New", "Load script", "Save script", "Script history", "Enable Query Assist" (which is turned on), and "Change view". The script editor contains the following code:

```
1 db.grades.remove( {} )
```

The output pane below shows the results of the command:

```
1 {  
2   "acknowledged" : true,  
3   "deletedCount" : 264.0  
4 }  
5
```

Explicación:

Elimina todos los documentos de la colección grades. El uso de {} como filtro significa que no se aplican restricciones de selección.

db.grades.drop()

The screenshot shows the MongoDB Intellishell interface. The title bar says "IntelliShell: Base Datos*". The top menu includes "New", "Load script", "Save script", "Script history", "Enable Query Assist", and "Change view". Below the menu, a code editor window displays the command "db.grades.drop()". The output pane shows the results of the command execution, which includes a warning about the deprecation of Collection.remove() and the use of deleteOne, deleteMany, findOneAndDelete, or bulkWrite instead. The output is as follows:

```

127 modifiedCount: 2,
128 upsertedCount: 0
129 }
130 {
131 acknowledged: true,
132 insertedId: null,
133 matchedCount: 6,
134 modifiedCount: 1,
135 upsertedCount: 0
136 }
137 { acknowledged: true, deletedCount: 1 }
138 { acknowledged: true, deletedCount: 1 }
139 { acknowledged: true, deletedCount: 10 }
140 DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.
141 { acknowledged: true, deletedCount: 1 }
142 { acknowledged: true, deletedCount: 4 }
143 { acknowledged: true, deletedCount: 264 }
144 true
145

```

Explicación:

Esta consulta elimina toda la colección grades, incluyendo tanto los datos como la estructura. Es una operación irreversible que elimina completamente la colección.

Consultas base de datos personal

Actualizar la dirección de un cliente

Supongamos que necesitamos actualizar la dirección de un cliente específico.

Usaremos updateOne() para hacer coincidir el cliente_id.

The screenshot shows the IntelliJ Shell: Mongo interface. At the top, there are tabs for 'Quickstart' and 'IntelliShell: Mongo*'. Below the tabs, the connection information is displayed as 'localhost:27017 > Libreria'. The main area contains a code editor with the following MongoDB update query:

```
1 db.clientes.updateOne(  
2   { "cliente_id": 1 }, // Filtro para encontrar al cliente  
3   {  
4     $set: { "direccion": "Calle Nueva 456, Ciudad" } // Actualización  
5   }  
6 );  
7
```

Below the code editor is a toolbar with various icons. The bottom section shows the 'Raw shell output' tab, which displays the results of the executed query:

```
1 {  
2   "acknowledged" : true,  
3   "insertedId" : null,  
4   "matchedCount" : 1.0,  
5   "modifiedCount" : 1.0,  
6   "upsertedCount" : 0.0  
7 }
```

Agregar un nuevo préstamo a un cliente existente

Para agregar un nuevo préstamo a un cliente, puedes usar el operador \$push, que se utiliza para agregar un elemento a un arreglo.

The screenshot shows the IntelliJ Shell interface for MongoDB. The top bar has tabs for 'Quickstart' and 'IntelliShell: Mongo*'. The main area is titled 'localhost:27017 > Libreria'. Below the title are buttons for 'New', 'Load script', 'Save script', 'Script history', and 'Enable Query'. The code editor contains the following MongoDB update query:

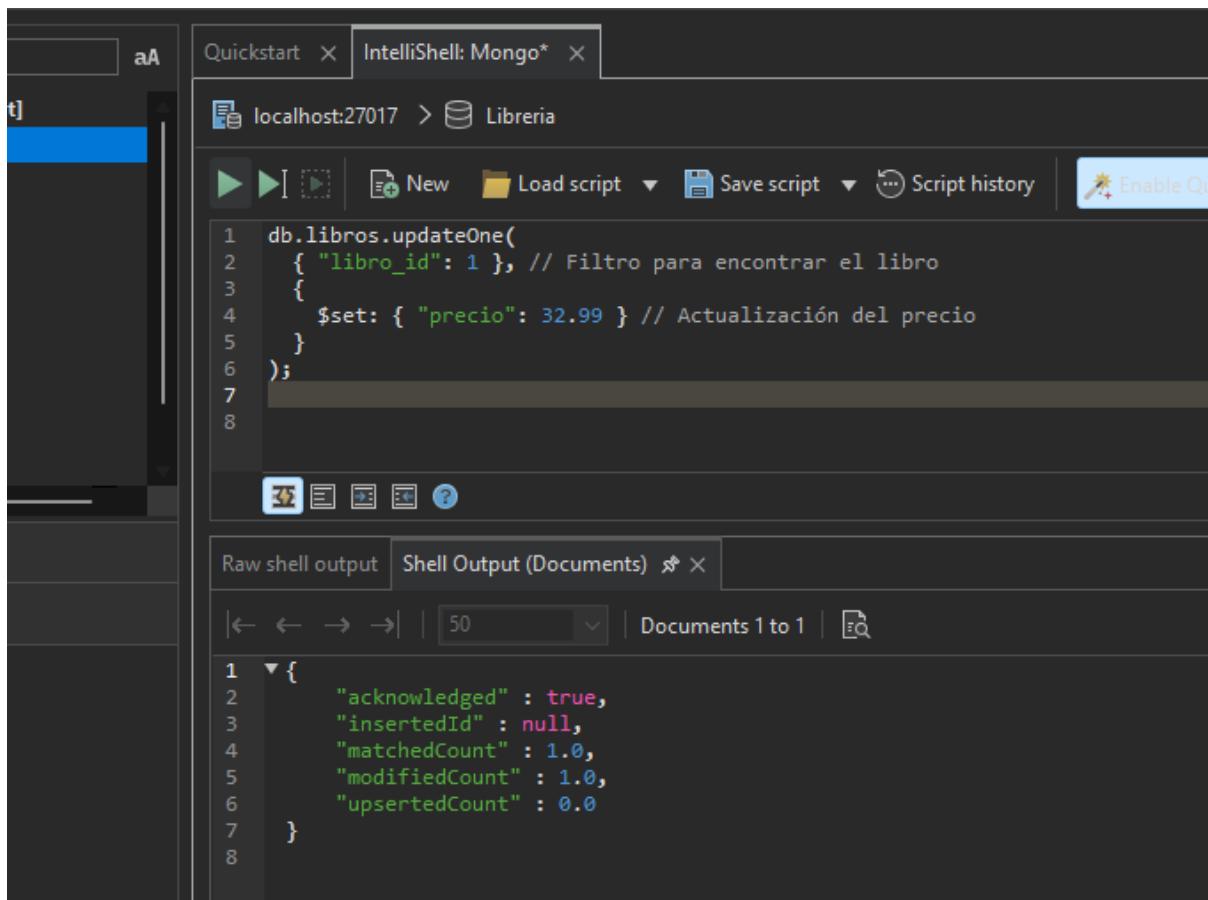
```
1 db.clientes.updateOne(
2   { "cliente_id": 1 }, // Filtro para encontrar al cliente
3   {
4     $push: {
5       "prestamos": {
6         "prestamo_id": 3,
7         "fecha_prestamo": "2024-10-10",
8         "fecha_devolucion": "2024-10-25",
9         "libros": [
10           {
11             "libro_id": 2,
12             "titulo": "El amor en los tiempos del cólera",
13             "cantidad": 1
14           }
15         ],
16       }
17     }
18   }
19 )
```

The output window below shows the response from the database:

```
1 { "acknowledged" : true,
2  "insertedId" : null,
3  "matchedCount" : 1.0,
4  "modifiedCount" : 1.0,
5  "upsertedCount" : 0.0
6 }
```

Actualizar la información de un libro existente

Supongamos que deseas actualizar el precio de un libro específico. Aquí utilizamos `updateOne()` para coincidir por `libro_id`.



The screenshot shows the IntelliJ IDEA interface with the MongoDB Intellishell plugin. The top navigation bar has tabs for 'Quickstart' and 'IntelliShell: Mongo*'. Below the bar, it says 'localhost:27017 > Libreria'. The main area contains a code editor with the following MongoDB update query:

```
1 db.libros.updateOne(  
2   { "libro_id": 1 }, // Filtro para encontrar el libro  
3   {  
4     $set: { "precio": 32.99 } // Actualización del precio  
5   }  
6 );  
7  
8
```

Below the code editor is a toolbar with icons for run, stop, and other shell operations. The bottom panel shows the 'Raw shell output' tab with the results of the update operation:

```
1 {  
2   "acknowledged" : true,  
3   "insertedId" : null,  
4   "matchedCount" : 1.0,  
5   "modifiedCount" : 1.0,  
6   "upsertedCount" : 0.0  
7 }  
8
```

Agregar un nuevo libro a un autor existente (embebido)

Si deseas agregar un nuevo libro a un autor, se utiliza el operador \$addToSet para evitar duplicados.

The screenshot shows the IntelliJ IDEA interface with the MongoDB Intellishell plugin. The top navigation bar has tabs for 'Quickstart' and 'IntelliShell: Mongo*'. The main workspace shows a connection to 'localhost:27017' and a database named 'Libreria'. Below the connection information are standard shell controls: play/pause, step, and a toolbar with 'New', 'Load script', 'Save script', and 'Script hist'. The code editor contains a MongoDB update query:

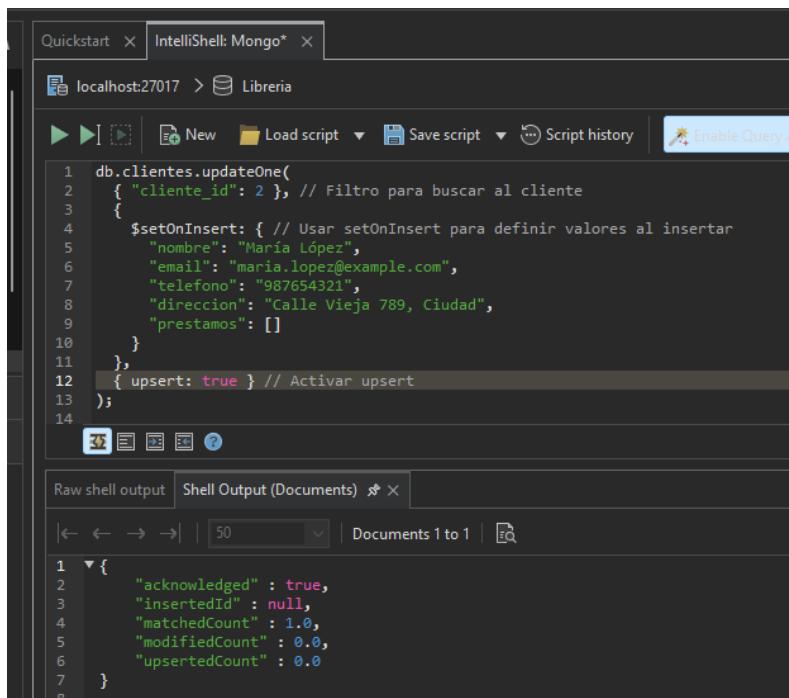
```
1 db.autores.updateOne(
2   { "autor_id": 1 }, // Filtro para encontrar el autor
3   {
4     $addToSet: {
5       "libros": {
6         "libro_id": 3,
7         "titulo": "Crónica de una muerte anunciada",
8         "anio_publicacion": 1981,
9         "genero": "Ficción"
10      }
11    }
12  }
13 );
14
```

Below the code editor is a toolbar with icons for copy, paste, and help. The bottom panel displays the 'Raw shell output' tab, which shows the response from the database:

```
1 { "acknowledged" : true,
2  "insertedId" : null,
3  "matchedCount" : 1.0,
4  "modifiedCount" : 1.0,
5  "upsertedCount" : 0.0
6 }
```

Usar upsert para agregar un nuevo cliente

Si deseas agregar un nuevo cliente solo si no existe, puedes usar `upsert: true`. Esto intentará encontrar el cliente por `cliente_id`, y si no lo encuentra, lo creará.



The screenshot shows the IntelliJ Shell interface with the title bar "IntelliShell: Mongo*". Below it, the URL "localhost:27017 > Librería" is displayed. The main area contains a code editor with the following MongoDB update script:

```

1 db.clientes.updateOne(
2   { "cliente_id": 2 }, // Filtro para buscar al cliente
3   {
4     $setOnInsert: { // Usar setOnInsert para definir valores al insertar
5       "nombre": "María López",
6       "email": "maria.lopez@example.com",
7       "telefono": "987654321",
8       "direccion": "Calle Vieja 789, Ciudad",
9       "prestamos": []
10    }
11  },
12  { upsert: true } // Activar upsert
13 );
14

```

Below the code editor is a "Raw shell output" tab showing the response from the database:

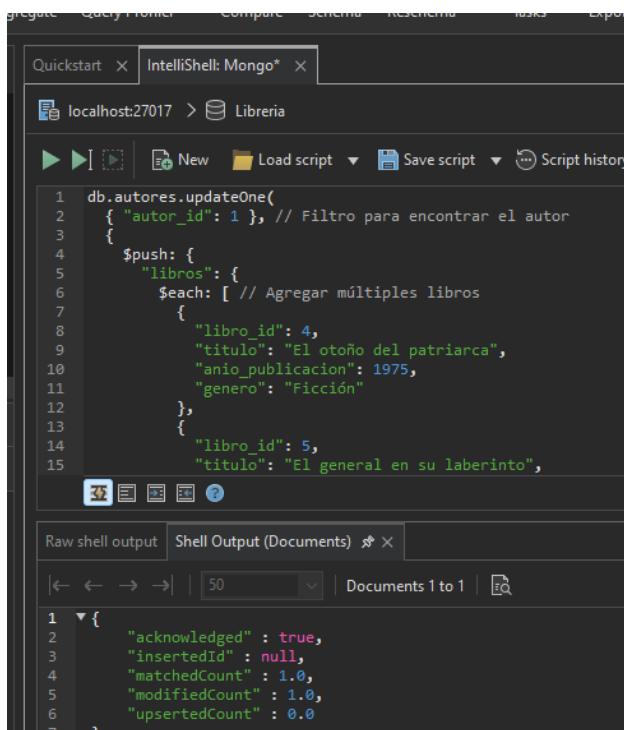
```

1 {
2   "acknowledged" : true,
3   "insertedId" : null,
4   "matchedCount" : 1.0,
5   "modifiedCount" : 0.0,
6   "upsertedCount" : 0.0
7 }

```

Agregar múltiples libros a un autor existente (usando \$each)

Si deseas agregar varios libros a un autor en una sola operación, puedes usar el operador \$push junto con \$each.



The screenshot shows the IntelliJ Shell interface with the title bar "IntelliShell: Mongo*". Below it, the URL "localhost:27017 > Librería" is displayed. The main area contains a code editor with the following MongoDB update script:

```

1 db.autores.updateOne(
2   { "autor_id": 1 }, // Filtro para encontrar el autor
3   {
4     $push: {
5       "libros": {
6         $each: [ // Agregar múltiples libros
7           {
8             "libro_id": 4,
9             "titulo": "El otoño del patriarca",
10            "anio_publicacion": 1975,
11            "genero": "Ficción"
12          },
13          {
14            "libro_id": 5,
15            "titulo": "El general en su laberinto",
16          }
17        ]
18      }
19    }
20  )
21

```

Below the code editor is a "Raw shell output" tab showing the response from the database:

```

1 {
2   "acknowledged" : true,
3   "insertedId" : null,
4   "matchedCount" : 1.0,
5   "modifiedCount" : 1.0,
6   "upsertedCount" : 0.0
7 }

```

Análisis y Discusión

Se logró crear y gestionar eficientemente varias bases de datos y colecciones, con un enfoque en operaciones CRUD. Las consultas realizadas permitieron analizar correctamente los datos almacenados y optimizar su estructura para mejorar el rendimiento.

El uso de embebidos en MongoDB permite un acceso eficiente a datos relacionados, como la relación entre autores y libros, o clientes y préstamos. Este enfoque mejora el rendimiento en consultas donde es necesario acceder a múltiples documentos de forma recurrente. Sin embargo, debe tenerse cuidado con la redundancia de datos, ya que embebidos pueden generar duplicación de información.

Conclusiones

MongoDB es una herramienta poderosa para la gestión de datos no estructurados y semi-estructurados. Permite una gran flexibilidad, especialmente cuando se trabaja con grandes volúmenes de datos distribuidos. El uso de herramientas gráficas como MongoDB Compass facilita enormemente la administración y consulta de bases de datos.

Recomendaciones

- Embeber datos solo cuando sea necesario para evitar la duplicación innecesaria.
- Usar índices apropiados para mejorar el rendimiento de las consultas.
- Mantener un esquema flexible para adaptarse a cambios futuros en la estructura de datos.

Referencias

- MongoDB Documentation: <https://www.mongodb.com/docs/>
- Studio 3T Documentation: <https://studio3t.com/>
- Curso de MongoDB avanzado: <https://www.example.com>
- GitHub documentación:
https://github.com/nathalysilva23/CRUD_MONGO_BASEDEDATOSPERSONAL.git