

# Informe Base de Datos

Jessica Khaterine Correa

Ingeniería de Sistemas, Instituto Tecnológico del Putumayo

BD Almacenamiento De Datos Masivo

Brayan Arcos

Octubre de 2024

## INDICE

<b>Resumen Ejecutivo.....</b>	<b>4</b>
<b>Introducción .....</b>	<b>5</b>
<b>Contexto y Motivación.....</b>	<b>5</b>
<b>Alcance del Informe.....</b>	<b>6</b>
<b>Objetivos .....</b>	<b>7</b>
<b>Metodología .....</b>	<b>8</b>
<b>Herramientas Utilizadas.....</b>	<b>8</b>
<b>Procedimientos .....</b>	<b>8</b>
<b>Creación base de datos personal.....</b>	<b>8</b>
<b>Métodos de captura.....</b>	<b>10</b>
<b>Consultas Realizadas y Sus resultados.....</b>	<b>13</b>
<b>Desarrollo del Informe .....</b>	<b>15</b>
<b>Esquemas .....</b>	<b>15</b>
<b>Diseño de Base de Datos Personal .....</b>	<b>17</b>
<b>Modelo de Datos:.....</b>	<b>17</b>
<b>Consideraciones de Diseño .....</b>	<b>18</b>
<b>Métodos de captura BD grupal.....</b>	<b>20</b>
<b>Esquema de las colecciones .....</b>	<b>27</b>
<b>Consultas.....</b>	<b>30</b>

<b>Análisis y Discusión.....</b>	<b>54</b>
<b>Interpretación de Resultados .....</b>	<b>54</b>
<b>Conclusiones .....</b>	<b>55</b>
<b>Recomendaciones .....</b>	<b>56</b>
<b>Bibliografía .....</b>	<b>57</b>

## **Resumen Ejecutivo**

Este informe presenta un análisis detallado sobre el uso de MongoDB, una base de datos NoSQL, para la gestión de grandes volúmenes de datos en diversas aplicaciones, una librería y un sistema escolar. A lo largo del documento se abordan aspectos clave como el diseño de bases de datos, las relaciones entre colecciones y la eficiencia en la ejecución de consultas.

MongoDB se destaca por su flexibilidad en el manejo de datos no estructurados y semi-estructurados, permitiendo el desarrollo de esquemas dinámicos que pueden ajustarse a las necesidades de cada aplicación sin la rigidez de las bases de datos relacionales tradicionales. En los casos analizados, se demuestra cómo MongoDB facilita la creación de relaciones uno a uno, uno a muchos y muchos a muchos entre diferentes colecciones, optimizando el rendimiento de las consultas sin duplicación de datos.

Además, se realizaron consultas avanzadas que incluyen la búsqueda, actualización y eliminación de datos, mostrando la capacidad de MongoDB para procesar información de forma rápida y eficiente, incluso cuando se trata de datos complejos y anidados. Las pruebas realizadas revelaron que MongoDB es una opción robusta para aplicaciones modernas que necesitan escalabilidad y flexibilidad, brindando un acceso rápido a los datos y una estructura eficiente para manejar relaciones entre múltiples colecciones.

El informe concluye con recomendaciones para mejorar el rendimiento de MongoDB mediante la optimización de índices, la automatización de consultas frecuentes y el monitoreo

del sistema a medida que los volúmenes de datos crecen. Se sugiere, además, una revisión continua de la estructura de las colecciones y de las consultas ejecutadas para garantizar la escalabilidad y eficiencia a largo plazo.

## **Introducción**

### **Contexto y Motivación**

El objetivo principal del informe es demostrar la utilidad y eficiencia de MongoDB, una base de datos NoSQL, en el manejo de grandes volúmenes de datos y en la adaptación a las necesidades de sistemas que requieren flexibilidad y escalabilidad. MongoDB es especialmente relevante para aplicaciones modernas que procesan datos no estructurados o semi-estructurados, sistemas de gestión escolar y tiendas en línea, donde el volumen de datos puede crecer rápidamente y donde se necesitan consultas complejas en tiempo real. A medida que las aplicaciones digitales aumentan en complejidad, las limitaciones de las bases de datos relacionales tradicionales se hacen más evidentes. Aquí es donde entra MongoDB, ofreciendo una alternativa que no solo soporta escalabilidad horizontal, sino que también permite un diseño de esquemas más dinámico y adaptable.

En este contexto, el informe se justifica por la creciente adopción de bases de datos NoSQL en aplicaciones de alto rendimiento. MongoDB permite almacenar datos en formato JSON, facilitando la integración con aplicaciones modernas, y ofrece una mejor gestión de grandes cantidades de datos sin las restricciones de esquemas rígidos, como en las bases de datos SQL tradicionales. Este informe, por lo tanto, ofrece una guía práctica sobre cómo MongoDB

puede mejorar los procesos de almacenamiento, actualización y consulta de datos en distintos escenarios reales.

### **Alcance del Informe**

El informe se centra en cubrir aspectos clave del uso de MongoDB en diversas aplicaciones. Incluye la explicación y ejemplos de cómo diseñar bases de datos utilizando colecciones, definir relaciones entre datos (uno a uno, uno a muchos, y muchos a muchos) y realizar consultas complejas para extraer información relevante. Se abordan tres estudios de caso específicos que muestran la aplicabilidad de MongoDB en escenarios reales:

**Librería :** El caso de estudio se enfoca en una librería donde se gestionan colecciones de clientes, libros, autores, categorías y compras. Se demuestra cómo MongoDB puede manejar transacciones y consultas sobre los productos disponibles, así como las compras de los clientes

**Sistema Escolar:** En este escenario, se utilizan consultas sobre datos de calificaciones de estudiantes, mostrando la flexibilidad de MongoDB para trabajar con datos anidados y realizar filtros específicos que faciliten el análisis del rendimiento académico(informe bd jueves).

Cada uno de estos casos resalta cómo MongoDB permite el manejo eficiente de datos en diferentes contextos, ajustándose a las necesidades particulares de cada aplicación.

## **Objetivos**

Este informe tiene los siguientes objetivos principales:

Demostrar la capacidad de MongoDB para manejar datos masivos y variados: A través de ejemplos prácticos, se muestra cómo MongoDB permite gestionar grandes volúmenes de información sin comprometer el rendimiento, permitiendo el acceso rápido a los datos.

Ilustrar la flexibilidad del diseño de esquemas en MongoDB: Se explica cómo MongoDB permite diseñar esquemas flexibles, donde las colecciones pueden evolucionar sin necesidad de alterar la estructura general de la base de datos, a diferencia de las bases de datos relacionales tradicionales.

Proporcionar ejemplos concretos de consultas y actualizaciones de datos en MongoDB: Se incluyen ejemplos de consultas NoSQL que destacan las ventajas de trabajar con un sistema orientado a documentos, así como casos de uso que involucran actualizaciones con métodos avanzados como `upsert`, `addToSet`, y `$each`

## **Metodología**

### **Herramientas Utilizadas**

Studio 3T: Herramienta visual para MongoDB, utilizada para diseñar el esquema y realizar las consultas.

MongoDB: Sistema de base de datos NoSQL utilizado para gestionar los datos.

MongoDB Compass: Herramienta gráfica para gestionar y visualizar datos en MongoDB.

JavaScript: Lenguaje de consulta para interactuar con la base de datos.

### **Procedimientos**

- Creación de la Base de Datos: Se creó la base de datos denominada Librería con cinco colecciones: clientes, libros, autores, categorías, y compras.
- Inserción de Datos: Se llenaron las colecciones con datos representativos de los clientes, libros, autores, categorías, y compras realizadas en la librería.
- Consultas NoSQL: Se ejecutaron consultas para extraer información de las colecciones, como los libros comprados por un cliente, las categorías de libros disponibles y los detalles de los autores.

## **Métodos**

### **Creación base de datos personal**

Estructura de la base de datos

1. Colección Libros: Representa los libros disponibles en la librería.
  - `_id`
  - `titulo`



- autor\_id (relación 1 a 1 con Autores)
  - categorias\_ids (relación muchos a muchos con Categorias)
  - precio
  - stock
2. Colección Autores: Representa a los autores de los libros.
- \_id
  - nombre
  - bio
  - nacionalidad
3. Colección Categorias: Representa las categorías o géneros a los que pertenecen los libros.
- \_id
  - nombre
4. Colección Clientes: Representa a los clientes de la librería.
- \_id
  - nombre
  - email
  - compras\_ids (relación 1 a muchos con Compras)
5. Colección Compras: Registra las compras realizadas por los clientes.
- \_id
  - cliente\_id (relación 1 a muchos con Clientes)
  - libros\_ids (relación muchos a muchos con Libros)

- fecha\_compra
- total

### Métodos de captura

- Creación de base de datos **librería**:

```
>_MONGOSH
> use Libreria;
< switched to db Libreria
Libreria> db.createCollection("")
```

- Inserción de datos de la tabla **autores**:

```
>_MONGOSH
> use Libreria;
< switched to db Libreria
> db.autores.insertMany([
  { _id: 1, nombre: "Gabriel García Márquez", bio: "Autor colombiano", nacionalidad: "Colombiano" },
  { _id: 2, nombre: "Isabel Allende", bio: "Autora chilena", nacionalidad: "Chilena" }
]);
< {
  acknowledged: true,
  insertedIds: {
    '0': 1,
    '1': 2
  }
}
Libreria>|
```

- Inserción y creación de datos en la colección **categorías**:

```
> db.categorías.insertMany([
  { _id: 1, nombre: "Realismo Mágico" },
  { _id: 2, nombre: "Novela Histórica" }
]);
< {
  acknowledged: true,
  insertedIds: {
    '0': 1,
    '1': 2
  }
}
Librería>
```

- Inserción y creación de la colección **libros y clientes**:

```
> _MONGOSH
}
}
> db.libros.insertMany([
  { _id: 1, titulo: "Cien años de soledad", autor_id: 1, categorías_ids: [1], precio: 20, stock: 100 },
  { _id: 2, titulo: "La casa de los espíritus", autor_id: 2, categorías_ids: [2], precio: 15, stock: 50 }
]);
< {
  acknowledged: true,
  insertedIds: {
    '0': 1,
    '1': 2
  }
}
> db.clientes.insertMany([
  { _id: 1, nombre: "Juan Pérez", email: "juan@example.com", compras_ids: [] },
  { _id: 2, nombre: "Ana Gómez", email: "ana@example.com", compras_ids: [] }
]);
< {
  acknowledged: true,
  insertedIds: {
    '0': 1,
    '1': 2
  }
}
Librería>
```

- Inserción y creación de la colección **compras**:

```
> db.compras.insertMany([
  { _id: 1, cliente_id: 1, libros_ids: [1, 2], fecha_compra: new Date(), total: 35 },
  { _id: 2, cliente_id: 2, libros_ids: [2], fecha_compra: new Date(), total: 15 }
]);
< {
  acknowledged: true,
  insertedIds: {
    '0': 1,
    '1': 2
  }
}
```

Libreria>

Visualización de toda la base de datos en general

Collection	Storage size	Documents	Avg. document size	Indexes	Total index size
autores	20.48 KB	2	98.00 B	1	20.48 KB
categorias	20.48 KB	3	43.00 B	1	36.86 KB
clientes	20.48 KB	2	83.00 B	1	20.48 KB
compras	20.48 KB	2	94.00 B	1	20.48 KB
libros					

## Consultas Realizadas y Sus resultados

### 1. Actualización de Datos con updateOne():

- Actualizar el autor de un libro con coincidencia puntual (1 a 1):

```
> _MONGOSH
> use Libreria
< switched to db Libreria
> db.libros.updateOne(
  { _id: 1 }, // Coincidencia puntual por _id del libro
  { $set: { autor_id: 2 } } // Cambiamos el autor a Isabel Allende
);
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Libreria>
```

Descripción: Inserta varios documentos en la colección autores, creando registros para Gabriel García Márquez e Isabel Allende, junto con información sobre su biografía y nacionalidad.

### 2. Actualizar o insertar una categoría utilizando upsert():

```
> db.categorias.updateOne(
  { _id: 3 }, // Coincidencia por ID de la categoría
  { $setOnInsert: { nombre: "Ciencia Ficción" } }, // Si no existe, insertar
  { upsert: true }
);
< {
  acknowledged: true,
  insertedId: 3,
  matchedCount: 0,
  modifiedCount: 0,
  upsertedCount: 1
}
Libreria>
```

**Descripción:** Busca una categoría con `_id` igual a 3. Si no se encuentra, se inserta un nuevo documento con el nombre "Ciencia Ficción". Este método combina la búsqueda y la inserción en una sola operación.

### 3. Agregar un libro a una compra utilizando `$each`:

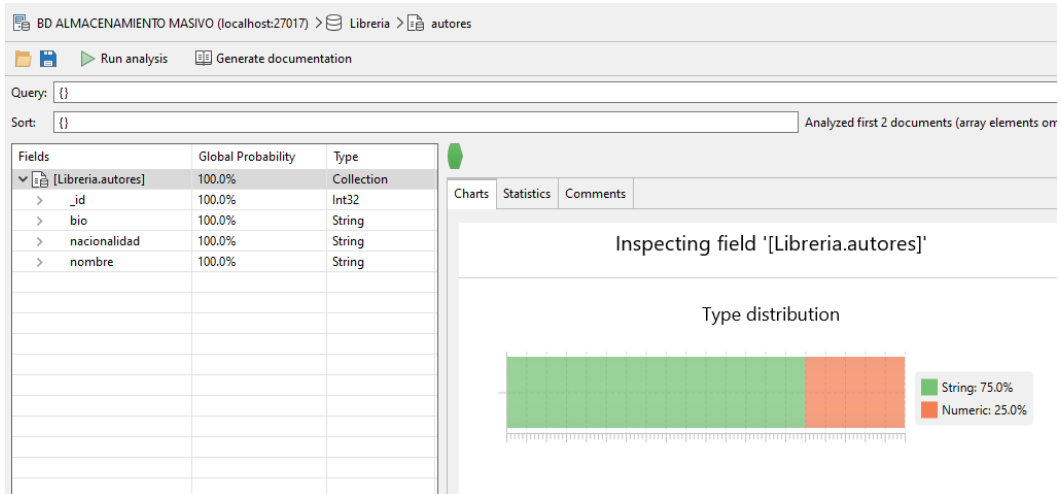
```
> db.compras.updateOne(
  { _id: 1 }, // Coincidencia por ID de la compra
  { $addToSet: { libros_ids: { $each: [3] } } } // Agregamos un nuevo libro a la lista
);
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Libreria>
```

**Descripción:** Busca una compra con `_id` igual a 1 y agrega el ID del libro 3 a la lista `libros_ids`, si aún no está presente. La operación `$addToSet` asegura que no se agreguen duplicados en el array.

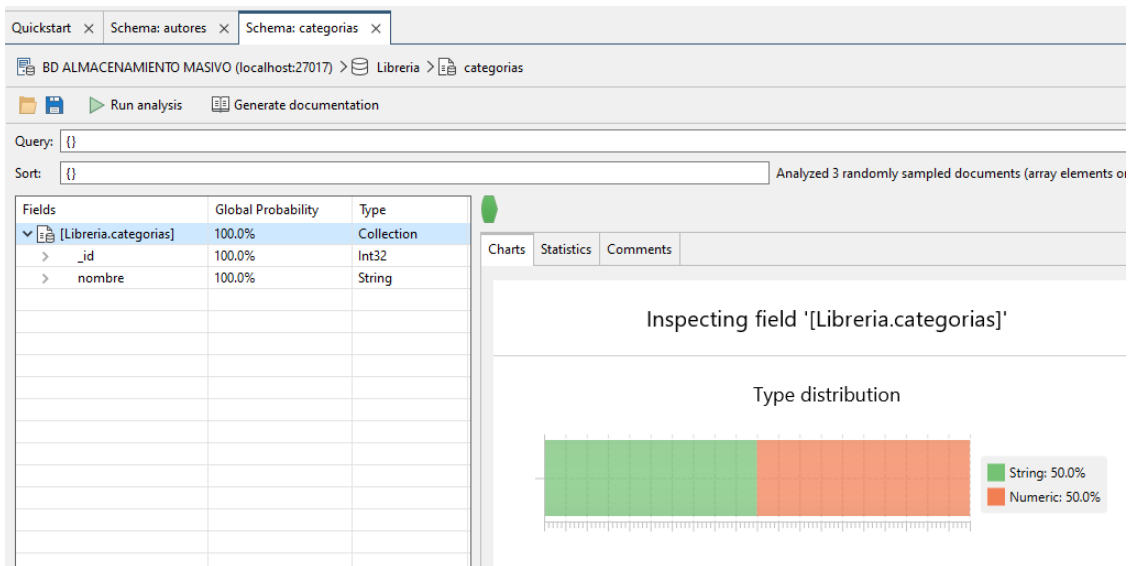
## Desarrollo del Informe

### Esquemas

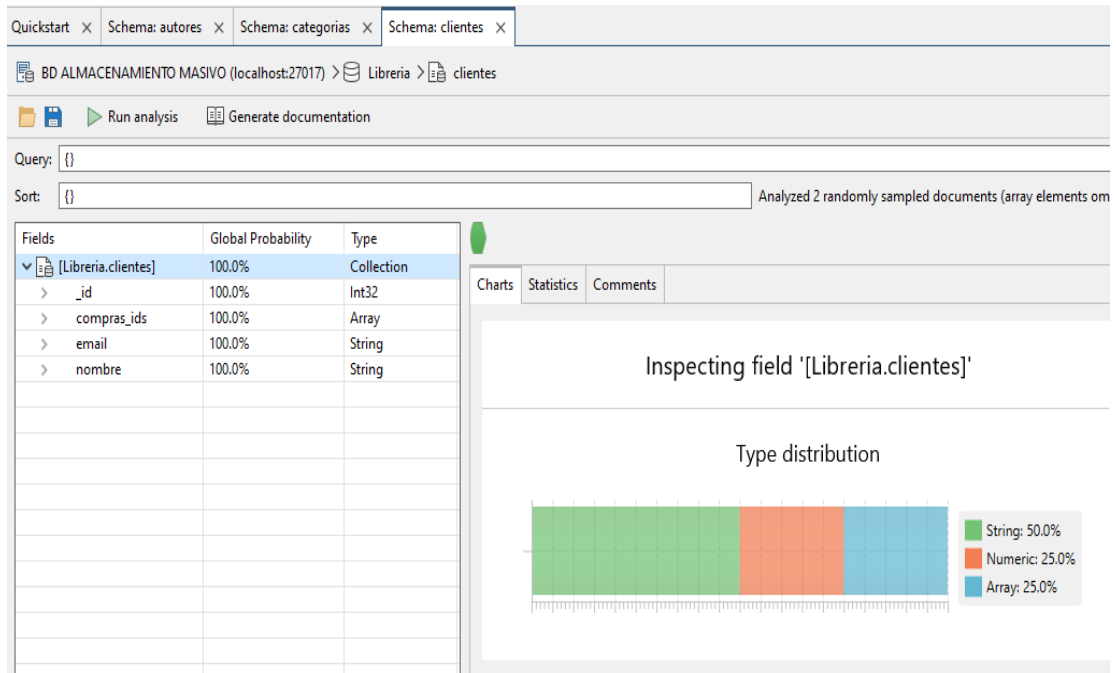
- Esquema de autores:



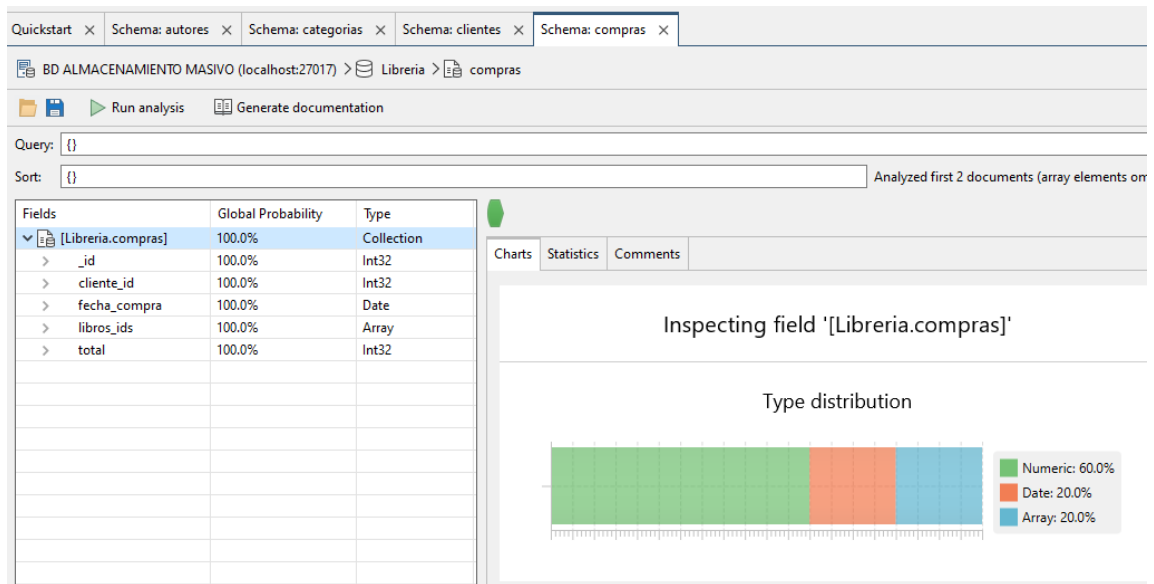
- Esquema Categorías:



- Esquema Clientes:

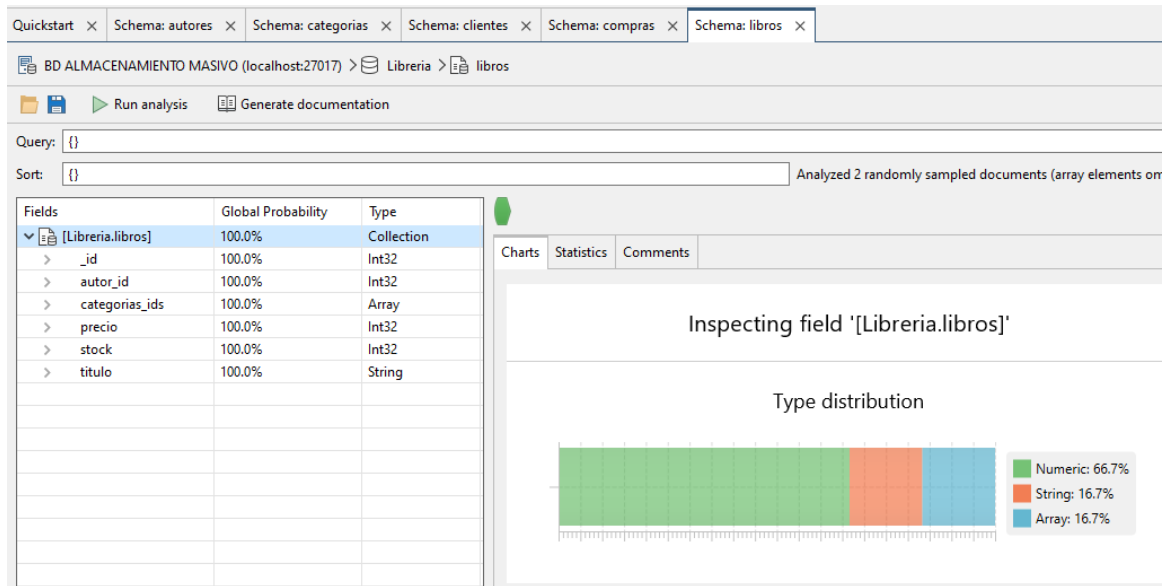


- Esquema Compras:





## - Esquema Libros:



## Diseño de Base de Datos Personal

### Modelo de Datos:

#### Relaciones y Cardinalidad:

- Clientes - Compras: Relación 1 a muchos.

Descripción: Un cliente puede realizar múltiples compras, pero cada compra pertenece a un solo cliente.

Decisión: No se embebe. En la colección clientes, las compras se referencian mediante un array compras\_ids. Esto permite mantener los datos de las compras separados y fácilmente consultables.

- Libros - Autores: Relación 1 a 1.

Descripción: Cada libro tiene un solo autor, y cada autor puede tener uno o más libros.

Decisión: No se embebe. Se usa una referencia en la colección libros hacia el campo autor\_id en autores. Esto permite modificar los datos del autor sin necesidad de actualizar todos los libros asociados.

- Libros - Categorías: Relación muchos a muchos.

Descripción: Un libro puede pertenecer a múltiples categorías, y una categoría puede tener múltiples libros.

Decisión: No se embebe. Se utiliza un campo categorias\_ids en la colección libros para referenciar los IDs de las categorías. Esto facilita la reutilización de las categorías y evita la duplicación de datos.

- Compras - Libros: Relación muchos a muchos.

Descripción: Una compra puede incluir varios libros, y un libro puede formar parte de varias compras.

Decisión: No se embebe. La colección compras contiene un array libros\_ids que referencia los libros comprados. Esto permite escalar las transacciones sin duplicar información.

## **Consideraciones de Diseño**

Relaciones entre colecciones:

- Clientes - Compras: Cada cliente tiene un campo compras\_ids que guarda una lista de las compras asociadas. La colección compras guarda el ID del cliente que realizó la compra en cliente\_id.
- Libros - Autores: En la colección libros, el campo autor\_id guarda el ID del autor correspondiente, mientras que los datos del autor están en la colección autores.

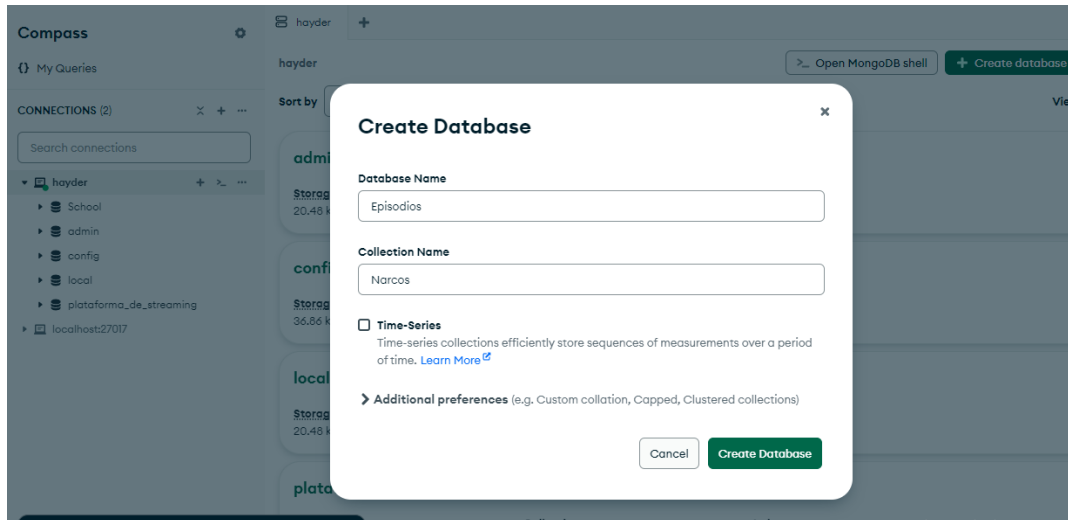
- Libros - Categorías: Los libros tienen un campo `categorías_ids` que guarda una lista de las categorías a las que pertenece el libro. La colección `categorías` no contiene referencias a los libros, lo que evita la duplicación de datos.
- Compras - Libros: En la colección `compras`, el campo `libros_ids` es un array que contiene los IDs de los libros que fueron adquiridos en cada compra.

Definición de campos comunes:

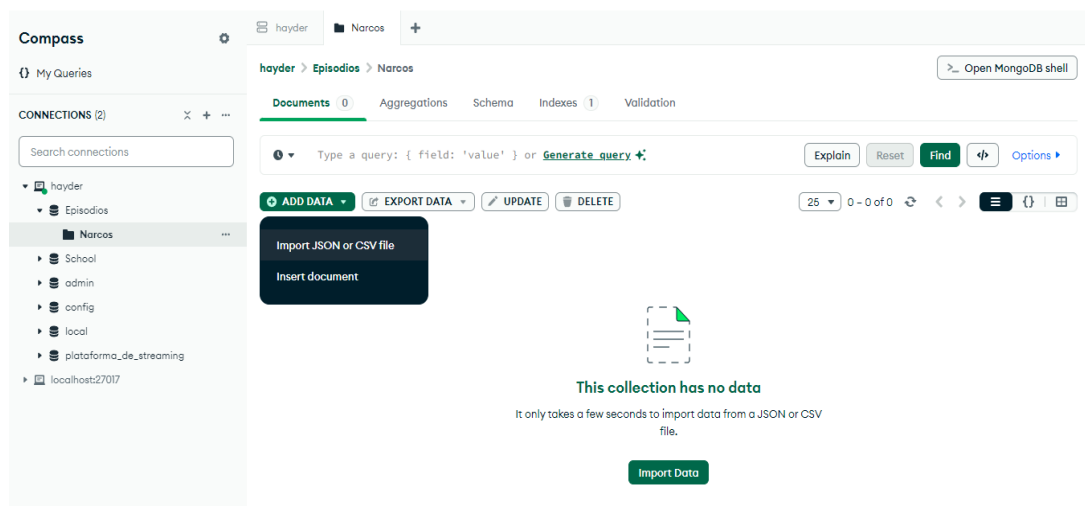
- ID: En todas las colecciones, el campo `_id` es utilizado como clave primaria y referencia entre documentos.
- Nombres: Los campos como `nombre` y `título` en las colecciones (`clientes`, `libros`, `autores`, `categorías`) se usan para identificar los elementos de manera clara.
- Fecha de compra: En la colección `compras`, el campo `fecha_compra` registra el momento en que se realizó la transacción.
- Precio y stock: En la colección `libros`, los campos `precio` y `stock` son importantes para gestionar la disponibilidad y costo de los productos.

## Métodos de captura BD grupal

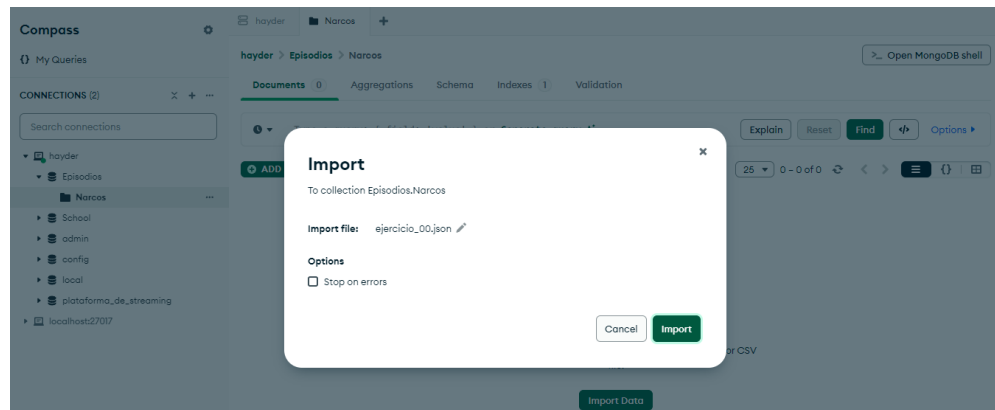
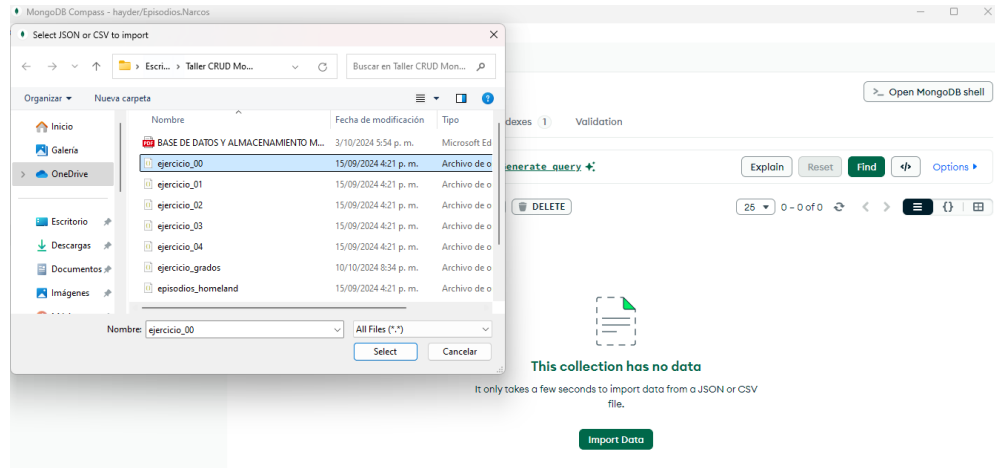
- Creación de la base de datos Episodios y de la primera colección llamada Narcos



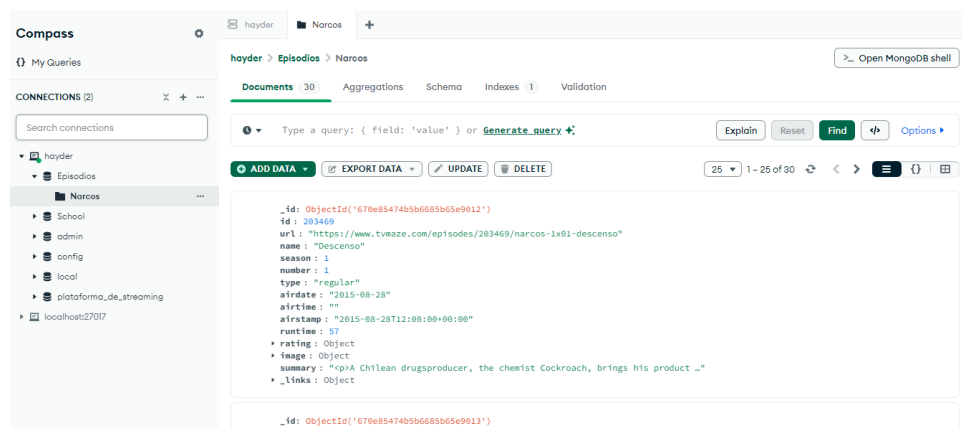
- En esta parte vamos a importar el primer archivo.json de la colección Narcos



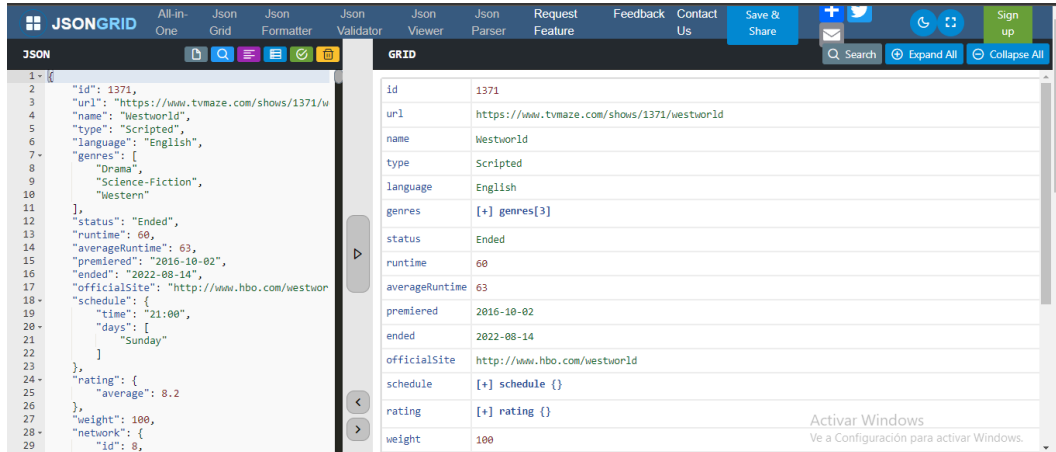
- Demostración de como Importar los archivos



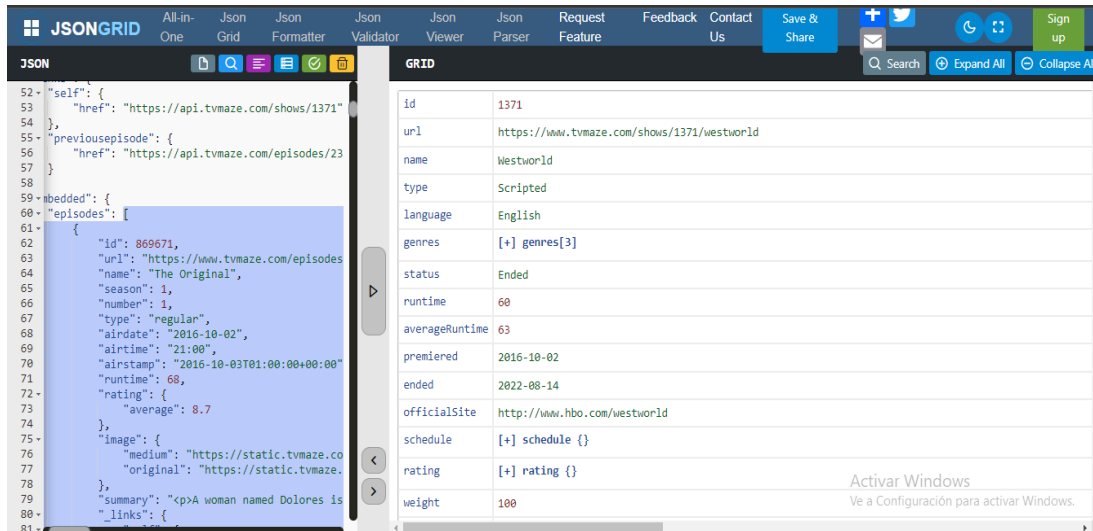
- Resultado de como queda el archivo .json ya importado y como se visualiza en Mongo compass.



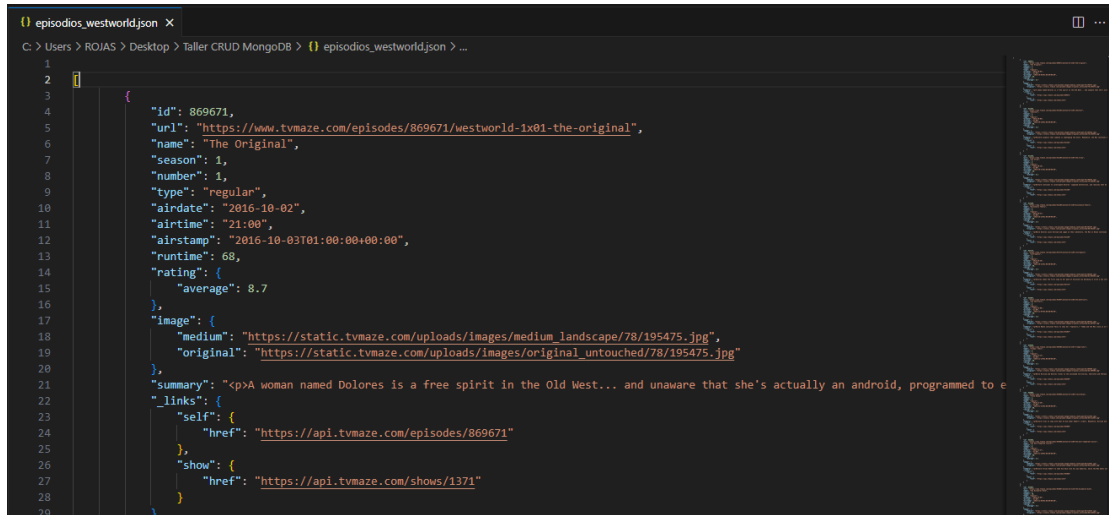
- Ahora añadiamos nuestro archivo llamado “ejercicio\_01” a nuestra herramienta online llamada JSONGRID.



- Luego de añadirlo nos ubicamos en la parte del archivo donde tenemos nuestros documentos de episodios y los copiamos todos.

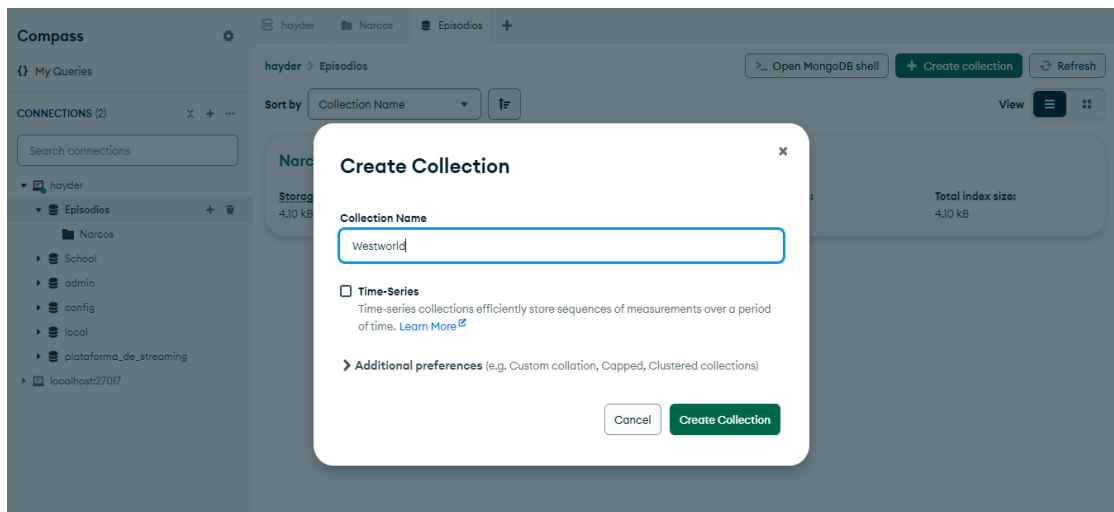


- Abrimos un editor de código y creamos un archivo llamado “episodios\_westworld.Json” para luego pegar los documentos de episodios copiados anteriormente.

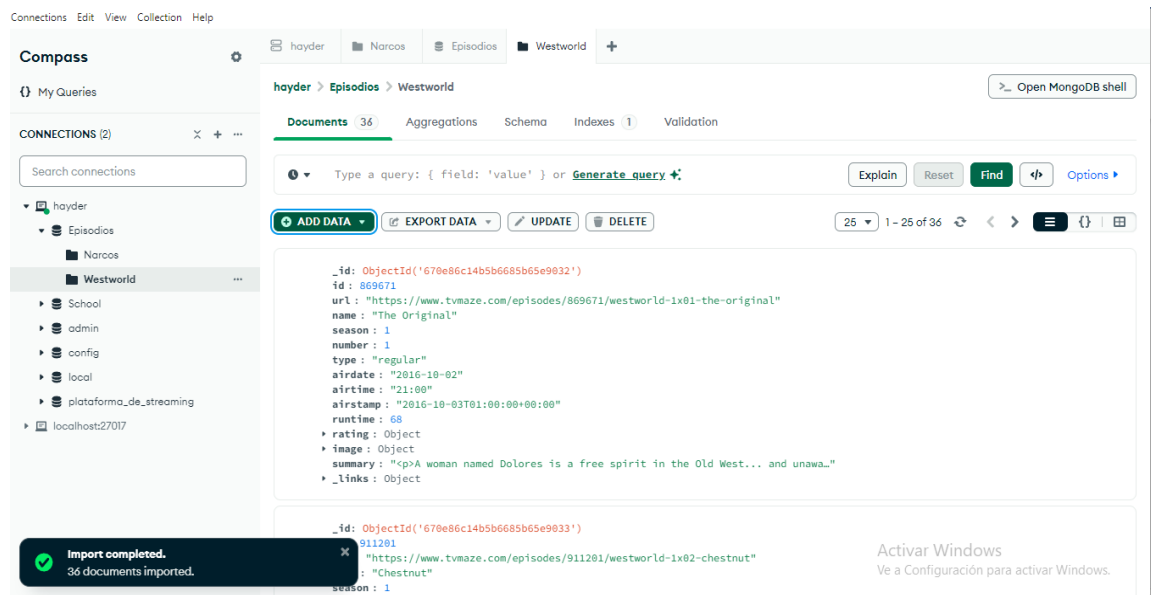


```
1 episodios_westworld.json
2
3
4 {
5   "id": 869671,
6   "url": "https://www.tvmaze.com/episodes/869671/westworld-1x01-the-original",
7   "name": "The Original",
8   "season": 1,
9   "number": 1,
10  "type": "Regular",
11  "airdate": "2016-10-02",
12  "airtime": "21:00",
13  "airstamp": "2016-10-03T01:00:00+00:00",
14  "runtime": 60,
15  "rating": {
16    "average": 8.7
17  },
18  "image": {
19    "medium": "https://static.tvmaze.com/uploads/images/medium_landscape/78/195475.jpg",
20    "original": "https://static.tvmaze.com/uploads/images/original_untouched/78/195475.jpg"
21  },
22  "summary": "<p>A woman named Dolores is a free spirit in the Old West... and unaware that she's actually an android, programmed to e",
23  "links": {
24    "self": {
25      "href": "https://api.tvmaze.com/episodes/869671"
26    },
27    "show": {
28      "href": "https://api.tvmaze.com/shows/1371"
29    }
30  }
31 }
```

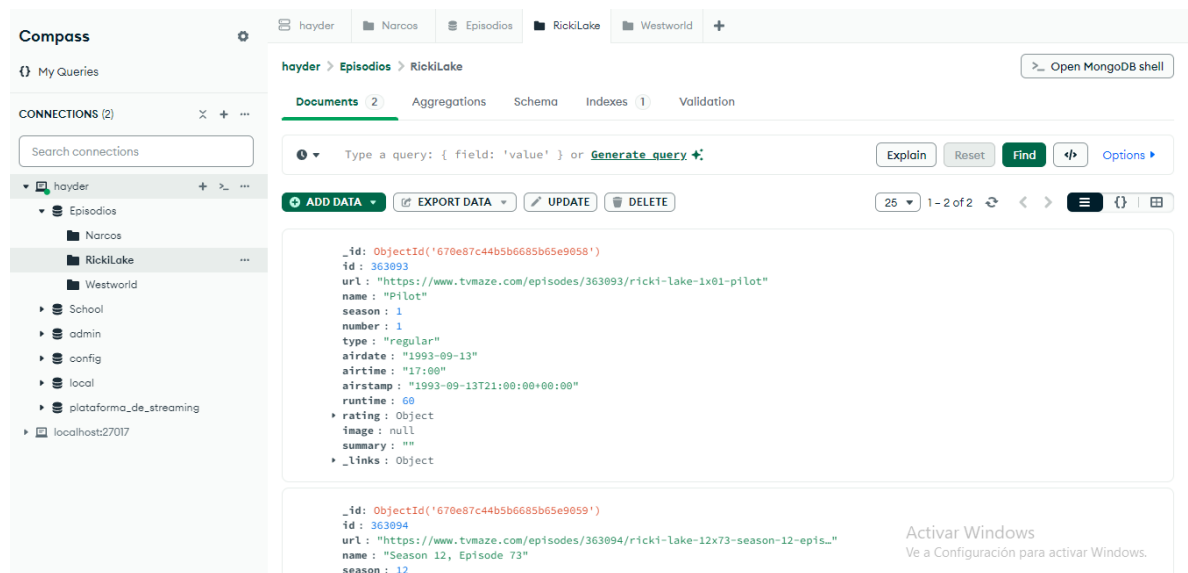
- Ahora en nuestra base de datos “Episodios” creamos otra colección llamada “Westworld”.



- Luego simplemente importamos a la colección el archivo “episodios\_wesworld.json” que creamos.

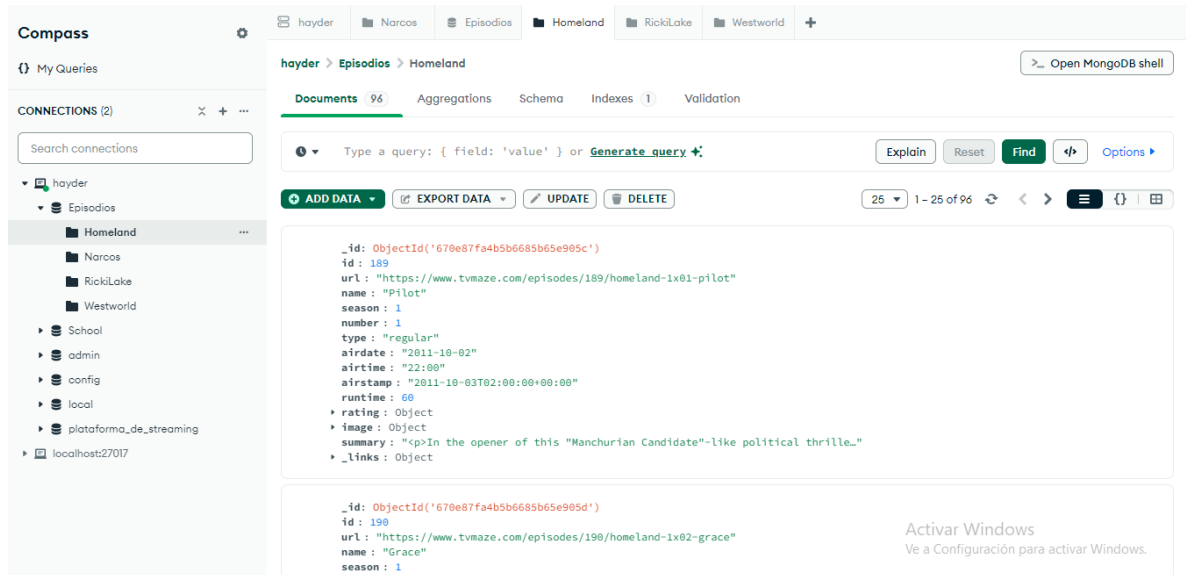


- Repite el paso anterior con el fichero “ejercicio\_02.json” pero esta vez la colección debe llamarse “RickiLake”.

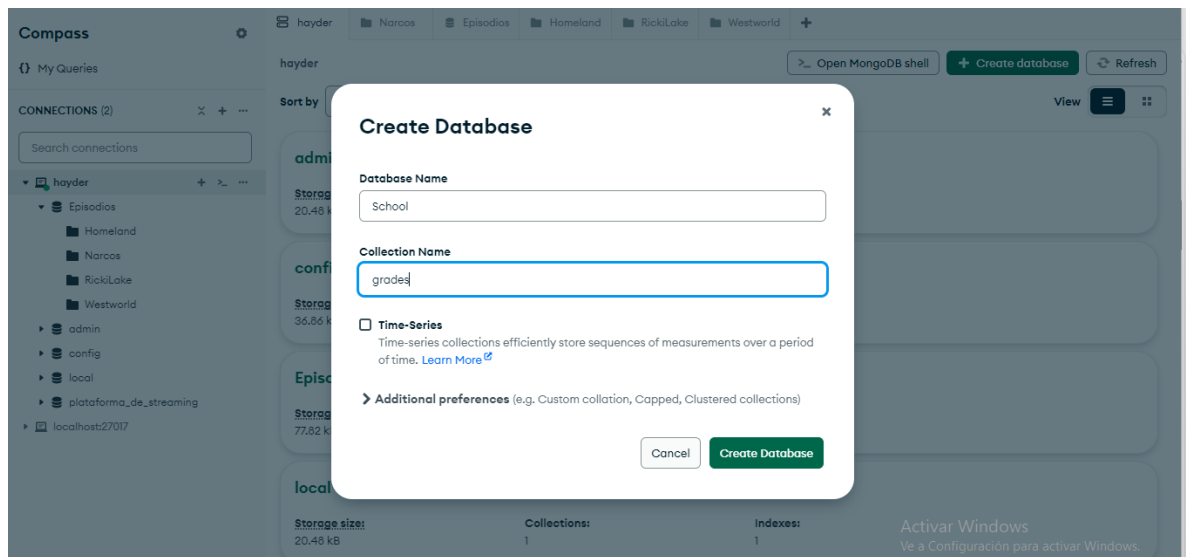




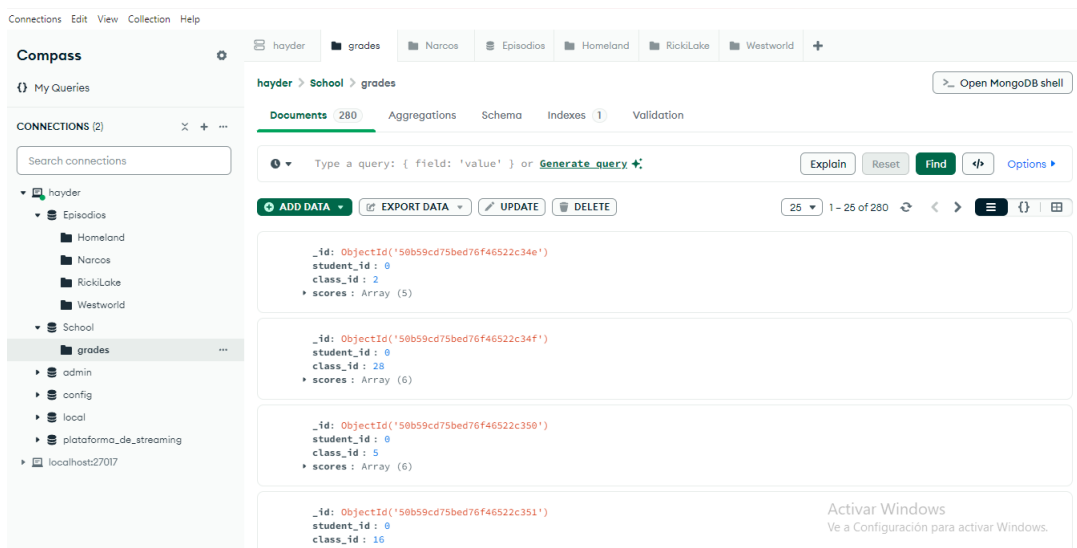
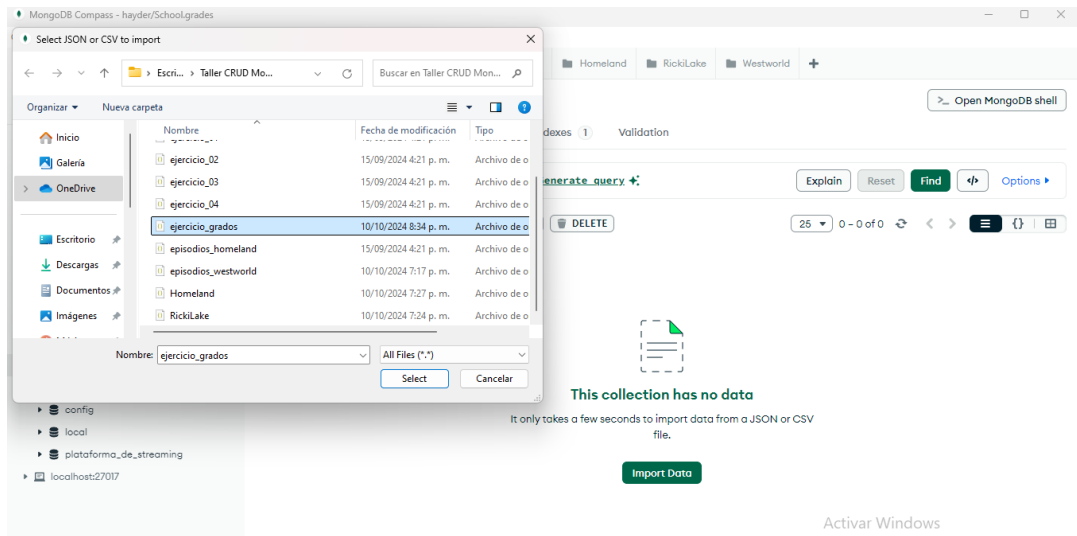
- Repite el paso anterior con el fichero “ejercicio\_03.json” pero esta vez la colección debe llamarse “Homeland”.



- Carga los datos del fichero “ejercicio\_grados.json” en la base de datos School sobre la colección grades

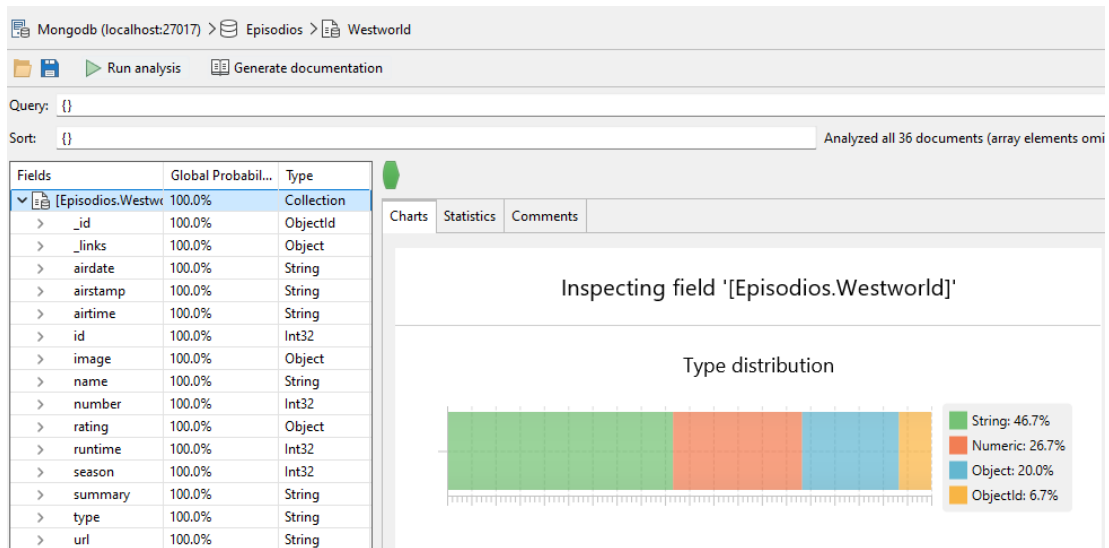


- Luego a nuestra colección “grades” importamos el archivo “ejercicio\_grades.json”.

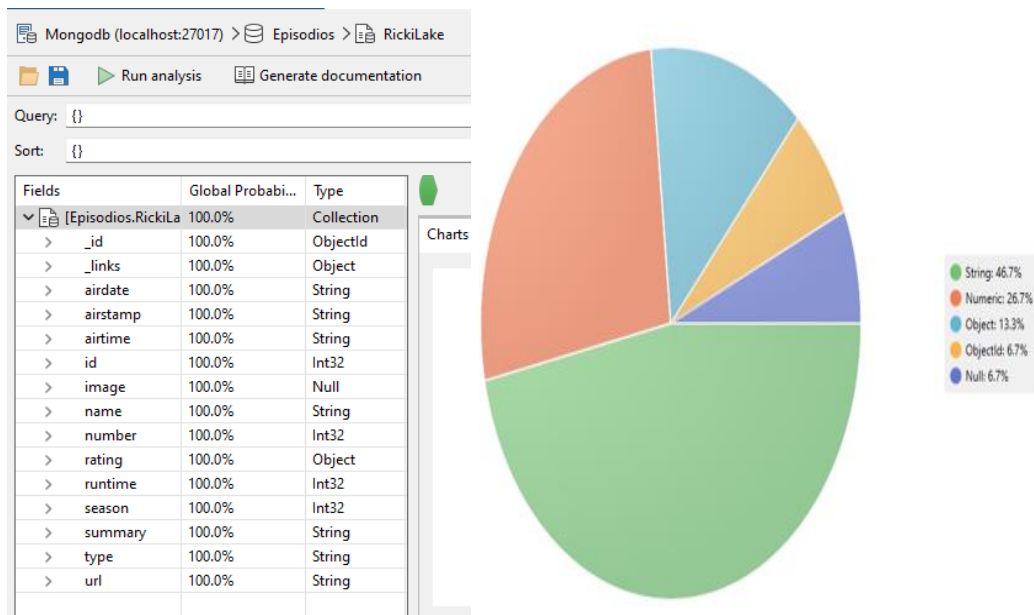


## Esquema de las colecciones

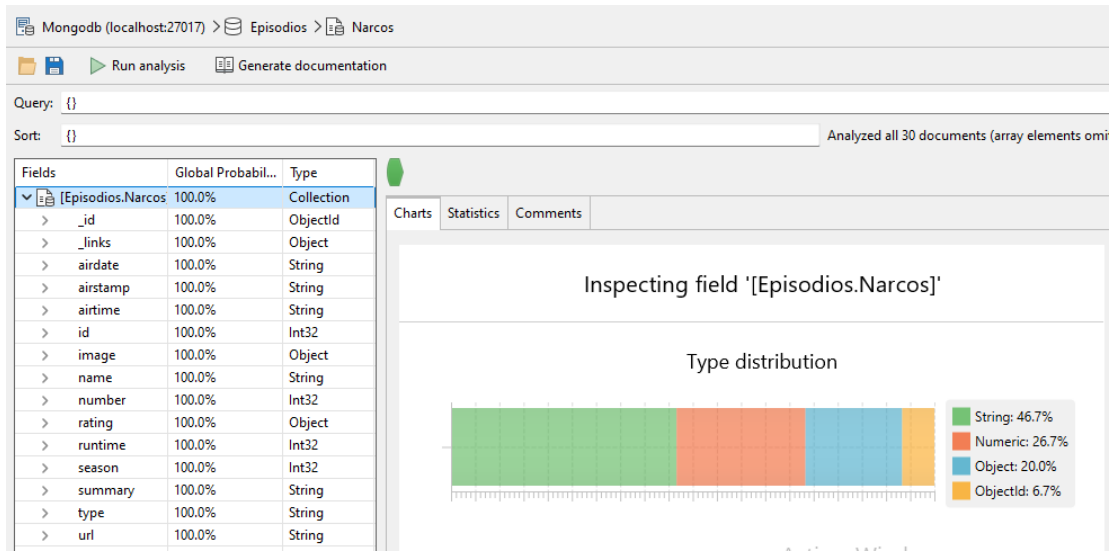
### - Colección “Westworld”:



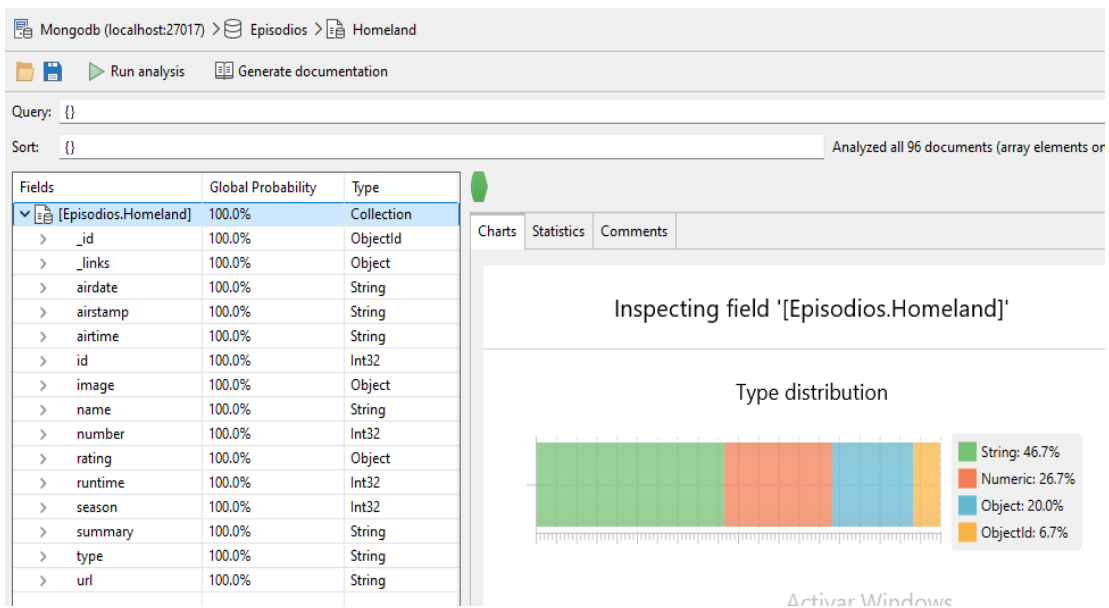
### - Colección “RickiLake”:



## - Colección “Narcos”



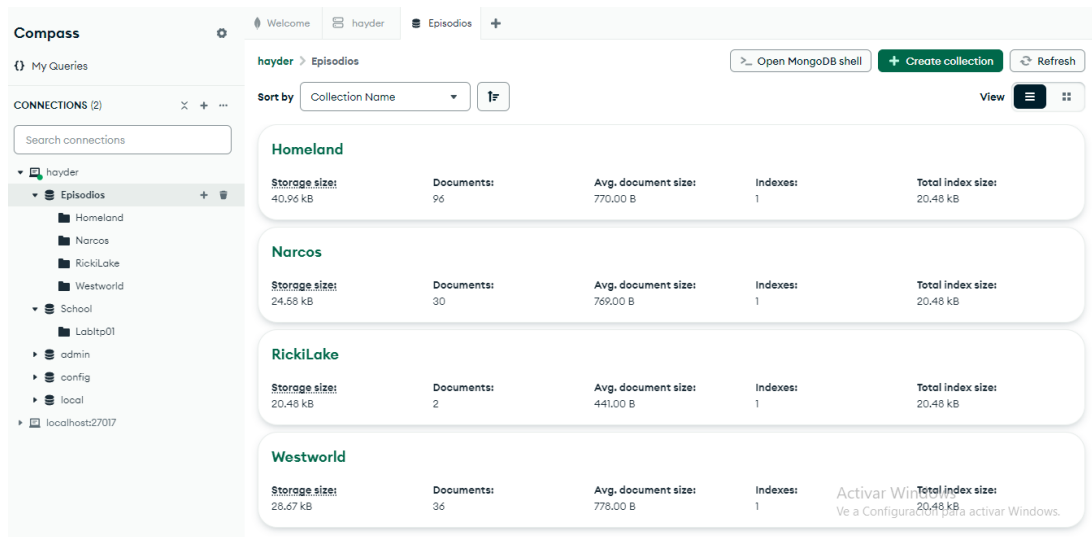
## - Colección “Homeland”



## ESTRUCTURA DE LA BASE DE DATOS Y SUS COLECCIONES

### Base de datos “Episodios”:

Colecciones: Homeland, Narcos, RickiLake, Westworld

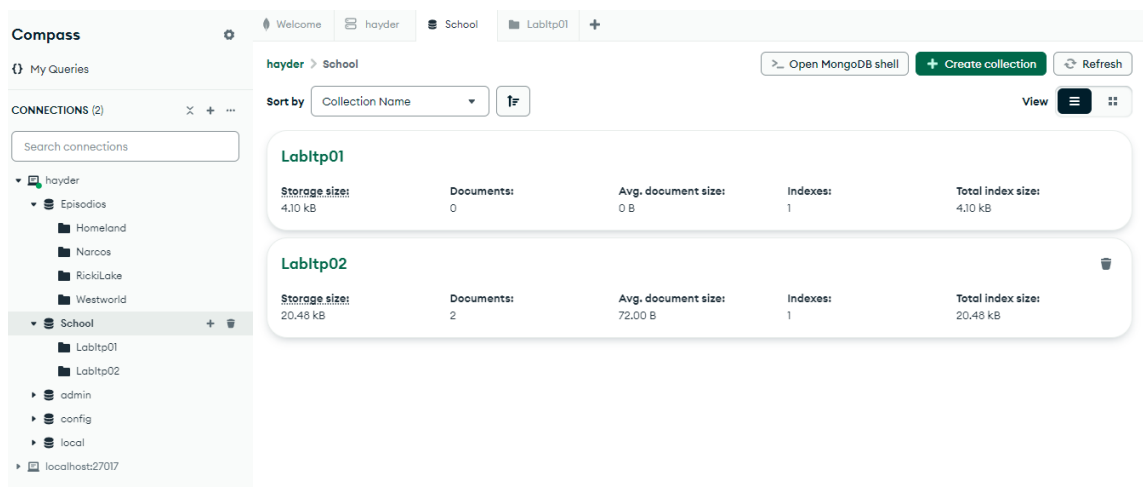


The screenshot shows the MongoDB Compass interface. The left sidebar displays the 'hayder' database with a collection named 'Episodios'. The main panel shows the 'Episodios' database with four collections: Homeland, Narcos, RickiLake, and Westworld. Each collection's details are shown in a card format.

Collection Name	Storage size	Documents	Avg. document size	Indexes	Total index size
Homeland	40.96 kB	96	770.00 B	1	20.48 kB
Narcos	24.58 kB	30	769.00 B	1	20.48 kB
RickiLake	20.48 kB	2	441.00 B	1	20.48 kB
Westworld	28.67 kB	36	778.00 B	1	20.48 kB

### Base de datos “School”:

Colecciones: grades, LabItp01, LabItp02



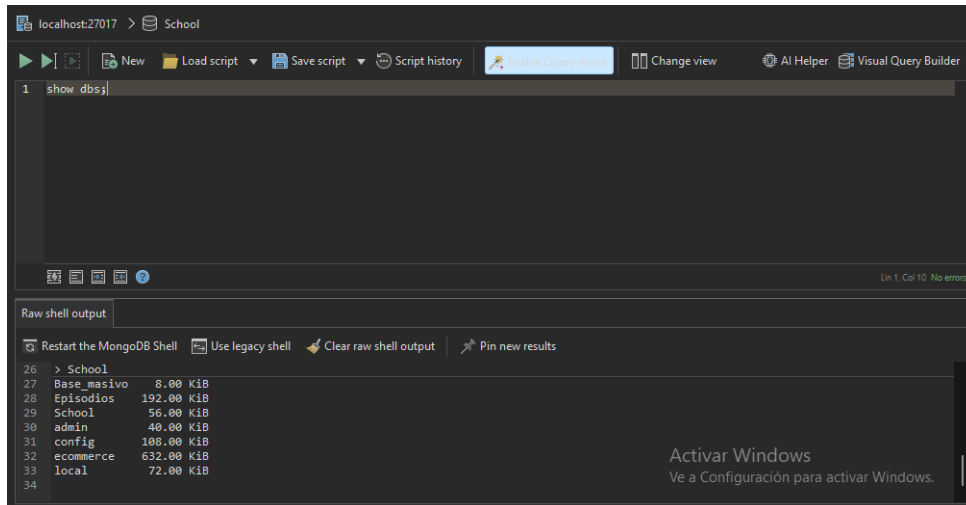
The screenshot shows the MongoDB Compass interface. The left sidebar displays the 'hayder' database with a collection named 'School'. The main panel shows the 'School' database with two collections: LabItp01 and LabItp02. Each collection's details are shown in a card format.

Collection Name	Storage size	Documents	Avg. document size	Indexes	Total index size
LabItp01	4.10 kB	0	0 B	1	4.10 kB
LabItp02	20.48 kB	2	72.00 B	1	20.48 kB

## Consultas

### Trabaja con READ

1. Visualiza todas las bases de datos existentes: show dbs

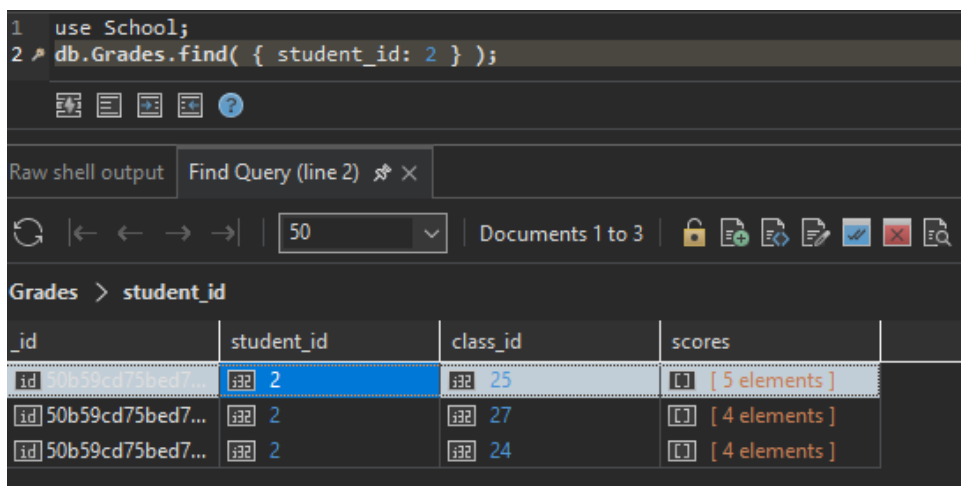


```
localhost:27017 > School
1 show dbs;

Raw shell output
> School
Base_masivo 8.00 KiB
Episodios 192.00 KiB
School 56.00 KiB
admin 40.00 KiB
config 108.00 KiB
ecommerce 632.00 KiB
local 72.00 KiB
```

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 } )**



```
1 use School;
2 db.Grades.find( { student_id: 2 } );

Raw shell output Find Query (line 2)
50 Documents 1 to 3

Grades > student_id

_id student_id class_id scores
[id] 2 [id] 25 [id] [ 5 elements ]
[id] 50b59cd75bed7... [id] 2 [id] 27 [id] [ 4 elements ]
[id] 50b59cd75bed7... [id] 2 [id] 24 [id] [ 4 elements ]
```

**Descripción:** Esta consulta busca todos los documentos en la colección `grades` donde el `student_id` es igual a 2. Recuperará todas las calificaciones asociadas al estudiante con ID 2.

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

The screenshot shows a MongoDB query interface with the following query: `use School;` and `db.grades.find( { "scores.0.score": { $lte: 10 } } );`. The results are displayed in a table view for the `grades` collection, filtered by `student_id`.

_id	student_id	class_id	scores
50b59cd75bed7...	0	24	[ 4 elements ]
50b59cd75bed7...	2	25	[ 5 elements ]
50b59cd75bed7...	3	13	[ 6 elements ]
50b59cd75bed7...	3	16	[ 3 elements ]
50b59cd75bed7...	4	5	[ 5 elements ]
50b59cd75bed7...	4	12	[ 4 elements ]
50b59cd75bed7...	6	22	[ 4 elements ]
50b59cd75bed7...	8	29	[ 3 elements ]
50b59cd75bed7...	11	25	[ 5 elements ]
50b59cd75bed7...	12	13	[ 6 elements ]

**Descripción:** Esta consulta busca documentos donde el primer puntaje en el arreglo `scores` (es decir, el puntaje con índice 0) sea menor o igual a 10. Es útil para identificar calificaciones bajas en el primer examen.

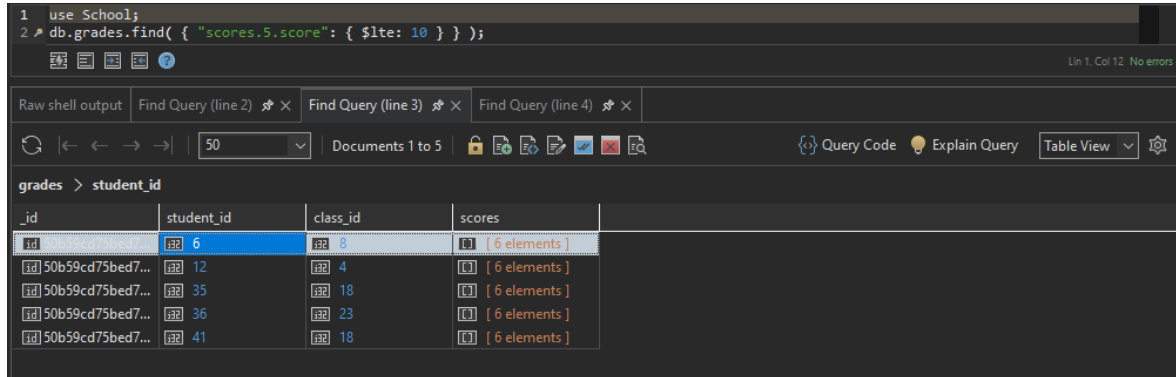
- `db.grades.find( { "scores.4.score": { $lte: 10 } } )`

The screenshot shows a MongoDB query interface with the following query: `use School;` and `db.grades.find( { "scores.4.score": { $lte: 10 } } );`. The results are displayed in a table view for the `grades` collection, filtered by `student_id`.

_id	student_id	class_id	scores
50b59cd75bed7...	0	5	[ 6 elements ]
50b59cd75bed7...	13	22	[ 6 elements ]
50b59cd75bed7...	15	21	[ 5 elements ]
50b59cd75bed7...	15	22	[ 5 elements ]
50b59cd75bed7...	17	19	[ 5 elements ]
50b59cd75bed7...	19	22	[ 5 elements ]
50b59cd75bed7...	24	22	[ 6 elements ]
50b59cd75bed7...	30	0	[ 6 elements ]
50b59cd75bed7...	41	0	[ 5 elements ]
50b59cd75bed7...	42	18	[ 6 elements ]

**Descripción:** Similar a la consulta anterior, esta busca documentos donde el puntaje en la posición 4 del arreglo `scores` sea menor o igual a 10. Esto permite identificar calificaciones deficientes en un examen específico, en este caso, el quinto.

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



The screenshot shows a MongoDB query interface with the following query in the shell:

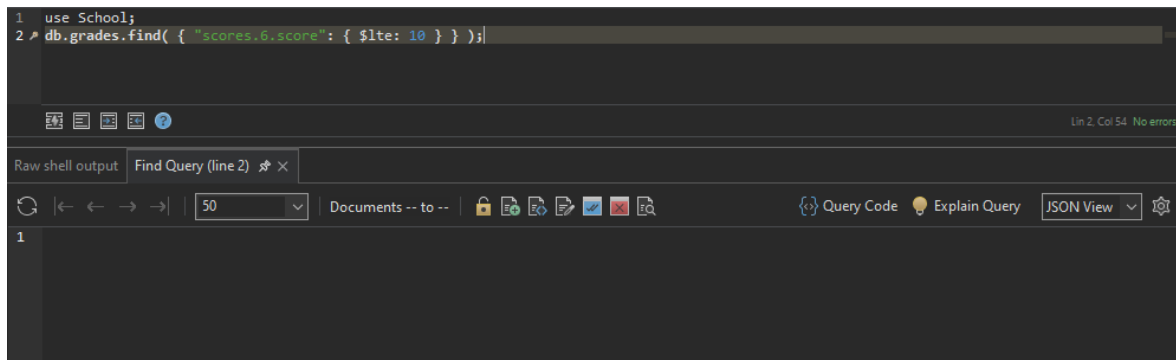
```
1 use School;
2 db.grades.find( { "scores.5.score": { $lte: 10 } } );
```

The results are displayed in a table view with the following columns: `_id`, `student_id`, `class_id`, and `scores`. The table contains 5 rows of data.

_id	student_id	class_id	scores
50b59cd75bed7...	6	8	[ 6 elements ]
50b59cd75bed7...	12	4	[ 6 elements ]
50b59cd75bed7...	35	18	[ 6 elements ]
50b59cd75bed7...	36	23	[ 6 elements ]
50b59cd75bed7...	41	18	[ 6 elements ]

**Descripción:** Esta consulta busca documentos donde el puntaje en la posición 5 del arreglo `scores` sea menor o igual a 10. Ayuda a encontrar estudiantes que tuvieron un mal rendimiento en un examen que ocupa esa posición.

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



The screenshot shows a MongoDB query interface with the following query in the shell:

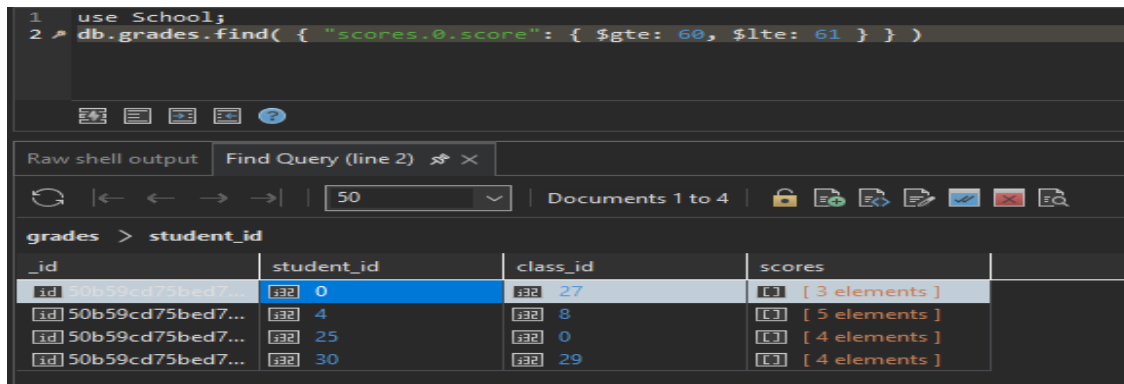
```
1 use School;
2 db.grades.find( { "scores.6.score": { $lte: 10 } } );
```

The results are displayed in a table view, but the table is empty, indicating that no documents were found matching the query criteria.

**Descripción:** En este caso la consulta no muestra nada porque no hay ningún documento de la colección que cumpla con la consulta. La consulta busca documentos donde el puntaje en la posición 6 del arreglo `scores` sea menor o igual a 10. Esto permite detectar calificaciones bajas en un examen específico.



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

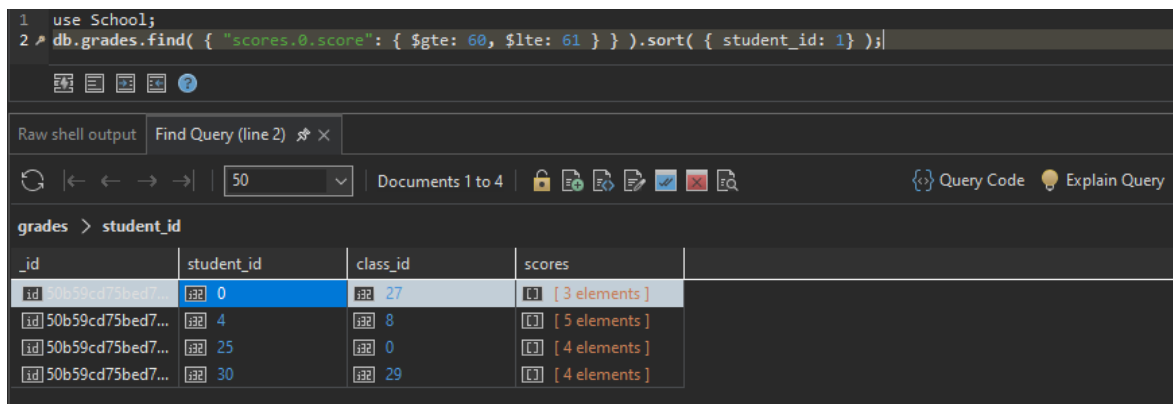


```
1 use School;
2 db.grades.find( { "scores.0.score": { $gte: 60, $lte: 61 } } )
```

_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 ]

**Descripción:** Esta consulta busca documentos donde el puntaje en la posición 0 del arreglo `scores` esté entre 60 y 61, inclusive. Se utiliza para identificar estudiantes que obtuvieron una calificación casi aprobatoria en el primer examen.

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



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

_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 ]

**Descripción:** Similar a la consulta anterior, pero añade un ordenamiento de los resultados por `student_id` en orden ascendente. Esto facilita la visualización de las calificaciones de estudiantes específicos.

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

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

Raw shell output Find Query (line 2) ✕

50 Documents 1 to 1

grades > student\_id

_id	student_id	class_id	scores
id 1635d75ba07	2	24	[ 4 elements ]

**Descripción:** Busca documentos donde `student_id` sea 2 y `class_id` sea 24. Esto permite obtener las calificaciones del estudiante 2 específicamente para la clase con ID 24.

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

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

Raw shell output Find Query (line 2) ✕

50 Documents 1 to 1

grades > student\_id

_id	student_id	class_id	scores
id 1635d75ba07	46	20	[ 5 elements ]

**Descripción:** Esta consulta busca documentos en la clase con ID 20, donde el puntaje en la posición 0 del arreglo `scores` sea mayor o igual a 15 y menor o igual a 30. Se utiliza para encontrar calificaciones que estén en un rango específico en el primer examen.

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

The screenshot shows a MongoDB query interface with the following query: `db.grades.find( { scores: { $elemMatch: { type: 'quiz', score: { $gte: 50 } } } } )`. The results are displayed in a table view with columns: `_id`, `student_id`, `class_id`, and `scores`. The first row is highlighted, showing a student with `student_id` 0 and `class_id` 28, who has 6 quiz scores, all of which are greater than or equal to 50.

_id	student_id	class_id	scores
50b59cd75bed7...	0	28	[ 6 elements ]
50b59cd75bed7...	0	5	[ 6 elements ]
50b59cd75bed7...	0	7	[ 5 elements ]
50b59cd75bed7...	0	27	[ 3 elements ]
50b59cd75bed7...	0	10	[ 4 elements ]
50b59cd75bed7...	1	18	[ 3 elements ]
50b59cd75bed7...	1	28	[ 5 elements ]
50b59cd75bed7...	1	13	[ 4 elements ]
50b59cd75bed7...	2	27	[ 4 elements ]

**Descripción:** Busca documentos donde al menos un elemento en el arreglo scores sea de tipo quiz y tenga un puntaje mayor o igual a 50. Esto permite identificar estudiantes que aprobaron al menos un examen tipo quiz.

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

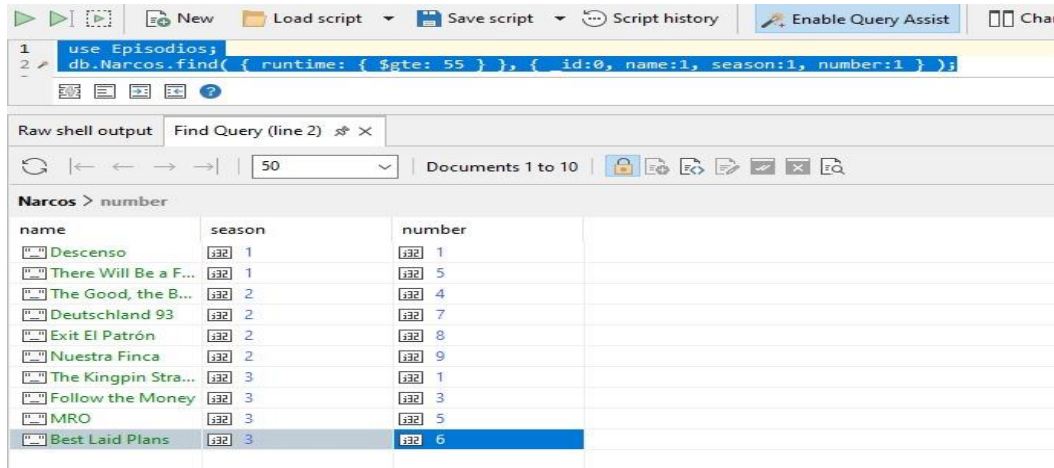
The screenshot shows a MongoDB query interface with the following query: `db.grades.find( { scores: { $elemMatch: { type: 'exam', score: { $gte: 50 } } } } )`. The results are displayed in a table view with columns: `_id`, `student_id`, `class_id`, and `scores`. The first row is highlighted, showing a student with `student_id` 0 and `class_id` 2, who has 5 exam scores, all of which are greater than or equal to 50.

_id	student_id	class_id	scores
50b59cd75bed7...	0	2	[ 5 elements ]
50b59cd75bed7...	0	5	[ 6 elements ]
50b59cd75bed7...	0	16	[ 5 elements ]
50b59cd75bed7...	0	6	[ 6 elements ]
50b59cd75bed7...	0	27	[ 3 elements ]
50b59cd75bed7...	0	11	[ 4 elements ]
50b59cd75bed7...	1	18	[ 3 elements ]
50b59cd75bed7...	1	16	[ 5 elements ]
50b59cd75bed7...	1	13	[ 4 elements ]

**Descripción:** Similar a la consulta anterior, pero busca documentos donde al menos un puntaje en el arreglo scores sea de tipo exam y tenga un puntaje mayor o igual a 50. Se utiliza para identificar estudiantes que aprobaron al menos un examen formal.

1. 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 a MongoDB shell window with the following content:

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

Below the script editor, the 'Raw shell output' tab is active, displaying the results of the query. The results are shown as a table with columns: name, season, and number. The table contains 10 rows of data, with the last row, 'Best Laid Plans', highlighted in blue.

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 Stra...	3	1
Follow the Money	3	3
MRO	3	5
Best Laid Plans	3	6

**Descripción:** Esta consulta busca documentos en la colección `Narcos` donde el `runtime` (duración) sea mayor o igual a 55 minutos. En la proyección de resultados, se excluye el campo `_id` (por eso se usa 0) y se incluyen los campos `name`, `season` y `number`. Esto permite obtener un listado de episodios que cumplen con el tiempo mínimo especificado.

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

The screenshot shows a MongoDB query interface. The query is: `use Episodios; db.Narcos.find( { runtime: { $gte: 15 } }, { _id:0, season:1, number:1 }).sort( { season:1, number:-1 } );`. The results are displayed in a table view for the collection 'Narcos' and field 'number'. The table has two columns: 'season' and 'number'. The results are sorted by 'season' in ascending order and 'number' in descending order.

season	number
1	10
1	9
1	8
1	7
1	6
1	5
1	4
1	3
1	2
1	1

**Descripción:** Esta consulta busca documentos donde el `runtime` sea mayor o igual a 15 minutos. En la proyección, se excluye `_id` y se incluyen `season` y `number`. Además, los resultados se ordenan por `season` en orden ascendente (1) y por `number` en orden descendente (-1). Esto es útil para listar episodios cortos en orden de temporada y con los episodios más recientes al inicio.

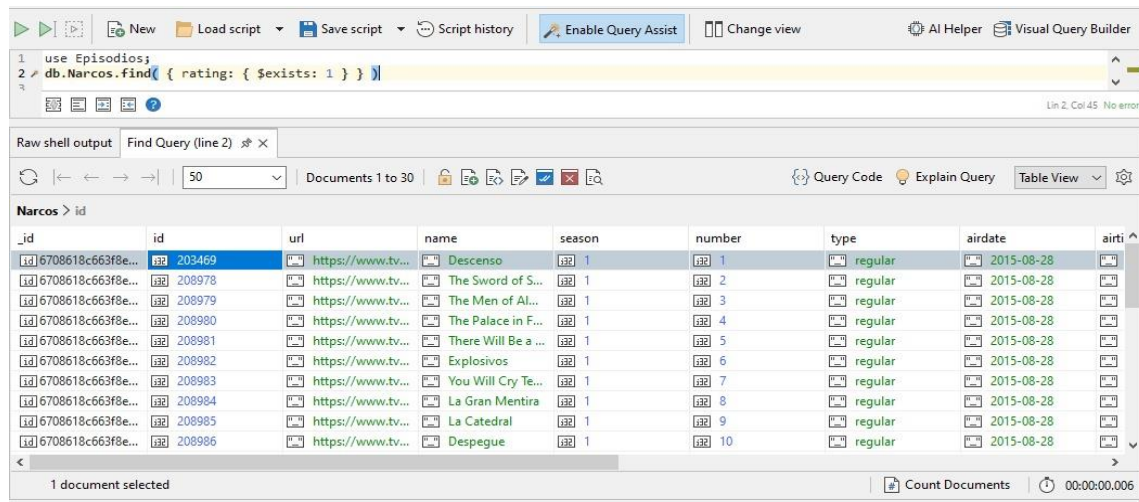
- **db.Narcos.find( { season: { \$type: 'number' } } )**

The screenshot shows a MongoDB query interface. The query is: `use Episodios; db.Narcos.find( { season: { $type: 'number' } } );`. The results are displayed in a table view for the collection 'Narcos' and field 'id'. The table has columns: '\_id', 'id', 'url', 'name', 'season', 'number', 'type', 'airdate', and 'airti'. The results are sorted by 'id' in ascending order.

_id	id	url	name	season	number	type	airdate	airti
6708618c663f8e...	203469	https://www.tv...	Descenso	1	1	regular	2015-08-28	
6708618c663f8e...	208978	https://www.tv...	The Sword of S...	1	2	regular	2015-08-28	
6708618c663f8e...	208979	https://www.tv...	The Men of Al...	1	3	regular	2015-08-28	
6708618c663f8e...	208980	https://www.tv...	The Palace in F...	1	4	regular	2015-08-28	
6708618c663f8e...	208981	https://www.tv...	There Will Be a ...	1	5	regular	2015-08-28	
6708618c663f8e...	208982	https://www.tv...	Explosivos	1	6	regular	2015-08-28	
6708618c663f8e...	208983	https://www.tv...	You Will Cry Te...	1	7	regular	2015-08-28	
6708618c663f8e...	208984	https://www.tv...	La Gran Mentira	1	8	regular	2015-08-28	
6708618c663f8e...	208985	https://www.tv...	La Catedral	1	9	regular	2015-08-28	
6708618c663f8e...	208986	https://www.tv...	Despegue	1	10	regular	2015-08-28	

**Descripción:** Esta consulta busca documentos donde el campo `season` sea del tipo `number`. Se utiliza para filtrar episodios y asegurarse de que el campo `season` está correctamente formateado como un número.`db.Narcos.find( { rating: { $exists: 1 } } )`

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



The screenshot shows the MongoDB Compass interface. The query editor at the top contains the following code:

```
1 use Episodios;
2 db.Narcos.find( { rating: { $exists: 1 } } )
```

Below the query editor, the 'Raw shell output' tab is active, displaying a table of results. The table has the following columns: `_id`, `id`, `url`, `name`, `season`, `number`, `type`, `airdate`, and `airti`. The results show 10 documents, all with `season` as a number and `type` as 'regular'.

_id	id	url	name	season	number	type	airdate	airti
6708618c663f8e...	203469	https://www.tv...	Descenso	1	1	regular	2015-08-28	
6708618c663f8e...	208978	https://www.tv...	The Sword of S...	1	2	regular	2015-08-28	
6708618c663f8e...	208979	https://www.tv...	The Men of AL...	1	3	regular	2015-08-28	
6708618c663f8e...	208980	https://www.tv...	The Palace in F...	1	4	regular	2015-08-28	
6708618c663f8e...	208981	https://www.tv...	There Will Be a ...	1	5	regular	2015-08-28	
6708618c663f8e...	208982	https://www.tv...	Explosivos	1	6	regular	2015-08-28	
6708618c663f8e...	208983	https://www.tv...	You Will Cry Te...	1	7	regular	2015-08-28	
6708618c663f8e...	208984	https://www.tv...	La Gran Mentira	1	8	regular	2015-08-28	
6708618c663f8e...	208985	https://www.tv...	La Catedral	1	9	regular	2015-08-28	
6708618c663f8e...	208986	https://www.tv...	Despegue	1	10	regular	2015-08-28	

**Descripción:** Busca documentos donde el campo `rating` existe. Esto es útil para identificar todos los episodios que tienen una calificación asignada, independientemente de su valor.`db.Narcos.find( { rating: { $exists: 1 }, rating: { $type: "string" } } )`

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

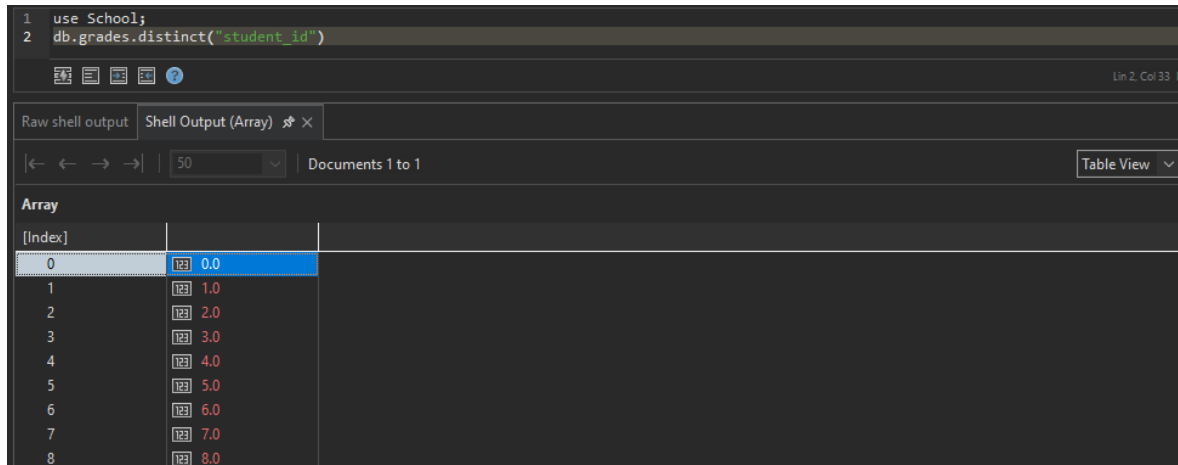
**Descripción:** Esta consulta busca documentos donde el campo `rating` existe y, además, es del tipo `string`. Esto puede ayudar a identificar episodios que tienen calificaciones almacenadas como cadenas de texto, lo que podría ser un error si se espera que sean números.

**LA CONSULTA NO FUNCIONA EN MONGO POR:** El problema con esta consulta es que estás intentando usar dos condiciones en el mismo campo (`rating`) al mismo tiempo. MongoDB no permite que un campo tenga múltiples condiciones en el mismo nivel. En este

caso, el primer criterio verifica si rating existe, y el segundo verifica si su tipo es string. Sin embargo, el segundo criterio sobrescribe el primero.

## 1. Consultas de la base de datos School:

- **db.grades.distinct("student\_id")**



```
1 use School;
2 db.grades.distinct("student_id")
```

Raw shell output | Shell Output (Array) ✖

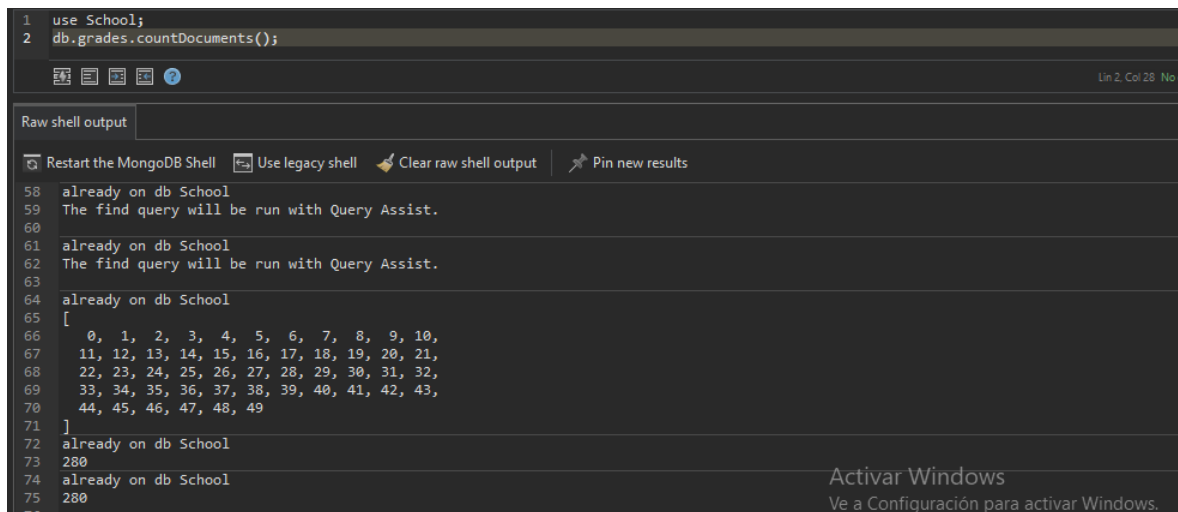
Table View

Array

[Index]	
0	123 0.0
1	123 1.0
2	123 2.0
3	123 3.0
4	123 4.0
5	123 5.0
6	123 6.0
7	123 7.0
8	123 8.0

**Descripción:** Esta consulta obtiene todos los valores únicos del campo `student_id` en la colección `grades`.

- **db.grades.countDocuments()**



```
1 use School;
2 db.grades.countDocuments();
```

Raw shell output

Restart the MongoDB Shell | Use legacy shell | Clear raw shell output | Pin new results

```
58 already on db School
59 The find query will be run with Query Assist.
60
61 already on db School
62 The find query will be run with Query Assist.
63
64 already on db School
65 [
66   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
67   11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
68   22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
69   33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43,
70   44, 45, 46, 47, 48, 49
71 ]
72 already on db School
73 280
74 already on db School
75 280
76
```

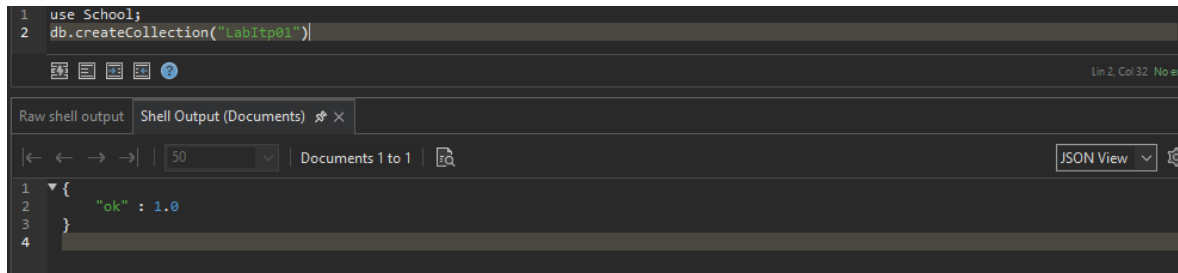
Activar Windows  
Ve a Configuración para activar Windows.

**Descripción:** Esta consulta cuenta el número total de documentos (registros) en la colección `grades`.

## Trabaja con CREATE

En la BD School crea la colección `LabItp`:

- `db.createCollection("LabItp01")`



```
1 use School;
2 db.createCollection("LabItp01")
```

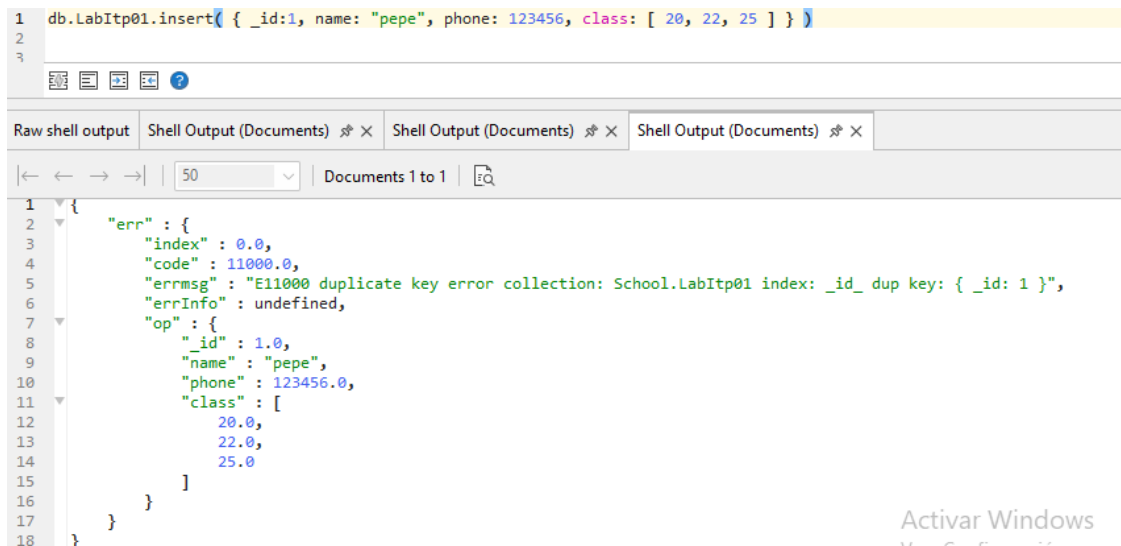
Raw shell output | Shell Output (Documents) ✕

Documents 1 to 1

```
1 {
2   "ok" : 1.0
3 }
4
```

**Descripción:** Crea una colección llamada "LabItp01" dentro de la base de datos actual. Las colecciones en MongoDB son equivalentes a las tablas en bases de datos relacionales.

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



```
1 db.LabItp01.insert( { _id:1, name: "pepe", phone: 123456, class: [ 20, 22, 25 ] } )
2
3
```

Raw shell output | Shell Output (Documents) ✕ | Shell Output (Documents) ✕ | Shell Output (Documents) ✕

Documents 1 to 1

```
1 {
2   "err" : {
3     "index" : 0.0,
4     "code" : 11000.0,
5     "errmsg" : "E11000 duplicate key error collection: School.LabItp01 index: _id_ dup key: { _id: 1 }",
6     "errInfo" : undefined,
7     "op" : {
8       "_id" : 1.0,
9       "name" : "pepe",
10      "phone" : 123456.0,
11      "class" : [
12        20.0,
13        22.0,
14        25.0
15      ]
16    }
17   }
18 }
```

**Descripción:** Inserta un documento en la colección `LabItp01` con los campos `_id` (identificador único), `name`, `phone` y `class` (una lista de números). La función `insert` es genérica y se puede usar para insertar un solo documento o varios.



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

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

Raw shell output | Shell Output (Documents) ✕

Documents 1 to 1

```
1 {
2   "acknowledged" : true,
3   "insertedId" : 2.0
4 }
5
```

**Descripción:** Inserta un único documento en la colección **LabItp01**, con los mismos campos que en el ejemplo anterior, pero con la función **insertOne**, que está diseñada específicamente para la inserción de 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] } ] )**

```
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 ] } ] )
2
```

Raw shell output | Shell Output (Documents) ✕ | Shell Output (Documents) ✕ | Shell Output (Documents) ✕

JSON View

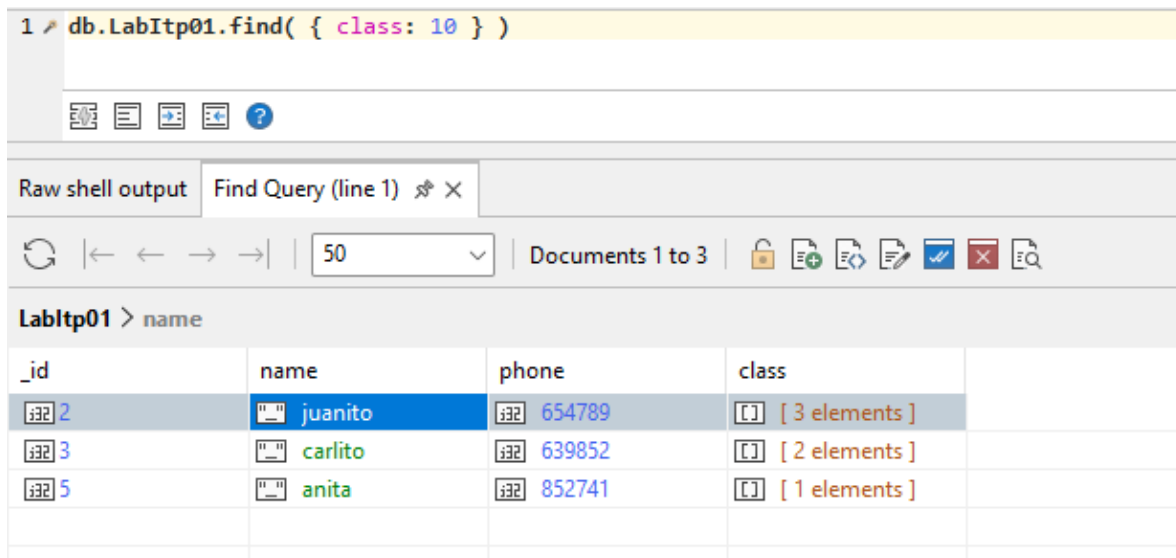
```
1 {
2   "err" : {
3     "index" : 0.0,
4     "code" : 11000.0,
5     "errmsg" : "E11000 duplicate key error collection: School.LabItp01 index: _id_ dup key: { _id: 3 }",
6     "errInfo" : undefined,
7     "op" : {
8       "_id" : 3.0,
9       "name" : "carlito",
10      "phone" : 639852.0,
11      "class" : [
12        11.0,
13        10.0
14      ]
15    }
16  }
17 }
```

Activar Windows

**Descripción:** Inserta varios documentos a la vez en la colección `LabItp01`. En este ejemplo, se están insertando cuatro documentos con diferentes valores de `_id`, `name`, `phone` y `class`.

Cabe destacar que hay dos documentos con el mismo `_id` de valor `5`, lo que causaría un error, ya que el campo `_id` debe ser único en la colección.

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



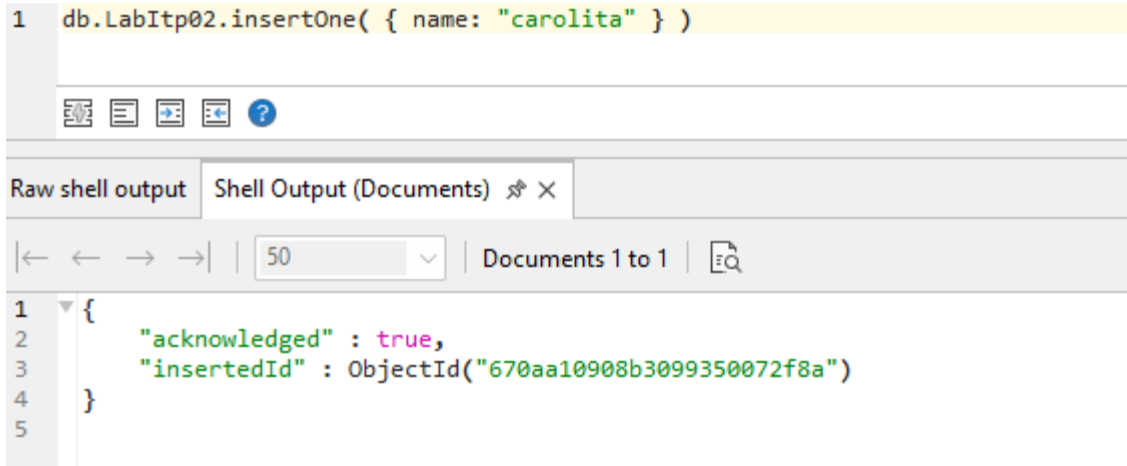
The screenshot shows a MongoDB shell interface. At the top, a command is entered: `1 db.LabItp01.find( { class: 10 } )`. Below the command, there are icons for raw shell output, find query, and a help icon. The main area displays the results of the query, which are three documents. The documents are shown in a table-like format with columns: `_id`, `name`, `phone`, and `class`. The first document has `_id` 2, `name` 'juanito', `phone` 654789, and `class` [ 3 elements ]. The second document has `_id` 3, `name` 'carlito', `phone` 639852, and `class` [ 2 elements ]. The third document has `_id` 5, `name` 'anita', `phone` 852741, and `class` [ 1 elements ].

_id	name	phone	class
2	juanito	654789	[ 3 elements ]
3	carlito	639852	[ 2 elements ]
5	anita	852741	[ 1 elements ]

**Descripción:** Busca todos los documentos en la colección `LabItp01` que contengan el valor `10` en el campo `class` (que es un array). Este tipo de consulta devuelve todos los documentos que contengan `10` como uno de los valores en la lista `class`.

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

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



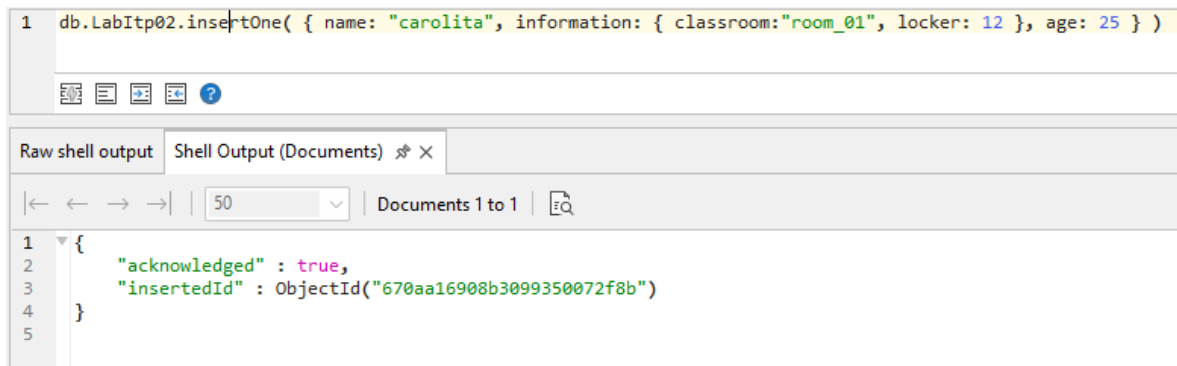
The screenshot shows the MongoDB Shell interface. The command `db.LabItp02.insertOne( { name: "carolita" } )` has been executed. The output is displayed in the 'Shell Output (Documents)' tab, showing a single document with the following structure:

```
1 {
2   "acknowledged" : true,
3   "insertedId" : ObjectId("670aa10908b3099350072f8a")
4 }
5
```

**Descripción:** Inserta un documento en una colección llamada **LabItp02**, que probablemente no existía antes, pero MongoDB la crea automáticamente. El documento tiene un único campo **name**.

- **db.LabItp02.insertOne( { name: "carolita", information: {  
classroom:"room\_01", locker: 12 }, age: 25 } )**

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



The screenshot shows the MongoDB Shell interface. The command `db.LabItp02.insertOne( { name: "carolita", information: { classroom:"room_01", locker: 12 }, age: 25 } )` has been executed. The output is displayed in the 'Shell Output (Documents)' tab, showing a single document with the following structure:

```
1 {
2   "acknowledged" : true,
3   "insertedId" : ObjectId("670aa16908b3099350072f8b")
4 }
5
```

**Descripción:** Inserta un documento en la colección **LabItp02**, pero esta vez con campos adicionales: **information**, que es un subdocumento que contiene **classroom** y **locker**, y **age**.

- **db.LabItp02.find()**

```
1 db.LabItp02.find()

Raw shell output Find Query (line 1) X
50 Documents 1 to 2

1 {
2   "_id" : ObjectId("670aa10908b3099350072f8a"),
3   "name" : "carolita"
4 }
5 {
6   "_id" : ObjectId("670aa16908b3099350072f8b"),
7   "name" : "carolita",
8   "information" : {
9     "classroom" : "room_01",
10    "locker" : NumberInt(12)
11  },
12   "age" : NumberInt(25)
13 }
14 }
```

**Descripción:** Recupera y muestra todos los documentos en la colección **LabItp02**. Como no se pasa ningún criterio de búsqueda, se devuelven todos los documentos.

## Trabaja con UPDATE

En la colección LabItp01 realiza las siguientes actualizaciones

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

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

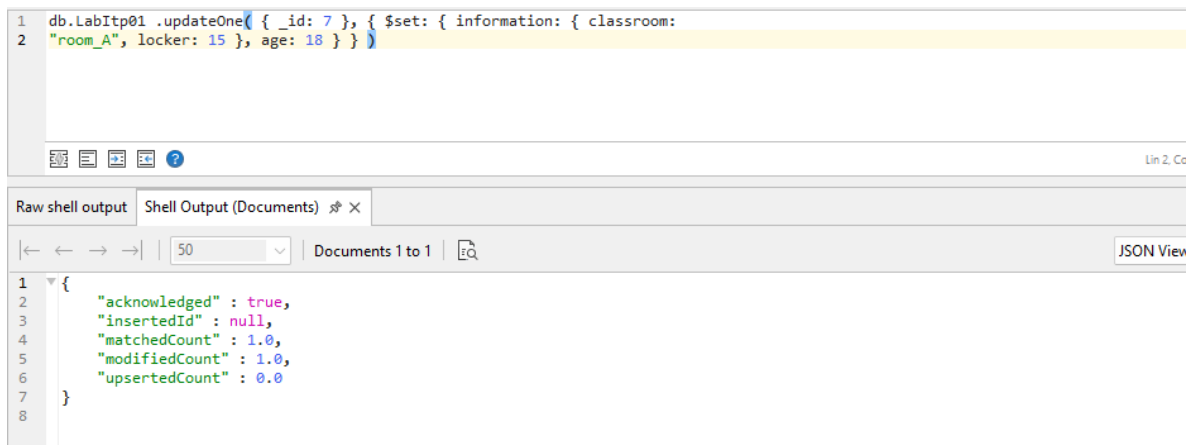
Raw shell output Shell Output (Documents) X
50 Documents 1 to 1 JSO

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

**Descripción:** Esta consulta actualiza el documento en la colección `LabItp01` cuyo `_id` es

6. El operador `$set` modifica el campo `virtues`, asignándole una nueva lista con los valores `'cheerful', 'funny', 'comprehensive', 'sociable', 'respectful'`. Si el campo `virtues` no existe, lo crea.

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



The screenshot shows a MongoDB Shell window with the following content:

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

Below the command input, there are tabs for "Raw shell output" and "Shell Output (Documents)". The "Shell Output (Documents)" tab is active, displaying the result of the update operation in JSON format:

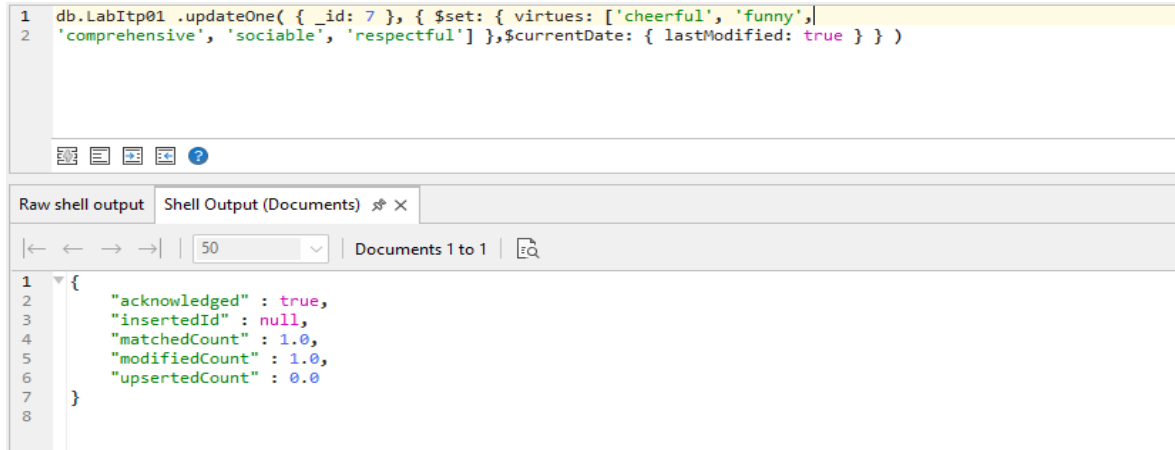
```
1 {
2   "acknowledged" : true,
3   "insertedId" : null,
4   "matchedCount" : 1.0,
5   "modifiedCount" : 1.0,
6   "upsertedCount" : 0.0
7 }
8
```

The interface also includes navigation controls (back, forward, search) and a "JSON View" button.

**Descripción:** Esta consulta actualiza el documento con `_id` 6, modificando o creando el campo `information` con un objeto que tiene las propiedades `classroom` y `locker`. También actualiza el campo `age` a 18. Si los campos no existen, los crea.

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

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

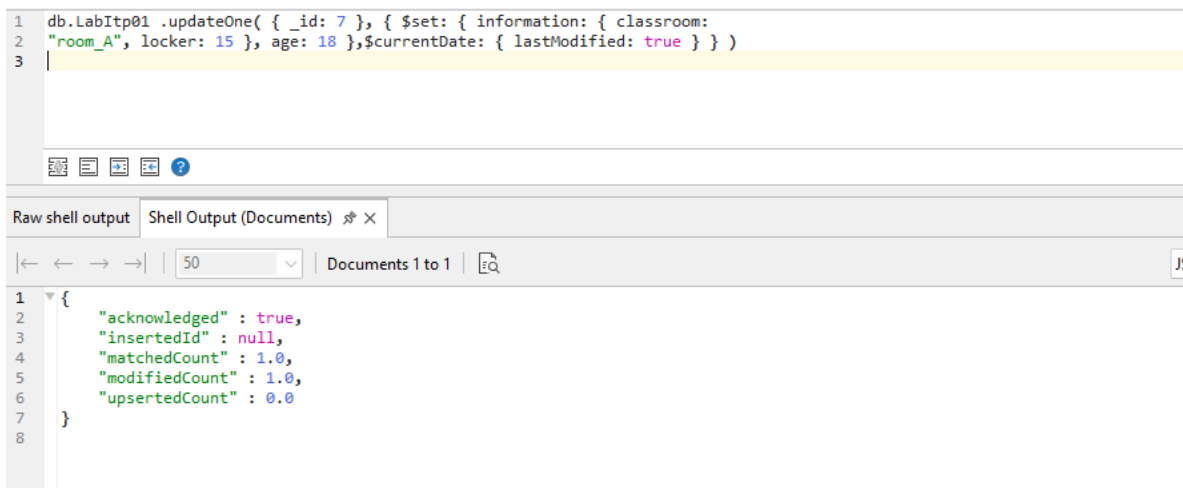


```
1 {
2   "acknowledged" : true,
3   "insertedId" : null,
4   "matchedCount" : 1.0,
5   "modifiedCount" : 1.0,
6   "upsertedCount" : 0.0
7 }
8
```

**Descripción:** Al igual que en la primera consulta, esta modifica el campo `virtues`. Sin embargo, también añade el operador `$currentDate`, que actualiza el campo `lastModified` con la fecha y hora actual. Esto se usa para llevar un control de cuándo fue la última modificación.

- **db.LabItp01 .updateOne( { \_id: 7 }, { \$set: { information: { classroom: "room\_A", locker: 15 }, age: 18 }, \$currentDate: { lastModified: true } } )**

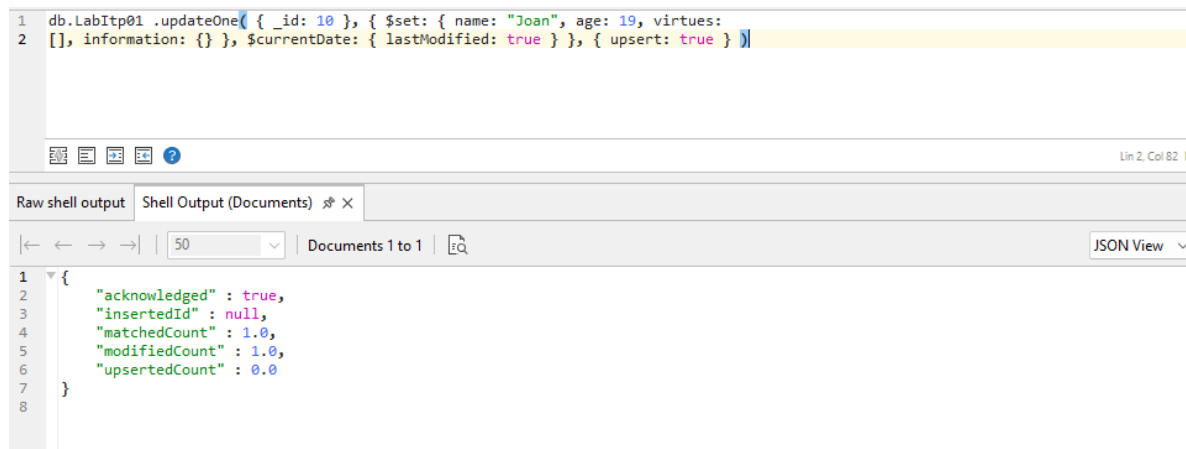
```
1 db.LabItp01 .updateOne( { _id: 7 }, { $set: { information: { classroom: "room_A", locker: 15 }, age: 18 }, $currentDate: { lastModified: true } } )
```



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

**Descripción:** Esta es una combinación de la segunda consulta y la tercera. Además de actualizar el campo `information` y `age`, también añade o actualiza el campo `lastModified` con la fecha y hora actuales.

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



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

Raw shell output | Shell Output (Documents) ✕

Documents 1 to 1 | JSON View

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

**Descripción:** Esta consulta busca un documento con `_id 10`. Si lo encuentra, actualiza los campos `name`, `age`, `virtues` (con una lista vacía), e `information` (con un objeto vacío), y también añade o actualiza `lastModified` con la fecha y hora actuales. Si no encuentra ningún documento con `_id 10`, crea un nuevo documento con esos valores debido a la opción `upsert: true`.

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

```
1 db.LabItp01.updateMany(  
2   { _id: { $in: [1, 2, 3, 4, 5, 6] } },  
3   { $set: { virtues: ['cheerful'] } }  
4 )
```

Raw shell output | Shell Output (Documents) ✕

← → | 50 | Documents 1 to 1 | EQ JS

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

Descripción: Todos los documentos con `_id` entre 1 y 6 tendrán el campo `virtues` actualizado con la lista `['cheerful']`. Si el campo ya existía, se sobrescribirá; si no existía, se añadirá.

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

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

Raw shell output | Shell Output (Documents) ✕

← → | 50 | Documents 1 to 1 | EQ JS

```
1 {  
2   "acknowledged" : true,  
3   "insertedId" : null,  
4   "matchedCount" : 7.0,  
5   "modifiedCount" : 7.0,  
6   "upsertedCount" : 0.0  
7 }  
8
```

**Descripción:** Esta consulta actualiza todos los documentos en la colección `LabItp01`, añadiendo o modificando el campo `status` con el valor `'A'`.



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

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

Raw shell output | Shell Output (Documents) ✕

50 | Documents 1 to 1

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

**Descripción:** Esta consulta actualiza todos los documentos en la colección LabItp01 donde el campo name es 'pepe' o 'camilito', asignándoles el valor 'student' en el campo role.

## Trabaja con DELETE

En la colección LabItp01 realiza las siguientes actualizaciones:

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

```
1 db.LabItp01.deleteOne( { name: "carlito" } )
2
```

Raw shell output | Shell Output (Documents) ✕

50 | Documents 1 to 1 | 🔍

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

**Descripción:** Elimina un único documento de la colección **LabItp01** donde el campo **name** es igual a **"carlito"**. Si hay múltiples documentos que cumplen con esta condición, solo se eliminará el primero encontrado.

- **db.grades.deleteOne( { student\_id: 0 } )**

```
1 db.grades.deleteOne( { student_id: 0 } )
2
```

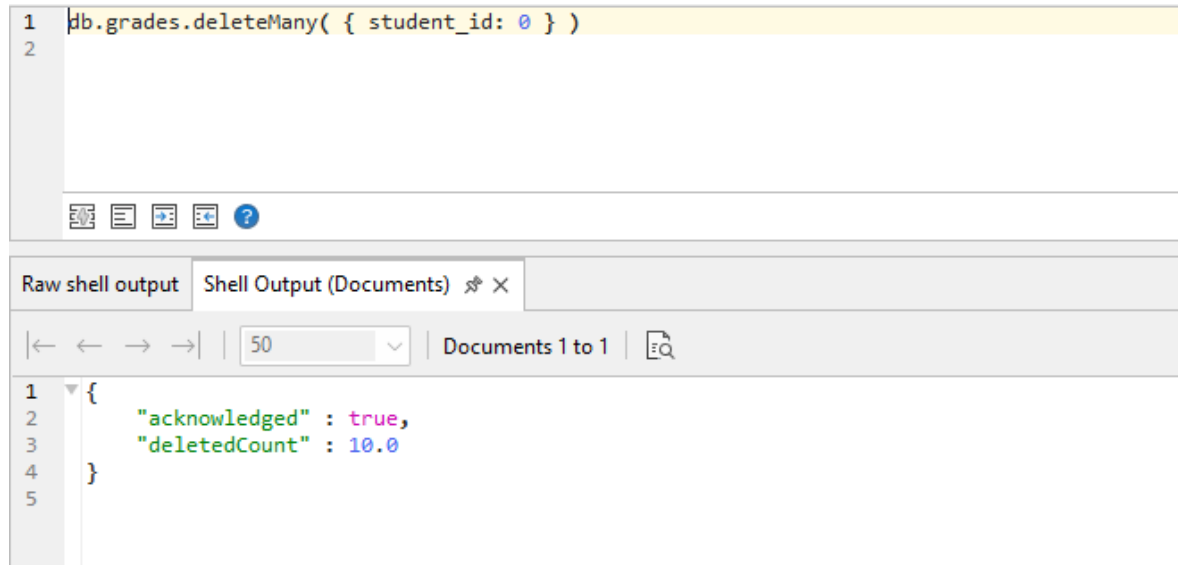
Raw shell output | Shell Output (Documents) ✕

50 | Do Shell Output (Documents)

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

**Descripción:** Elimina un único documento de la colección **grades** donde el campo **student\_id** es igual a 0. Similar a la consulta anterior, solo se eliminará el primer documento que coincida con la condición.

- **db.grades.deleteMany( { student\_id: 0 } )**



```
1 db.grades.deleteMany( { student_id: 0 } )
2
```

Raw shell output | Shell Output (Documents) ✖

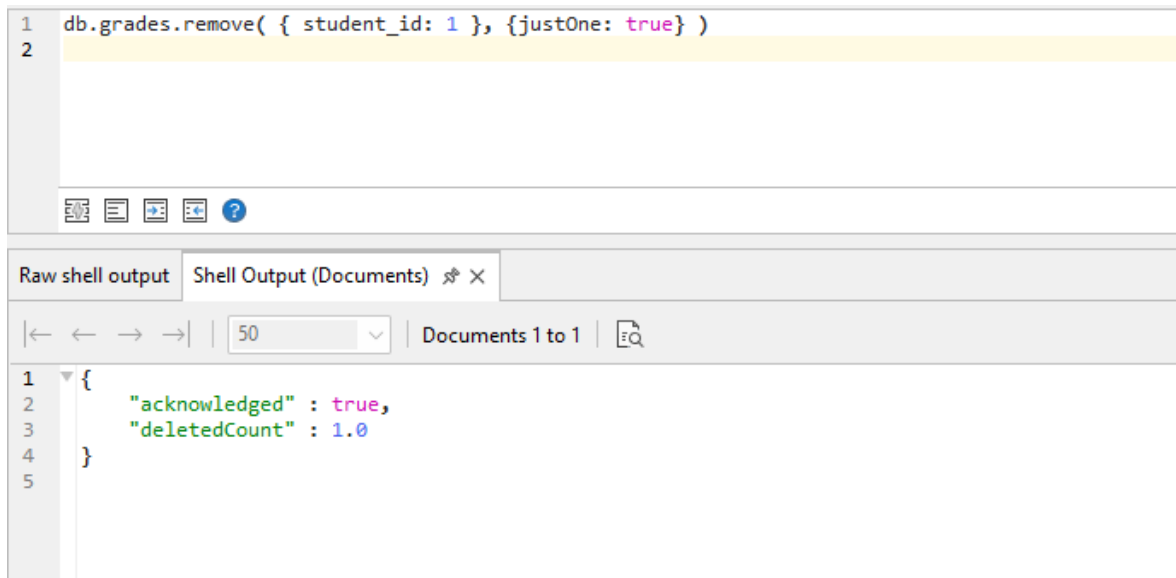
← ← → → | 50 | Documents 1 to 1 | 🔍

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

**Descripción:** Elimina todos los documentos de la colección **grades** que tienen el campo **student\_id** igual a 0. A diferencia de **deleteOne**, esta operación afectará a todos los documentos que coincidan con la condición.

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

```
1 db.grades.remove( { student_id: 1 }, {justOne: true} )
2
```

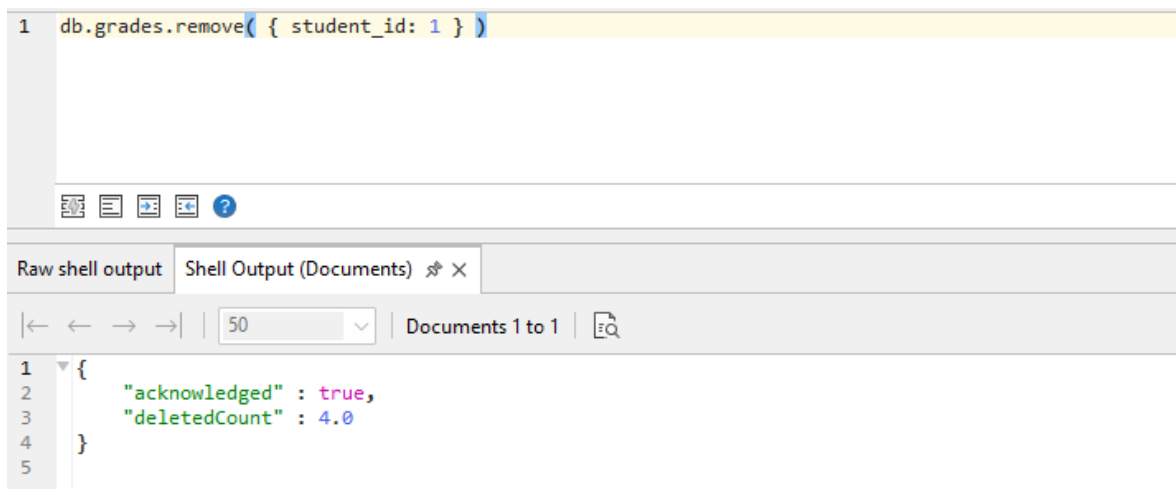


The screenshot shows the MongoDB Shell interface. The command `db.grades.remove( { student_id: 1 }, {justOne: true} )` is entered in the shell. The output, displayed in the 'Shell Output (Documents)' tab, is a JSON object: `{ "acknowledged" : true, "deletedCount" : 1.0 }`. The interface includes navigation buttons and a search bar.

**Descripción:** Elimina un solo documento en la colección **grades** donde el **student\_id** es 1. El uso de `{ justOne: true }` indica que solo se debe eliminar uno, aunque esta opción es obsoleta y se recomienda utilizar `deleteOne` en su lugar.

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

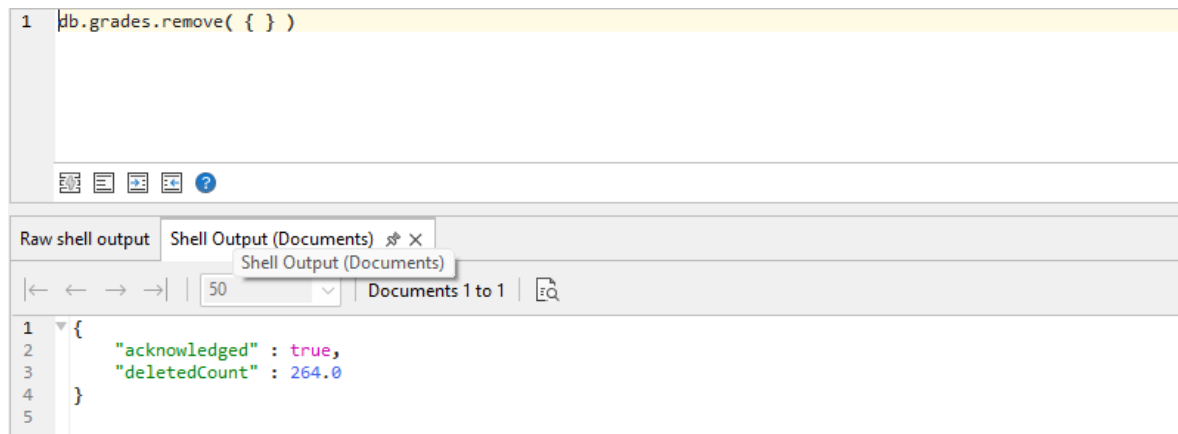
```
1 db.grades.remove( { student_id: 1 } )
```



The screenshot shows the MongoDB Shell interface. The command `db.grades.remove( { student_id: 1 } )` is entered in the shell. The output, displayed in the 'Shell Output (Documents)' tab, is a JSON object: `{ "acknowledged" : true, "deletedCount" : 4.0 }`. The interface includes navigation buttons and a search bar.

**Descripción:** Elimina todos los documentos en la colección **grades** donde el **student\_id** es 1. A diferencia de la consulta anterior, aquí no se limita a un solo documento.

- **db.grades.remove( { } )**



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

Raw shell output | Shell Output (Documents) X

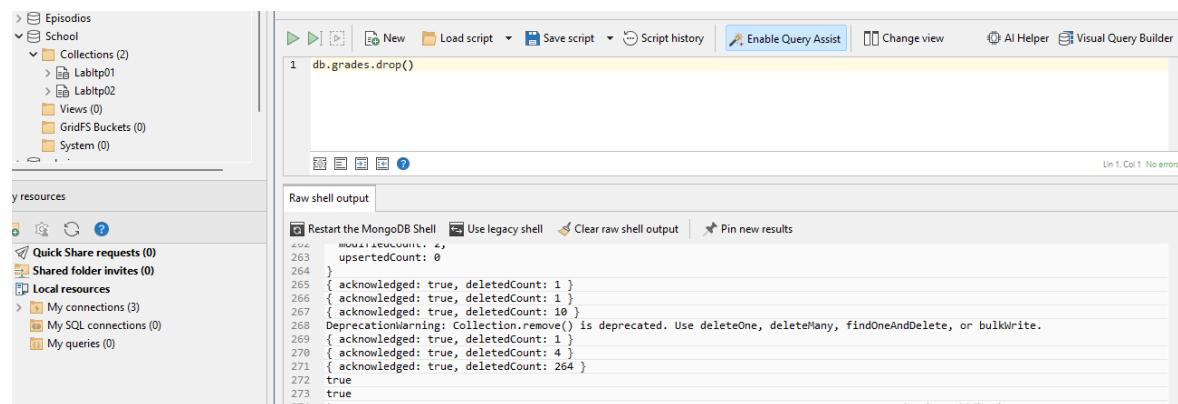
Shell Output (Documents)

50 | Documents 1 to 1

```
1 {
2   "acknowledged" : true,
3   "deletedCount" : 264
4 }
5
```

**Descripción:** Elimina todos los documentos en la colección **grades**. Dado que no se especifica ninguna condición, todos los documentos en la colección serán eliminados.

- **db.grades.drop()**



```
1 db.grades.drop()
```

Raw shell output

Restart the MongoDB Shell | Use legacy shell | Clear raw shell output | Pin new results

```
265 { acknowledged: true, deletedCount: 1 }
266 { acknowledged: true, deletedCount: 1 }
267 { acknowledged: true, deletedCount: 10 }
268 DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany, findOneAndDelete, or bulkWrite.
269 { acknowledged: true, deletedCount: 1 }
270 { acknowledged: true, deletedCount: 4 }
271 { acknowledged: true, deletedCount: 264 }
272 true
273 true
274 true
```

**Descripción:** Elimina la colección **grades** por completo. Esto no solo elimina todos los documentos en la colección, sino que también elimina la colección en sí, lo que significa que no podrás acceder a ella nuevamente a menos que la vuelvas a crear.

## **Análisis y Discusión**

### **Interpretación de Resultados**

Los resultados obtenidos a partir de las consultas y actualizaciones realizadas muestran claramente las ventajas de MongoDB en comparación con las bases de datos relacionales tradicionales. El uso de agregaciones y referencias optimiza el acceso a datos sin duplicación innecesaria, permitiendo un manejo ágil de grandes volúmenes de datos multimedia, como películas y géneros.

En el caso de la librería, se observó cómo las referencias entre las colecciones de autores, libros y categorías permiten actualizaciones precisas y consultas complejas sin afectar el rendimiento del sistema. Por ejemplo, la relación muchos a muchos entre libros y categorías permite identificar los géneros más populares de manera rápida, mientras que las transacciones pueden gestionarse sin necesidad de duplicar datos. Esto es especialmente útil en entornos donde las consultas relacionadas con ventas y stock se realizan constantemente.

Por último, el caso del sistema escolar mostró cómo MongoDB maneja de manera eficiente los datos anidados y permite realizar consultas precisas sobre datos de calificaciones de estudiantes. Utilizando índices y filtros en arreglos, fue posible extraer información relevante sobre el desempeño académico de los estudiantes, facilitando la identificación de patrones de rendimiento bajo o la clasificación de estudiantes por sus logros en diferentes exámenes. Esta flexibilidad para gestionar datos jerárquicos sin sacrificar la capacidad de realizar consultas rápidas es una ventaja clave de MongoDB.

## **Conclusiones**

El análisis y los casos presentados demuestran que MongoDB es una solución robusta para la gestión de datos no estructurados y semi-estructurados en aplicaciones que requieren flexibilidad y escalabilidad. La capacidad de MongoDB para manejar relaciones complejas mediante referencias en lugar de duplicar información garantiza que los datos sean fáciles de actualizar y consultar, sin comprometer el rendimiento del sistema. Además, MongoDB permite la manipulación de datos anidados de forma sencilla, lo que lo convierte en una opción ideal para aplicaciones que necesitan almacenar y procesar datos jerárquicos o de múltiples niveles.

Se concluye que, gracias a su flexibilidad en el diseño de esquemas y su capacidad para manejar grandes volúmenes de datos de manera eficiente, MongoDB es una opción excelente para aplicaciones modernas que requieren tanto velocidad como escalabilidad. Las consultas y operaciones avanzadas realizadas en los tres casos estudiados resaltan cómo MongoDB ofrece un rendimiento óptimo al manejar relaciones complejas entre colecciones, como en el caso de la librerías.

## **Recomendaciones**

Mejorar la estructura de índices: Dado que MongoDB permite crear índices para acelerar las consultas, es recomendable crear índices en las colecciones que se consultan con mayor frecuencia, como las de usuarios, compras y películas. Esto mejorará significativamente el rendimiento del sistema cuando se trabaje con grandes volúmenes de datos.

Automatización de consultas complejas: En aplicaciones donde se realicen consultas complejas de forma recurrente, como la recuperación de calificaciones o transacciones de compras, se recomienda automatizar estos procesos mediante scripts o jobs programados que ejecuten las consultas de manera periódica. Esto no solo ahorrará tiempo, sino que también garantizará que los datos sean procesados y mostrados en tiempo real.

Monitoreo de rendimiento y ajuste de la base de datos: A medida que el volumen de datos en MongoDB crece, es importante monitorear constantemente el rendimiento de la base de datos para identificar posibles cuellos de botella. Herramientas como MongoDB Atlas pueden ayudar a ajustar el rendimiento a medida que se amplían las necesidades del sistema, asegurando una escalabilidad continua sin comprometer la integridad o la velocidad de las consultas



## **Bibliografia**

Link del repositorio:

[https://github.com/jessica123c/DB\\_ALMACENAMIENTO\\_MASIVO](https://github.com/jessica123c/DB_ALMACENAMIENTO_MASIVO)

MongoDB Inc. (2023). MongoDB Manual. Disponible en: <https://docs.mongodb.com/manual/>

Chodorow, K. (2013). MongoDB: The Definitive Guide. O'Reilly Media.

MongoDB University. (2023). *MongoDB aggregation framework*.

<https://university.mongodb.com/>

MongoDB Inc. (2023). Best Practices for MongoDB Performance Tuning. Disponible en:

<https://www.mongodb.com/blog/post/performance-best-practices>.

Fowler, M. (2012). NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley.

Studio 3T. (2023). Studio 3T documentation. <https://studio3t.com/>.