Informe Base de Datos

Hayder Arley Rojas

Ingeniería de Sistemas, Instituto Tecnológico del Putumayo

BD Almacenamiento De Datos Masivo

Brayan Arcos

Octubre de 2024

**INDICE**

[**Resumen Ejecutivo** 4](#_Toc179975247)

[**Introducción** 5](#_Toc179975248)

[**Contexto y Motivación** 5](#_Toc179975249)

[**Alcance del Informe** 6](#_Toc179975250)

[**Objetivos** 7](#_Toc179975251)

[**Metodología** 8](#_Toc179975252)

[**Herramientas Utilizadas** 8](#_Toc179975253)

[**Procedimientos** 8](#_Toc179975254)

[**Creación base de datos personal** 8](#_Toc179975255)

[**Métodos de captura** 10](#_Toc179975256)

[**Consultas Realizadas y Sus resultados** 13](#_Toc179975257)

[**Desarrollo del Informe** 15](#_Toc179975258)

[**Esquemas** 15](#_Toc179975259)

[**Diseño de Base de Datos Personal** 17](#_Toc179975260)

[**Modelo de Datos:** 17](#_Toc179975261)

[**Consideraciones de Diseño** 18](#_Toc179975262)

[**Métodos de captura BD grupal** 20](#_Toc179975263)

[**Esquema de las colecciones** 27](#_Toc179975264)

[**Consultas** 30](#_Toc179975265)

[**Análisis y Discusión** 54](#_Toc179975266)

[**Interpretación de Resultados** 54](#_Toc179975267)

[**Conclusiones** 55](#_Toc179975268)

[**Recomendaciones** 56](#_Toc179975269)

[**Bibliografia** 57](#_Toc179975270)

**Link de GitHub**…………………………………………………………………………57

# **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 plataforma de streaming 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:

Plataforma de Streaming : El caso de estudio se enfoca en una platafroma de Streaming donde se gestionan colecciones de Usuarios, Suscripciones, Peliculas, Generos y lista de reproducciones. Se demuestra cómo MongoDB puede manejar transacciones y consultas sobre los productos disponibles, así como las suscripciones de los Usuarios a la plataforma.

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​(informebd 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 Plataforma de Streaming con cinco colecciones principales: **usuarios**, **suscripciones**, **géneros**, **películas**, y **lista de reproducciones**.
* **Inserción de Datos**: Se llenaron las colecciones con datos representativos de los usuarios, las suscripciones activas, los géneros de las películas, el catálogo de películas disponibles, y las listas de reproducción creadas por los usuarios.
* **Consultas NoSQL**: Se ejecutaron consultas para extraer información de las colecciones, tales como las películas vistas por un usuario, los géneros más populares, los detalles de las suscripciones activas y el historial de reproducción de cada usuario.

**Métodos**

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

**1. Estructura de la base de datos**

**Colecciones:**

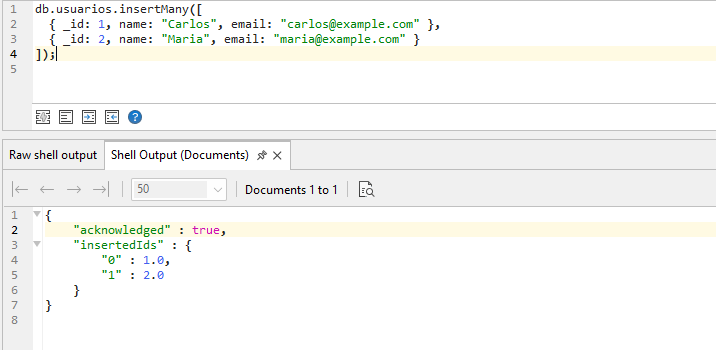
1. **Usuarios** (Un usuario puede tener una suscripción)
2. **Suscripciones** (Una suscripción puede estar relacionada con un solo usuario)
3. **Películas** (Una película puede tener varios géneros y puede estar en muchas listas de reproducción)
4. **Géneros** (Un género puede tener muchas películas)
5. **Listas de Reproducción** (Una lista de reproducción pertenece a un usuario y contiene varias películas)

**Relaciones:**

* **Relación 1 a 1**: Entre Usuarios y Suscripciones (un usuario tiene una suscripción).
* **Relación 1 a Muchos**: Entre Usuarios y Listas de Reproducción (un usuario puede tener varias listas de reproducción).
* **Relación Muchos a Muchos**: Entre Películas y Géneros, y entre Películas y Listas de Reproducción.

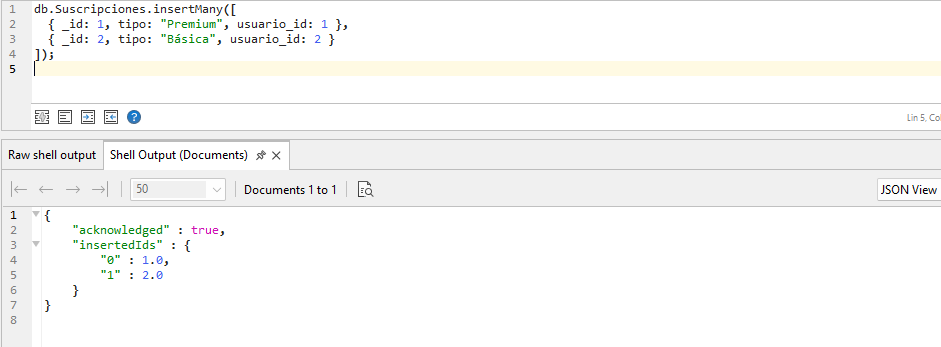
### **Métodos de captura**

* **Usuarios:**



**Descripción**: Inserta dos usuarios, "Carlos" y "Maria", en la colección Usuarios. Cada documento contiene un ID único, el nombre del usuario y su email.

* **Suscripciones:**



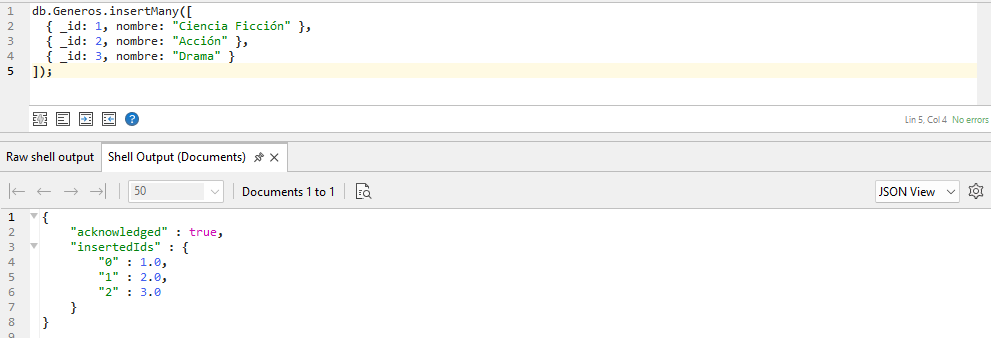
**Descripción**: Inserta dos suscripciones en la colección Suscripciones. Cada suscripción está asociada a un usuario a través de usuario\_id. Carlos tiene una suscripción "Premium", y Maria una "Básica".

* **Películas:**

****

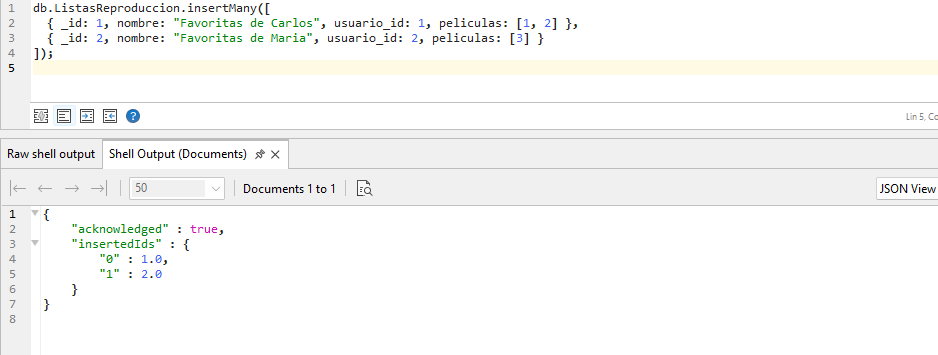
**Descripción**: Inserta tres películas en la colección Peliculas. Cada película tiene un título y una duración en minutos.

* **Géneros:**

****

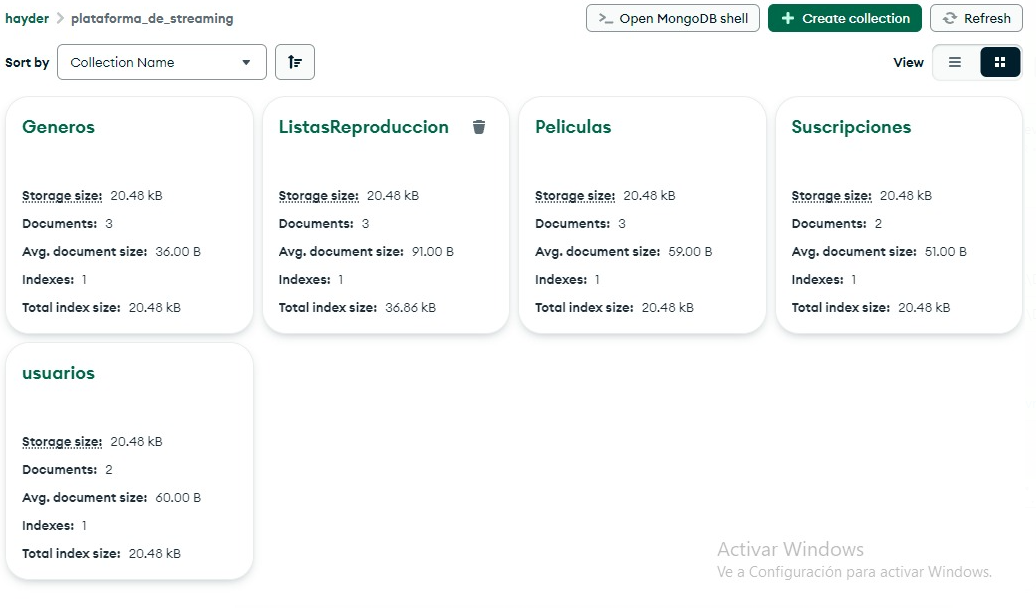
**Descripción**: Inserta tres géneros de películas en la colección Generos, como "Ciencia Ficción", "Acción" y "Drama".

* **Listas de Reproducción:**

****

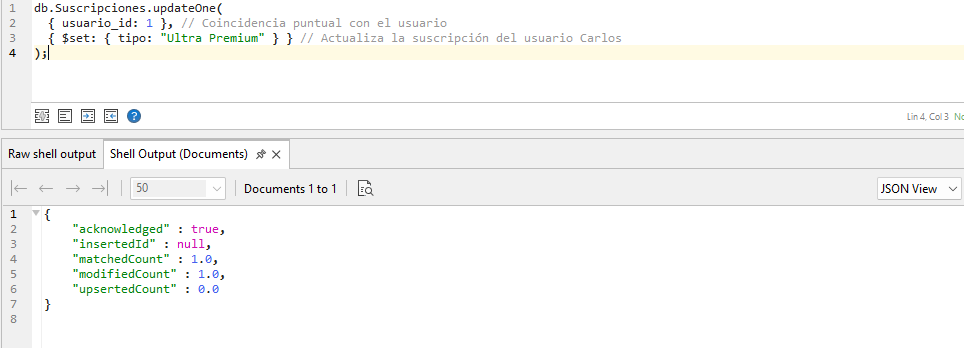
**Descripción**: Inserta dos listas de reproducción en la colección ListasReproduccion. Carlos tiene una lista con las películas "Inception" y "Matrix", y Maria tiene una lista con "Interstellar".

Visualización de toda la base de datos en general



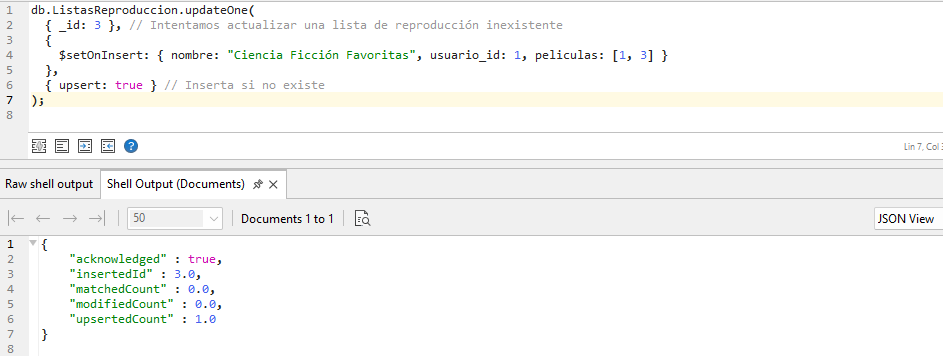
### **Consultas Realizadas y Sus resultados**

* **1. Actualizar la suscripción de un usuario con updateOne() y coincidencias puntuales(relación de 1 a 1):**

****

**Descripción**: Actualiza la suscripción de Carlos (usuario\_id: 1), cambiando el tipo de suscripción de "Premium" a "Ultra Premium". Se utiliza una coincidencia puntual (usuario\_id: 1).

* 2. **Agregar una nueva lista de reproducción a un usuario usando updateOne() con upsert(relación de 1 a Muchos):**

****

**Descripción**: Intenta actualizar la lista de reproducción con \_id: 3. Si no existe, inserta una nueva lista con el nombre "Ciencia Ficción Favoritas", asociada al usuario Carlos (usuario\_id: 1) y con las películas "Inception" e "Interstellar". El upsert permite insertar si no encuentra coincidencias.

* 3. **Agregar géneros a una película usando $each (Relación de Muchos a Muchos):**

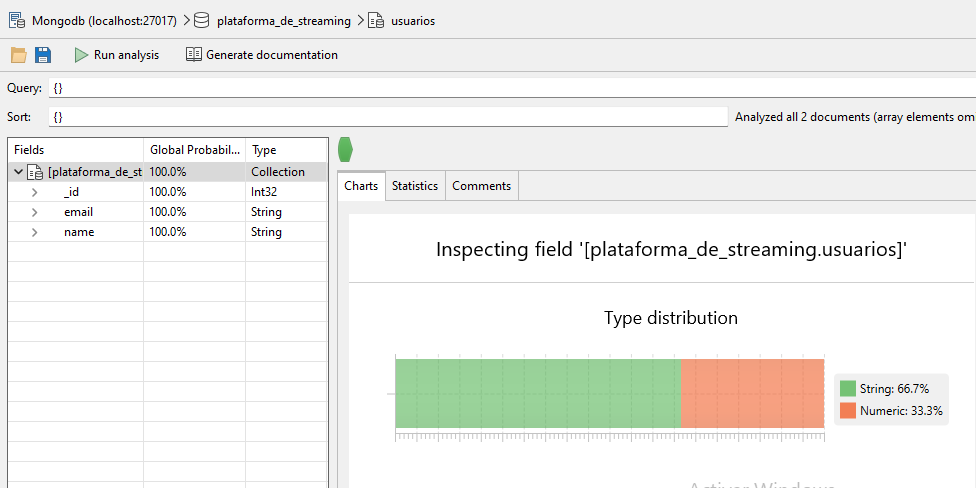
****

**Descripción**: Actualiza la película "Inception" (id: 1) añadiendo los géneros "Ciencia Ficción" (id: 1) y "Drama" (id: 3) a su lista de géneros. Se usa $each para añadir varios elementos al array y $addToSet para evitar duplicados.

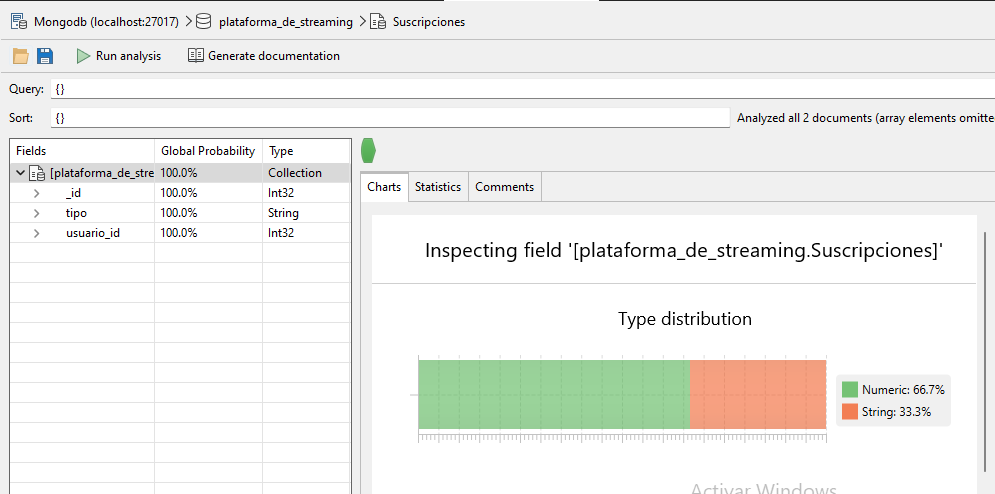
# **Desarrollo del Informe**

## **Esquemas**

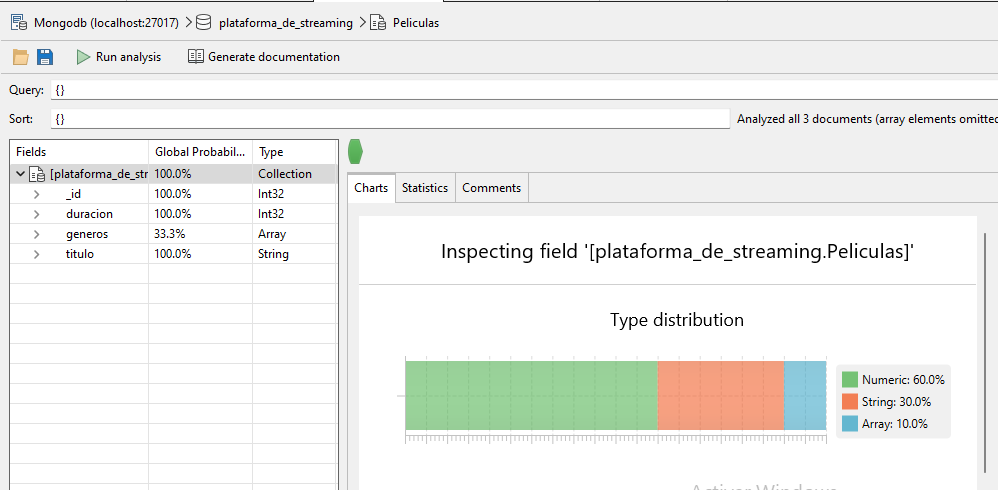
* **Colección “Usuarios”:**

****

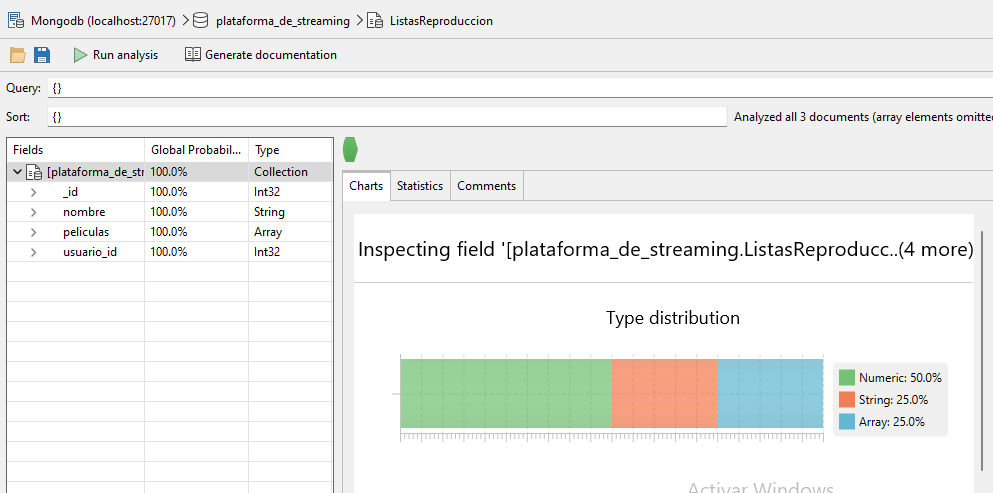
* **Colección “Suscripciones”:**

****

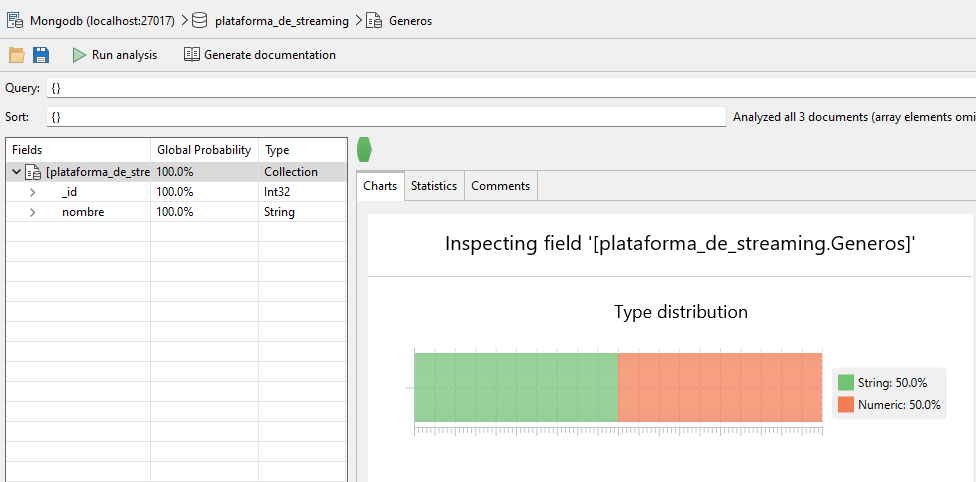
* **Colección “Películas”:**

****

* **Colección “ListasReproduccion”:**

****

* **Colección “Generos”:**

****

## **Diseño de Base de Datos Personal**

1**. Modelo de Datos: Normalización y Cardinalidad**

**Normalización:**

En MongoDB, la normalización es más flexible que en bases de datos relacionales tradicionales, ya que podemos optar por "embeber" documentos o "referenciar" colecciones dependiendo de la relación entre los datos.

En este caso, las decisiones de normalización dependerán de:

Acceso frecuente a los datos: Se prefiere embebido si los datos son frecuentemente leídos juntos.

Redundancia: Se evita la redundancia excesiva mediante la referencia si los datos cambian con frecuencia o son compartidos.

**Cardinalidad y Decisiones de Embebido o Referencia:**

* **Relación 1 a 1:** Usuarios y Suscripciones

**Cardinalidad:** Uno a Uno.

**Decisión:** No embebido, se prefiere mantener las colecciones separadas.

**Justificación:** Las suscripciones pueden cambiar de estado o ser eliminadas/actualizadas con más frecuencia que los datos del usuario. Mantenerlos separados facilita estas actualizaciones.

* **Relación 1 a Muchos:** Usuarios y Listas de Reproducción

**Cardinalidad:** Uno a Muchos.

**Decisión:** No embebido.

**Justificación:** Un usuario puede tener varias listas, y estas listas pueden cambiar (añadir o quitar películas) de forma independiente al usuario. Mantenerlas en una colección separada facilita consultas relacionadas con listas y películas.

* **Relación Muchos a Muchos:** Películas y Géneros

**Cardinalidad:** Muchos a Muchos.

**Decisión:** No embebido (Referencias a géneros en la colección de películas).

**Justificación:** Los géneros pueden compartirse entre muchas películas y pueden cambiar con el tiempo (nuevos géneros o eliminaciones). Además, es más eficiente manejar los géneros como una referencia en las películas para evitar duplicación.

* **Relación Muchos a Muchos:** Películas y Listas de Reproducción

**Cardinalidad:** Muchos a Muchos.

**Decisión:** No embebido (Referencias a películas en las listas de reproducción).

**Justificación:** Varias películas pueden aparecer en muchas listas de reproducción. Dado que las películas tienen un tamaño considerable (datos de duración, actores, etc.), se opta por referenciarlas para evitar embebido y reducir el uso innecesario de espacio.

**2.** **Consideraciones de Diseño**

**Relaciones entre Colecciones:**

* **Usuarios y Suscripciones:**

Se mantiene una relación 1 a 1 mediante una referencia en Suscripciones hacia Usuarios (usuario\_id).

Las suscripciones no se embeben en los usuarios porque podrían necesitar actualizaciones frecuentes o ser desactivadas.

* **Usuarios y Listas de Reproducción:**

Se mantiene una relación 1 a muchos usando una referencia en la colección ListasReproduccion hacia Usuarios (usuario\_id).

Las listas de reproducción no se embeben en los usuarios, ya que cada lista puede cambiar de forma independiente al usuario.

* **Películas y Géneros:**

Se mantiene una relación muchos a muchos con referencias a géneros en las películas.

No se embeben los géneros directamente en las películas para evitar redundancia, ya que un mismo género puede aplicarse a muchas películas.

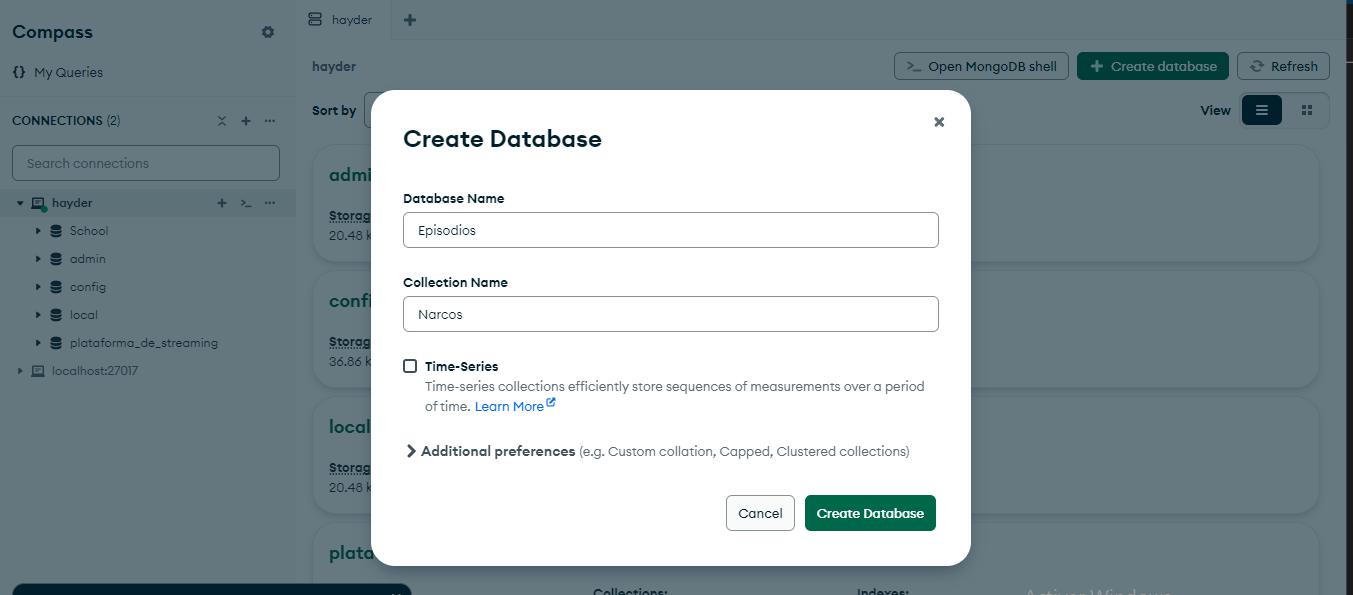
* **Películas y Listas de Reproducción:**

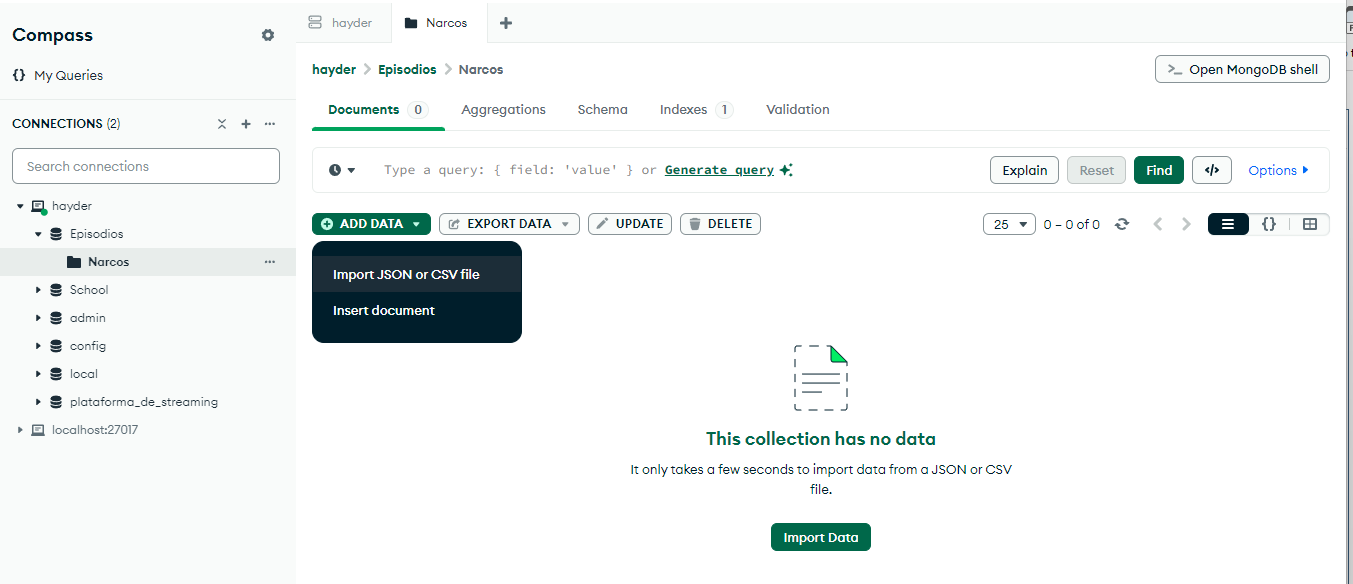
Se mantiene una relación muchos a muchos mediante referencias a películas en las listas de reproducción.

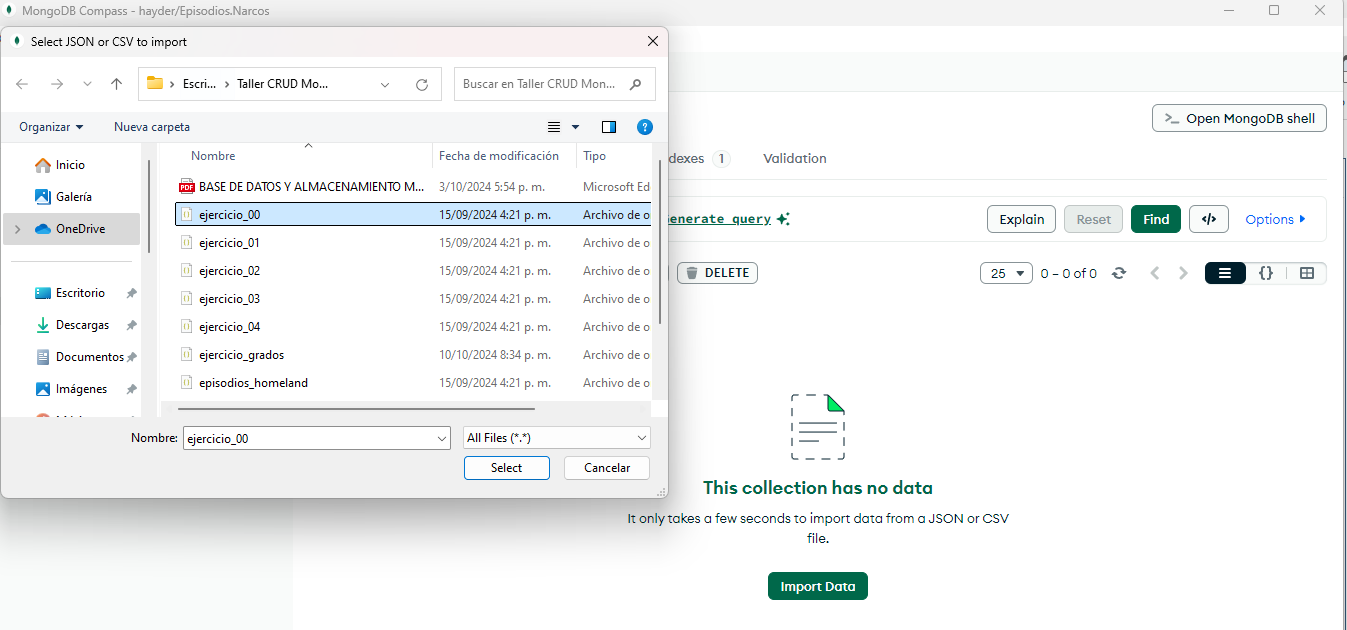
Las películas no se embeben en las listas porque su contenido puede ser amplio y aparecer en múltiples listas.

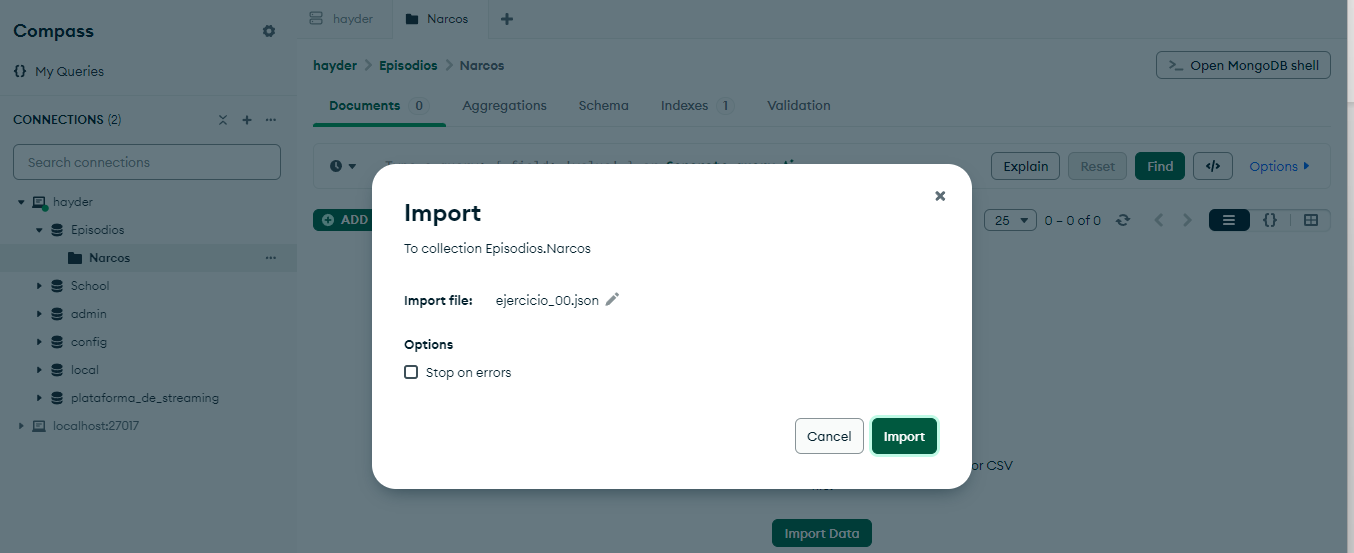
**Métodos de captura BD grupal**

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

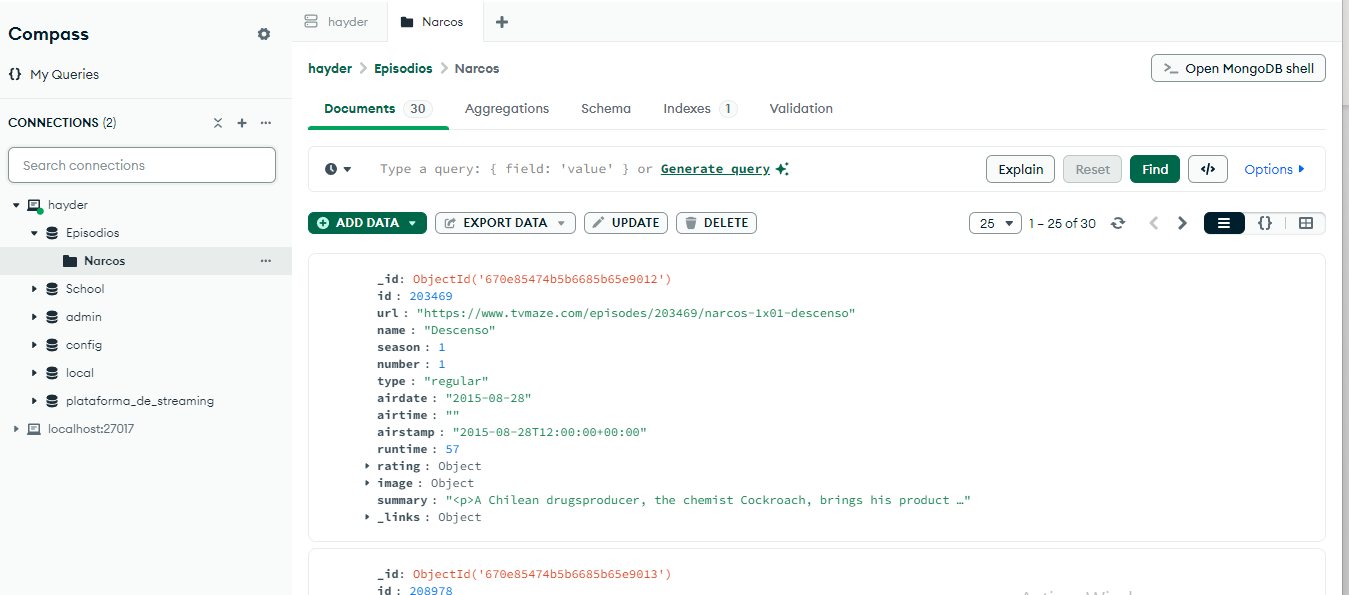
****

* En esta parte vamos a importar el primer archivo.json de la colección Narcos
* Demostración de como Importar los archivos

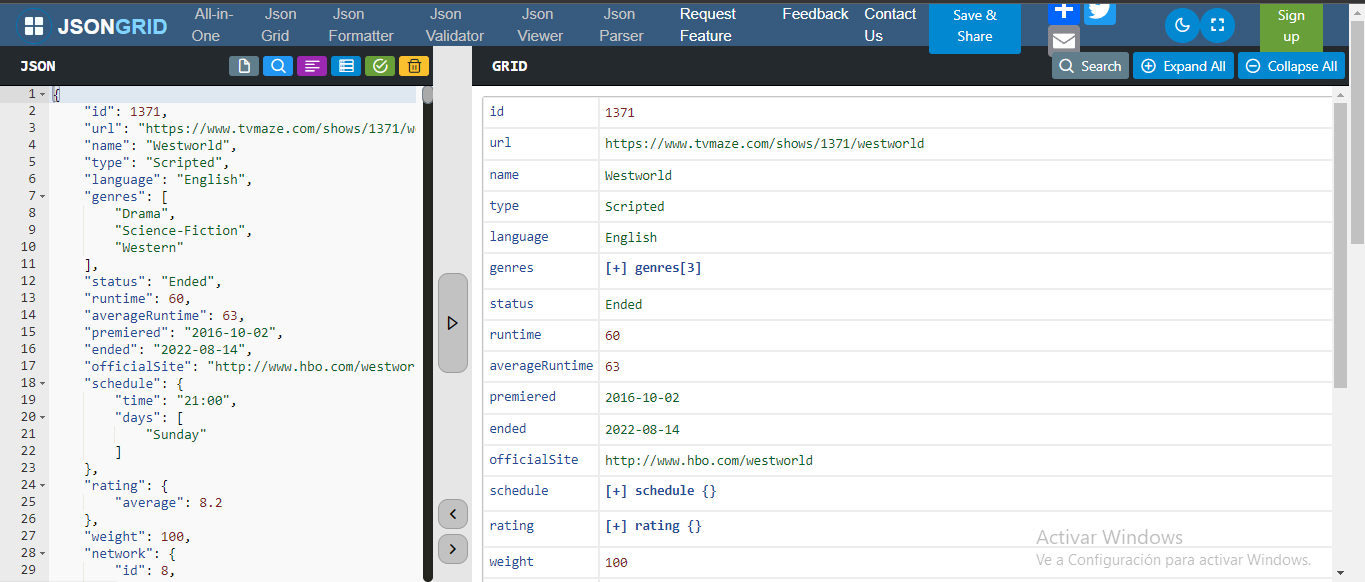




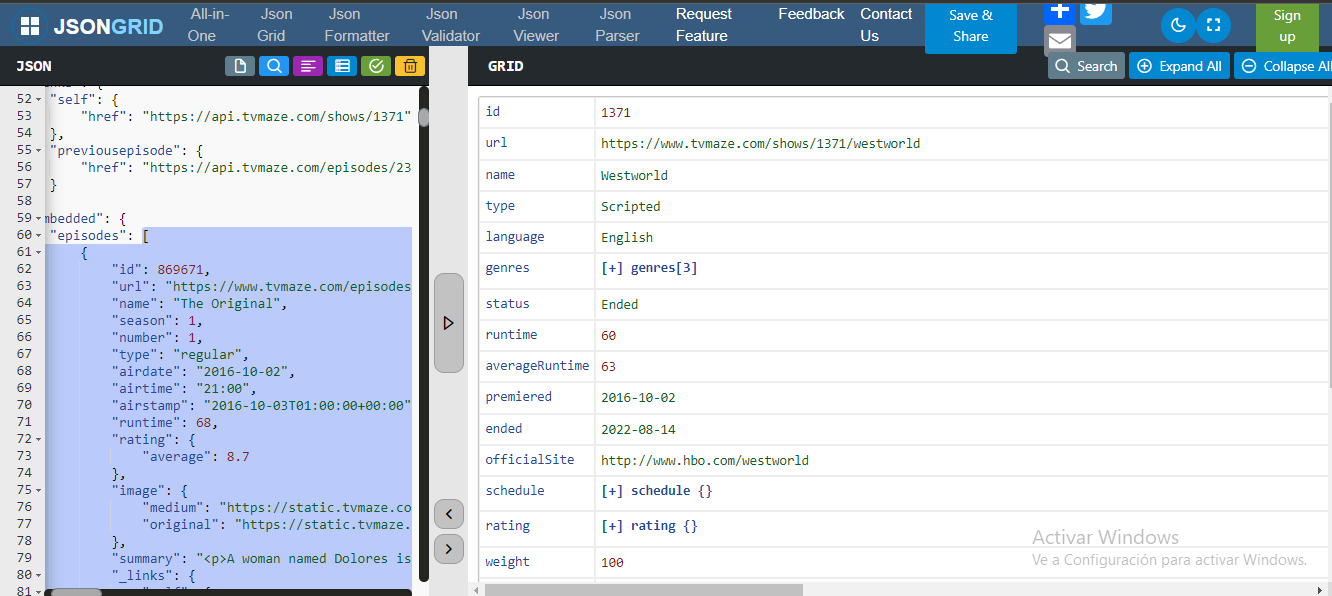
* Resultado de como queda el archivo .json ya importado y como se visualiza en Mongo compass.

****

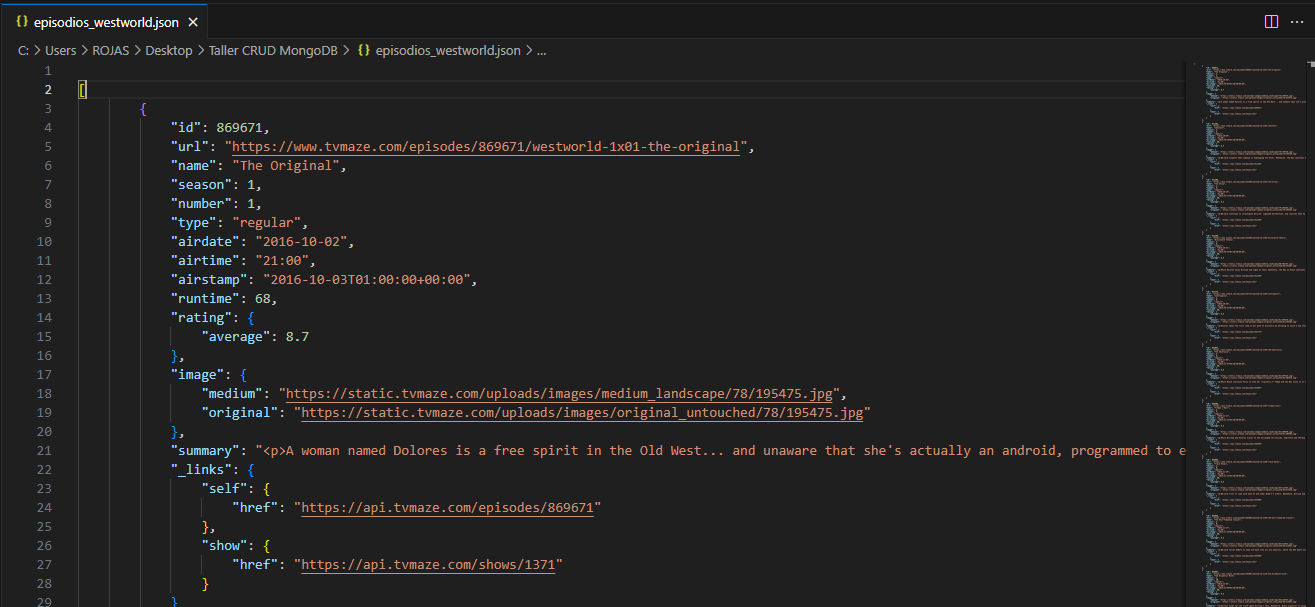
* Ahora añadiomos nuestro archivo llamado “ejercicio\_01” a nuestra herramienta online llamada JSONGRID.

****

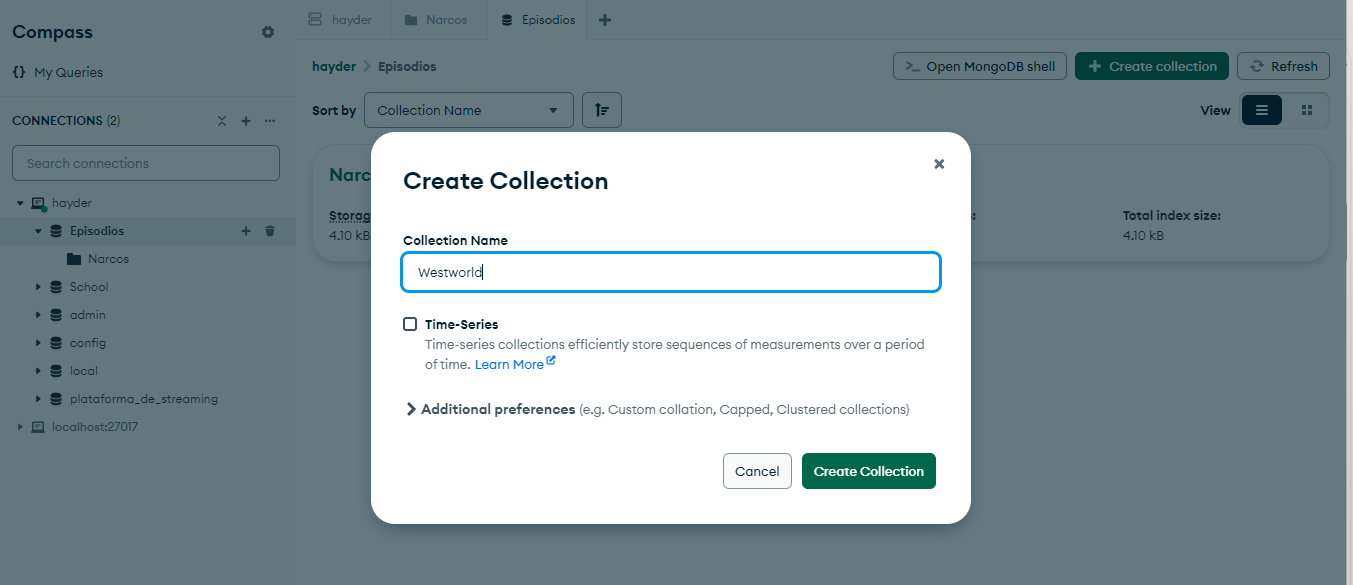
* Luego de añadirlo nos ubicamos en la parte del archivo donde tenemos nuestros documentos de episodios y los copiamos todos.

****

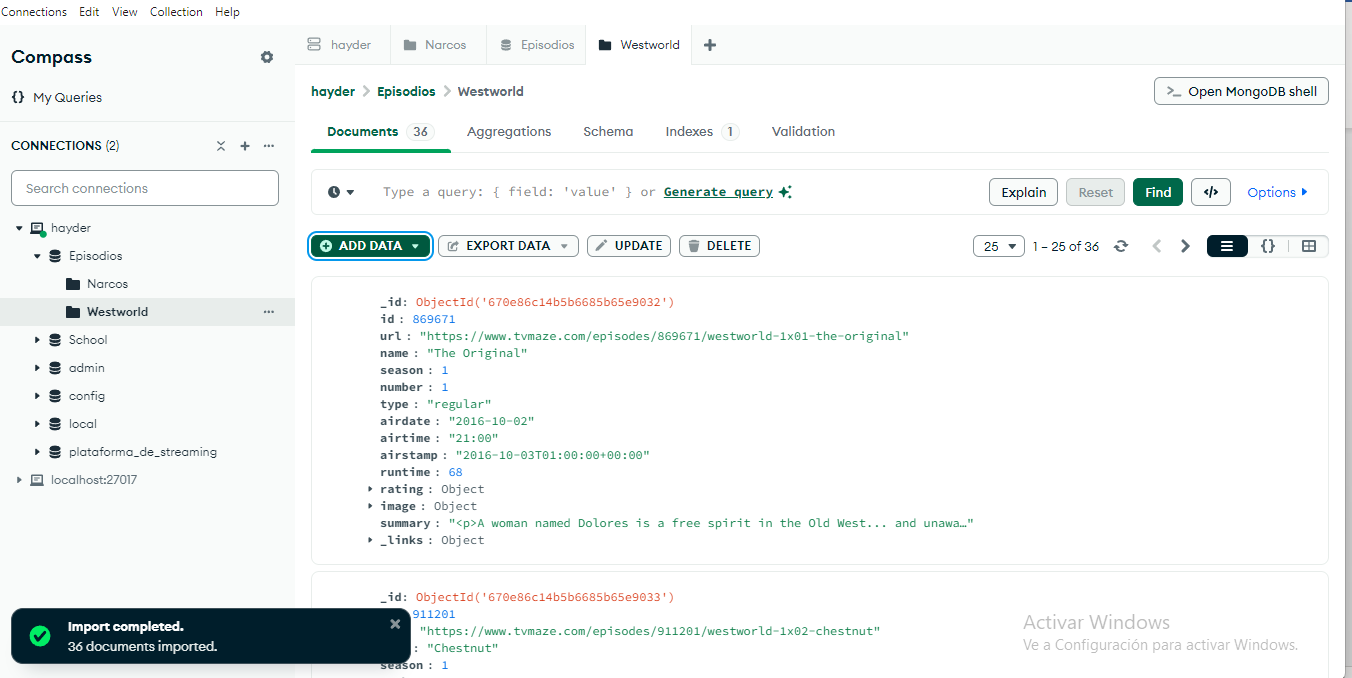
* Abrimos un editor de codigo y creamos un archivo llamado “episodios\_westworld.Json” para luego pegar los documentos de episodios copiados anteriormente.

****

* Ahora en nuestra base de datos “Episodios” creamos otra colección llamada “Westwolrd”.

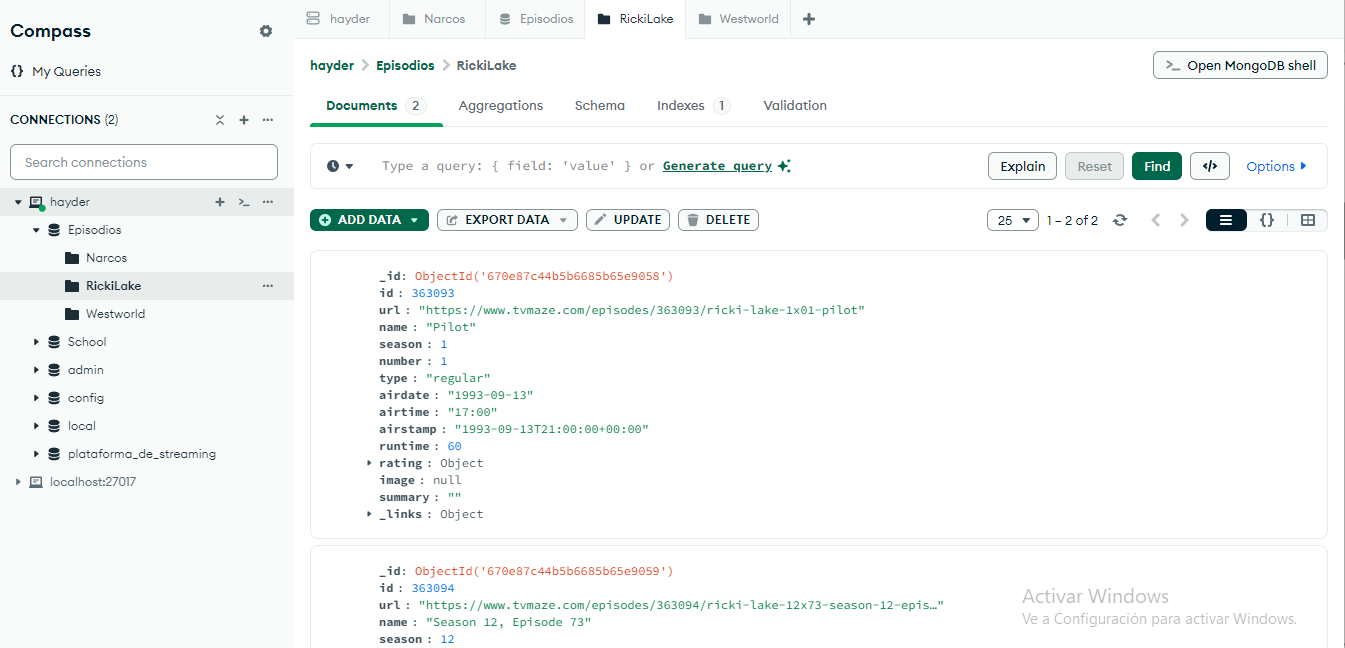
****

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

****

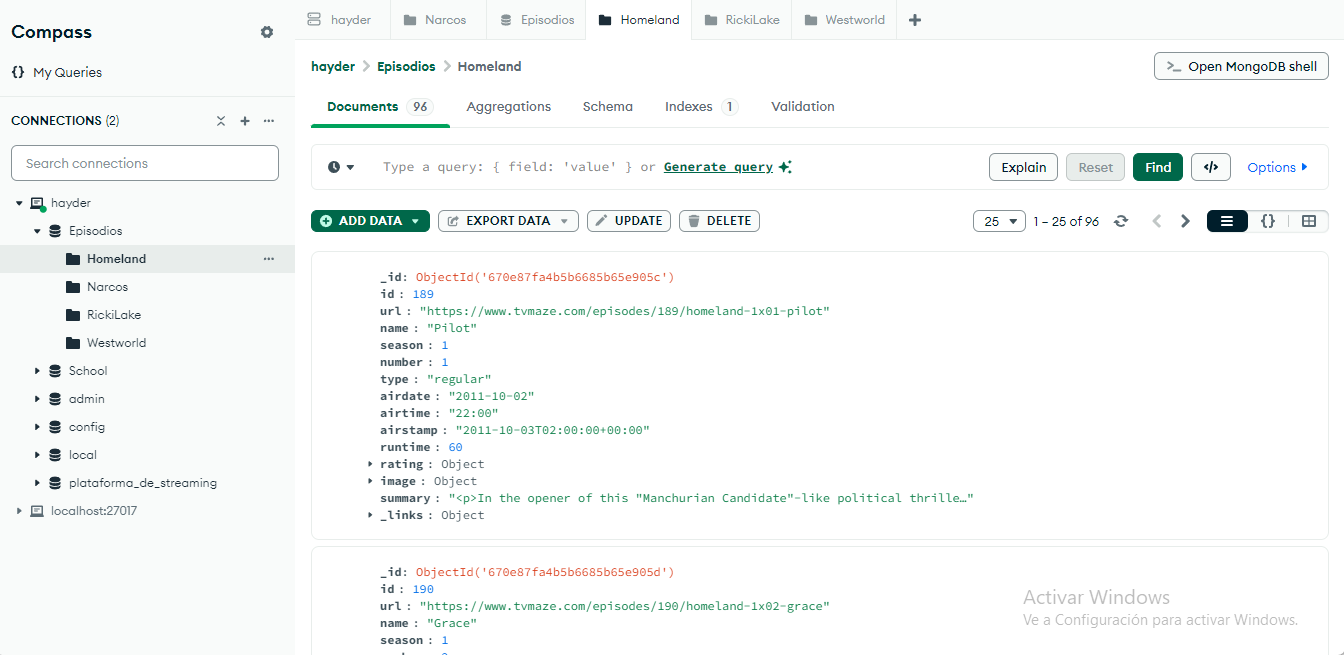
* Repite el paso anterior con el fichero “ejercicio\_02.json” pero esta vez la colección debe

llamarse “RickiLake”.



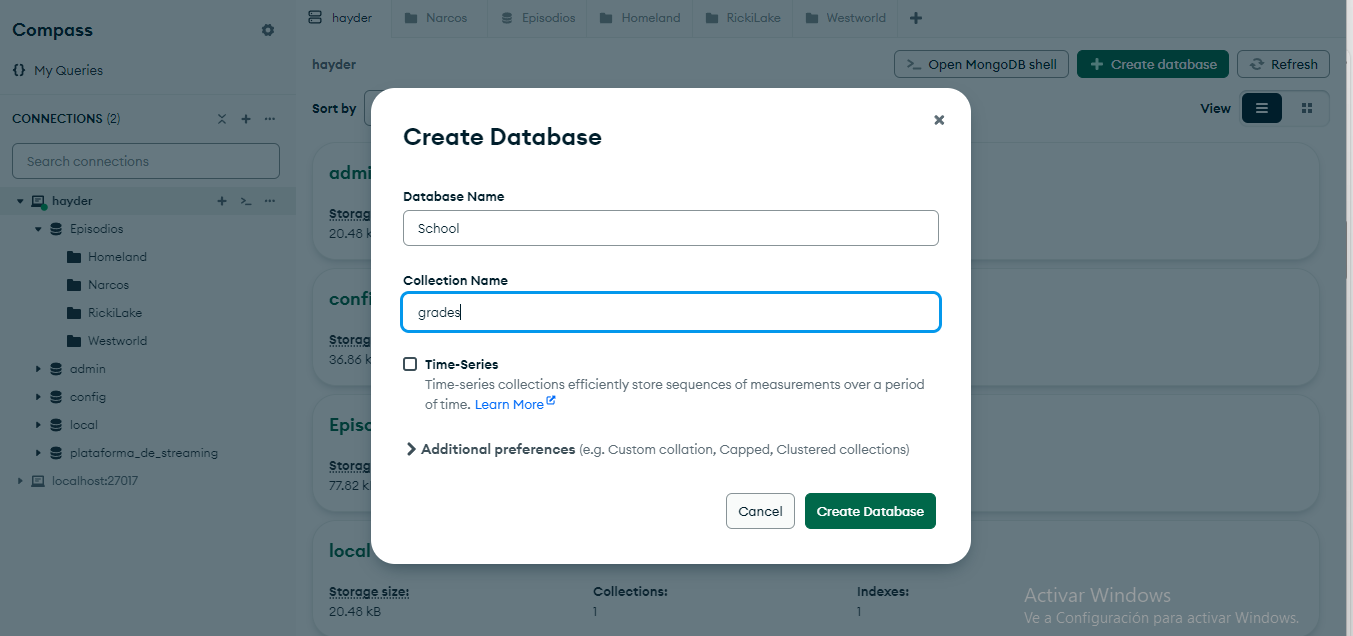
* Repite el paso anterior con el fichero “ejercicio\_03.json” pero esta vez la colección debe

llamarse “Homeland”.

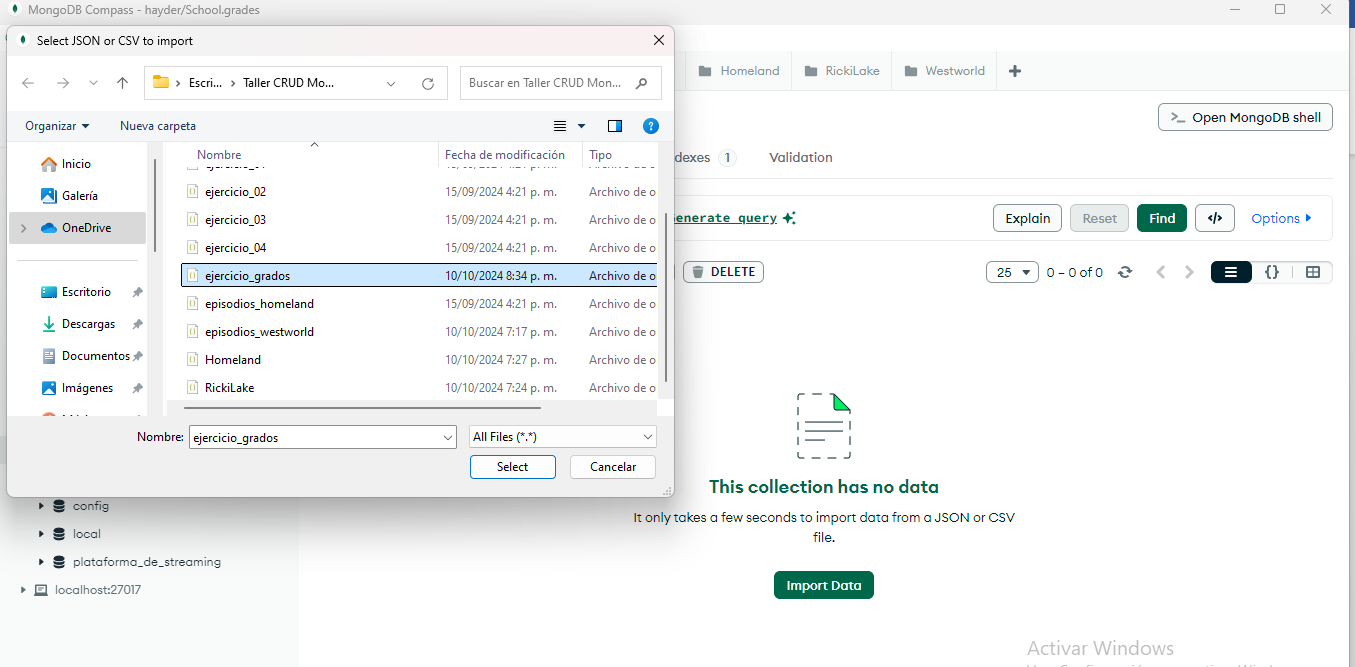


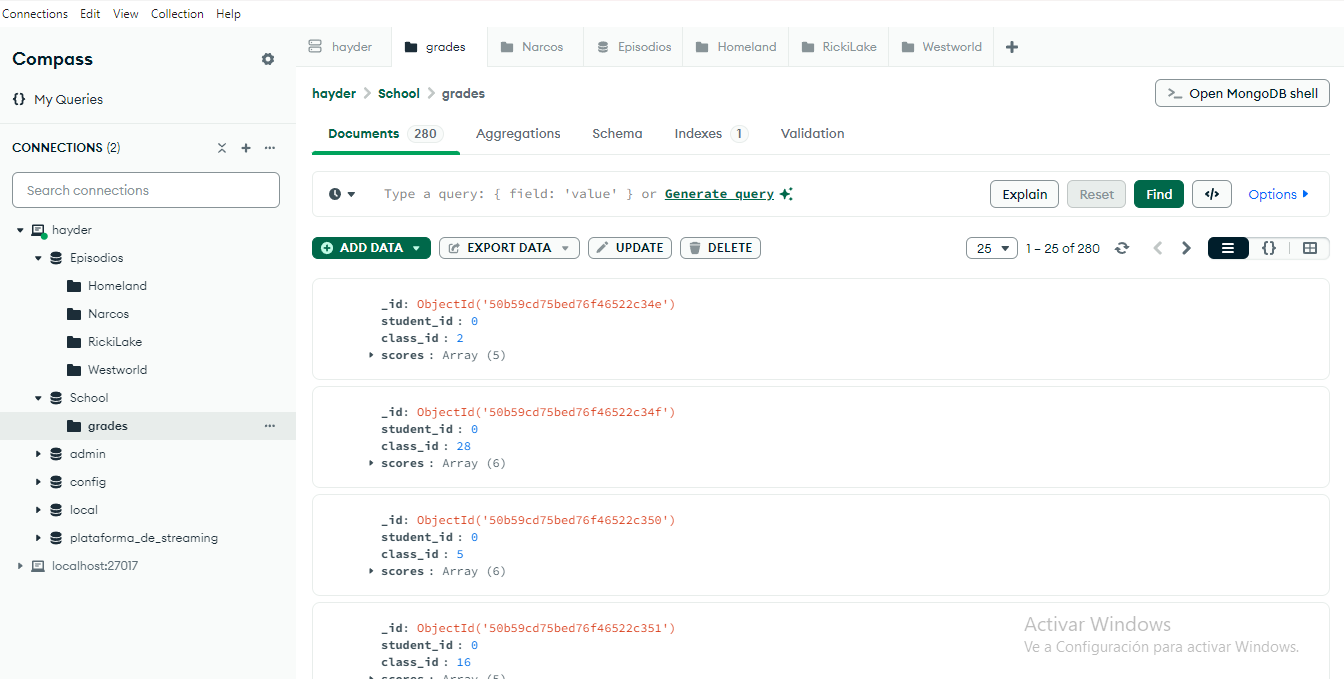
* Carga los datos del fichero “ejercicio\_grados.json” en la base de datos School sobre

la colección grades

****

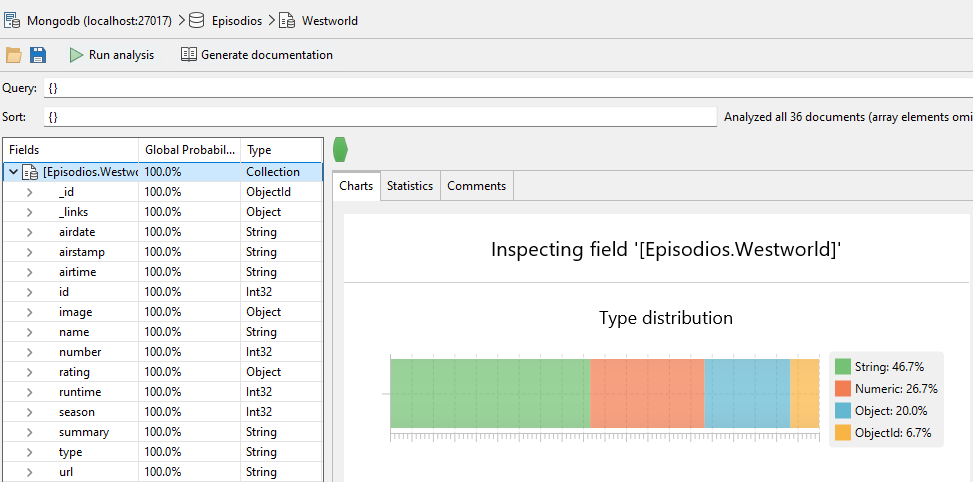
* Luego a nuestra colección “grades” importamos el archivo “ejercicio\_grades.json”.

****

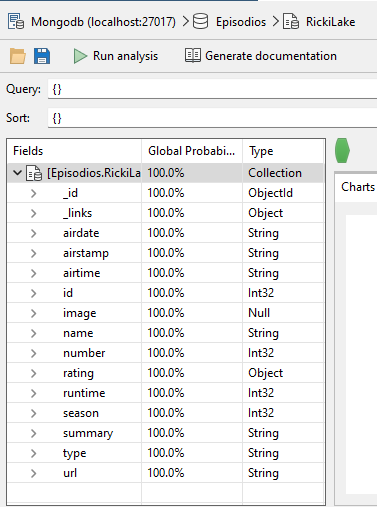
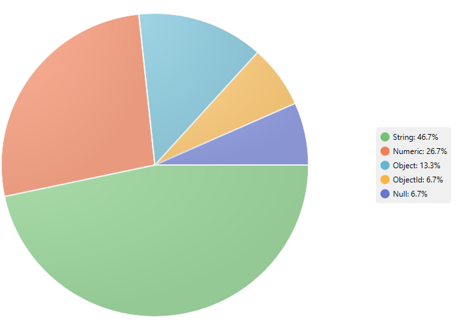
****

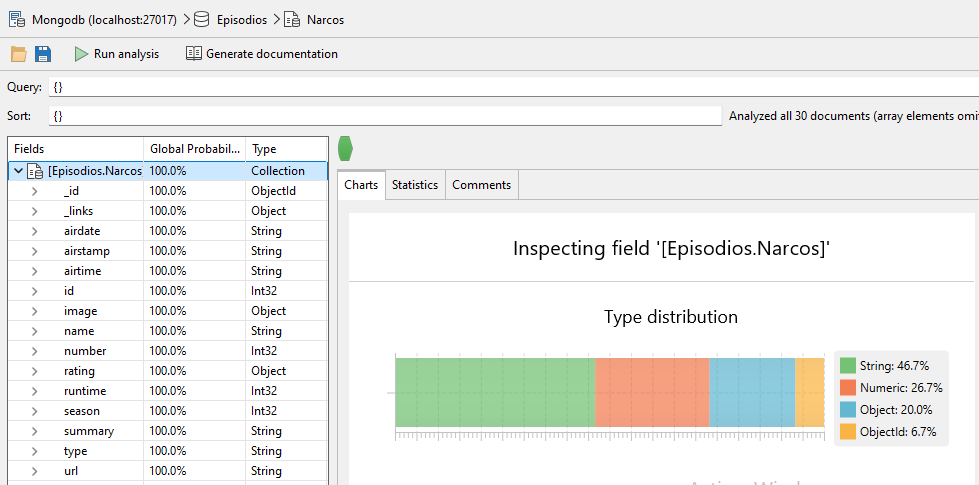
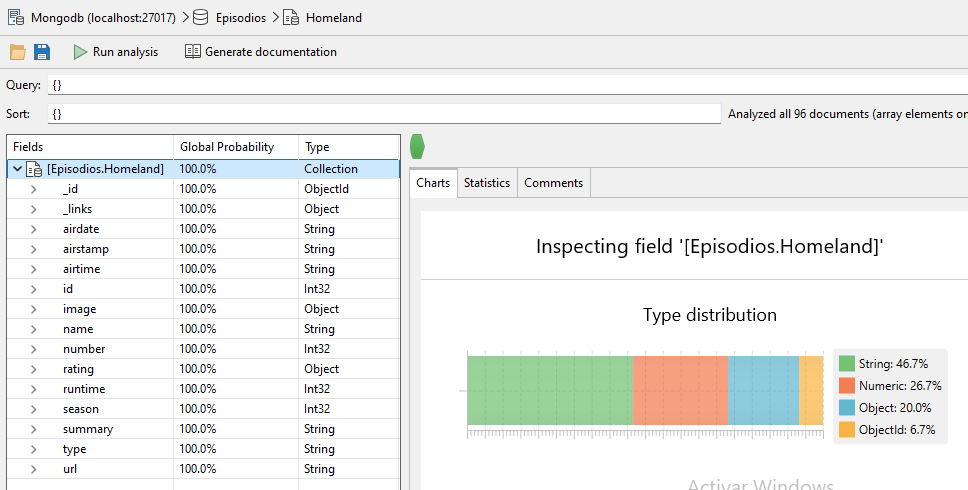
### **Esquema de las colecciones**

* **Colección “Westworld”:**

****

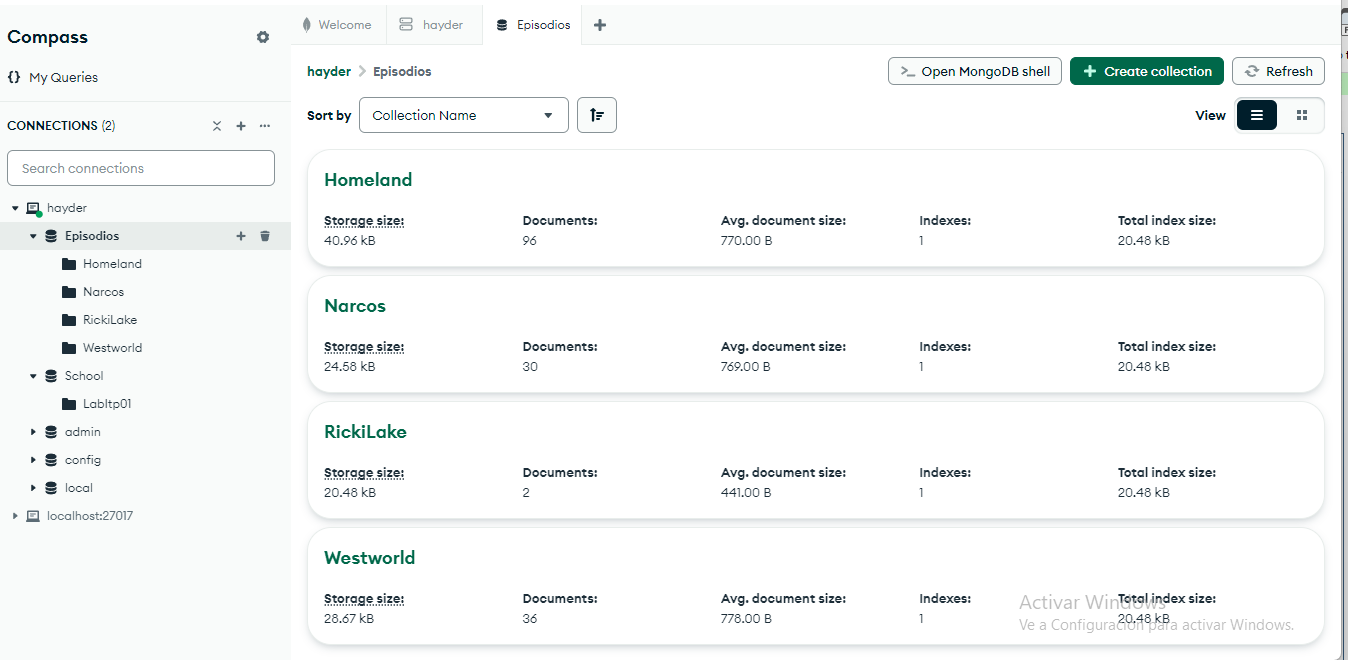
* **Colección “RickiLake”:**

**** ****

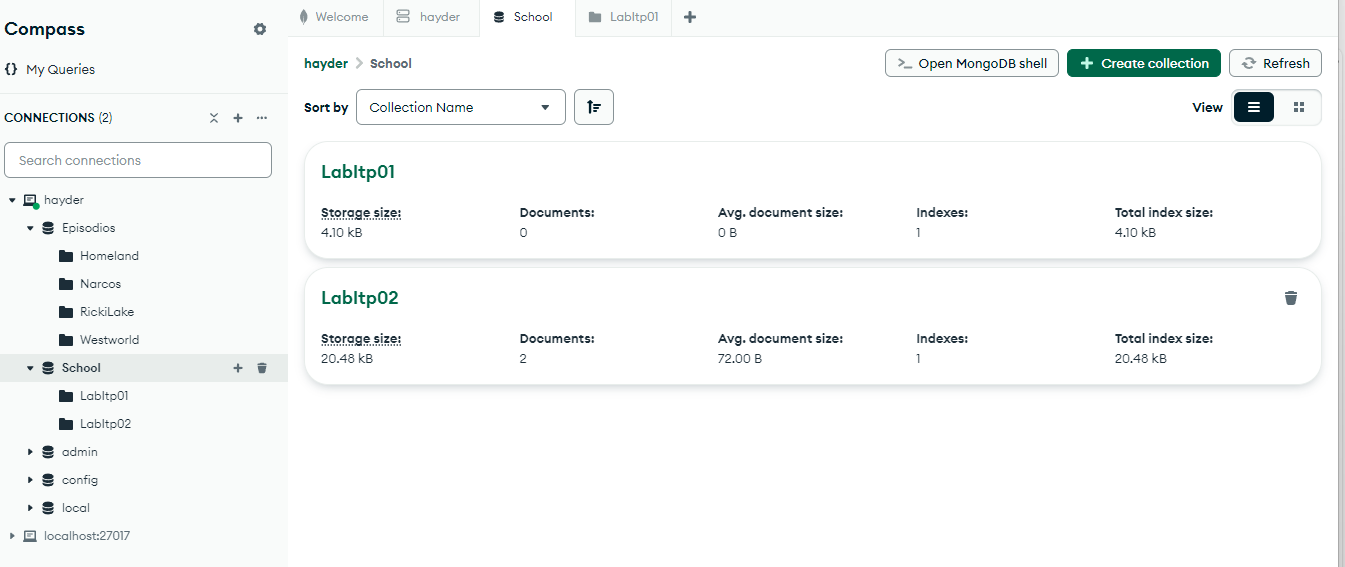
* **Colección “Narcos”**
* **Colección “Homeland”**

ESTRUCTURA DE LA BASE DE DATOS Y SUS COLECCIONES

**Base de datos “Episodios”:**

****Colecciones: Homeland, Narcos, RickiLake, Westworld

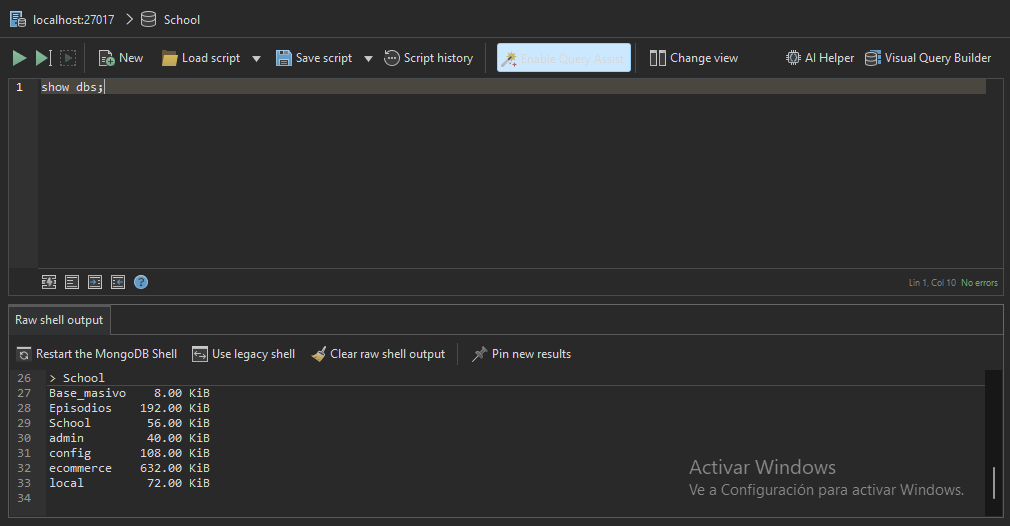
**Base de datos “School”:**

****Colecciones: grades, LabItp01, LabItp02

### **Consultas**

#### **Trabaja con READ**

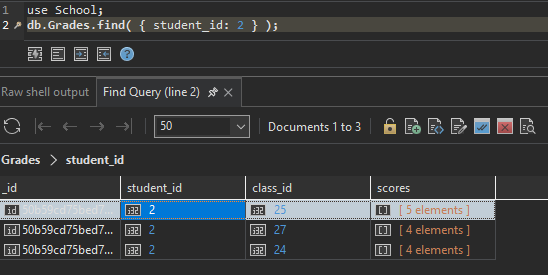
1. Visualiza todas las bases de datos existentes: show dbs



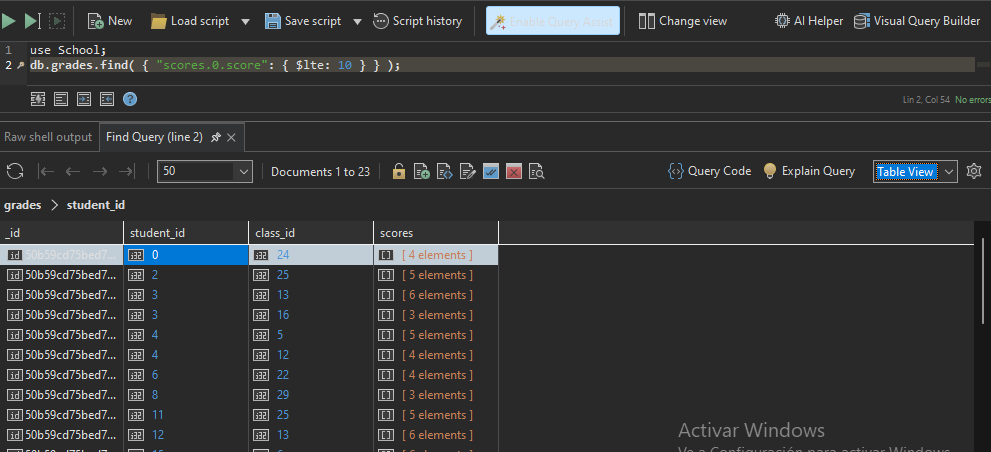
1. Localiza la base de datos School, ejecuta las siguientes consultas y describe qué

hace cada una de ellas:

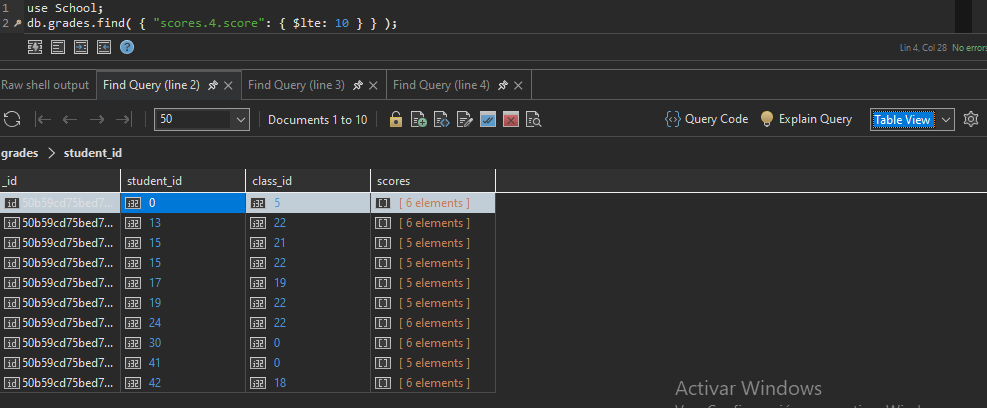
* **db.grades.find( { student\_id: 2 }** )

****

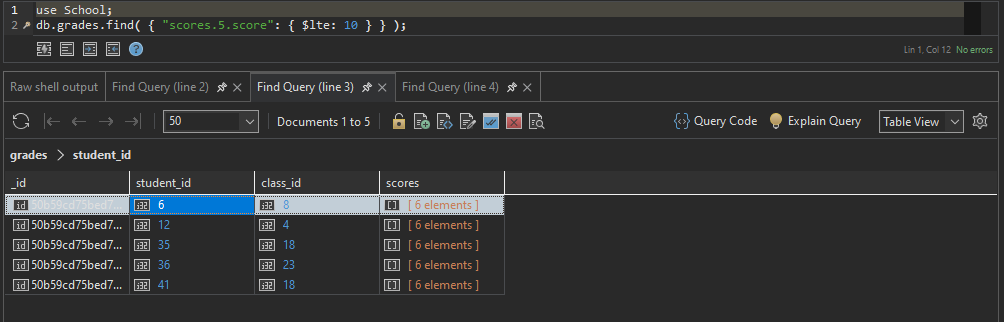
**Descripción**: Esta consulta busca todos los documentos en la colección grades donde el student\_id es igual a 2. Recuperará todas las calificaciones asociadas al estudiante con ID 2.

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

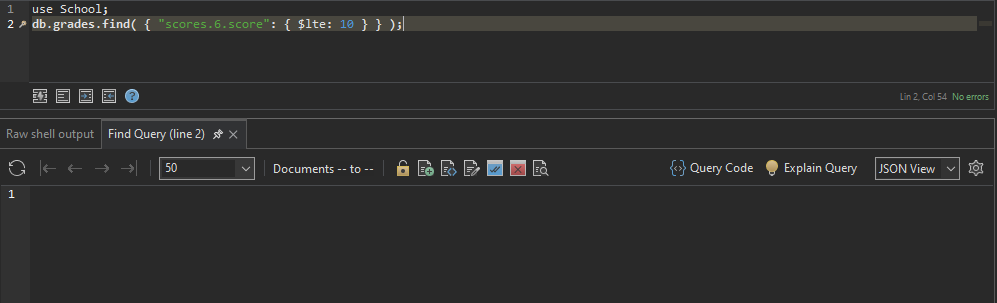
**Descripción**: Esta consulta busca documentos donde el primer puntaje en el arreglo scores (es decir, el puntaje con índice 0) sea menor o igual a 10. Es útil para identificar calificaciones bajas en el primer examen.

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

**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.

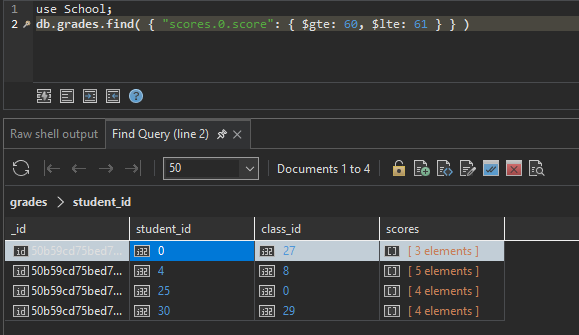
* **db.grades.find( { "scores.5.score": { $lte: 10 } } )**

**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.

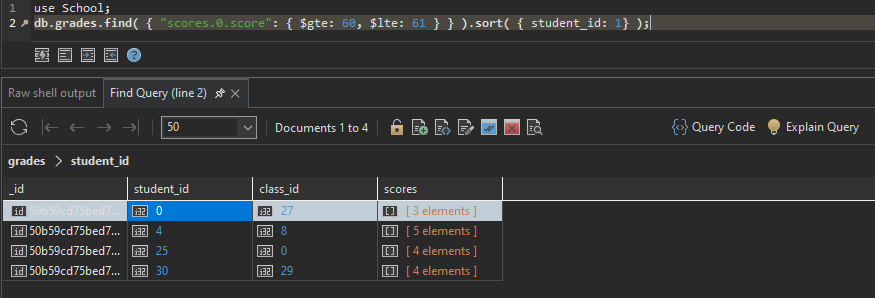
* **db.grades.find( { "scores.6.score": { $lte: 10 } } )**

**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.

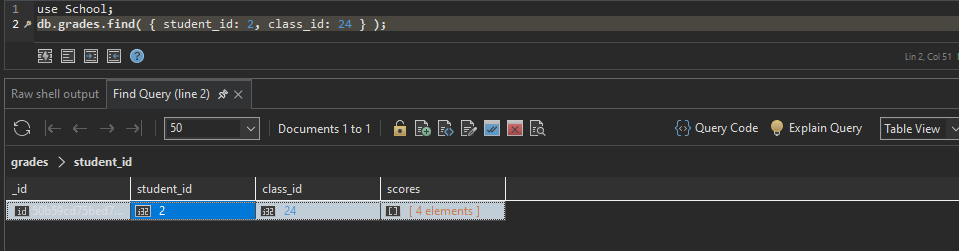
* **db.grades.find( { "scores.0.score": { $gte: 60, $lte: 61 } } )**

****

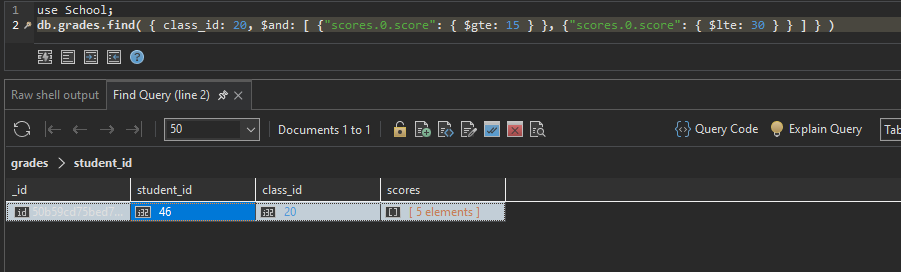
**Descripción**: Esta consulta busca documentos donde el puntaje en la posición 0 del arreglo scores esté entre 60 y 61, inclusive. Se utiliza para identificar estudiantes que obtuvieron una calificación casi aprobatoria en el primer examen.

* **db.grades.find( { "scores.0.score": { $gte: 60, $lte: 61 } } ).sort( { student\_id: 1} )**

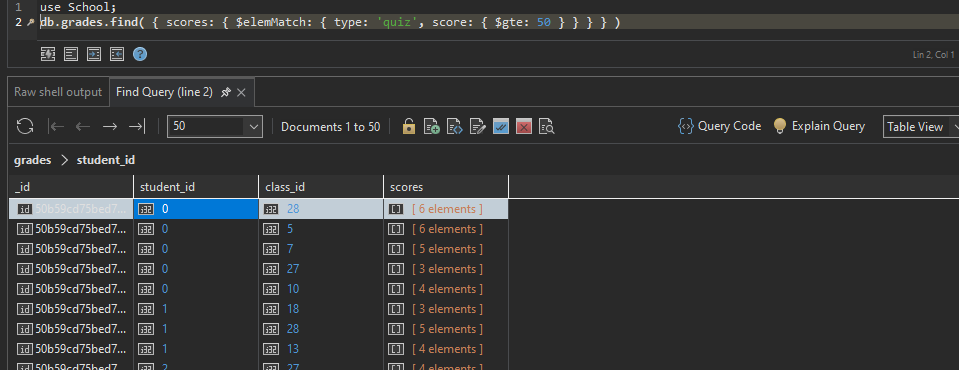
**Descripción**: Similar a la consulta anterior, pero añade un ordenamiento de los resultados por student\_id en orden ascendente. Esto facilita la visualización de las calificaciones de estudiantes específicos.

* **db.grades.find( { student\_id: 2, class\_id: 24 } )**

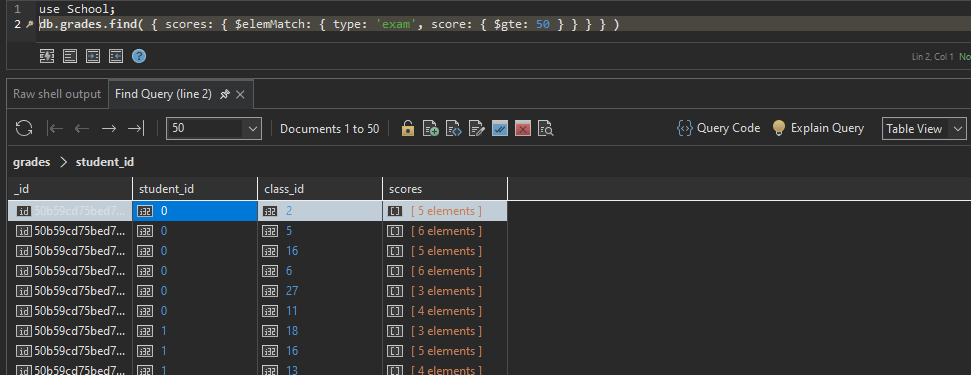
**Descripción**: Busca documentos donde student\_id sea 2 y class\_id sea 24. Esto permite obtener las calificaciones del estudiante 2 específicamente para la clase con ID 24.

* **db.grades.find( { class\_id: 20, $and: [ {"scores.0.score": { $gte: 15 } }, {"scores.0.score": { $lte: 30 } } ] } )**

**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 } } } } )**

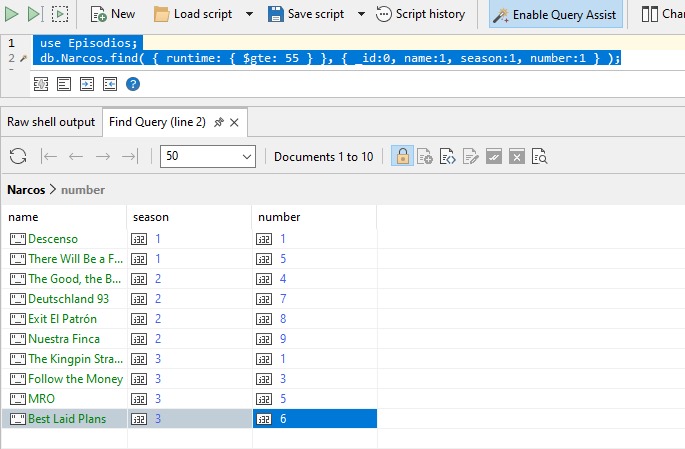
**Descripción**: Busca documentos donde al menos un elemento en el arreglo scores sea de tipo quiz y tenga un puntaje mayor o igual a 50. Esto permite identificar estudiantes que aprobaron al menos un examen tipo quiz.

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

**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.

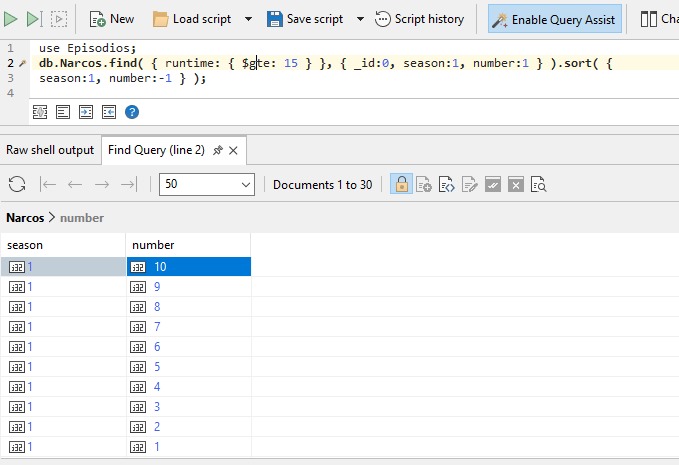
1. Busca la colección Narcos, ejecuta las siguientes consultas y describe qué hace cada una de ellas:

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



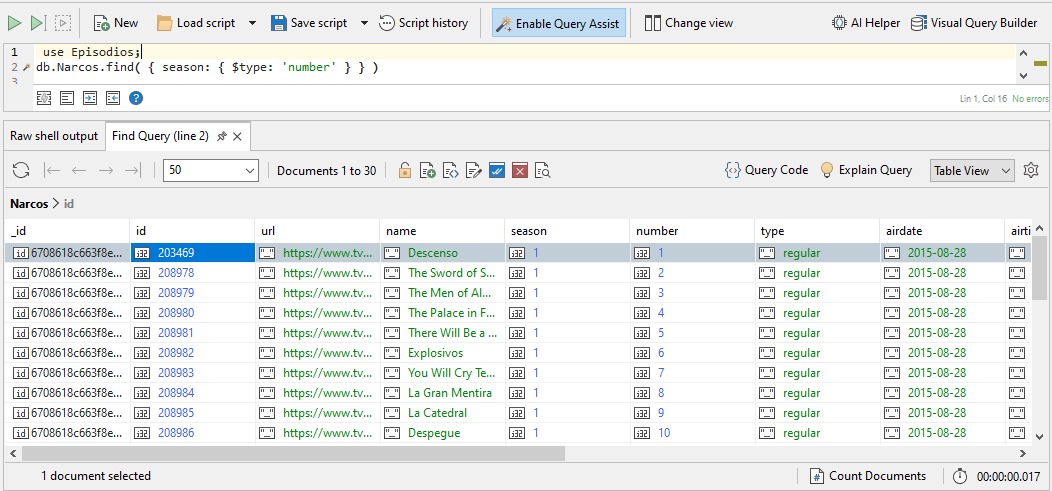
**Descripción**: Esta consulta busca documentos en la colección Narcos donde el runtime (duración) sea mayor o igual a 55 minutos. En la proyección de resultados, se excluye el campo \_id (por eso se usa 0) y se incluyen los campos name, season y number. Esto permite obtener un listado de episodios que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

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

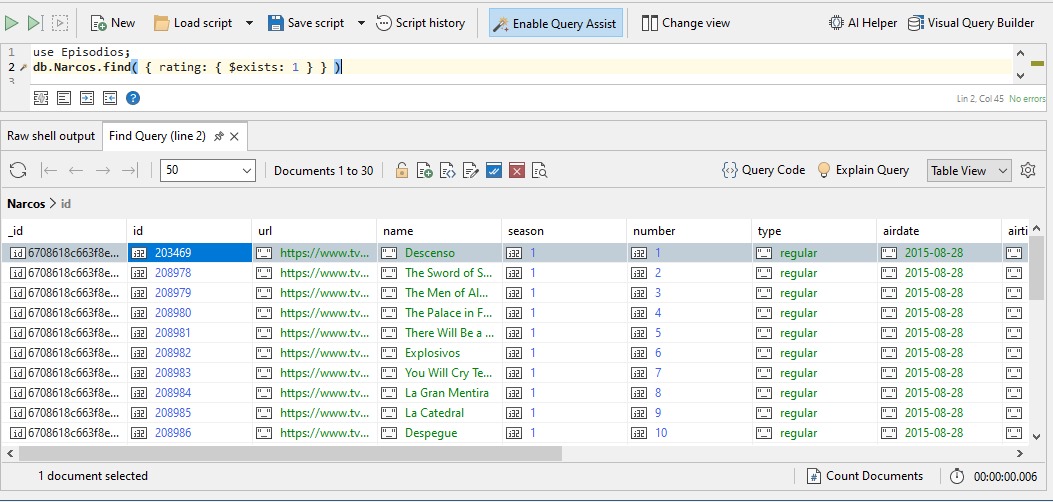


**Descripción**: Esta consulta busca documentos donde el runtime sea mayor o igual a 15 minutos. En la proyección, se excluye \_id y se incluyen season y number. Además, los resultados se ordenan por season en orden ascendente (1) y por number en orden descendente (-1). Esto es útil para listar episodios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

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



**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 } } )**

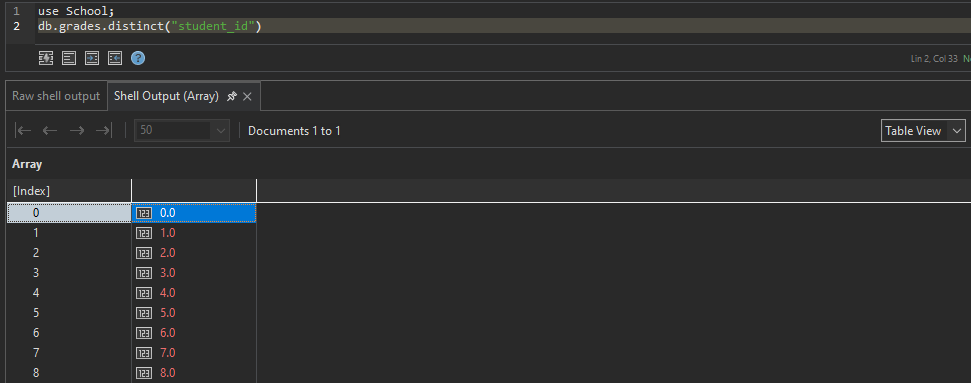
**Descripción**: Busca documentos donde el campo rating existe. Esto es útil para identificar todos los episodios que tienen una calificación asignada, independientemente de su valor.db.Narcos.find( { rating: { $exists: 1 }, rating: { $type: "string" } } )

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

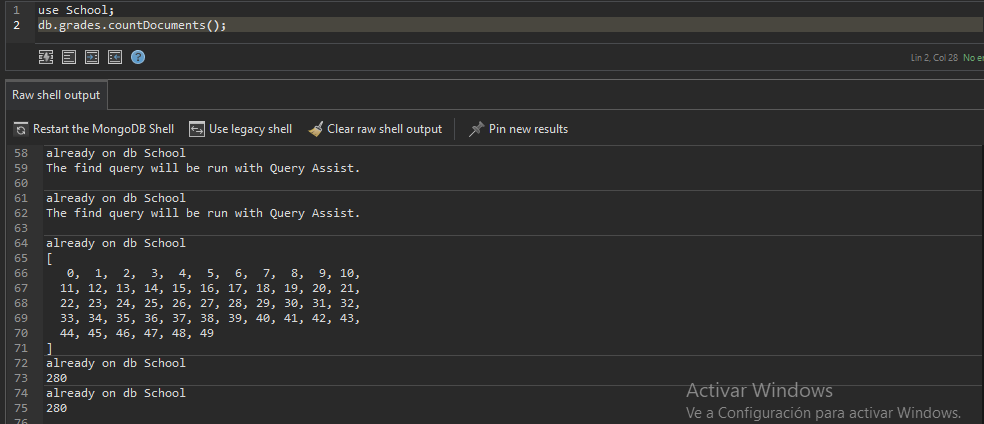
**Descripción**: Esta consulta busca documentos donde el campo rating existe y, además, es del tipo string. Esto puede ayudar a identificar episodios que tienen calificaciones almacenadas como cadenas de texto, lo que podría ser un error si se espera que sean números.

**LA CONSULTA NO FUNCION 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")**

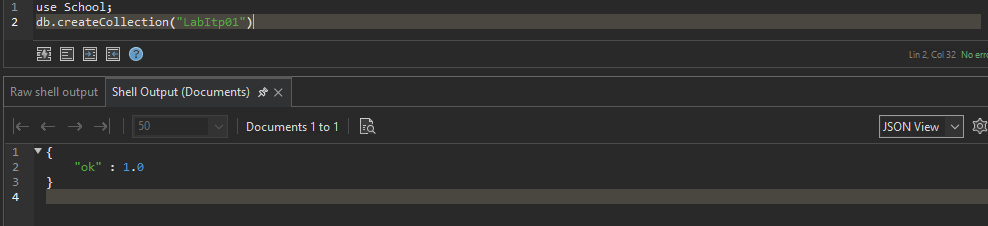
**Descripción**: Esta consulta obtiene todos los valores únicos del campo student\_id en la colección grades.

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

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

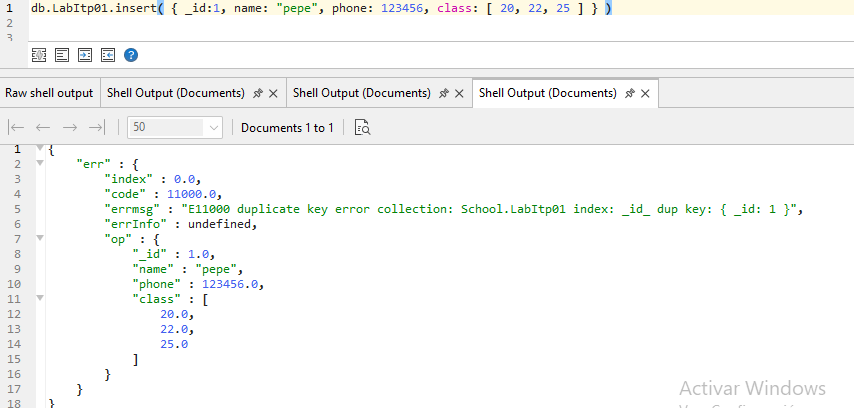
#### **Trabaja con CREATE**

En la BD School crea la colección LabItp:

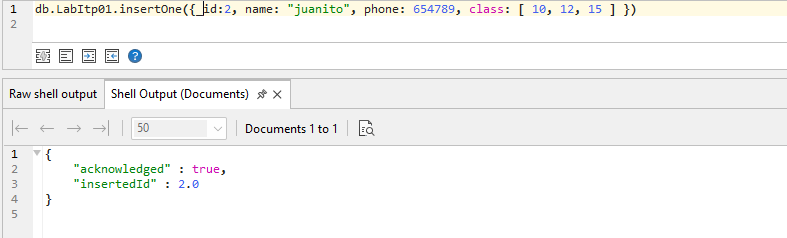
* **db.createCollection("LabItp01")**

**Descripcion:** 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.

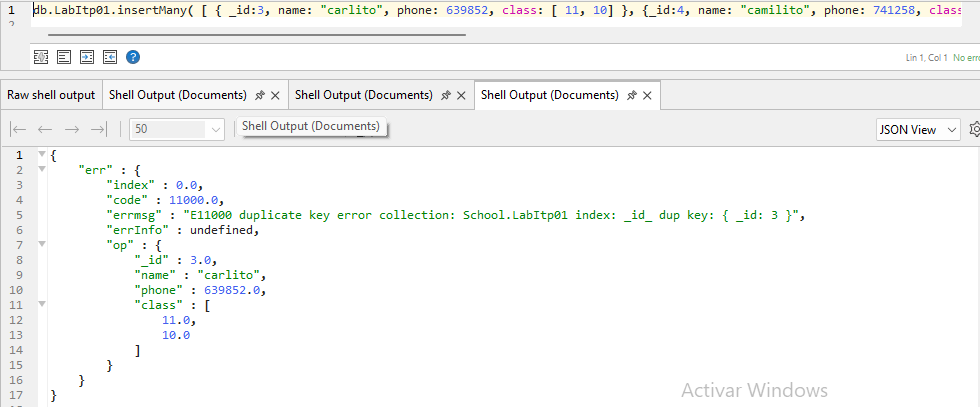
* **db.LabItp01.insert( { \_id:1, name: "pepe", phone: 123456, class: [ 20, 22, 25 ] } )**

****

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.

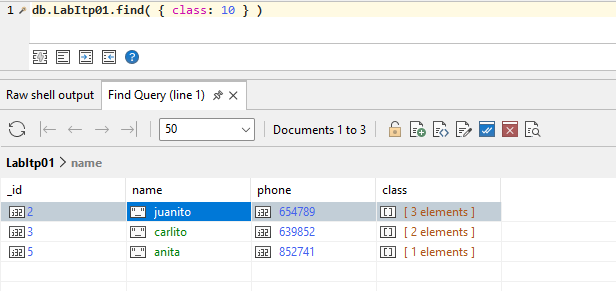
* **db.LabItp01.insertOne({\_id:2, name: "juanito", phone: 654789, class: [ 10, 12, 15 ] })**

**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] } ] )**

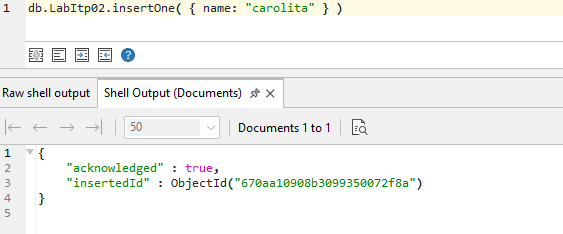
**Descripción:** Inserta varios documentos a la vez en la colección **LabItp01**. En este ejemplo, se están insertando cuatro documentos con diferentes valores de **\_id**, **name**, **phone** y **class**.

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.

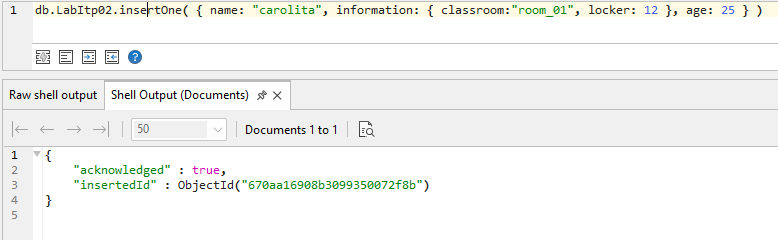
* **db.LabItp01.find( { class: 10 } )**

**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**.

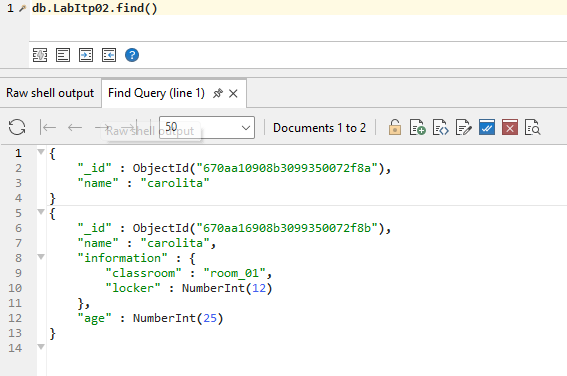
* **db.LabItp02.insertOne( { name: "carolita" } )**

****

**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 } )**

**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**.

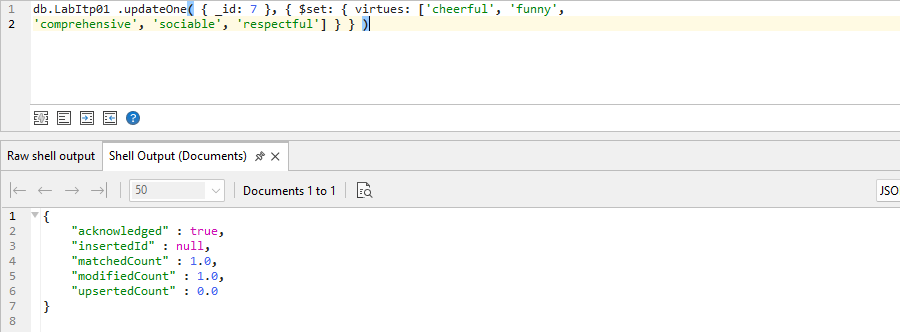
* **db.LabItp02.find()**

**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**

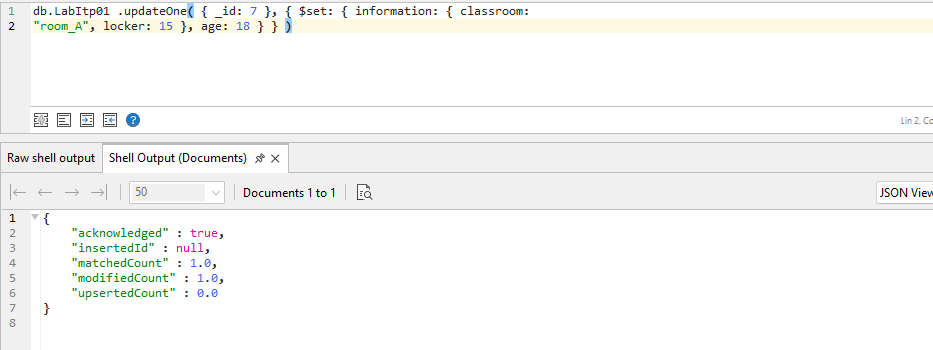
En la colección LabItp01 realiza las siguientes actualizaciones

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

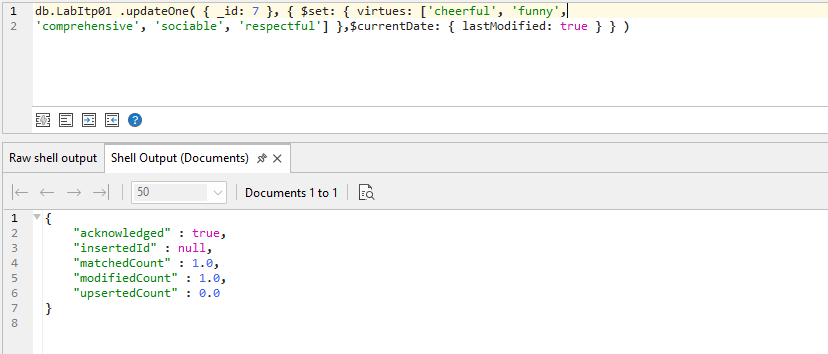
****

**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.

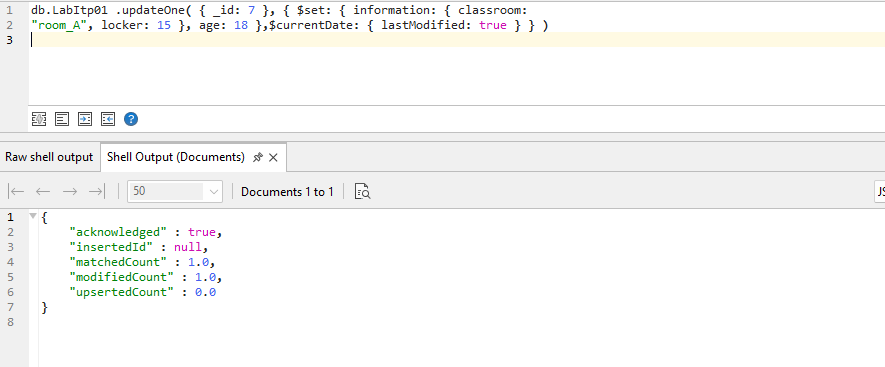
* **db.LabItp01 .updateOne( { \_id: 7 }, { $set: { information: { classroom:"room\_A", locker: 15 }, age: 18 } }**

****

**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.

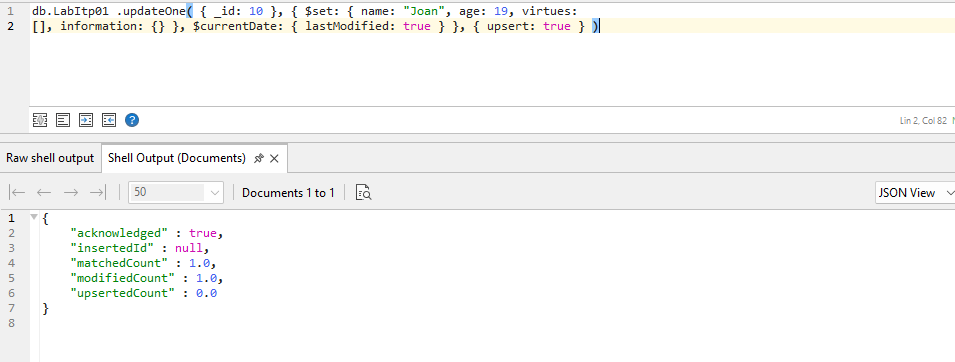
* **db.LabItp01 .updateOne( { \_id: 7 }, { $set: { virtues: ['cheerful', 'funny','comprehensive', 'sociable', 'respectful'] },$currentDate: { lastModified: true } } )**

**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 } } )**

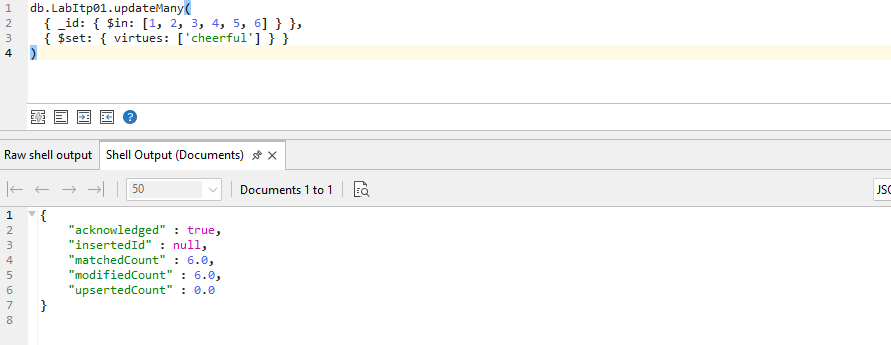
**Descripción:** Esta es una combinación de la segunda consulta y la tercera. Además de actualizar el campo information y age, también añade o actualiza el campo lastModified con la fecha y hora actuales.

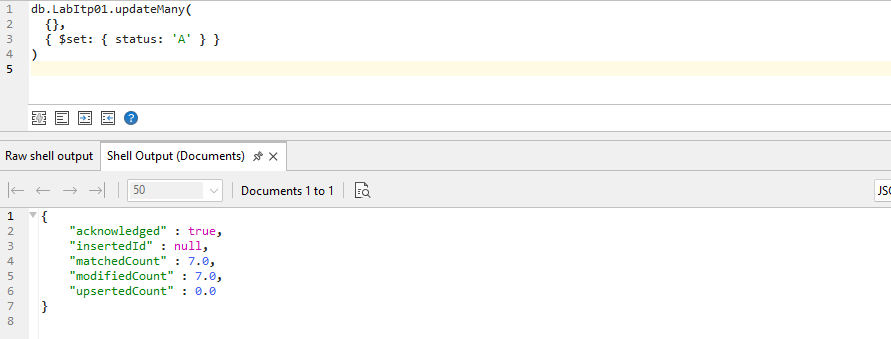
* **db.LabItp01 .updateOne( { \_id: 10 }, { $set: { name: "Joan", age: 19, virtues:[], information: {} }, $currentDate: { lastModified: true } }, { upsert: true } )**

****

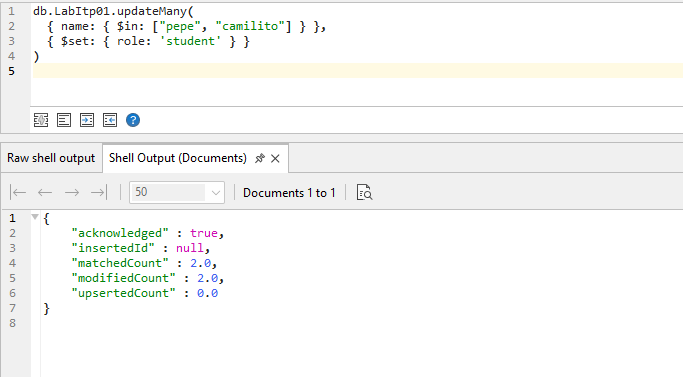
**Descripción:** Esta consulta busca un documento con \_id 10. Si lo encuentra, actualiza los campos name, age, virtues (con una lista vacía), e information (con un objeto vacío), y también añade o actualiza lastModified con la fecha y hora actuales. Si no encuentra ningún documento con \_id 10, crea 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'].**

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'.**

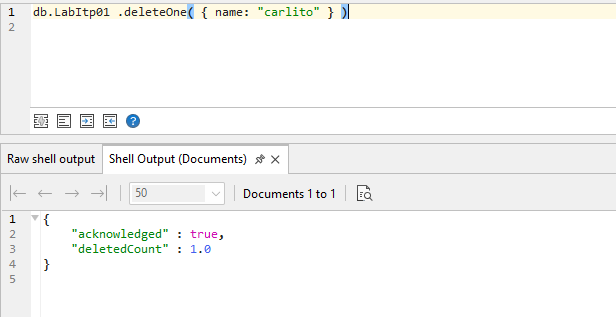
**Descripción:** Esta consulta actualiza todos los documentos en la colección LabItp01, añadiendo o modificando el campo status con el valor 'A'.

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

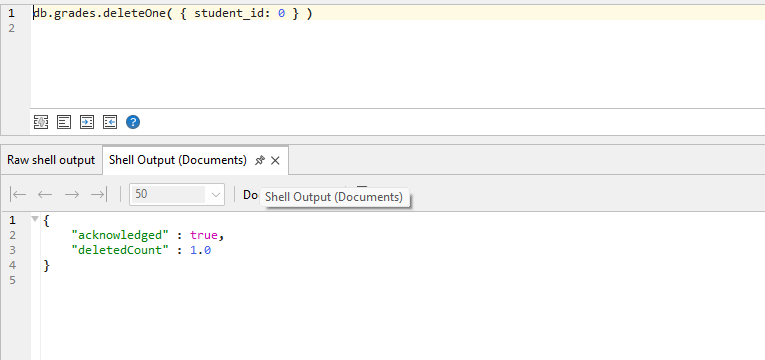
**Descripción:** Esta consulta actualiza todos los documentos en la colección LabItp01 donde el campo name es 'pepe' o 'camilito', asignándoles el valor 'student' en el campo role.

#### **Trabaja con DELETE**

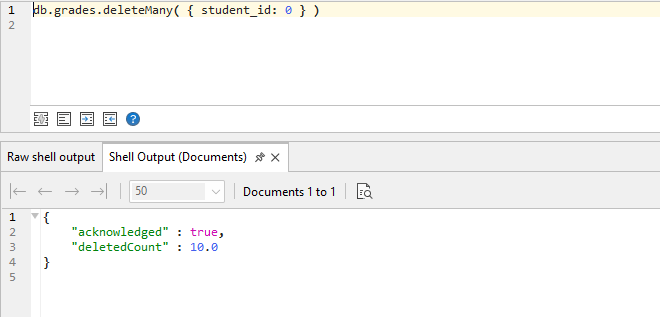
En la colección LabItp01 realiza las siguientes actualizaciones:

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

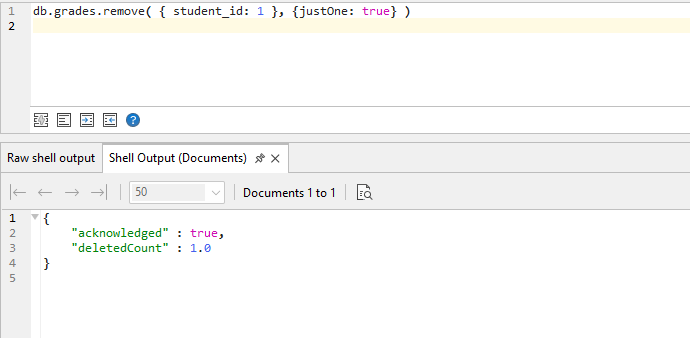
**Descripción:** Elimina un único documento de la colección **LabItp01** donde el campo **name** es igual a **"carlito"**. Si hay múltiples documentos que cumplen con esta condición, solo se eliminará el primero encontrado.

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

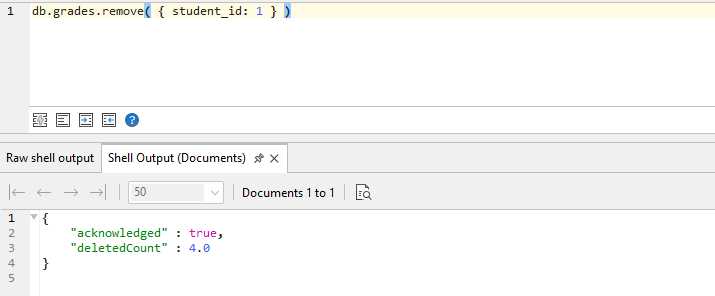
**Descripción:** Elimina un único documento de la colección **grades** donde el campo **student\_id** es igual a **0**. Similar a la consulta anterior, solo se eliminará el primer documento que coincida con la condición.

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

**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.

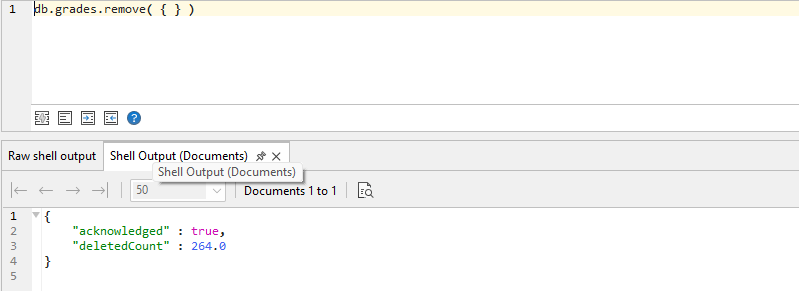
* **db.grades.remove( { student\_id: 1 }, {justOne: true} )**

**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.

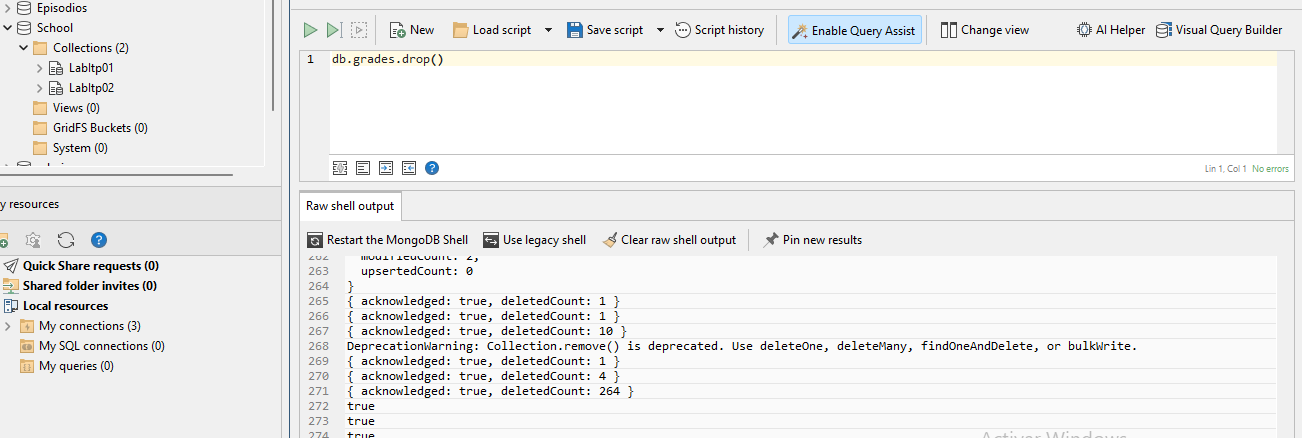
* **db.grades.remove( { student\_id: 1 } )**

**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( { } )**

****

**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()**

**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**

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/>.

**LINK DE GITHUB:** <https://github.com/hayderrojas84/BASE-DE-DATOS-Y-ALMACENAMIENTO-MASIVO.git>