TALLER CRUD Y BASE PERSONAL

ESTUDIANTE:

YEINER ADRIAN ANGULO ORDOÑEZ

DOCENTE:

BRAYAN IGNACIO ARCOS

INSTITUTO TECNOLOGICO DEL PUTUMAYO

MOCOA-PUTUMAYO

2024

Contenido

[RESUMEN EJECUTIVO 3](#_Toc179832759)

[INTRODUCCION 3](#_Toc179832760)

[METODOLOGIA 4](#_Toc179832761)

[HERRAMIENTAS USADAS 4](#_Toc179832762)

[PROCEDIMIENTOS 4](#_Toc179832763)

[DESARROLLO 5](#_Toc179832764)

[READ 6](#_Toc179832765)

[SCHOOL 6](#_Toc179832766)

[NARCOS: 11](#_Toc179832767)

[CREATE 15](#_Toc179832768)

[SCHOOL 15](#_Toc179832769)

[UPDATE 19](#_Toc179832770)

[DELETE: 24](#_Toc179832771)

[ESTADISTICAS DE CADA COLECCION: 27](#_Toc179832772)

[CONSIDERACION DE DISEÑO 32](#_Toc179832773)

[ANALISIS Y DISCUSIÓN 32](#_Toc179832774)

[CONCLUCION 33](#_Toc179832775)

[RECOMENDACIONES 33](#_Toc179832776)

[REFERENCIAS 33](#_Toc179832777)

[REPOSITORIO DE ALOJAMIENTO 33](#_Toc179832778)

# RESUMEN EJECUTIVO

El presente documento presenta el taller de practica enfocada en el uso de mongoDB, que es una base NoSQL que es ampliamente utilizada. A través de una serie de ejercicios vamos a profundizar en el manejo de las operaciones CRUD (crear, leer, actualizar y eliminar) mediante la manipulación de colecciones de datos en formato JSON. Se emplearán herramientas como son mongoDB Compass y JSON viewer para el cargue y la visualización de datos. Este taller va a permitir adquirir las habilidades necesarias para la gestión de datos y realizar consultas eficientes.

# INTRODUCCION

El presente taller tiene como objetivo profundizar en el uso practico de mongoDB, una presente base de datos documental ampliamente utilizada por una flexibilidad y capacidad de trabajar con datos no estructurados. A lo largo de esta actividad, se buscará guiar en la ejecución de dichas operaciones CRUD (crear, leer, actualizar y eliminar) en mongoDB, lo que permitirá comprender de manera más sólida sus particularidades y aplicaciones. Para esto, se utilizará herramientas como son la consola de mongoDB Compass, trabajando en diversos archivos JSON simulando bases reales.

# METODOLOGIA

La metodología que se aplico para este taller se basa en el aprendizaje practico y guiado. Se proporcionan varios ejemplos que simulan el manejo de datos no estructurados en mongoDB utilizando los archivos JSON como una fuente de información. A través del cargue, consultas, modificación y eliminación de datos en varias colecciones de la base lograremos desarrollar las habilidades claves para trabajar con bases de datos documentales.

# HERRAMIENTAS USADAS

MondoDB: Base de datos NoSQL utilizada para gestionar colecciones y documentos.

MongoDB Compass: interfaz grafica que permite la visualización y manipulación de datos en MongoDB

Mongosh o consola mongo: terminal interactiva para ejecutar comandos en mongoDB

JSON Viewer: herramienta online de visualización y manipulación de archivos JSON.

Archivos JSON: Se utilizaron archivos de formato JSON como ejemplo “ejercicio\_00. json”, etc.

Database-tool: para la exportación de las bases que manejamos.

# PROCEDIMIENTOS

\*Carga de datos JSON en diferentes colecciones dentro de la base de datos episodio y school.

\*Visualización y edición de datos utilizando herramientas como JSON Viewer

\*Operadores CRUD sobre las colecciones, incluyendo la inserción de datos con InsertOne e InsertMany, consultar con find y eliminación con delectOne y deleteMany.

\*Creacion y manejo de diferentes colecciones como narcos, westworld, rickilake y homeland para agrupar los datos en función de ejercicios.

# DESARROLLO

Descripción de la Base de Datos Personal

La base de datos personal diseñada para este taller se centra en la gestión de información de un restaurante. Se implementaron diferentes colecciones, como clientes, menú y pedidos, que representan distintos aspectos de la operación del restaurante, lo que permite una fácil manipulación y consulta de datos ya que está diseñada para que tenga relaciones de 1 a 1, de 1 a muchos y de muchos a muchos.

1. Esquema de la Base

La unión de colecciones se llevó a cabo utilizando Studio 3T, donde se analizó el esquema de las colecciones. Este análisis permite entender cómo se interrelacionan los datos entre clientes, los elementos del menú y los pedidos realizados. Se creó un esquema que define las relaciones entre las diferentes colecciones y sus documentos, asegurando que los datos estén organizados de manera eficiente. Por ejemplo, cada pedido se asocia con un cliente específico y puede incluir múltiples elementos del menú.

Diseño de la Base de Datos Personal

1. Modelo de Datos

El modelo de datos se desarrolló siguiendo los principios de normalización para evitar la redundancia de datos. Se evaluó la cardinalidad de las relaciones entre las colecciones, considerando si se debería embeber documentos (como los detalles del pedido en la colección de pedidos) o mantener referencias entre ellos, lo que afecta directamente la eficiencia de las consultas. La colección de clientes contiene información básica sobre cada cliente, la colección de menú incluye los platillos disponibles, y la colección de pedidos registra las órdenes realizadas, junto con la referencia a los clientes y a los elementos del menú.

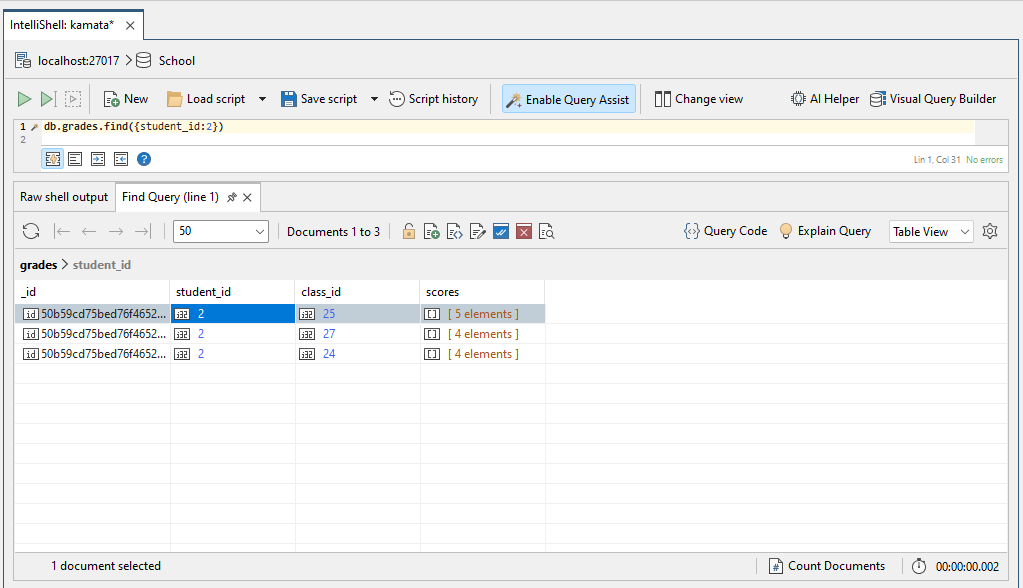
2. Consideraciones de Diseño

Se definieron relaciones entre las colecciones, asegurando que se utilizaran campos comunes (como el ID del cliente y el ID del menú) para facilitar la integración y la consulta de datos. Las decisiones de diseño se basaron en las necesidades específicas del restaurante y en las mejores prácticas para la gestión de bases de datos NoSQL, garantizando un acceso eficiente a la información.

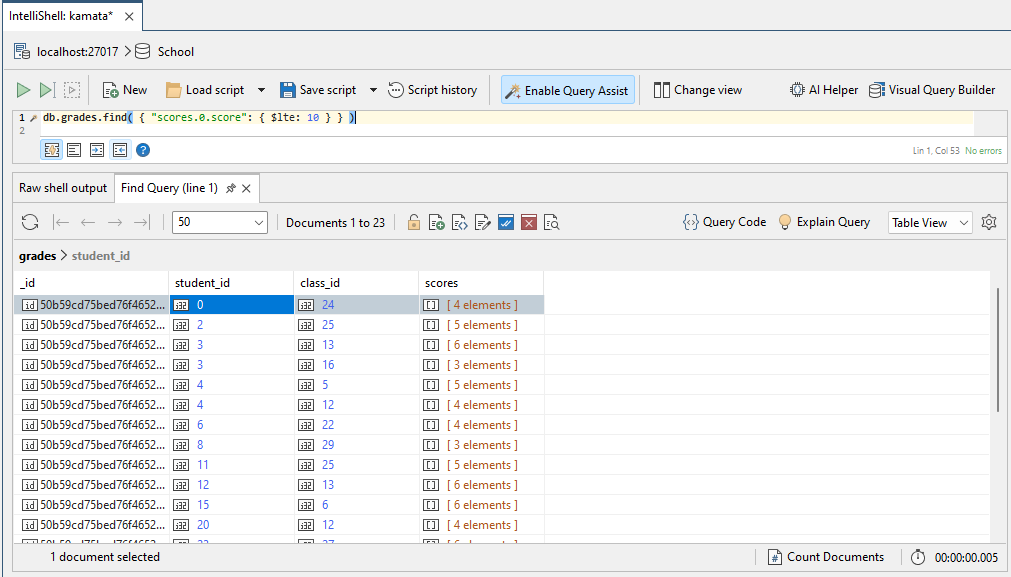
# READ

## SCHOOL

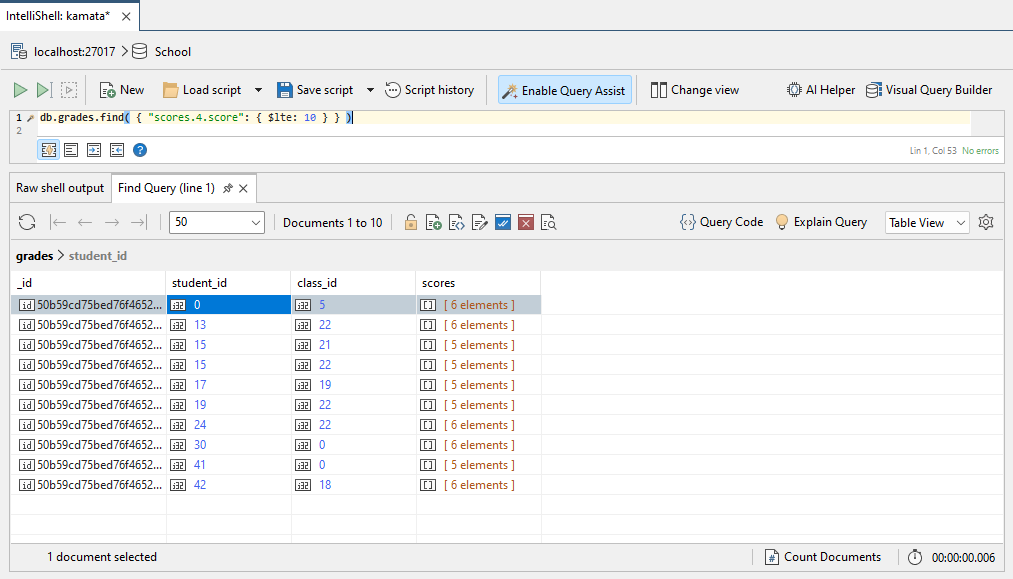
\* db.grades.find({student\_id:2}): Busca todos los documentos en la colección grades donde el campo student\_id es igual a 2.



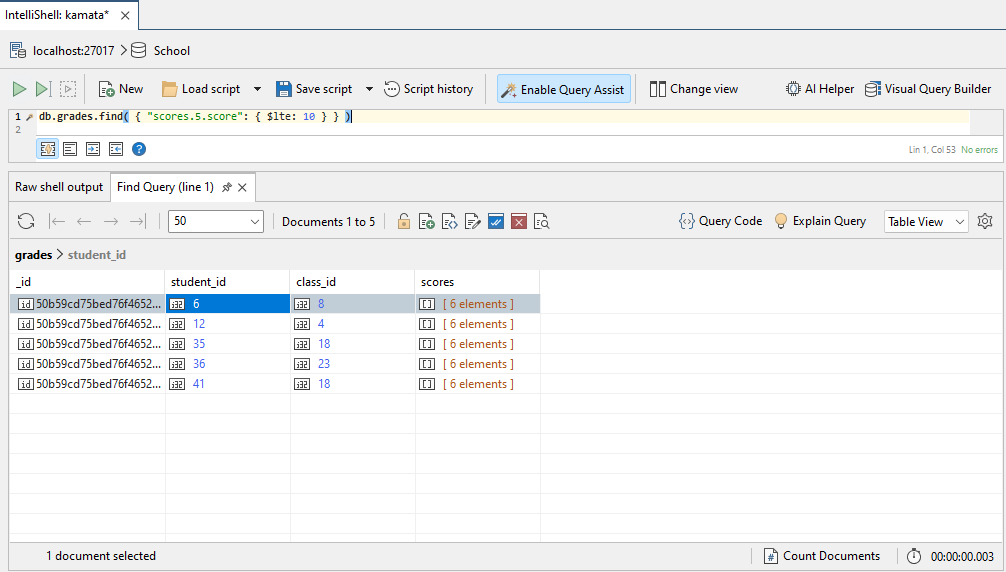
\* db.grades.find( { "scores.0.score": { $lte: 10 } } ):busca los documentos donde el primer elemento del array scores tiene un campo score menor o igual a 10.



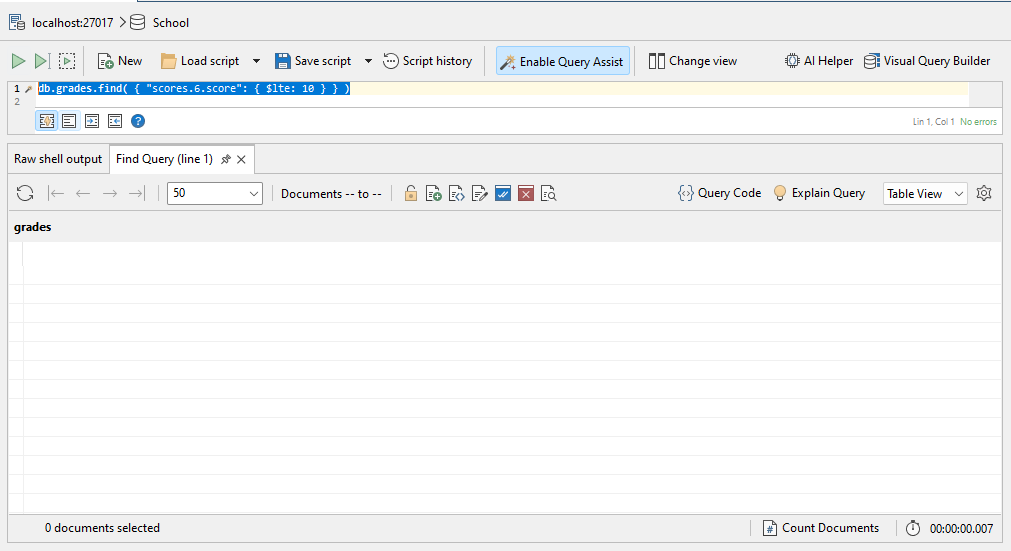
\* db.grades.find( { "scores.4.score": { $lte: 10 } } ): busca documentos donde el quinto elemento del array scores tiene un campo score menor o igual a 10

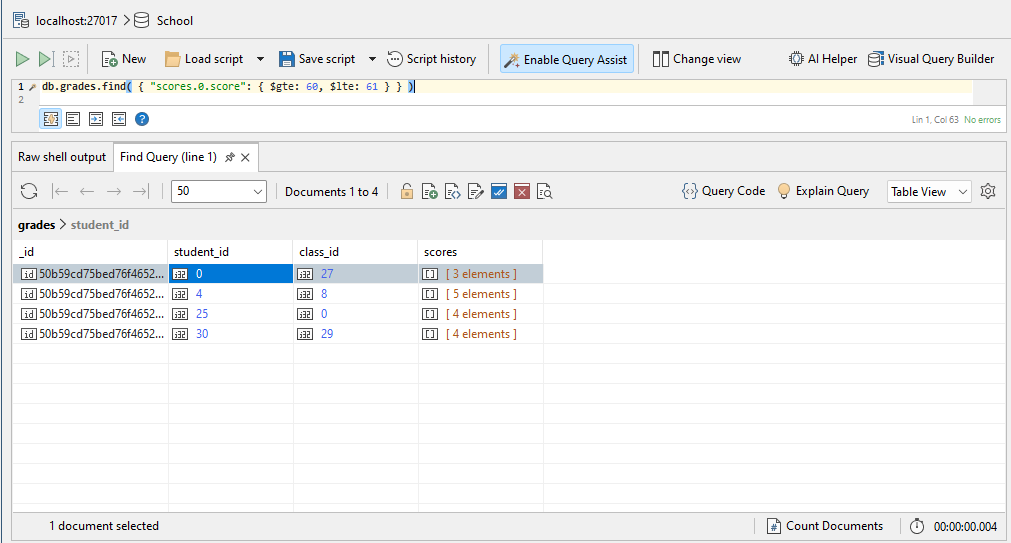


\* db.grades.find( { "scores.5.score": { $lte: 10 } } ):busca los documentos donde el sexto elemento del array scores tiene un campo score menor o igual a 10.

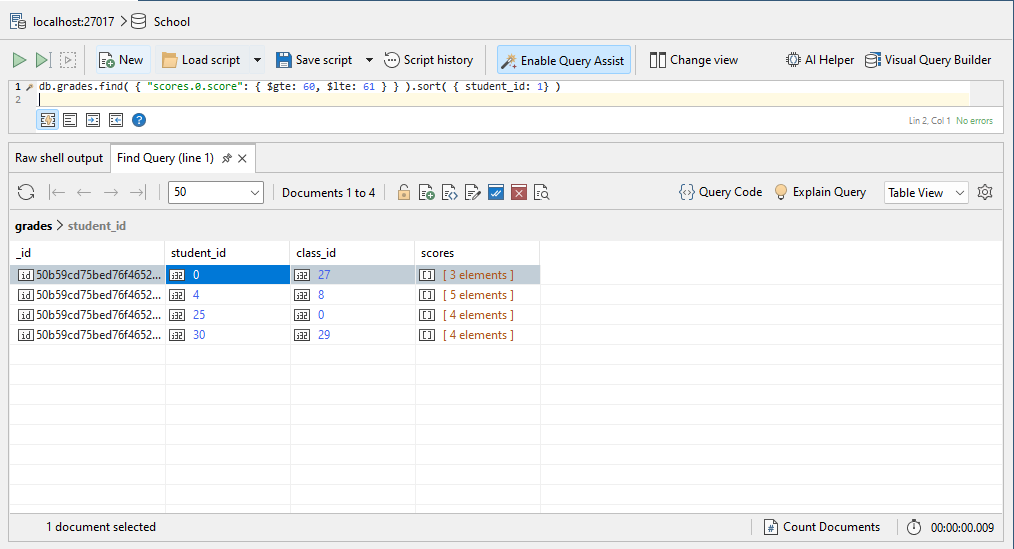


\* db.grades.find( { "scores.6.score": { $lte: 10 } } ): busca documentos donde el séptimo elemento del array scores tiene como campo score menor o igual a 10.

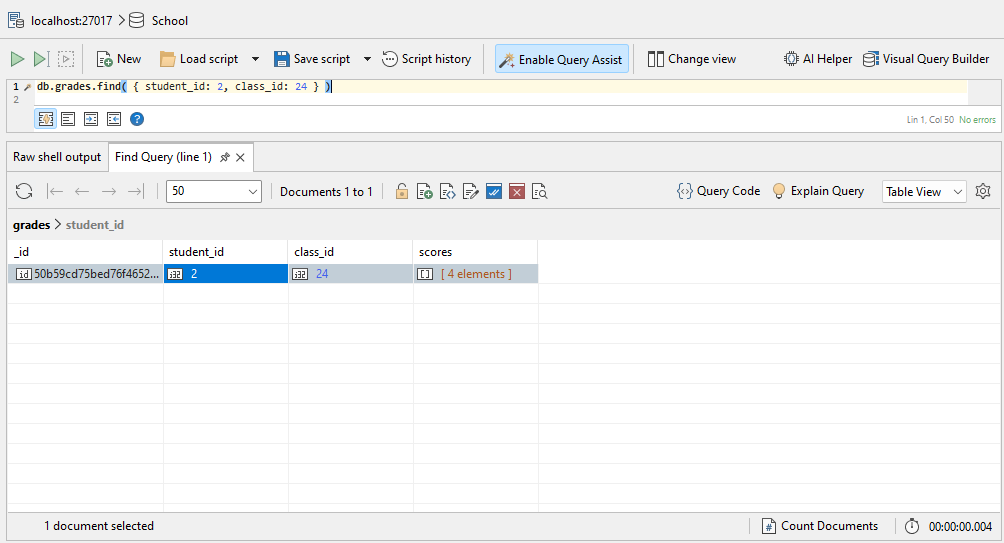


\* db.grades.find( { "scores.0.score": { $gte: 60, $lte: 61 } } ): busca documentos donde el séptimo elemento del array scores tiene un campo score en el rango entre 60 y 61.

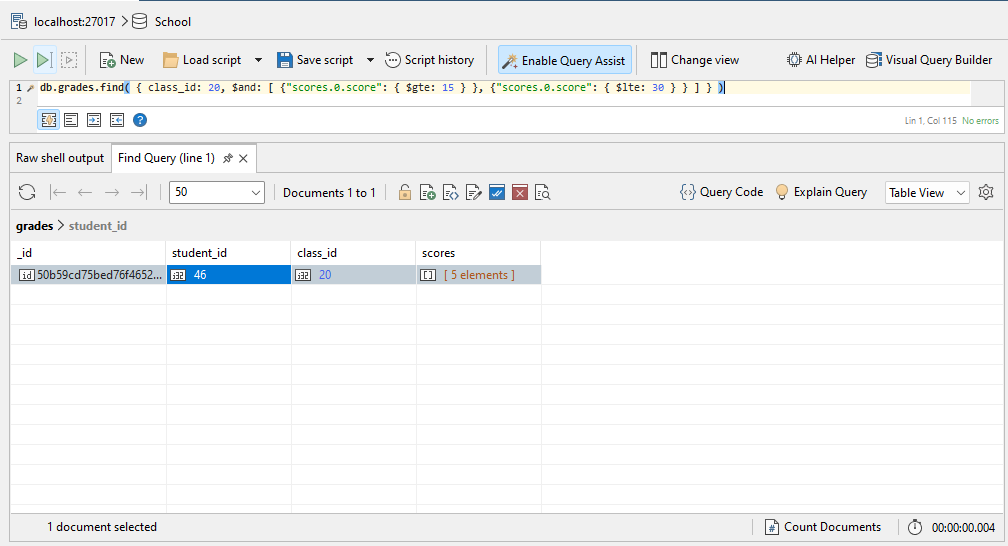
\* db.grades.find( { "scores.0.score": { $gte: 60, $lte: 61 } } ).sort( { student\_id: 1} ):realiza la misma búsqueda que la consulta anterior pero ordena los resultados por el campo student\_id en orden ascendente.



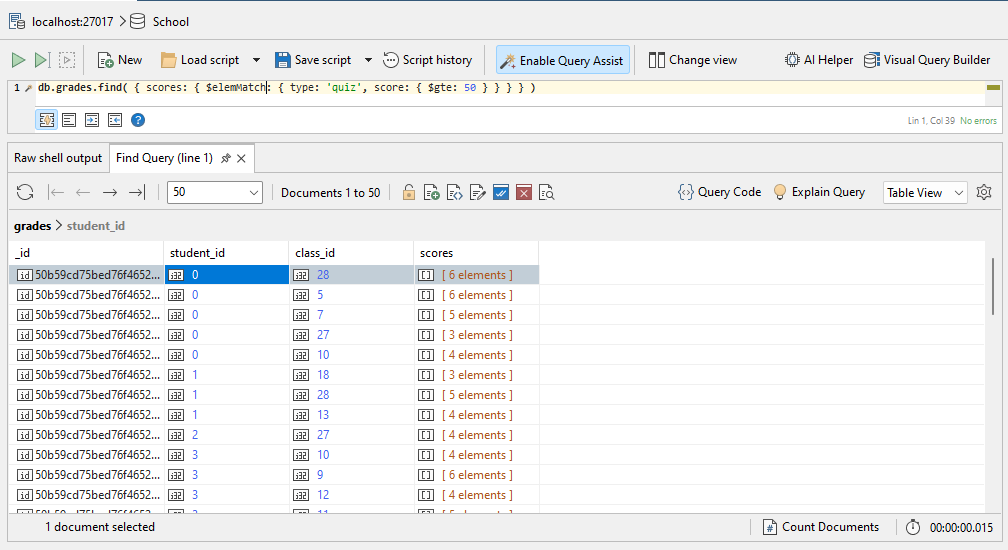
\* db.grades.find( { student\_id: 2, class\_id: 24 } ):busca documentos donde el student\_id es 2 y el class\_id es 24.



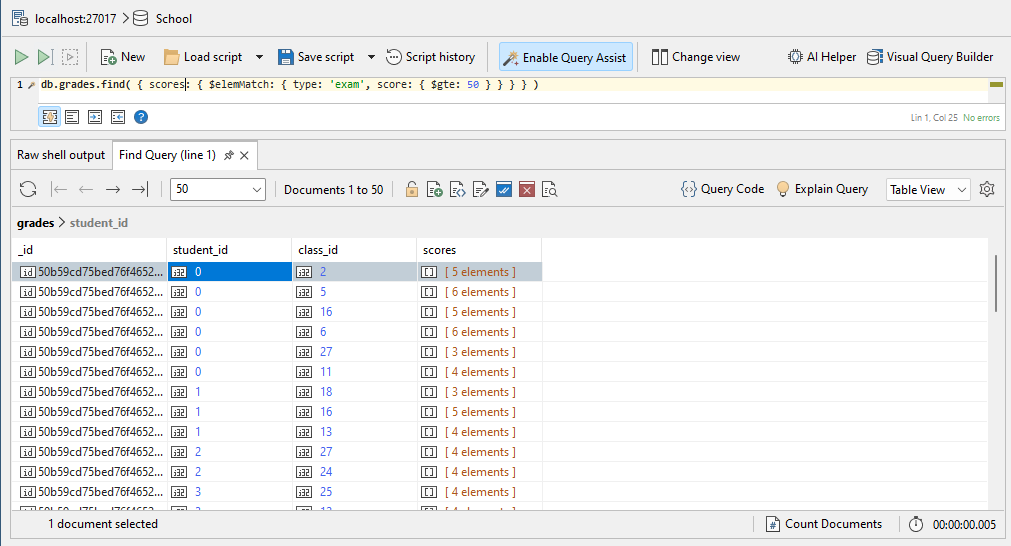
\* db.grades.find( { class\_id: 20, $and: [ {"scores.0.score": { $gte: 15 } }, {"scores.0.score": { $lte: 30 } } ] } ):busca documentos donde el class\_id es 20 y el primer elemento del array scores tiene un score entre 15 y 30.



\*db.grades.find( { scores: { $elemMatch: { type: 'quiz', score: { $gte: 50 } } } } ):busca documentos donde el array scores hay al menos un elemento de tipo quiz con un score mayor o igual a 50

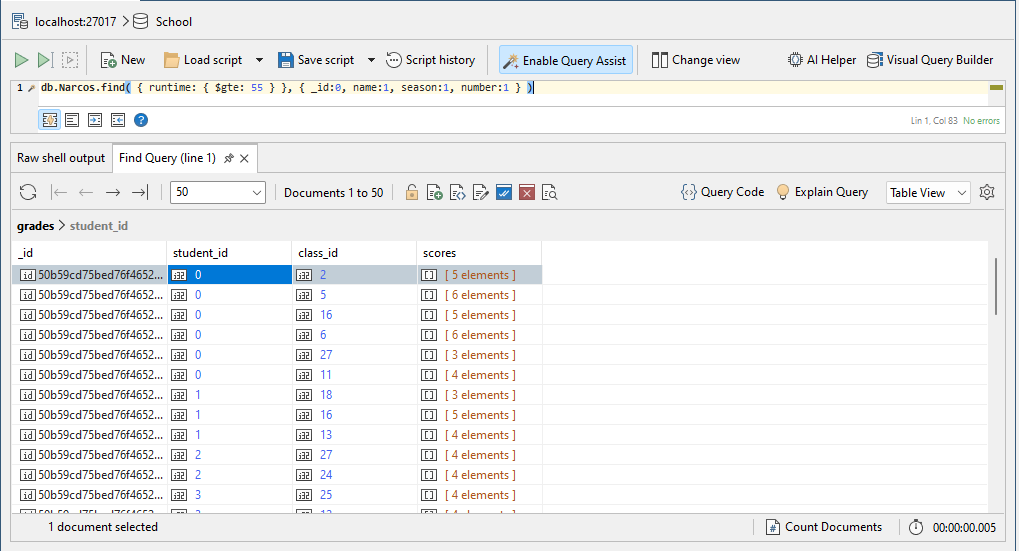


\* db.grades.find( { scores: { $elemMatch: { type: 'exam', score: { $gte: 50 } } } } ): busca documentos donde el array scores hay al menos un elemento de tipo exam con un score mayor o igual a 50

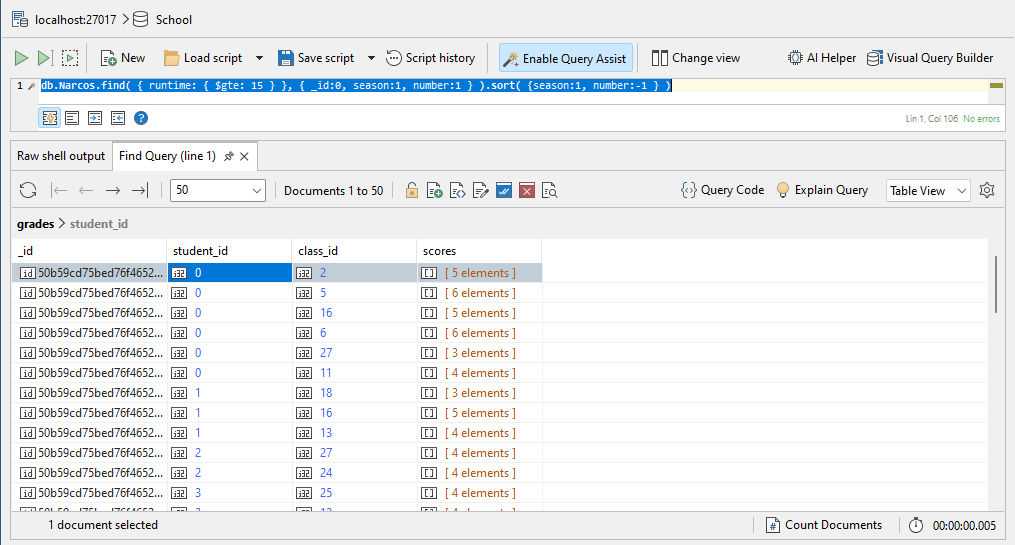


## NARCOS:

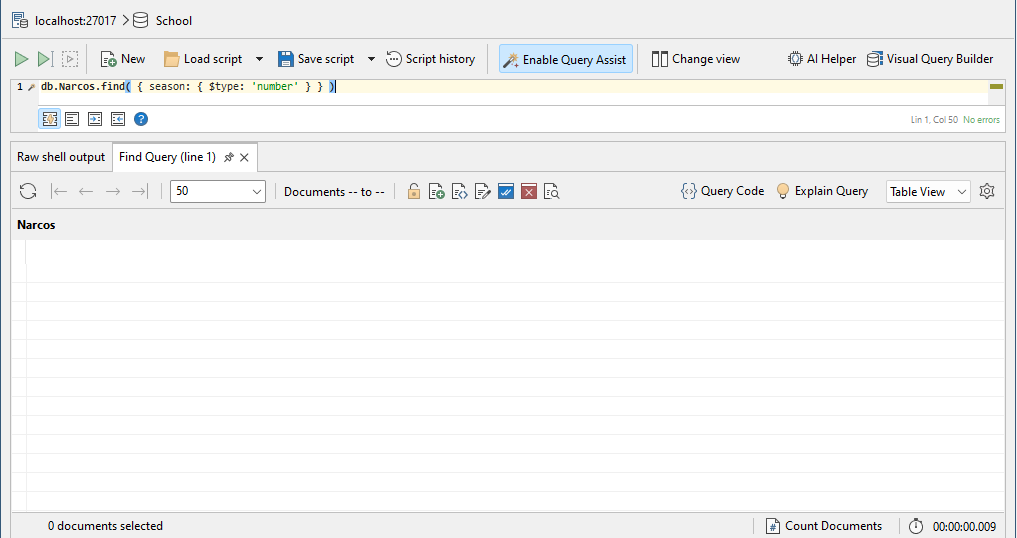
\* db.Narcos.find( { runtime: { $gte: 55 } }, { \_id:0, name:1, season:1, number:1 } ): busca episodios de la serie narcos donde el tiempo de ejecución (runtime) es mayor o igual a 55 minutos y solo muestra los campos name, season y number



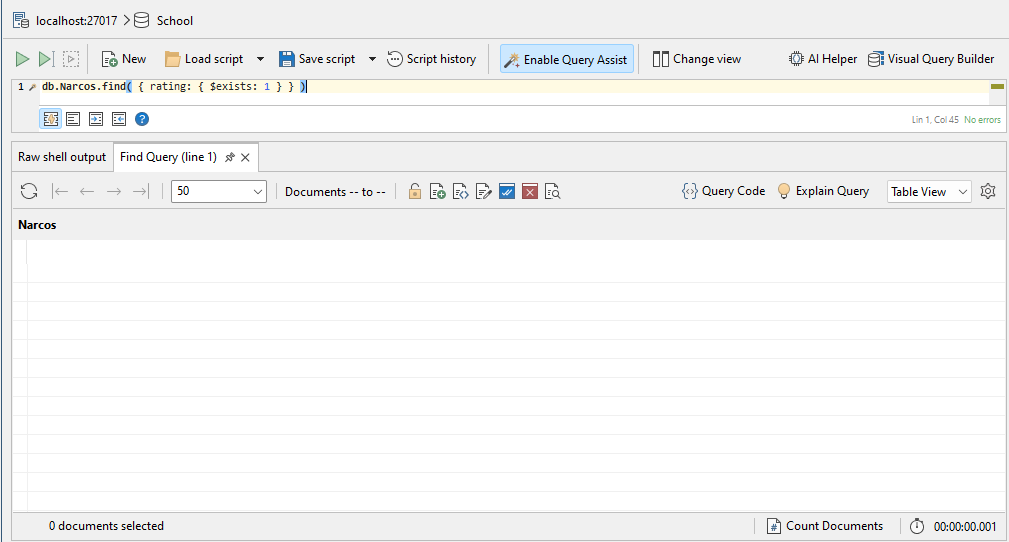
\* db.Narcos.find( { runtime: { $gte: 15 } }, { \_id:0, season:1, number:1 } ).sort( {season:1, number:-1 } ): busca episodios con un runtime mayor o igual a 15 minutos, muestra los campos season y number y los ordena de manera ascendente por season y descendente por number



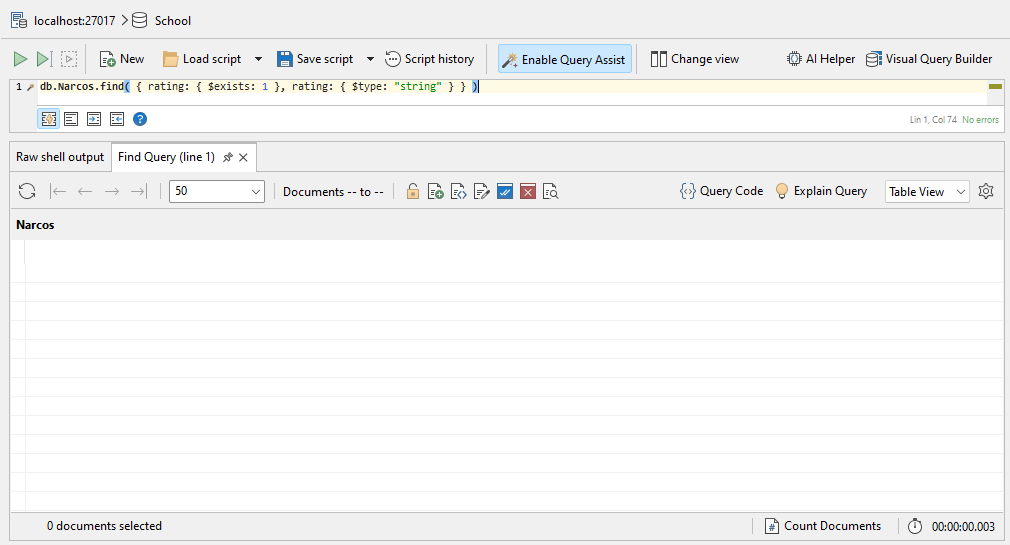
\* db.Narcos.find( { season: { $type: 'number' } } ):busca documentos donde el campo season es de tipo numérico.



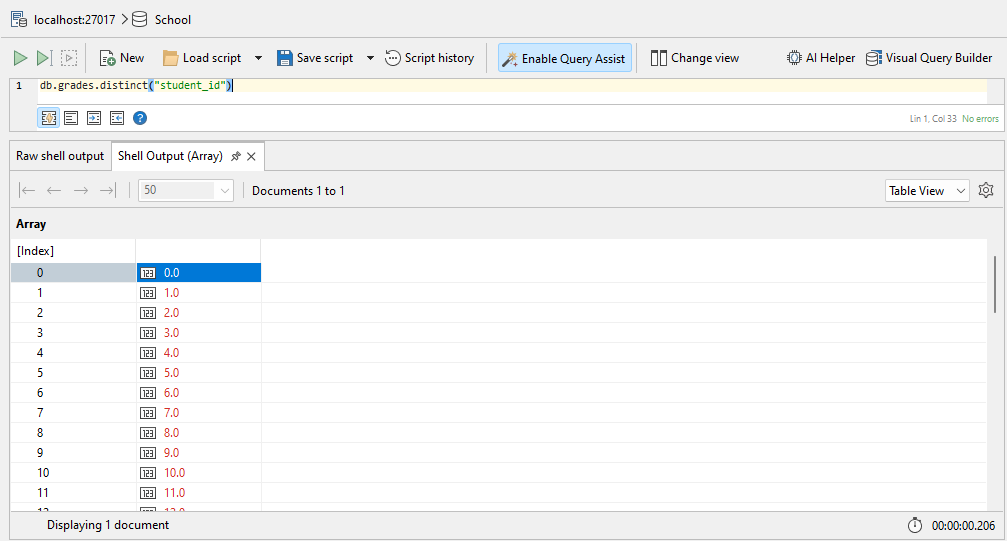
\* db.Narcos.find( { rating: { $exists: 1 } } ): busca documentos donde el campo rating existe.



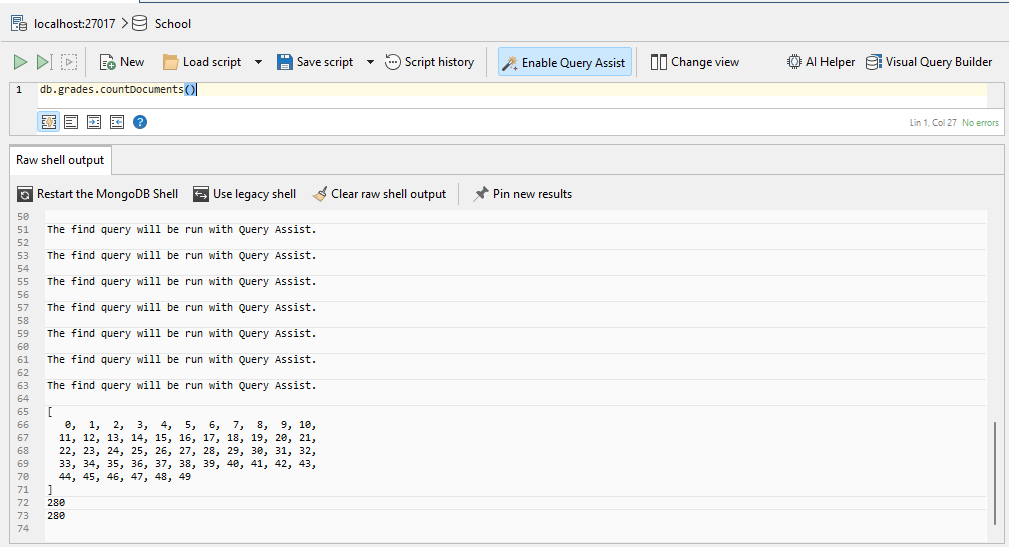
\*db.Narcos.find( { rating: { $exists: 1 }, rating: { $type: "string" } } ): busca documentos donde el campo rating existe y es de tipo string.



\* db.grades.distinct("student\_id"): retorna una lista de valores únicos para el campo student\_id en la colección grades.



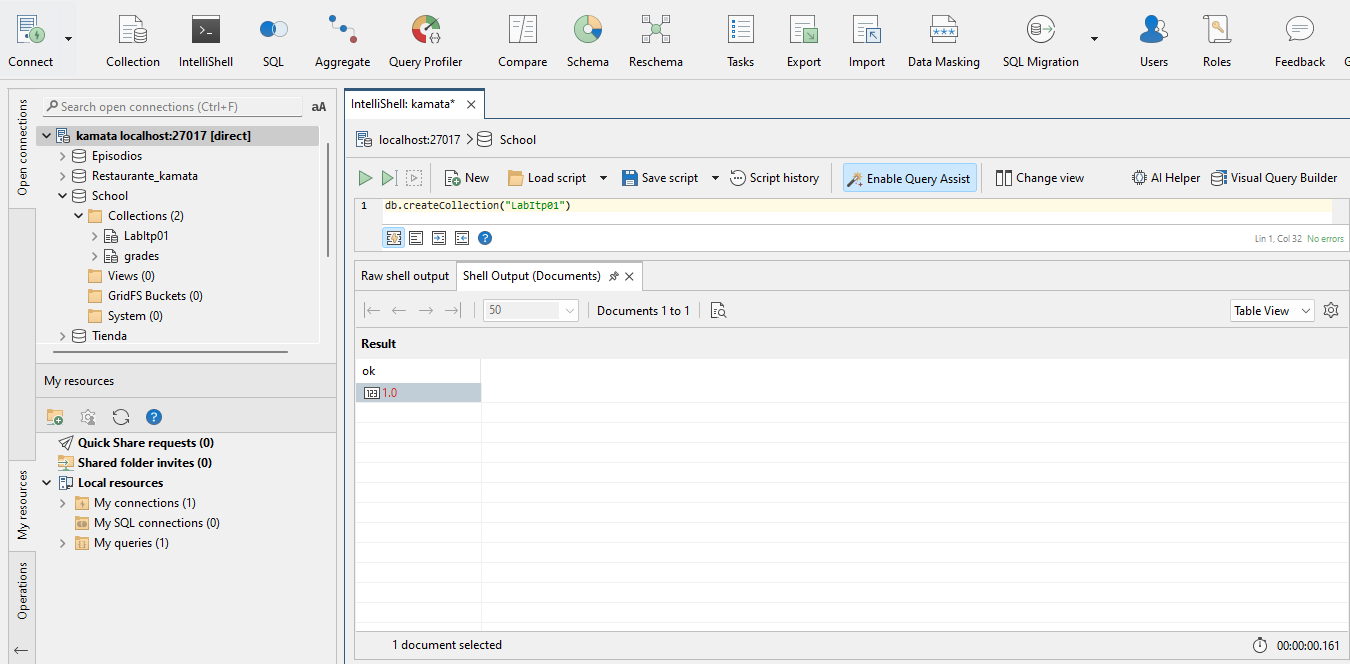
\* db.grades.countDocuments(): cuenta el numero total de documentos en la colección grades.



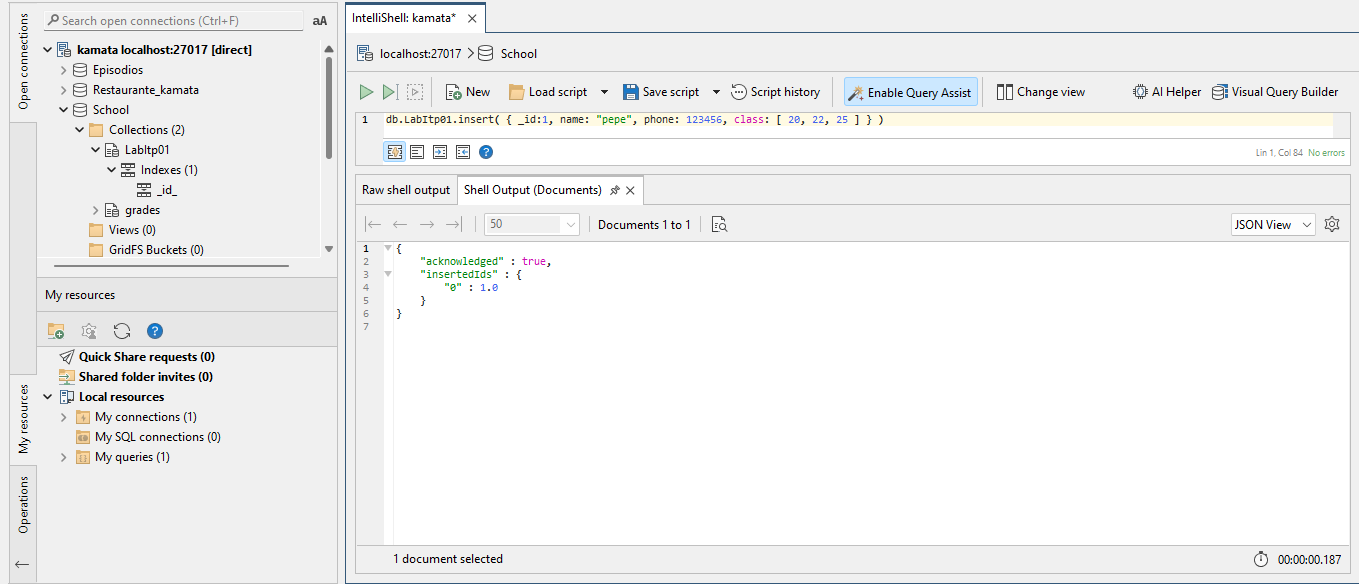
# CREATE

## SCHOOL

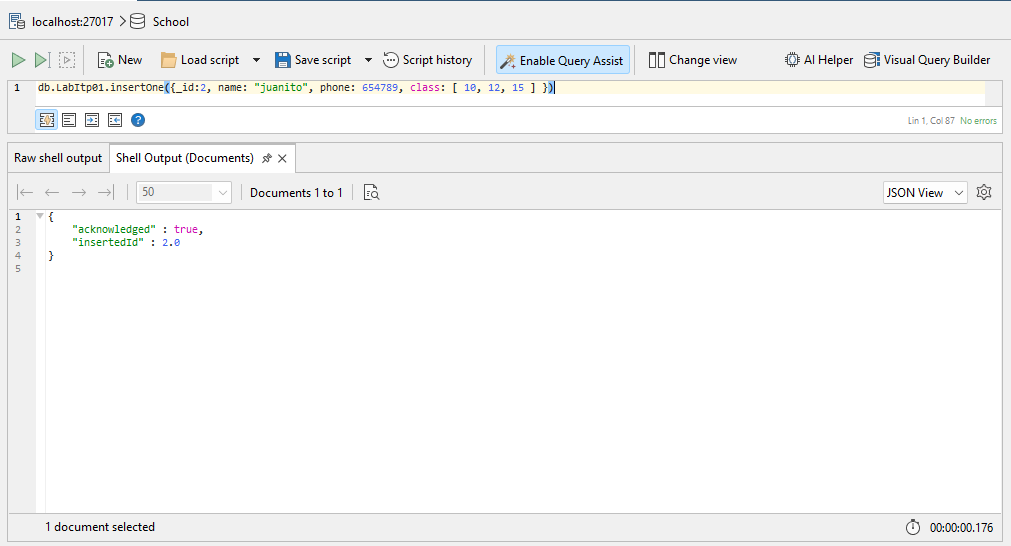
\* db.createCollection("LabItp01"): crea una colección llamada LapItp01.



\*db.LabItp01.insert( { \_id:1, name: "pepe", phone: 123456, class: [ 20, 22, 25 ] } ):inserta un documento con el name pepe y otros datos.

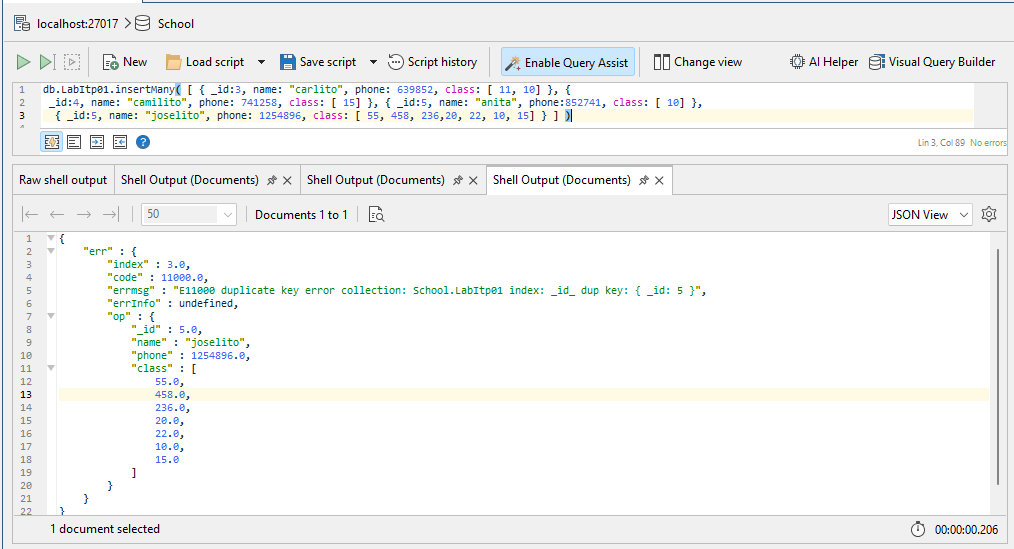


\* db.LabItp01.insertOne({\_id:2, name: "juanito", phone: 654789, class: [ 10, 12, 15 ] }):inserta un documento con el name Juanito

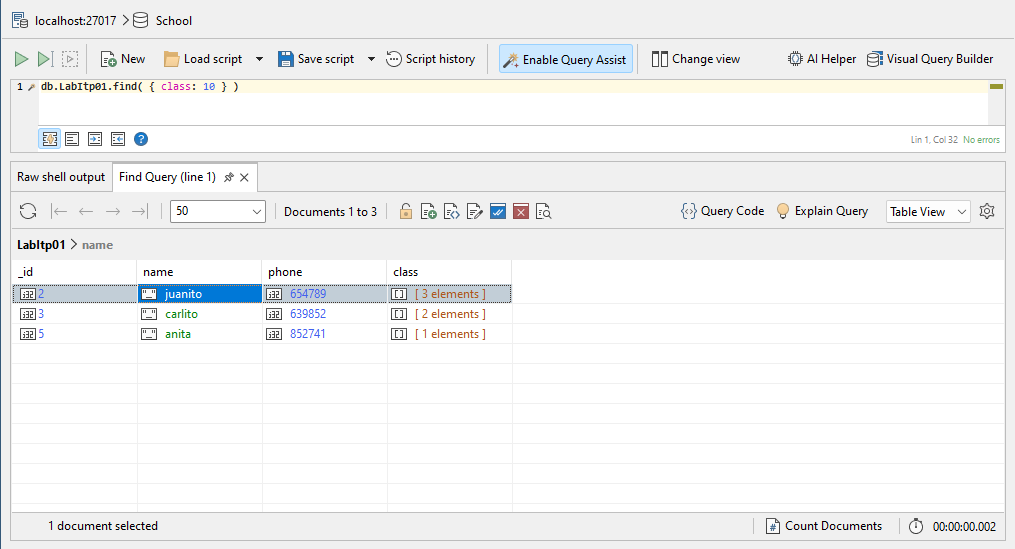


\* db.LabItp01.insertMany( [ { \_id:3, name: "carlito", phone: 639852, class: [ 11, 10] }, {

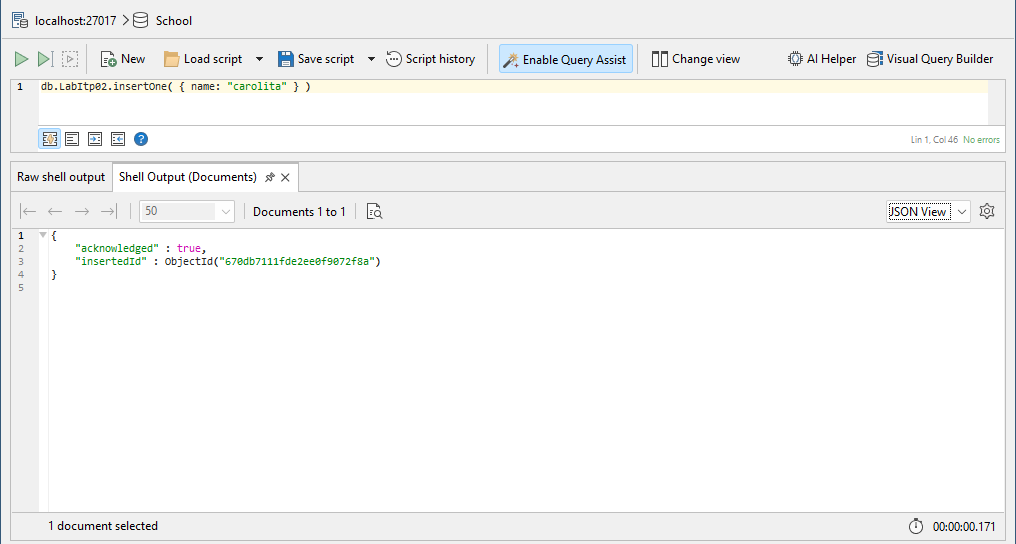
\_id:4, name: "camilito", phone: 741258, class: [ 15] }, { \_id:5, name: "anita", phone:852741, class: [ 10] },{ \_id:5, name: "joselito", phone: 1254896, class: [ 55, 458, 236,20, 22, 10, 15] } ]): inserta varios documentos a la vez.



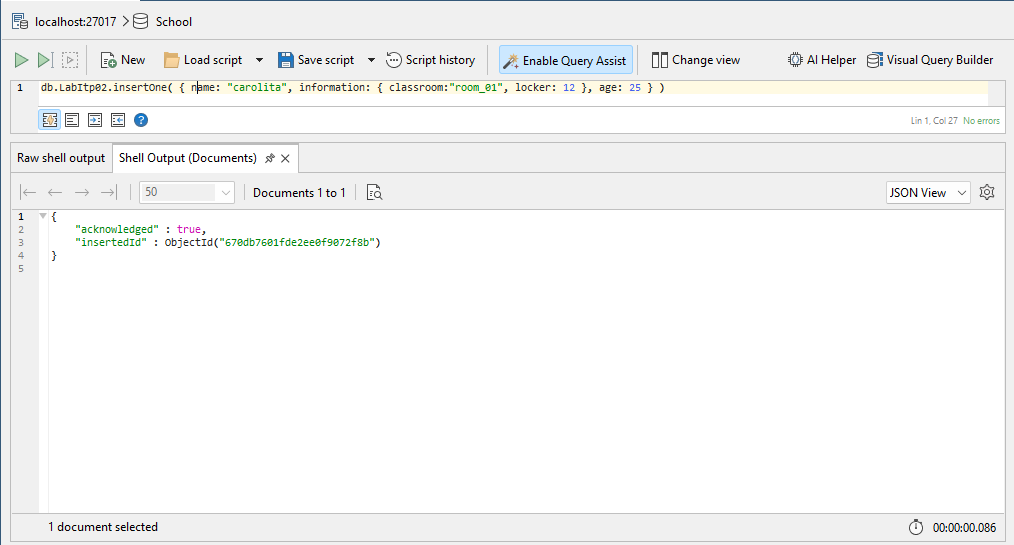
\* db.LabItp01.find( { class: 10 } ): busca documentos donde el array class contiene el valor 10



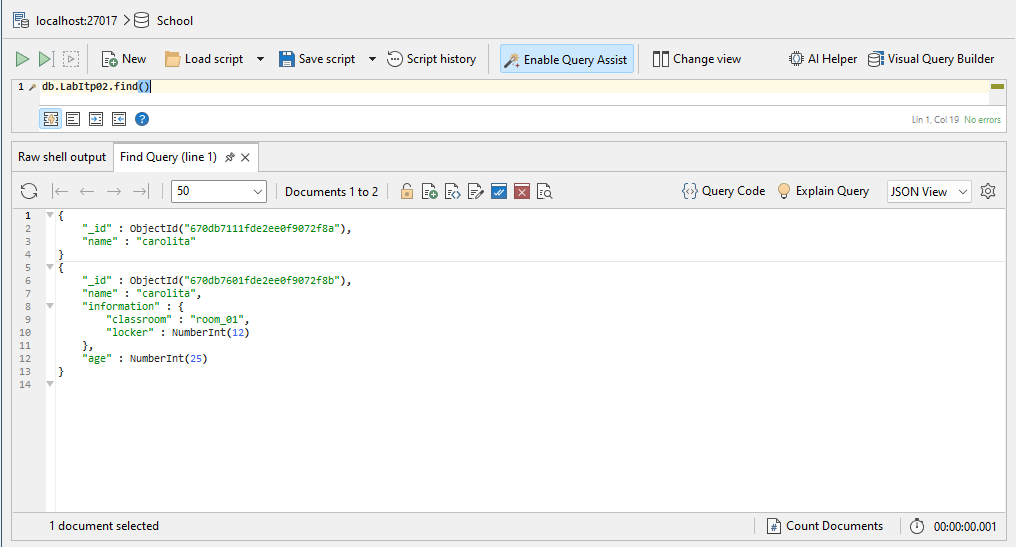
\* db.LabItp02.insertOne( { name: "carolita" } ):inserta un documento en la colección LabItp con el campo name carolina.



\* db.LabItp02.insertOne( { name: "carolita", information: { classroom:"room\_01", locker: 12 }, age: 25 } ):inserta un documento con campos anidados en la colección labitp02

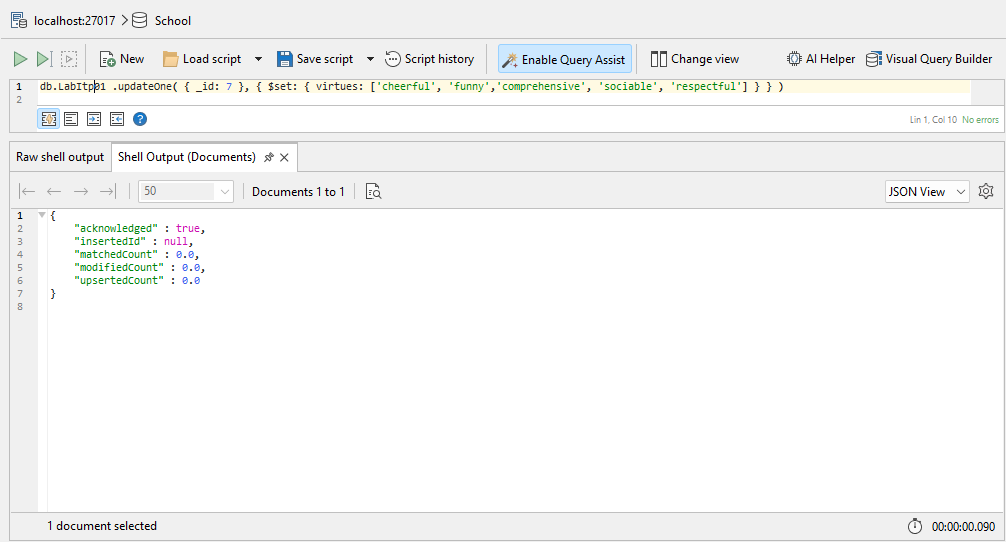


\* db.LabItp02.find(): muestra todos los documentos en la colección labitp02.

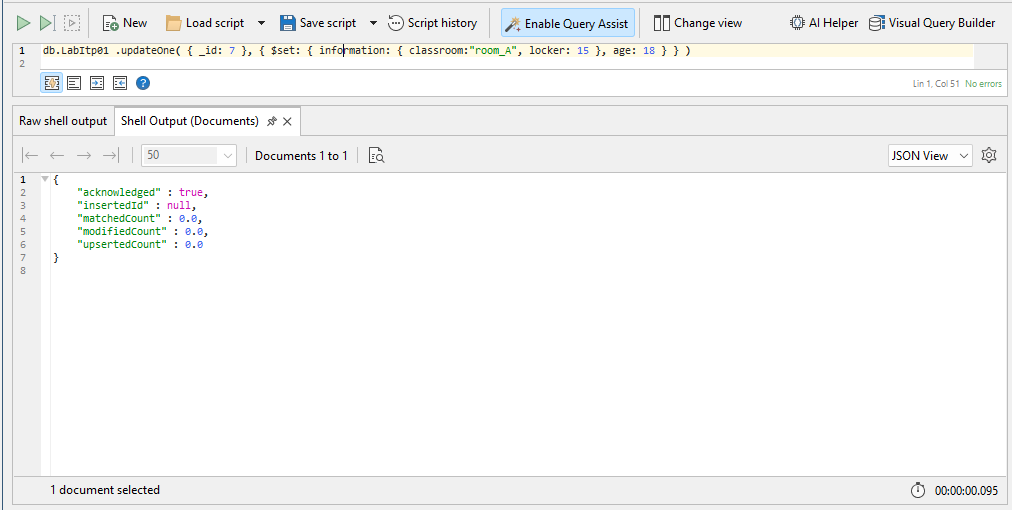


# UPDATE

\* db.LabItp01 .updateOne( { \_id: 7 }, { $set: { virtues: ['cheerful', 'funny','comprehensive', 'sociable', 'respectful'] } } ) : actualiza el documento con el \_id 7 agregando el campo virtues con los valores indicados



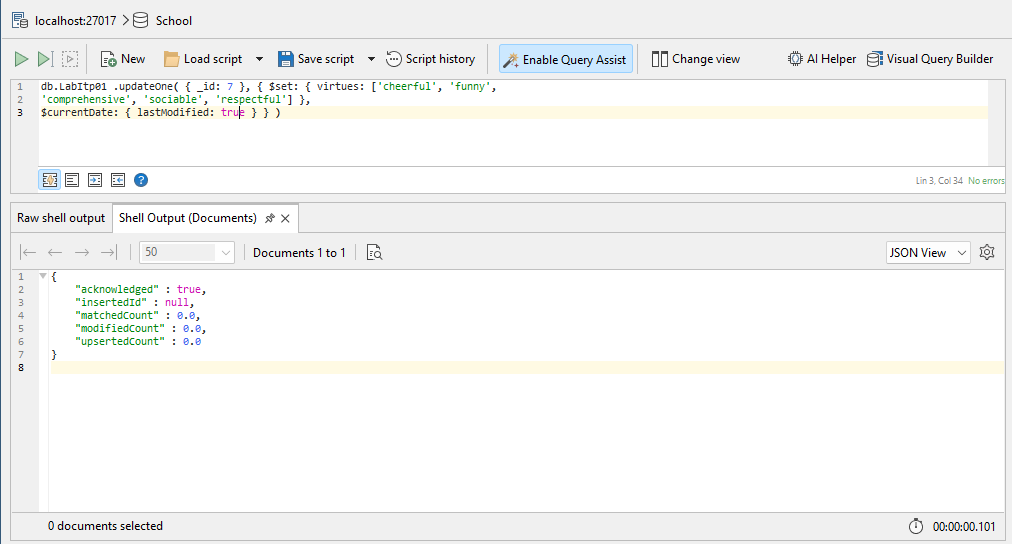
\* db.LabItp01 .updateOne( { \_id: 7 }, { $set: { information: { classroom:"room\_A", locker: 15 }, age: 18 } } ):actualiza o agrega información anidada para el campo información en el documento con el \_id 7.



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

'comprehensive', 'sociable', 'respectful'] },

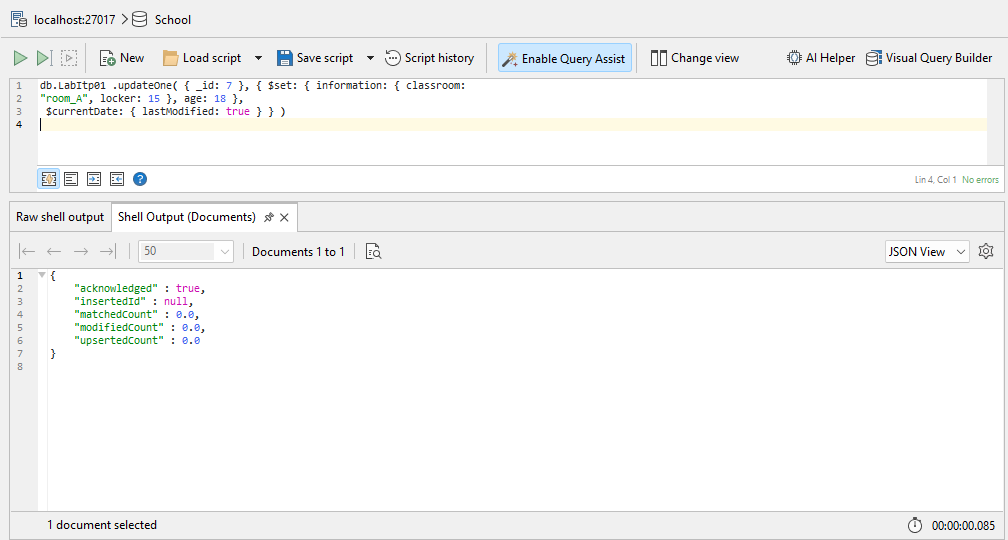
$currentDate: { lastModified: true } } ): actualiza el documento con el \_id 7 agregando el campo virtues con los valores indicados



\* db.LabItp01 .updateOne( { \_id: 7 }, { $set: { information: { classroom:

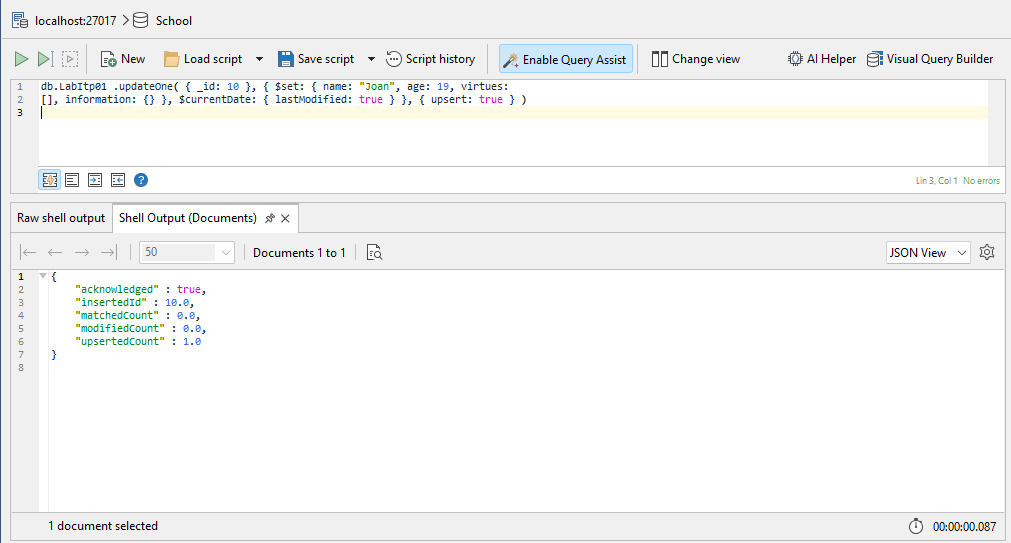
"room\_A", locker: 15 }, age: 18 },

$currentDate: { lastModified: true } } ): actualiza o agrega información anidada para el campo information en el documento con el \_id 7.



\* db.LabItp01 .updateOne( { \_id: 10 }, { $set: { name: "Joan", age: 19, virtues:

[], information: {} }, $currentDate: { lastModified: true } }, { upsert: true } ): si no existe un documento con el \_id 10, crea uno nuevo con los valores que se indican.



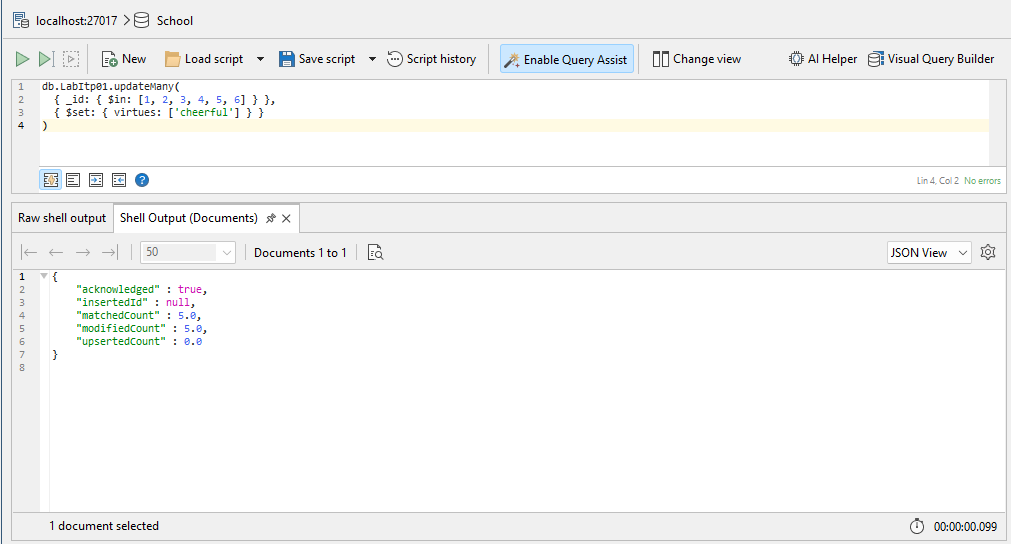
\* Actualiza los documentos con \_id 1 – 6 y agrega el campo virtues con un array que contenga un único valor, el que decidas de la lista siguiente: ['cheerful', 'funny', 'comprehensive', 'sociable', 'respectful'].

db.LabItp01.updateMany(

{ \_id: { $in: [1, 2, 3, 4, 5, 6] } },

{ $set: { virtues: ['cheerful'] } }

)



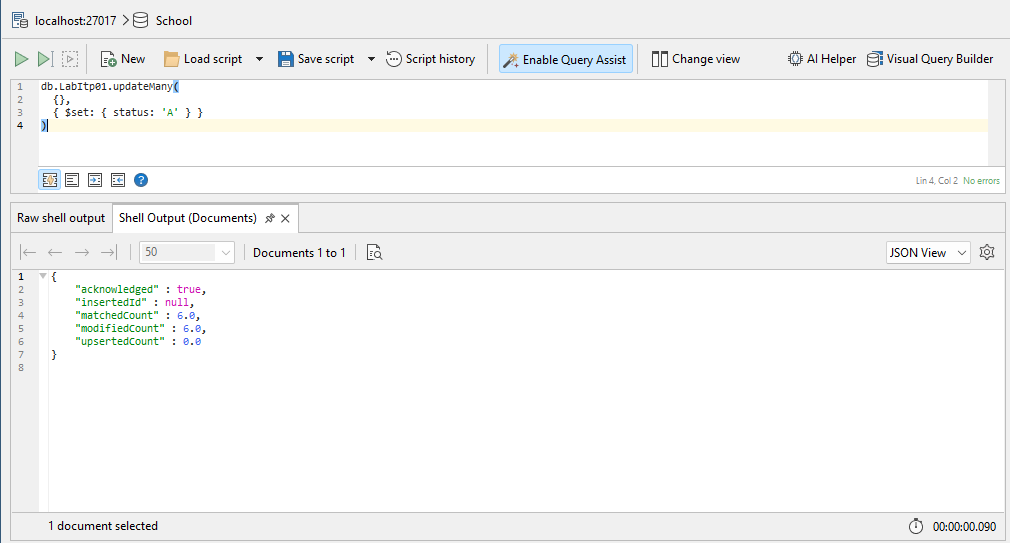
\*Actualiza todos los documentos con una única instrucción y agrega el siguiente campo: status: 'A'.

db.LabItp01.updateMany(

{},

{ $set: { status: 'A' } }

)



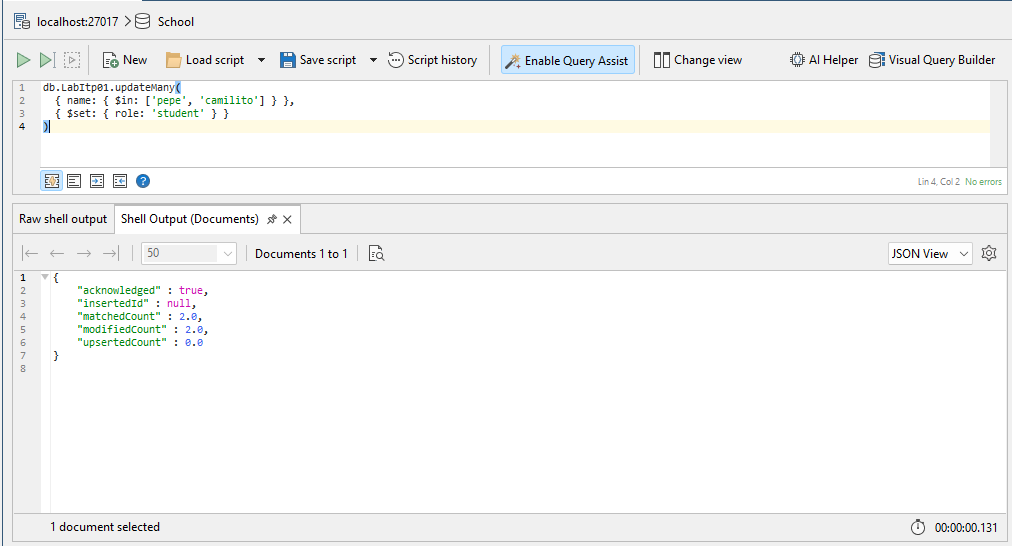
\*Actualiza los documentos de “pepe” y “camilito” y agrega el siguiente campo: role: 'student'.

db.LabItp01.updateMany(

{ name: { $in: ['pepe', 'camilito'] } },

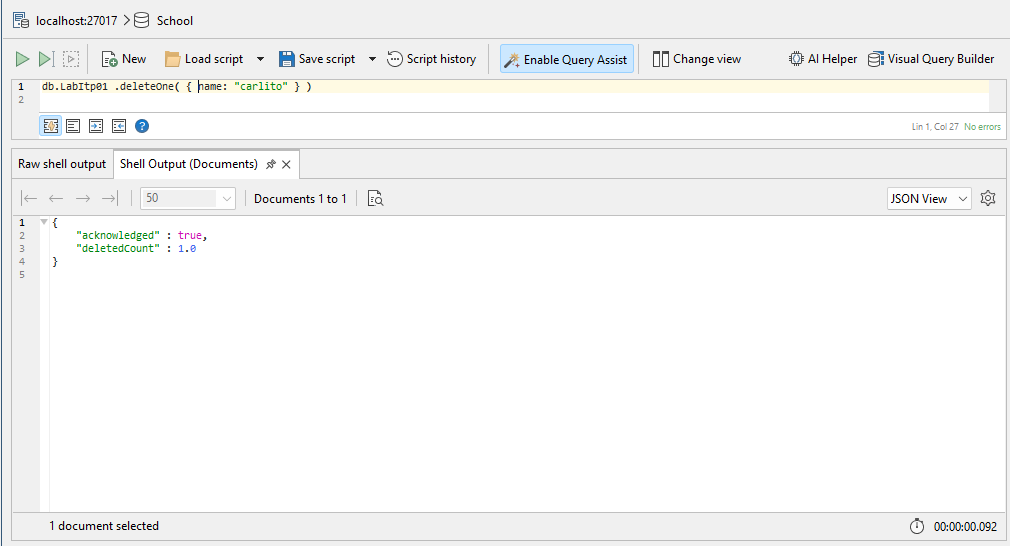
{ $set: { role: 'student' } }

)

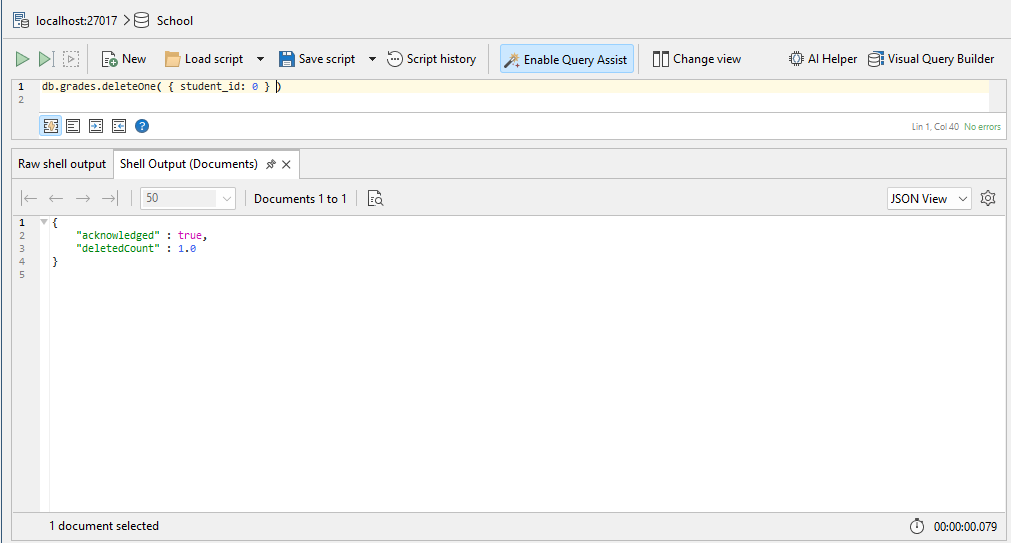


# DELETE:

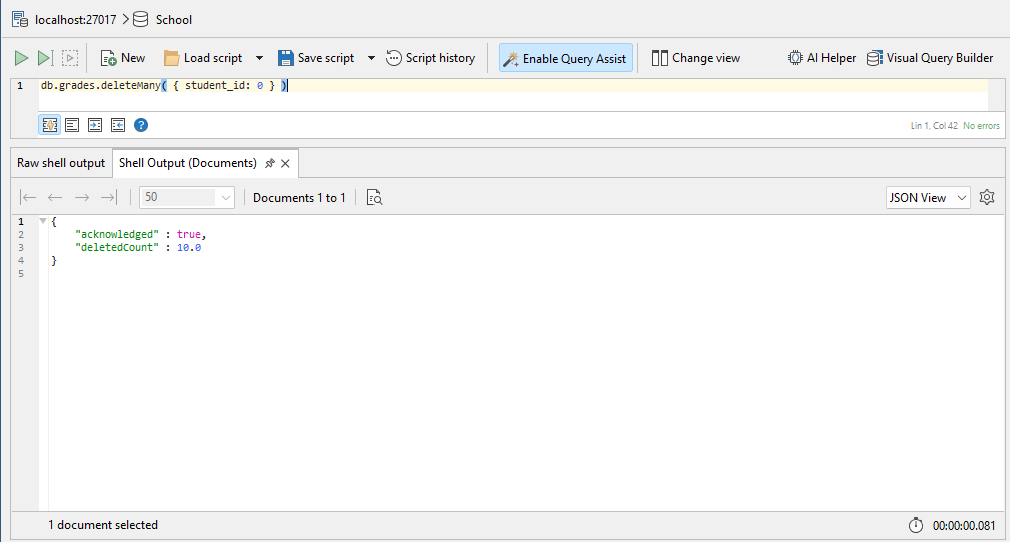
\* db.LabItp01 .deleteOne( { name: "carlito" } ): elimina un documento donde el campo name es carlito



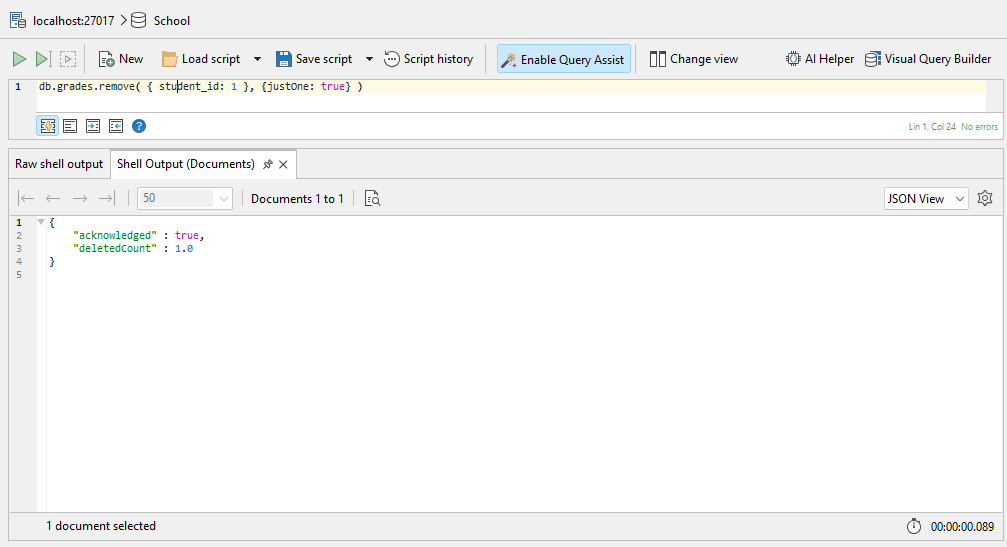
\* db.grades.deleteOne( { student\_id: 0 } ): elimina el documento donde el student\_id es 0.



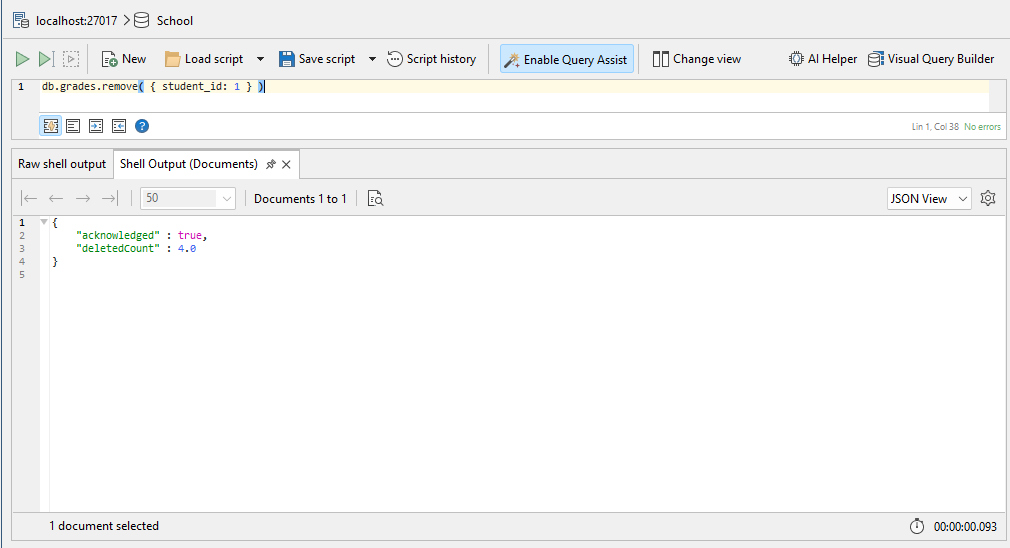
\* db.grades.deleteMany( { student\_id: 0 } ): elimina todos los documentos donde el student\_id es 0.



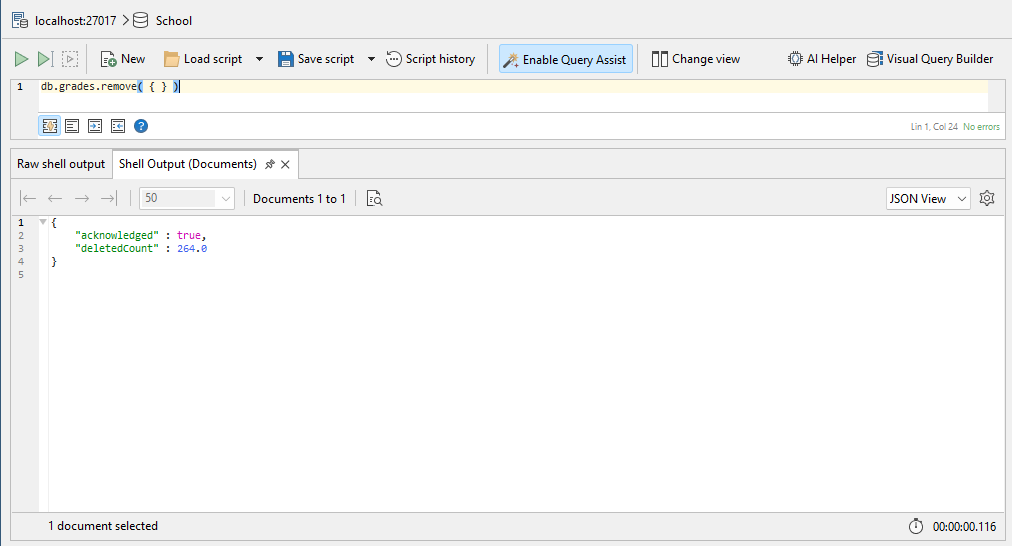
\* db.grades.remove( { student\_id: 1 }, {justOne: true} ): elimina solo un documento donde el student\_id es 1.



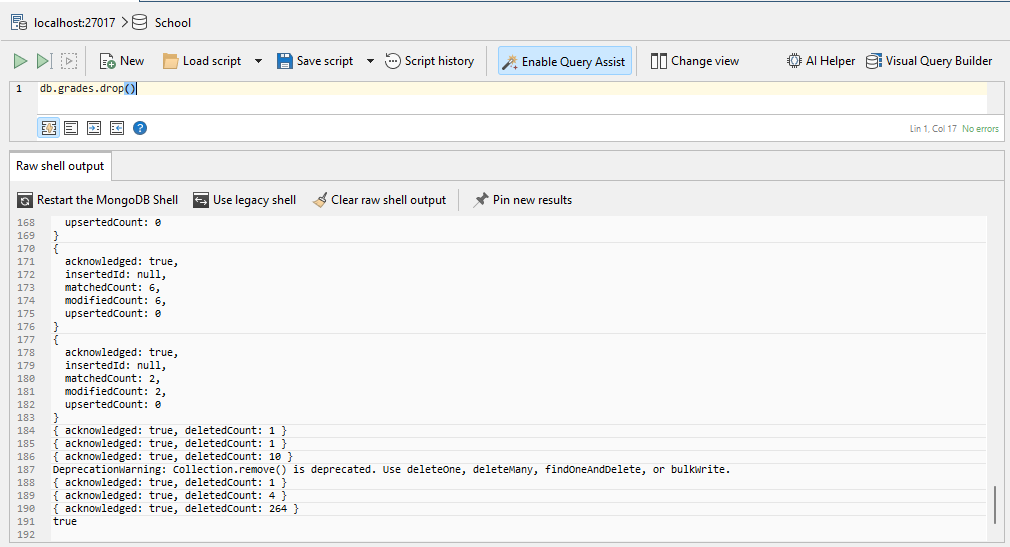
\* db.grades.remove( { student\_id: 1 } ): elimina todos los documentos donde el student\_id es 1.



\* db.grades.remove( { } ): elimina todos los documentos en la coleccion.



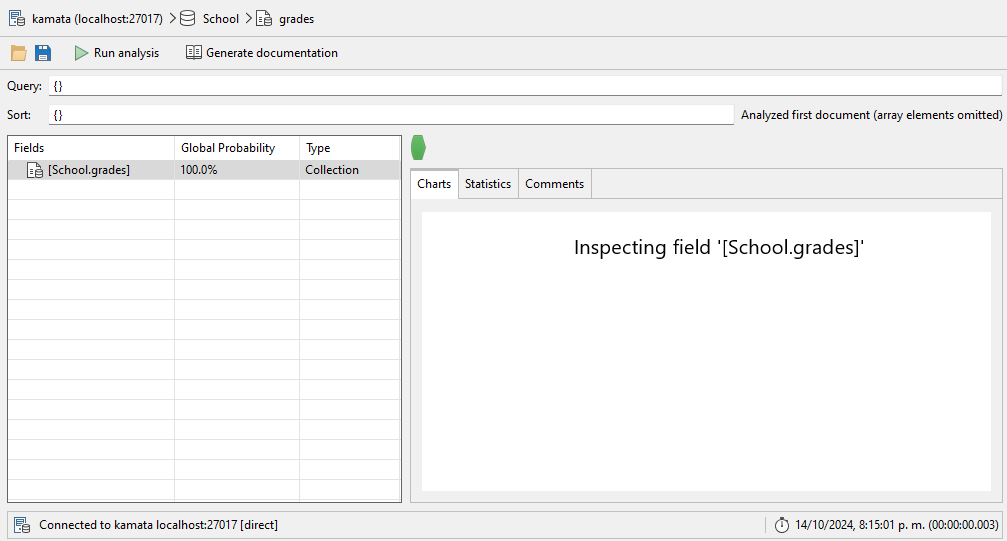
\* db.grades.drop():elimina toda la colección grades



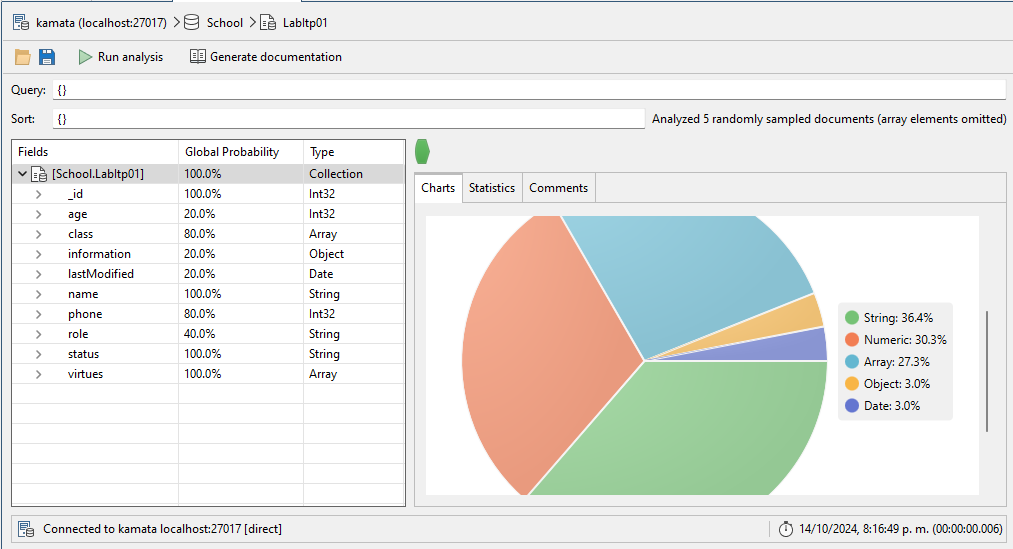
# ESTADISTICAS DE CADA COLECCION:

SCHOOL:

GRADE:

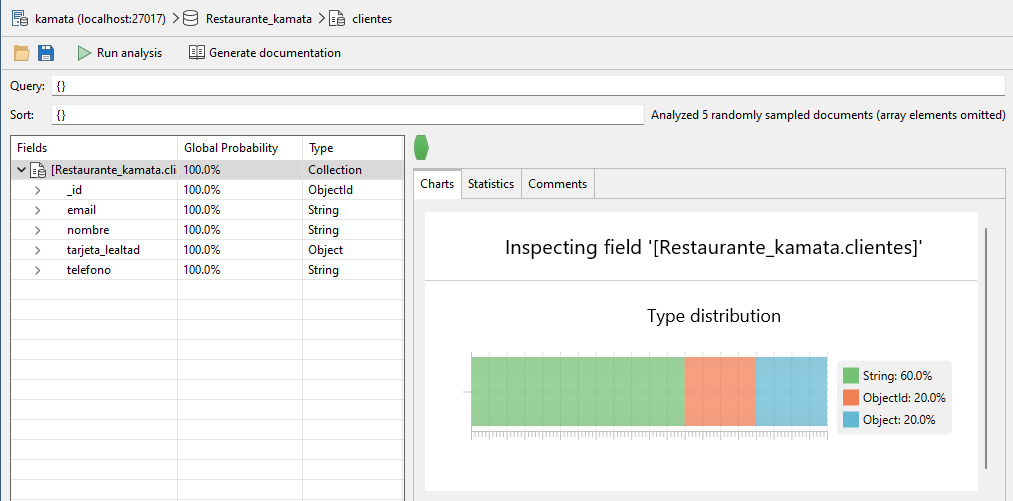


LABITP01:

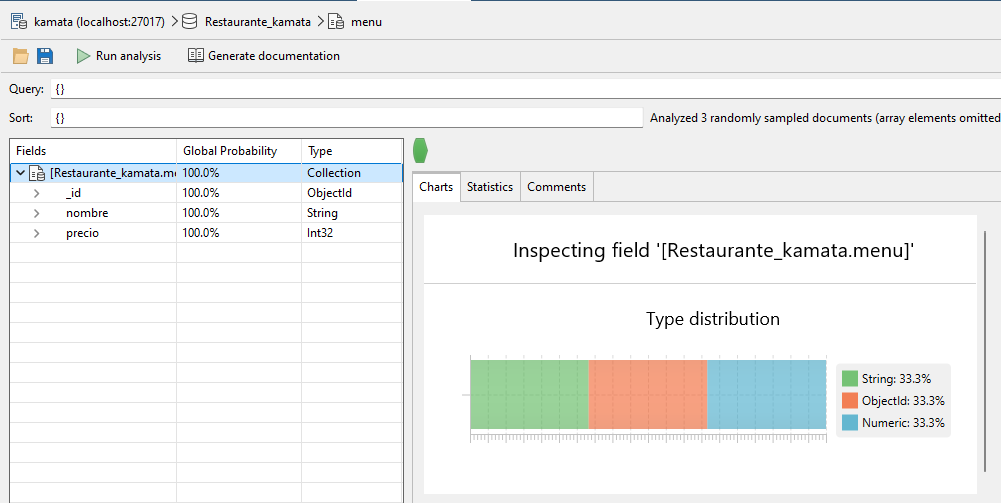


RESTAURANTE-KAMATA

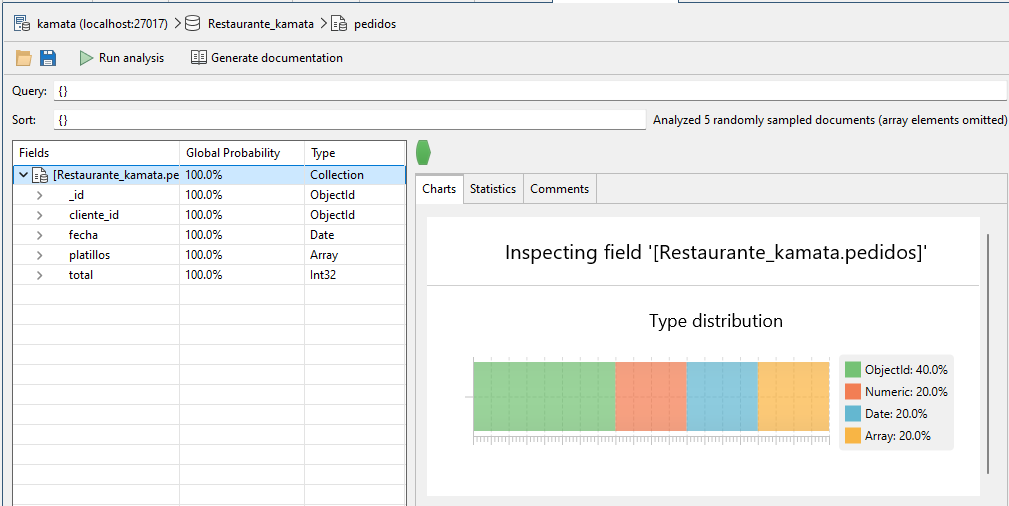
CLIENTES:



MENU:

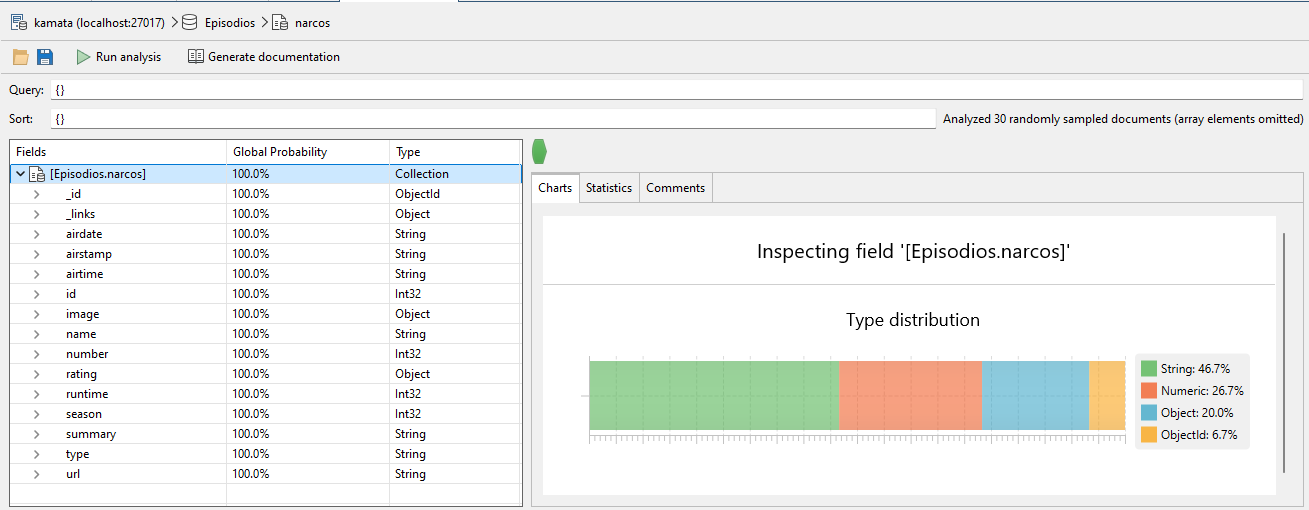


PEDIDOS:

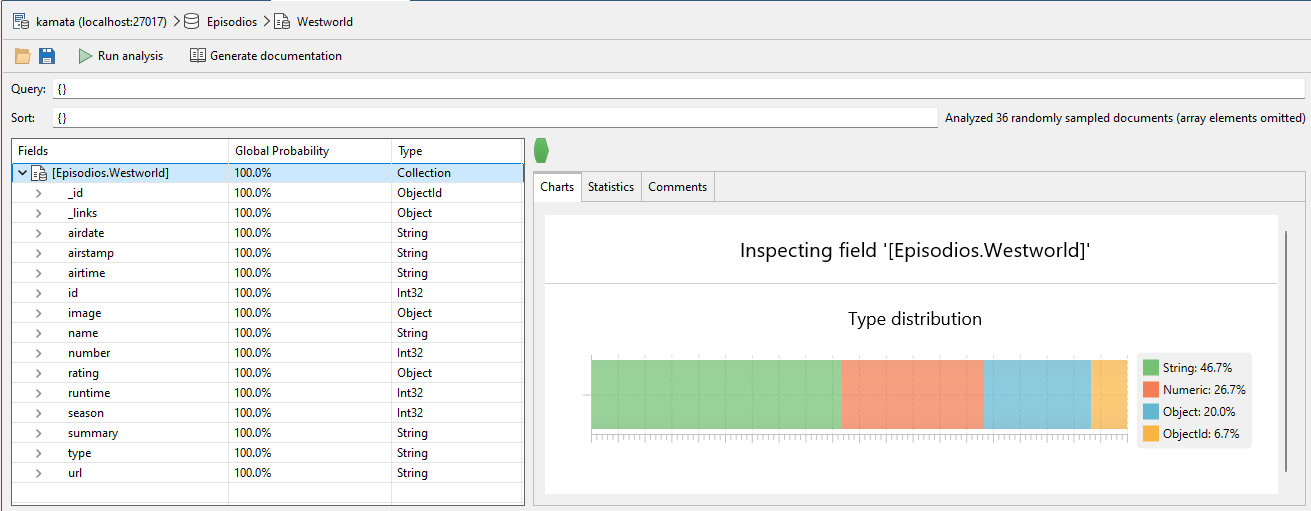


EPISODIOS:

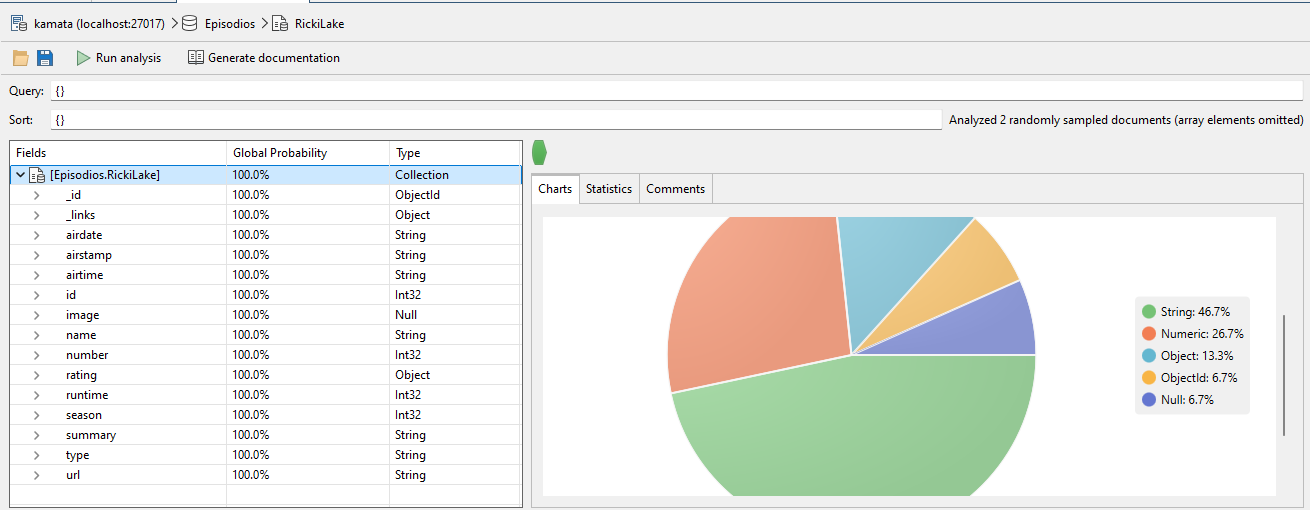
NARCOS:



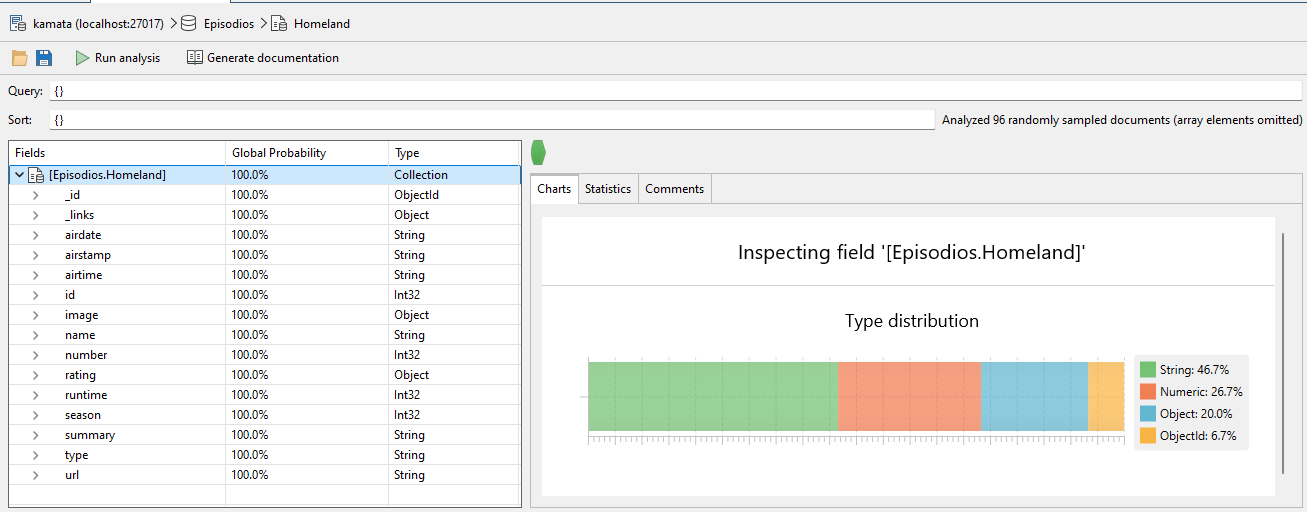
WESTWORD:



RICKILAKE:



HOMELAND:



# CONSIDERACION DE DISEÑO

Al diseñar el taller, se considero un enfoque que nos permite trabajar de manera mas practica con datos reales, simulando situaciones de uso diario en la gestión de bases de datos no relacionales. Las colecciones se estructuraron de una manera que se representen agrupaciones de los datos relacionados, lo que nos facilita la comprensión de mongoDB en diferentes escenarios.

# ANALISIS Y DISCUSIÓN

A lo largo del taller, hemos comprender las ventajas que tiene utilizar mongoDB para las bases de datos no estructurales, por su capacidad de almacenar grandes cantidades de volúmenes de información de manera más flexible y eficiente. Se destaco las estructuras de las colecciones para la optimización de las consultas y reducir la complejidad de la manipulación de los datos. El uso de JSON Viewer y mongoDB Compass facilito la identificación de las estructuras de datos y permitió la visualización clara de la información almacenada.

# CONCLUCION

Al finalizar este taller se habrá ganado experiencia en el manejo de las principales operaciones CRUD en mongoDB, comprendiendo mejor cómo funciona las inserciones, consultas, actualizaciones y eliminación en la base de datos documental. Además, habrá trabajado con datos reales en formato JSON, lo que permitirá visualizar el potencial de mongoDB para gestionar grades volúmenes de información no estructurada de forma eficiente.

# RECOMENDACIONES

\*Familiarizarse con MongoDB Compass para tener una medida de avance en el uso de mongoDB se recomienda a los estudiantes dominar herramientas graficas como son Compass para una mayor eficiencia.

\*Optimizar las consultas para los proyectos mas complejos ya que es esencial la optimización de las consultas mediante el uso de índices y estructuras adecuadas para garantizar un rendimiento más eficiente.

\*Conocer las operaciones del CRUD con más profundización las operaciones básicas de mongoDB ayudara a ser mas competentes en la gestión de las bases NoSQL.

# REFERENCIAS

MongoDB Inc. (2024). MongoDB Documentation. Recuperado de https://docs.mongodb.com

Brown, P. (2023). Practical MongoDB. O'Reilly Media.

Jain, N. (2023). Mastering MongoDB 6.x. Packt Publishing.

Chodorow, K. (2022). MongoDB: The Definitive Guide. O'Reilly Media.

JSON Viewer. (2024). JSONGrid Tool. Recuperado de https://jsongrid.com/json-viewer

# REPOSITORIO DE ALOJAMIENTO