

Verificación de consultas

Jhonier Arley Pasos Perengues

Brayan Arcos

Instituto Tecnológico del Putumayo

Facultad de Ingeniería y Ciencias Básicas

Ingeniería de Software

Mocoa

2024

índice

Resumen ejecutivo	3
Introducción	4
Metodología	5
1. Herramientas Utilizadas	5
 Desarrollo del informe.....	8
Descripción de la base de datos	8
Modelo de Datos: Normalización y Relaciones.....	10
Métodos de captura	12
Consultas	13
 Análisis y discusión.....	35
Conclusión	36
Recomendaciones.....	38

Resumen ejecutivo

Este informe presenta un análisis detallado sobre la implementación de bases de datos en MongoDB, aplicado a un contexto educativo. Se cubren aspectos clave como la creación de un esquema de base de datos personal para una escuela, el diseño de colecciones, la ejecución de consultas, y el análisis de resultados. A través de estas actividades, se busca entender mejor las capacidades de MongoDB y su aplicabilidad en el diseño de bases de datos NoSQL.

Introducción

Contexto y Motivación:

Este informe fue elaborado con el propósito de demostrar la capacidad de diseño, manejo y consulta de bases de datos en MongoDB, utilizando una base de datos escolar personalizada. La relevancia de este tema radica en la creciente adopción de bases de datos NoSQL en la industria tecnológica, especialmente en aplicaciones que requieren alta flexibilidad y escalabilidad, como en entornos educativos.

Alcance del Informe:

El informe cubre los conceptos fundamentales de MongoDB, incluyendo la creación de una base de datos personal para una escuela, la ejecución de consultas complejas, el diseño de colecciones y la normalización. Se exploran técnicas de captura de datos y los métodos usados para organizar la información dentro de MongoDB.

Objetivos:

El objetivo principal de este informe es diseñar y analizar una base de datos en MongoDB que represente un sistema escolar. A través de este ejercicio, se pretende realizar consultas eficientes y optimizar el diseño de la base de datos para mejorar el almacenamiento y la recuperación de datos. Además, se busca evaluar las consultas ejecutadas durante un taller y contrastarlas con las realizadas sobre la base de datos personalizada.

Metodología

1. Herramientas Utilizadas

Para la creación y gestión de la base de datos, se emplearon varias herramientas que facilitaron el manejo de datos y la realización de consultas:

- **MongoDB Compass:** Esta herramienta fue utilizada para la importación de archivos JSON a MongoDB y la gestión visual de las colecciones. MongoDB Compass permite visualizar la estructura de la base de datos, realizar consultas y administrar los documentos de manera intuitiva.
- **JSONGrid:** Se utilizó para visualizar y validar los archivos JSON antes de su importación a MongoDB. JSONGrid facilita la inspección de los datos y asegura que los archivos estén correctamente formateados para ser cargados en el sistema.
- **Database Tools:** Un conjunto de herramientas de análisis que permitieron examinar el rendimiento de las consultas y monitorear la eficiencia de la base de datos. Estas herramientas se usaron para evaluar las operaciones CRUD (Create, Read, Update, Delete) y el comportamiento de la base de datos en escenarios de carga.

Procedimientos

1.1. Creación de la Base de Datos Personal

La primera fase del proyecto consistió en el diseño y creación de una base de datos personalizada que representara la estructura organizativa de una escuela. Este proceso incluyó la creación de colecciones para almacenar la información de estudiantes, profesores, clases, matrículas y administrativos.

Se definió una estructura normalizada, donde las relaciones entre colecciones se manejaron principalmente a través de identificadores (IDs). Para evitar redundancia de datos, se utilizaron referencias en lugar de documentos embebidos cuando era necesario.

1.2. Visualización y Validación de Datos

Antes de proceder a la importación, los datos fueron validados utilizando JSONGrid. Esta herramienta permitió revisar los archivos JSON que contenían la información de estudiantes, profesores y clases, asegurando que la estructura y los campos eran los correctos. Esta validación fue esencial para evitar problemas durante la importación a MongoDB.

1.3. Importación de Datos

La importación de los archivos JSON a MongoDB se realizó mediante MongoDB Compass, que permitió cargar los datos de manera visual y sencilla en las colecciones correspondientes. Una vez importados, se revisaron las colecciones para confirmar que todos los datos fueron cargados correctamente.

1.4. Realización del Taller

El siguiente paso fue ejecutar consultas y operaciones de prueba en la base de datos creada, lo que incluyó:

Inserciones de nuevos documentos en las colecciones de estudiantes, profesores y clases.

Actualizaciones de información ya existente, como números de teléfono y asignaciones de clases.

Eliminaciones controladas de documentos para probar las funcionalidades de MongoDB.

Consultas complejas para verificar las relaciones entre estudiantes y clases.

Todas estas operaciones se ejecutaron tanto a través de MongoDB Compass como del shell de MongoDB.

1.5. Análisis de Rendimiento con Database Tools

Finalmente, se realizó un análisis del rendimiento de la base de datos utilizando Database Tools, un conjunto de herramientas para monitorear el uso de recursos y optimizar las consultas. Se evaluó la velocidad de las operaciones de lectura y escritura, así como la eficiencia en la recuperación de datos en escenarios con múltiples relaciones, como el registro de matrículas entre estudiantes y clases.

Estas herramientas también permitieron identificar posibles cuellos de botella en las consultas más complejas y sugerir el uso de índices en algunos campos clave como `estudiante_id` y `clase_id` para mejorar el rendimiento en el futuro.

Desarrollo del informe

Descripción de la base de datos

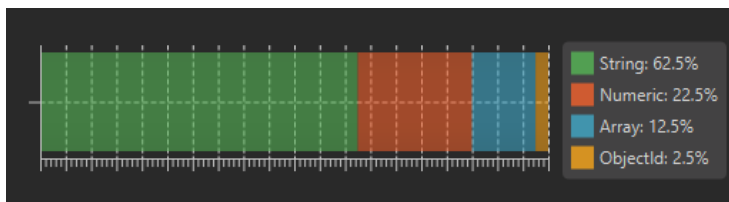
La base de datos de la escuela creada está diseñada para gestionar la información esencial de una institución educativa, incluyendo estudiantes, profesores, clases y matrículas. Las relaciones entre las colecciones permiten un control eficaz de las inscripciones, la administración del personal docente y el seguimiento académico de los estudiantes.

Las colecciones principales son

- ESTUDIANTES
- PROFESORES
- CLASES
- MATRICULAS

ESTUDIANTES: La colección estudiante contiene la información personal de cada estudiante, como nombre, apellidos, edad, teléfono, dirección y correo electrónico. Se priorizó el

Fields	Global Probability	Type
▼ [escuela.estudiantes]	100.0%	Collection
> _id	100.0%	...
> apellido1	100.0%	String
> apellido2	100.0%	String
> direccion	100.0%	String
> edad	100.0%	Int32
> email	100.0%	String
> nombre	100.0%	String
> telefono	100.0%	Array

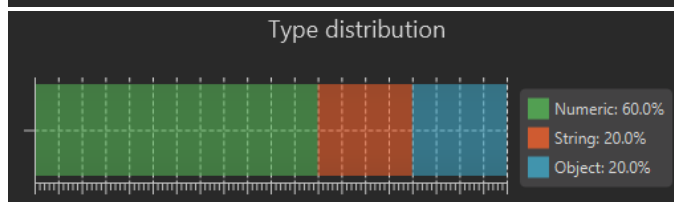
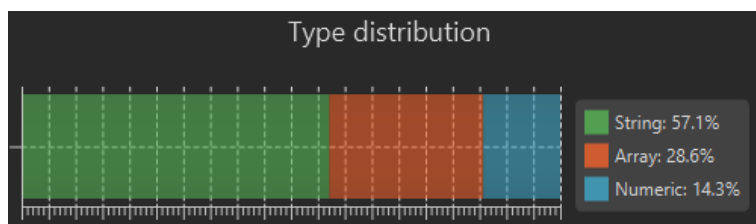


uso de un array para almacenar múltiples teléfonos de cada estudiante, lo que permite una mayor flexibilidad sin crear campos adicionales.

PROFESORES: En la colección profesores, se almacena la información de cada profesor de manera similar a los estudiantes. Sin embargo, se decidió evitar redundancias y eliminar el campo clases dentro de los documentos de profesores, ya que esa relación se maneja mejor desde la colección clases.

Fields	Global Probability	Type
▼ [escuela.profesores]	100.0%	Collection
> _id	100.0%	Int32
> apellido1	100.0%	String
> apellido2	100.0%	String
> clases	100.0%	Array
> email	100.0%	String

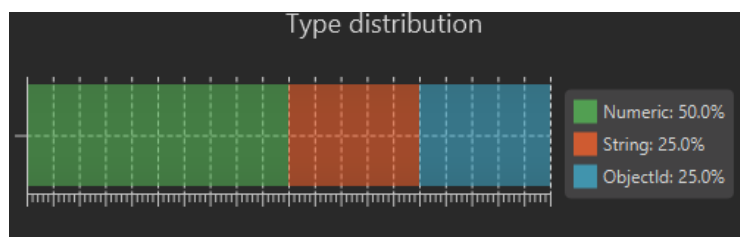
Fields	Global Probability	Type
▼ [escuela.clases]	100.0%	Collection
> _id	100.0%	Int32
> credits	100.0%	Int32
> horario	100.0%	Object
> nombre_clase	100.0%	String
> profesor	100.0%	Int32



CLASES: La colección clases tiene como objetivo almacenar información sobre las asignaturas que se imparten en la escuela. Cada clase está relacionada directamente con un profesor a través de su ID, y contiene datos como el horario, los créditos y el nombre de la clase. Esta estructura asegura una consulta eficiente de los profesores que imparten clases y facilita la asignación de estudiantes a cada materia.

MATRICULAS: La colección matriculas se utiliza como una tabla intermedia para gestionar la relación de muchos a muchos entre estudiantes y clases. Esto permite a los estudiantes inscribirse en varias clases sin duplicar datos. Cada documento en esta colección contiene el ID

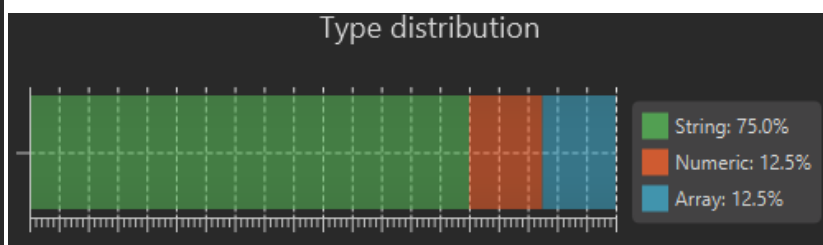
Fields	Global Probability	Type
✓ [escuela.matriculas]	100.0%	Collection
> _id	100.0%	ObjectId
> clase_id	100.0%	Int32
> estudiante_id	100.0%	Int32
> fecha_matricula	100.0%	String



del estudiante y el ID de la clase, así como la fecha de inscripción, lo que proporciona un historial de las inscripciones.

ADMINISTRATIVOS: Esta colección almacenará la información de los miembros del personal administrativo de la escuela. Además del nombre, apellidos y datos de contacto, se añadirá un campo rol que especificará su función en la institución y tendrá su respectiva contraseña en caso de separar el sistema.

Fields	Global Probability	Type
✓ [escuela.administrativo]	100.0%	Collection
> _id	100.0%	Int32
> apellido1	100.0%	String
> apellido2	100.0%	String
> contraseña	100.0%	String
> email	100.0%	String
> nombre	100.0%	String
> rol	100.0%	String
> telefono	100.0%	Array



Modelo de Datos: Normalización y Relaciones

Se ha implementado la **normalización** para minimizar la duplicación de datos. Por ejemplo:

- Los teléfonos se almacenan como un array dentro de cada documento de estudiante o profesor, lo que permite flexibilidad sin crear múltiples campos.
- Las clases están vinculadas a los profesores a través de una referencia (ID), manteniendo las relaciones claras sin necesidad de repetir información innecesaria.

Las **relaciones uno a muchos** se maneja con referencias entre documentos. Un profesor puede impartir varias clases (relación uno a muchos), y un estudiante puede estar inscrito en varias clases (relación muchos a muchos), gestionado a través de la colección **matriculas**.

Decisión de Embebido

Se tomó la decisión de utilizar **referencias** para evitar la duplicación de datos, especialmente entre las clases y los profesores. Por ejemplo:

- En lugar de embeber toda la información del profesor dentro de cada clase, solo se almacena su ID. Esto permite que los datos del profesor solo necesiten actualizarse en un lugar, lo que mejora la consistencia y el rendimiento.

Por otro lado, se **empeñó en evitar redundancia** al no incluir un campo que almacene las clases dentro de los documentos de los profesores, ya que esa relación se gestiona mejor desde la colección **clases**.

Relaciones y Cardinalidad

Las relaciones entre colecciones están estructuradas de la siguiente manera:

- **Uno a muchos:** Un profesor puede impartir varias clases.
- **Muchos a muchos:** Los estudiantes pueden inscribirse en múltiples clases, y cada clase puede tener múltiples estudiantes. Esto se gestiona a través de la colección **matriculas**.

Métodos de captura

Para poblar la base de datos, se utilizaron archivos JSON que contenían datos estructurados de estudiantes, profesores y clases. Estos archivos fueron cargados en MongoDB utilizando el comando `insertMany()`. A partir de ahí, se insertaron los documentos necesarios en cada colección, asegurando que los datos estuvieran correctamente organizados y formateados.

```
1 db.profesores.insertMany([
2   {
3     "_id":1,
4     "nombre": "María",
5     "apellido1": "Rodríguez",
6     "apellido2": "Martínez",
7     "telefono": ["98765432"],
8     "email": "maria.rodriguez@escuela.com",
9     "clases": []
10  }
11  ];
```

1. Importación de Datos

Los datos se importaron desde archivos JSON que contenían información de estudiantes, profesores y clases. Esto permitió poblar rápidamente la base de datos con información relevante, sin la necesidad de ingresar los datos manualmente.

2. Preparación de los Datos

Los datos se estructuraron para garantizar que estuvieran alineados con el esquema definido para cada colección. Por ejemplo, se garantizó que cada clase tuviera una referencia válida a un profesor y que cada inscripción (matrícula) estuviera correctamente asociada con el estudiante y la clase correspondientes.

Consultas

Consultas de base de datos escuela.

Insert

1.en esta consulta insertamos un nuevo profesor con id: 4

```
db.profesores.insertOne({
  "_id": 5,
  "nombre": "Sandra",
  "apellido1": "Herrera",
  "apellido2": "Velásquez",
  "telefono": ["87654321"],
  "email": "sandra.herrera@escuela.com"
})
```

acknowledged	insertedId
T/F true	id 4.0

2.En esta consulta registramos una nueva matricula de un estudiante en la tabla de muchos a muchos

```
db.matriculas.insertOne({
  "estudiante_id": NumberInt(3),
  "clase_id": NumberInt(2),
  "fecha_matricula": "2024-02-10"
})
```

acknowledged	insertedId
T/F true	id 6711183588816...

update

1.en esta consulta actualizamos la edad del estudiante con id: 4 a la edad de 17años

```
db.estudiantes.updateOne(
  { "_id": 4 },
  { $set: { "edad": 17 } }
)
```

_id	nombre	apellido	edad
4	Clara	Fernández	17

2.en esta consulta se actualiza los números de teléfono del docente en un array de números.

```
db.profesores.updateOne(
  { "_id": 4 },
  { $set: { "telefono": ["65432189", "2323213", "232414312"] } }
)
```

read

1. Consultar todos los administrativos con el rol de rector:

```
1 db.clases.find({ "profesor": 1 })
```

clases > nombre_clase				
_id	nombre_clase	profesor	creditos	horario
1	Matemáticas 101	1	4	{ 3 fields }



Esta consulta devuelve todos los administrativos que tienen el rol de rector, proporcionando acceso a sus datos de contacto y roles

delete

1. en esta consulta eliminamos la matrícula de una materia de un estudiante

```
db.matriculas.deleteOne({ "estudiante_id": 3, "clase_id": 5 })
```

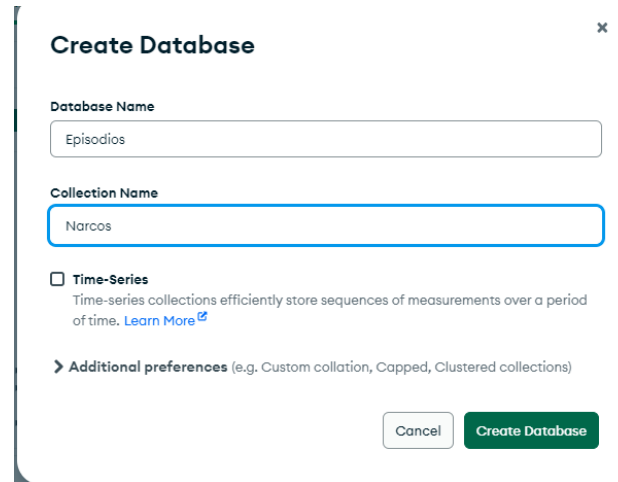
Result > deletedCount

acknowledged	deletedCount
 true	 1.0

Consultas y creación base de taller

1. creacion de la base de datos y cargue de ficheros de “ejercicio_00.json” a la base de datos

Episodios y Narcos



Create Database

Database Name
Episodios

Collection Name
Narcos

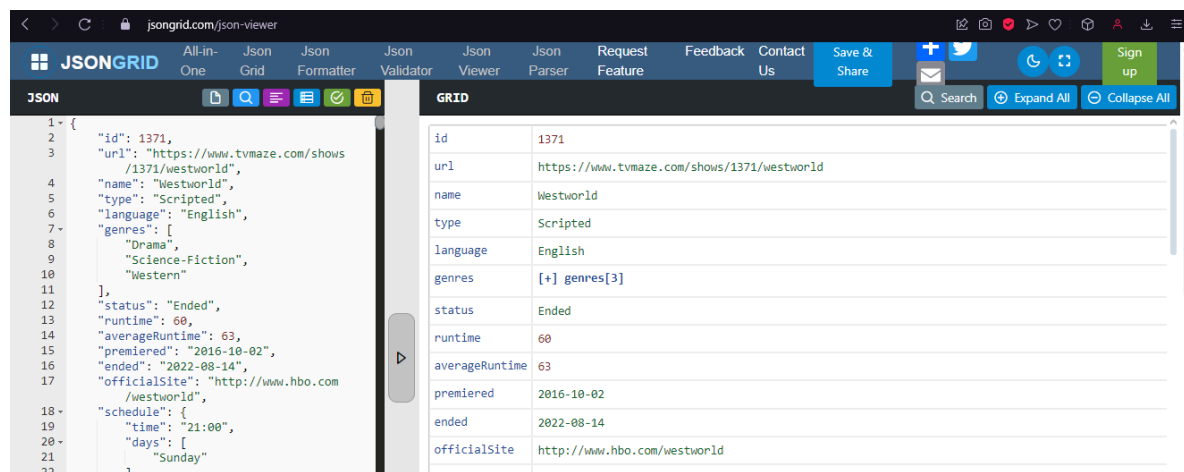
☐ Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. [Learn More](#)

> Additional preferences (e.g. Custom collation, Capped, Clustered collections)

Cancel Create Database

2

Cargara los datos a la plataforma <https://jsongrid.com/json-viewer>.



JSON

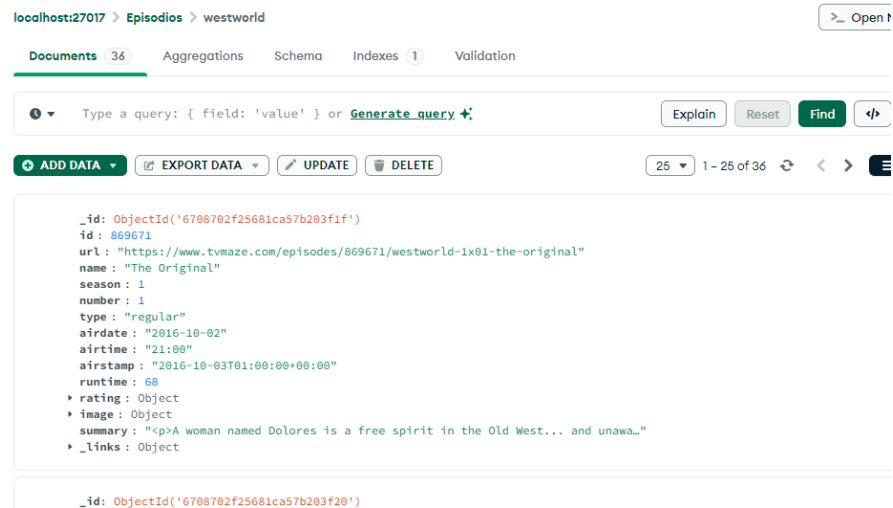
```

1 {
2   "id": 1371,
3   "url": "https://www.tvmaze.com/shows/1371/westworld",
4   "name": "Westworld",
5   "type": "Scripted",
6   "language": "English",
7   "genres": [
8     "Drama",
9     "Science-Fiction",
10    "Western"
11  ],
12  "status": "Ended",
13  "runtime": 60,
14  "averageRuntime": 63,
15  "premiered": "2016-10-02",
16  "ended": "2022-08-14",
17  "officialSite": "http://www.hbo.com/westworld",
18  "schedule": {
19    "time": "21:00",
20    "days": [
21      "Sunday"
22    ]
23  }
24 }
```

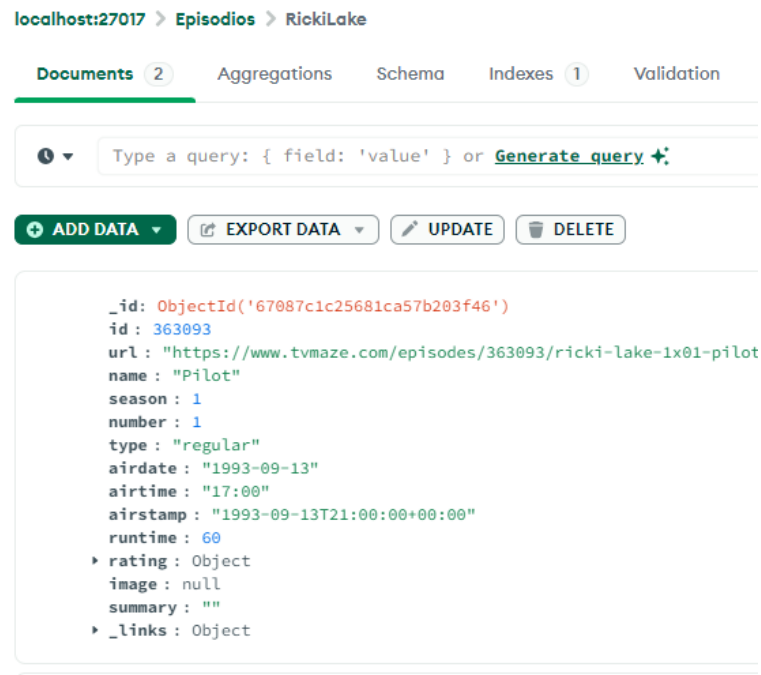
GRID

id	1371
url	https://www.tvmaze.com/shows/1371/westworld
name	Westworld
type	Scripted
language	English
genres	[+] genres[3]
status	Ended
runtime	60
averageRuntime	63
premiered	2016-10-02
ended	2022-08-14
officialSite	http://www.hbo.com/westworld

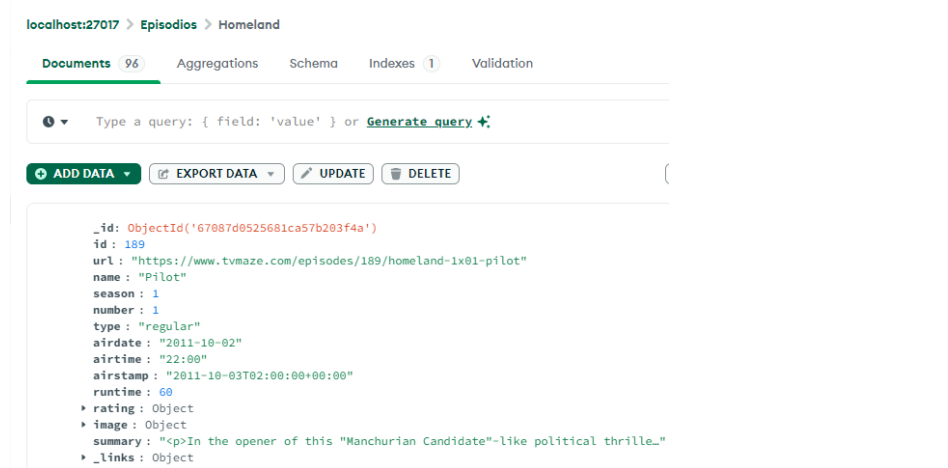
Y creo una lista solo con los episodios, llamado westworld



3. se repite el proceso con el ficher “ejercicio_02.json” y lo llamamos la colección “RickiLake”



4. se repite el proceso con el ficher “ejercicio_03.json” y lo llamamos la colección “Homeland”



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

Create Database

Database Name

Collection Name

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

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

CRUD

READ

Visualiza todas las bases de datos existentes: **show dbs**

```
> show databases
< Episodios 192.00 KiB
  School    56.00 KiB
  admin     40.00 KiB
  config    108.00 KiB
  ecommerce 592.00 KiB
  local     80.00 KiB
  tienda    288.00 KiB
> use School
< switched to db School
School> |
```

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

encuentra todos los documentos donde el id del estudiante sea igual a 2.

Grades > student_id

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

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

encuentra el documentos donde el primer elemento en el array scores (índice 0) tenga un campo score menor o igual a 10

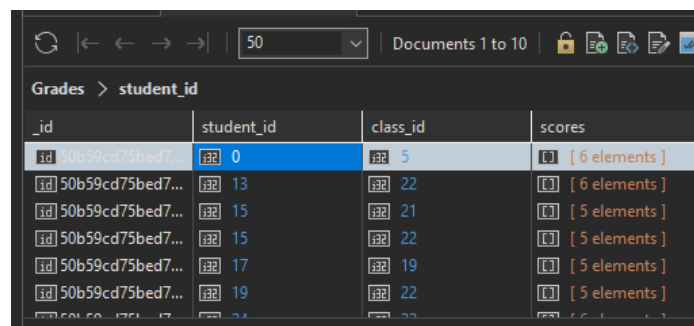
```

{
  "_id" : ObjectId("50b59cd75bed76f46522c352"),
  "student_id" : NumberInt(0),
  "class_id" : NumberInt(24),
  "scores" : [
    {
      "type" : "exam",
      "score" : 4.444435759027499
    },
    {
      "type" : "quiz",
      "score" : 28.63057857803885
    }
  ]
}

```

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

encuentra los documentos donde el quinto elemento en el array scores (índice 4) tenga un campo score menor o igual a 10.



_id	student_id	class_id	scores
50b59cd75bed76f46522c352	0	5	[6 elements]
50b59cd75bed76f46522c353	13	22	[6 elements]
50b59cd75bed76f46522c354	15	21	[5 elements]
50b59cd75bed76f46522c355	15	22	[5 elements]
50b59cd75bed76f46522c356	17	19	[5 elements]
50b59cd75bed76f46522c357	19	22	[5 elements]

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

encuentra los documentos donde el sexto elemento en el array scores (índice 5) tenga un campo score menor o igual a 10.

```

{
  "type" : "homework",
  "score" : 49.51759142259007
},
{
  "type" : "homework",
  "score" : 9.48049922977977
}
]

```

Vemos que el ultimo score es 9.4 por lo cual si entraría en esta consulta.

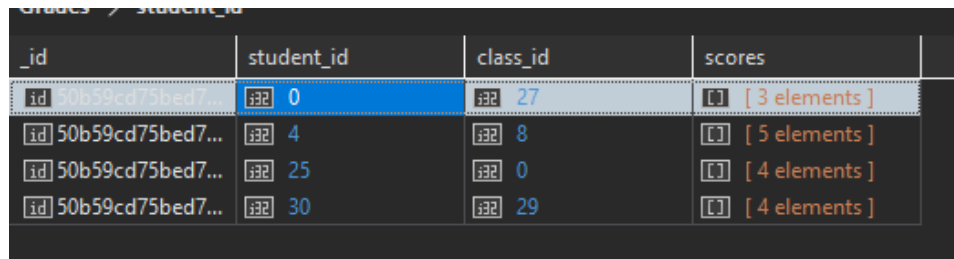
5. `db.grades.find({ "scores.6.score": { $lte: 10 } })`

encuentra los documentos donde el septimo elemento en el array scores (índice 6) tenga un campo score menor o igual a 10.

No encuentra resultados por que la cantidad de scores de cada documento solo es 6

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

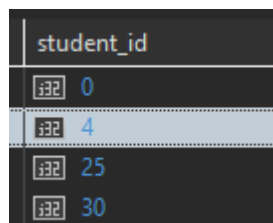
Busca documentos donde el primer elemento en el array scores (índice 0) tenga un score entre 60 y 61



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

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

Busca los mismos documentos que en la consulta anterior, pero los ordena en función del “student_id” en orden ascendente.



student_id
0
4
25
30

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

busca el documento donde el estudiante id sea 2 y la class_id sea 24

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

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

Busca documentos donde el `class_id` sea 20, El primer elemento en el array `scores` (índice 0) tenga un `score` entre 15 y 30 y usando **\$and** para combinar ambas condiciones.

Obtenemos Todos los registros donde la clase sea 20 y la primera calificación en el array `scores` esté entre 15 y 30.

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

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

se Usa **\$elemMatch** para buscar documentos donde al menos un elemento en el array `scores` sea de tipo `quiz` y tenga un `score` mayor o igual a 50.

grades > student_id			
_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]
50b59cd75bed7...	3	10	[4 elements]
50b59cd75bed7...	3	9	[6 elements]
50b59cd75bed7...	3	12	[4 elements]

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

Usa **\$elemMatch** para buscar documentos donde al menos un elemento en el array scores

grades > student_id

_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]

sea de tipo “**exam**” y tenga un score mayor o igual a 50

Punto 3.

Consulta dentro de **NARCOS**

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

Busca en la colección Narcos todos los episodios cuyo tiempo de ejecución (runtime) sea mayor o igual a 55 minutos. Solo devuelve los campos name (nombre del episodio), season (número de temporada) y number (número del episodio), excluyendo el campo _id.

Narcos > season

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

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

Busca episodios en la colección Narcos cuyo tiempo de ejecución sea mayor o igual a 15 minutos. Solo devuelve los campos `season` y `number`, excluyendo el `_id`, y ordena los resultados en orden ascendente por temporada (`season:1`) y descendente por número de episodio (`number:-1`).

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

1 document selected

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

Busca en la colección Narcos todos los documentos donde el campo `season` sea de tipo numérico.

_id	id	url	name	season	number	type	airdate
67086eac25681c...	203469	https://www.tv...	Descenso	1	1	regular	2015-08-28
67086eac25681c...	208978	https://www.tv...	The Sword of S...	1	2	regular	2015-08-28
67086eac25681c...	208979	https://www.tv...	The Men of AL...	1	3	regular	2015-08-28
67086eac25681c...	208980	https://www.tv...	The Palace in F...	1	4	regular	2015-08-28
67086eac25681c...	208981	https://www.tv...	There Will Be a ...	1	5	regular	2015-08-28
67086eac25681c...	208982	https://www.tv...	Explosivos	1	6	regular	2015-08-28
67086eac25681c...	208983	https://www.tv...	You Will Cry Te...	1	7	regular	2015-08-28
67086eac25681c...	208984	https://www.tv...	La Gran Mentira	1	8	regular	2015-08-28
67086eac25681c...	208985	https://www.tv...	La Catedral	1	9	regular	2015-08-28
67086eac25681c...	208986	https://www.tv...	Despegue	1	10	regular	2015-08-28
67086eac25681c...	832098	https://www.tv...	Free at Last	2	1	regular	2016-09-02

1 document selected

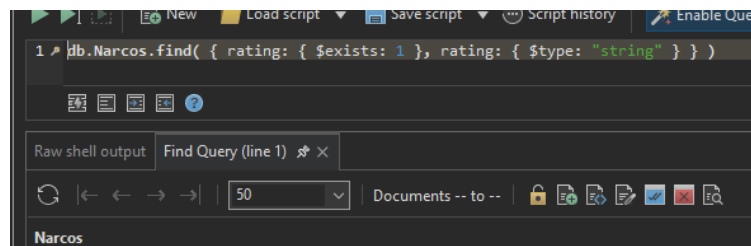
4. `db.Narcos.find({ rating: { $exists: 1 } })`

Busca en la colección Narcos todos los documentos que contienen el campo rating, es decir, donde el campo rating existe.

_id	id	url	name	season	number	type	airdate
67086eac25681c...	203469	https://www.tv...	Descenso	1	1	regular	2015-08-28
67086eac25681c...	208978	https://www.tv...	The Sword of S...	1	2	regular	2015-08-28
67086eac25681c...	208979	https://www.tv...	The Men of Al...	1	3	regular	2015-08-28
67086eac25681c...	208980	https://www.tv...	The Palace in F...	1	4	regular	2015-08-28
67086eac25681c...	208981	https://www.tv...	There Will Be a ...	1	5	regular	2015-08-28
67086eac25681c...	208982	https://www.tv...	Explosivos	1	6	regular	2015-08-28
67086eac25681c...	208983	https://www.tv...	You Will Cry Te...	1	7	regular	2015-08-28
67086eac25681c...	208984	https://www.tv...	La Gran Mentira	1	8	regular	2015-08-28
67086eac25681c...	208985	https://www.tv...	La Catedral	1	9	regular	2015-08-28
67086eac25681c...	208986	https://www.tv...	Despegue	1	10	regular	2015-08-28
67086eac25681c...	832098	https://www.tv...	Free at Last	2	1	regular	2016-09-02

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

Busca en la colección Narcos todos los documentos que contienen el campo rating y donde el valor de rating es de tipo cadena de texto (string).

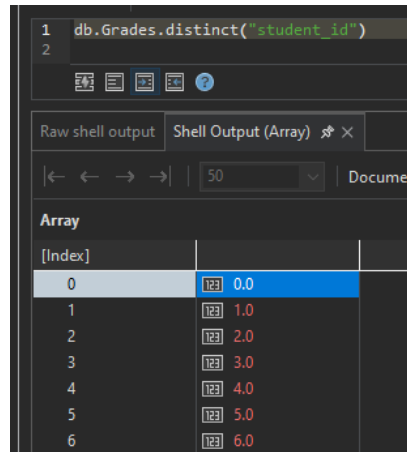


Pero como el campo rating no es del tipo string, sino un objeto no nos arroja ningún resultado.

Punto 4

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

Busca y devuelve una lista con los valores únicos del campo `student_id` en la colección `grades`. Esto significa que no se repiten los mismos `student_id` en el resultado, y solo se muestra cada valor una vez.



```

1 db.Grades.distinct("student_id")
2

```

Raw shell output | Shell Output (Array) ✖

|< < > >| 50 | Document

Array

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

2. `db.grades.countDocuments()`

Cuenta el número total de documentos presentes en la colección `grades`. Esta consulta devuelve un número que representa la cantidad de documentos que existen en dicha colección.

```

5 >
6 > School
7 0
8 280
9

```

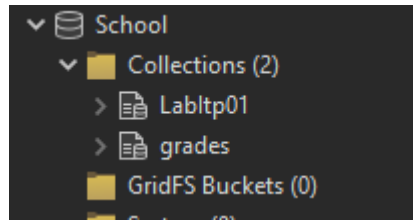
CREATE

PUNTO 1

1.En la BD School crea la colección LabItp:

`db.createCollection("LabItp01")`

Crea una nueva colección llamada LabItp01.



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

Inserta un nuevo documento en la colección LabItp01 con `_id:1`, nombre "pepe", teléfono 123456, y una lista de clases [20, 22, 25].

A screenshot of a MongoDB interface showing the 'LabItp01' collection. The table has columns: _id, name, phone, and class. The first document is highlighted with a blue background.

_id	name	phone	class
1	pepe	123456	[3 elements]

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

Inserta un documento en la colección con `_id:2`, nombre "juanito", teléfono 654789, y clases [10, 12, 15].

A screenshot of a MongoDB interface showing the 'LabItp01' collection. The table has columns: _id, name, phone, and class. Two documents are listed, with the second one highlighted in blue.

_id	name	phone	class
1	pepe	123456	[3 elements]
2	juanito	654789	[3 elements]

```
4 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 en la colección, incluyendo nombres, teléfonos, y listas de clases. Como hay una duplicación de `_id:5` solo crea el primero que sería anita y el segundo que es Joselito no se crea.

LabItp01 > name			
_id	name	phone	class
1	pepe	123456	[3 elements]
2	juanito	654789	[3 elements]
3	carlito	639852	[2 elements]
4	camilito	741258	[1 elements]
5	anita	852741	[1 elements]

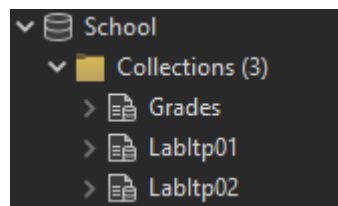
```
4 db.LabItp01.find( { class: 10 } )
```

Busca documentos en la colección donde el valor 10 esté presente en el array class.

LabItp01 > name			
_id	name	phone	class
2	juanito	654789	[3 elements]
3	carlito	639852	[2 elements]
5	anita	852741	[1 elements]

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

Inserta un documento en la colección LabItp02 con el campo name: "carolita"



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

con esta consulta le insertamos documentos a la colección carolita, pero dejamos que mongo le asigne in id

Result > insertedId

acknowledged	insertedId
T/F true	id 670d8eecfbcf1...

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

Inserta un documento en la colección LabItp02 con el nombre "carolita", información adicional (aula y casillero), y la edad 25.

_id	name	information	age
id 670d8e73fbcf16...	carolita		
id 670d8eecfbcf16...	carolita	{ 2 fields }	25
id 670d9095fbcf16...	carolita	{ 2 fields }	25

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

Recupera todos los documentos de la colección LabItp02.

LabItp02 > name

_id	name	information	age
id 670d8e73fbcf16...	carolita		
id 670d8eecfbcf16...	carolita	{ 2 fields }	25
id 670d9095fbcf16...	carolita	{ 2 fields }	25

UPDATE

En la colección LabItp01 realiza las siguientes actualizaciones.

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

Actualiza el documento con `_id: 7`, añadiendo o modificando el campo `virtues` con un array que contiene los valores `['cheerful', 'funny', 'comprehensive', 'sociable', 'respectful']`.

Pero como no existe un documento con `id 7` no actualiza nada

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

actualiza el documento con `_id: 7`, añadiendo o modificando el campo `information` con el aula `room_A` y el casillero `15`, y establece la edad en `18`, pero pasa lo mismo que en la anterior

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

Si el documento con `_id: 10` existe, lo actualiza añadiendo los campos `name`, `age`, `virtues` (vacío), e `information` (vacío). Si no existe, crea un nuevo documento con estos valores (`upsert: true`).

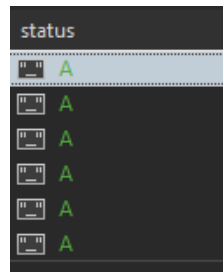
Result > insertedId				
acknowledged	insertedId	matchedCount	modifiedCount	upsertedCount
true	null	1.0	1.0	0.0

Actualiza los documentos con `_id: 1` a `_id: 6`, agregando el campo `virtues` con un valor único de la lista `['cheerful', 'funny', 'comprehensive', 'sociable', 'respectful']`.

{Document id}	0
1	respectful
2	respectful
3	respectful
4	respectful
5	respectful

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

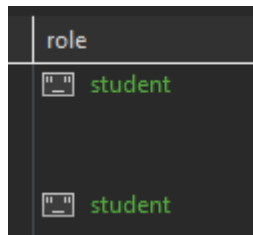
Actualiza todos los documentos de la colección LabItp01, añadiendo el campo status con el valor "A".



status	
11 11	A
11 11	A
11 11	A
11 11	A
11 11	A

Actualiza los documentos cuyo nombre sea "pepe" o "camilito", añadiendo el campo role con el valor "student".

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



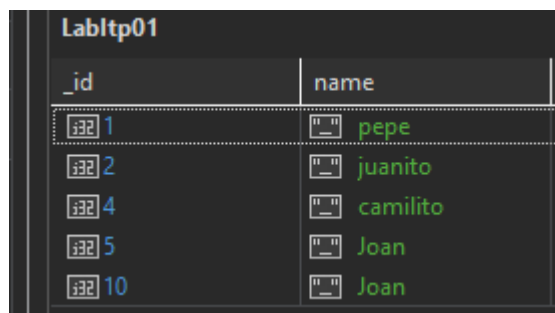
role	
11 11	student
11 11	student

DELETE

En la colección LabItp01 realiza las siguientes actualizaciones:

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

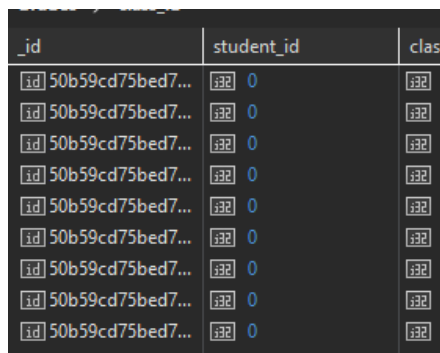
Elimina un único documento de la colección LabItp01 donde el campo name sea "carlito".



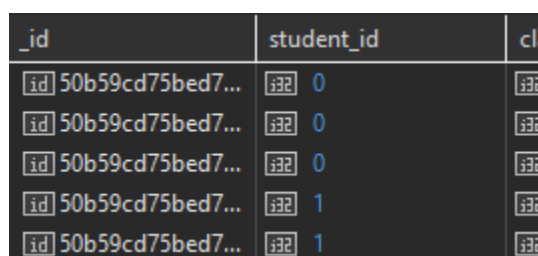
_id	name
1	pepe
2	juanito
4	camilito
5	Joan
10	Joan

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

Elimina solo 1 documento de la colección grades donde el campo student_id sea 0



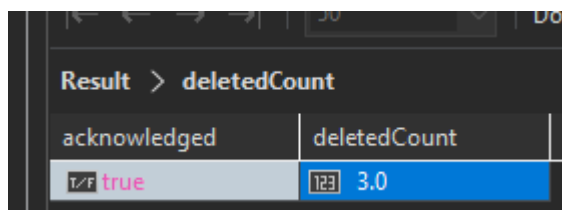
_id	student_id	clas
50b59cd75bed7...	0	
50b59cd75bed7...	0	
50b59cd75bed7...	0	
50b59cd75bed7...	0	
50b59cd75bed7...	0	
50b59cd75bed7...	0	
50b59cd75bed7...	0	
50b59cd75bed7...	0	
50b59cd75bed7...	0	



_id	student_id	cla
50b59cd75bed7...	0	
50b59cd75bed7...	0	
50b59cd75bed7...	0	
50b59cd75bed7...	1	
50b59cd75bed7...	1	

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

Elimina todos los documentos de la colección grades donde el campo student_id sea 0



Result > deletedCount	
acknowledged	deletedCount
<input checked="" type="checkbox"/> true	3.0

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

Elimina un solo documento de la colección grades donde student_id sea 1. El parámetro justOne: true asegura que solo se elimine un documento.

Result > deletedCount		
acknowledged	deletedCount	
T/F true	123 1.0	

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

Elimina todos los documentos de la colección grades donde `student_id` sea 1.

Grades > student_id		
_id	student_id	class_id
[id] 50b59cd75bed7...	2	25
[id] 50b59cd75bed7...	2	27
[id] 50b59cd75bed7...	2	24
[id] 50b59cd75bed7...	3	10
[id] 50b59cd75bed7...	3	9
[id] 50b59cd75bed7...	3	25

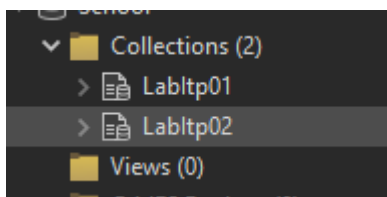
➤ `db.grades.remove({ })`

Elimina todos los documentos de la colección grades, ya que no se especifica ninguna condición de búsqueda.

Result > deletedCount		
acknowledged	deletedCount	
T/F true	123 264.0	

➤ `db.grades.drop()`

Elimina por completo la colección grades de la base de datos, incluyendo todos sus documentos y el esquema asociado.



Ya no aparece la colección al refrescar.

Análisis y discusión

Interpretación de los Resultados

La implementación de la base de datos para la escuela en MongoDB ha permitido diseñar un sistema flexible y eficiente para gestionar la información de estudiantes, profesores, clases y matrículas. A través de las consultas realizadas, se ha verificado que el diseño cumple con las expectativas y permite llevar a cabo tareas administrativas comunes, como el registro de nuevos estudiantes, la asignación de clases a profesores y el control de matrículas.

1. Eficiencia en la Recuperación de Datos

El uso de referencias entre las colecciones en lugar de duplicar datos ha demostrado ser una solución eficiente. Las consultas para obtener las clases de un profesor o los estudiantes inscritos en una clase han sido ejecutadas rápidamente, mostrando la ventaja de mantener relaciones bien estructuradas entre las colecciones.

Además, la consulta para actualizar información de contacto, como los números de teléfono de los profesores, mostró que el uso de arrays para almacenar múltiples valores proporciona flexibilidad sin redundancia. Esto también es evidente en las relaciones de muchos a muchos (como las de estudiantes y clases), gestionadas mediante la colección matriculas.

2. Control de Acceso y Gestión de Roles

La implementación de la colección administrativos añade un nivel de control esencial sobre los accesos al sistema. Cada usuario administrativo tiene un rol específico, como rector, director académico o administrador del sistema, lo que permite ajustar los permisos de acceso según sus responsabilidades. Esta separación de roles facilita la administración y seguridad de la

información, evitando que todos los usuarios tengan acceso irrestricto a todas las funciones del sistema.

3. Análisis de la Normalización

El diseño de la base de datos sigue las normas de normalización, evitando la duplicación de datos. En lugar de almacenar información redundante en diferentes colecciones, se han utilizado referencias entre documentos. Por ejemplo, en lugar de repetir los datos del profesor en cada clase, se almacena un ID que lo referencia en la colección profesores. Esto reduce el tamaño de los documentos y facilita la actualización de la información sin causar inconsistencias.

4. Resultados de Consultas Complejas

Las consultas más complejas, como la asignación de múltiples clases a los profesores o la verificación de si un teléfono ya está registrado antes de actualizarlo, se realizaron sin problemas, lo que demuestra la flexibilidad y el poder de MongoDB para manejar datos no relacionales. Por ejemplo, la lógica condicional para verificar si un número de teléfono ya estaba registrado en la colección de profesores y actualizarlo solo cuando era necesario es una muestra de cómo MongoDB puede gestionar eficientemente estos procesos.

Conclusión

La creación e implementación de una base de datos escolar en MongoDB proporciona una estructura potente y escalable para gestionar eficazmente las relaciones entre estudiantes, profesores, clases y administradores. Usar enlaces entre colecciones en lugar de incrustar datos puede hacer que la base de datos sea más eficiente y más fácil de mantener.

MongoDB demuestra ser una herramienta ideal para este tipo de proyectos debido a su flexibilidad y capacidad para manejar grandes cantidades de datos no estructurados. La base de datos desarrollada es capaz de realizar consultas complejas, actualizaciones de información eficientes y una gestión intuitiva y rápida de asignaciones de clases y registros.

El uso de colecciones administrativas y gestión de roles añade una importante capa de seguridad al sistema. Permite que los usuarios accedan solo a las áreas que necesitan según su rol dentro de la organización.

Recomendaciones

Monitoreo del Rendimiento: A medida que la base de datos siga creciendo, es importante monitorear el rendimiento de las consultas, especialmente aquellas que involucran relaciones de muchos a muchos (como las matrículas). MongoDB ofrece herramientas de análisis de rendimiento que pueden ser útiles para ajustar el diseño y optimizar el uso de índices.

Ampliación de Funcionalidades: Se sugiere implementar funcionalidades adicionales, como el registro de **notas de los estudiantes, calificaciones por clase o asistencia**, para que el sistema sea más completo y abarque más aspectos del control escolar.

Mejorar la Seguridad: Si se prevé el uso del sistema en un entorno de producción, es importante considerar la implementación de **autenticación y encriptación de datos** para proteger la información sensible, como los datos personales de estudiantes y profesores.

Referencias

JSONGrid. (n.d.). *JSONGrid: JSON data viewer & validator*. <https://jsongrid.com>

MongoDB Inc. (n.d.). *MongoDB Manual*. MongoDB.

<https://www.mongodb.com/docs/manual/>

MongoDB Inc. (n.d.). *MongoDB Compass*. <https://www.mongodb.com/products/compass>

Studio 3T. (n.d.). *Studio 3T Database GUI for MongoDB*. <https://studio3t.com/>

Jhonier Pasos (2024). Repository for MONGO and SQL [GitHub Repository]. Retrieved from https://github.com/jhonierp/repository_sql.git