

Base de Datos NoSQL - Librería

Presentado Por:

Maicol Sebastián Guerrero López

Presentado A:

Ing. Brayan Ignacio Arcos Burbano

Instituto Tecnológico Del Putumayo
Facultad De Ingeniería Y Ciencias Básicas
Ingeniería De Sistemas
Bases De Datos Y Almacenamiento Masivo
Mocoa
2024

Índice

Resumen Ejecutivo.....	4
Introducción.....	5
Metodología.....	6
MongoDB:.....	6
MongoDB Compass:.....	6
Studio 3T:.....	6
Base de Datos NoSQL Librería	7
Esquema.....	7
Modelo de Datos: Normalización y Cardinalidad	7
1. Authors	7
2. Books.....	8
3. Roles.....	8
4. Users	9
5. Orders.....	10
Actualizaciones.....	10
• \$set:.....	10
• \$setOnInsert:	10
• \$push:.....	10
• \$each:.....	11

• upsert:.....	11
1. Authors	11
2. Books.....	12
3. Orders.....	13
4. Roles.....	14
5. Users	15
Consultas NoSQL Taller	16
1. Operaciones de Lectura (READ)	16
2. Operaciones de Creación (CREATE).....	23
3. Operaciones de Actualización (UPDATE)	26
4. Operaciones de Eliminación (DELETE)	29
Análisis y Discusión.....	32
Conclusiones	33
• Normalización eficiente:.....	33
• Escalabilidad:.....	33
• Consultas rápidas:	33
• Reducción de redundancia:	33
Referencias	34

Resumen Ejecutivo

Este informe describe un taller práctico sobre MongoDB enfocado en las operaciones CRUD (Crear, Leer, Actualizar y Eliminar). Durante el taller, se utilizaron varios conjuntos de datos en formato JSON para practicar la manipulación de datos en una base de datos NoSQL. El objetivo fue consolidar conocimientos sobre la inserción, consulta, actualización y eliminación de documentos en MongoDB, destacando su flexibilidad y eficiencia en el manejo de datos.

Introducción

La creciente necesidad de gestionar grandes volúmenes de datos ha impulsado el uso de bases de datos NoSQL como MongoDB, que destaca por su almacenamiento eficiente y su estructura flexible. MongoDB se ha vuelto popular en el desarrollo de aplicaciones gracias a su capacidad para manejar datos semiestructurados y ofrecer un alto rendimiento en entornos con grandes cantidades de información. Este informe documenta un taller práctico centrado en la aplicación de operaciones CRUD en MongoDB, utilizando colecciones de datos en formato JSON. A través de estas actividades, se busca consolidar el conocimiento sobre cómo MongoDB permite una manipulación rápida y eficiente de datos no relacionales.

Metodología

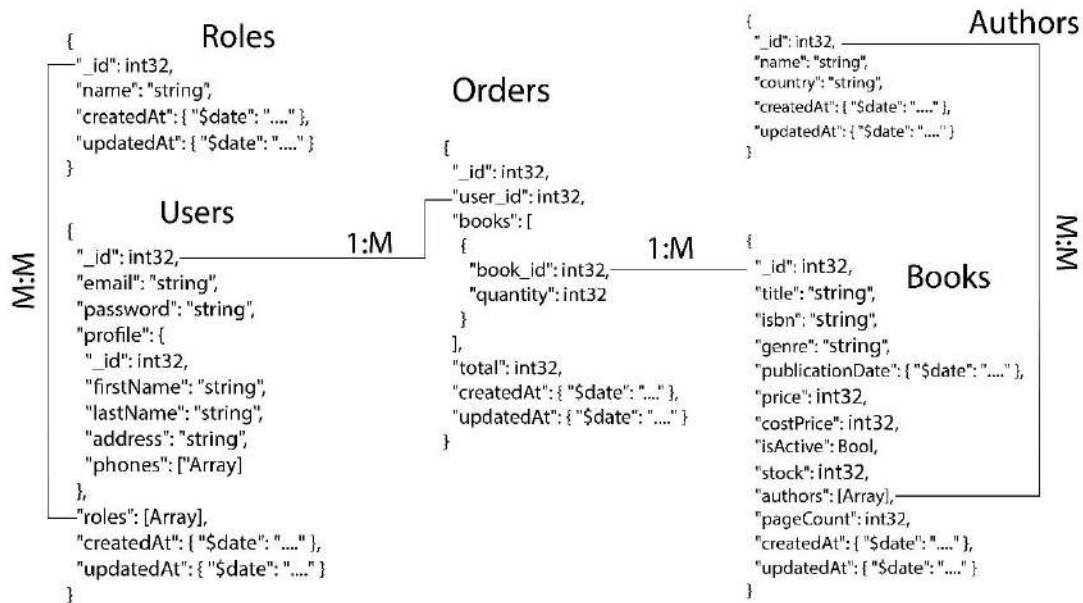
MongoDB: Sistema de gestión de bases de datos NoSQL que permite el almacenamiento eficiente y flexible de datos no estructurados. Facilita operaciones de lectura y escritura en grandes volúmenes de información.

MongoDB Compass: Herramienta gráfica que simplifica la visualización de datos, permitiendo a los usuarios explorar esquemas, realizar consultas y editar documentos sin necesidad de comandos complejos.

Studio 3T: Plataforma avanzada para el desarrollo y administración de MongoDB, que ofrece funcionalidades como la creación visual de consultas, la importación y exportación de datos, y un editor de JavaScript para scripts personalizados.

Base de Datos NoSQL Librería

Esquema



Modelo de Datos: Normalización y Cardinalidad

1. Authors

```
{
  "_id": 1,
  "name": "Miguel de Cervantes",
  "country": "España",
  "createdAt": { "$date": "2024-10-15T00:00:00Z" },
  "updatedAt": { "$date": "2024-10-15T00:00:00Z" }
}
```

Normalización: Se optó por no embebir los datos de los autores en los libros. En lugar de duplicar la información del autor en cada libro, se utiliza una referencia al autor. Esto promueve la normalización de datos, lo que implica que los detalles del autor solo se almacenan en una ubicación centralizada dentro de la colección authors. Esto facilita la actualización de datos y evita inconsistencias.

2. Books

```
{
  "_id": 1,
  "title": "El Quijote",
  "isbn": "978-3-16-148410-0",
  "genre": "Ficción",
  "publicationDate": { "$date": "1605-01-16" },
  "price": 59900,
  "costPrice": 45900,
  "isActive": true,
  "stock": 30,
  "authors": [1],
  "pageCount": 274,
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-15" }
}
```

Normalización: Los libros tampoco embeben datos de los autores, ya que los autores pueden tener varios libros y duplicar su información no es eficiente. En cambio, la colección books contiene una lista de referencias a la colección authors mediante el campo authors. Los campos relacionados con el precio (price, costPrice) y el inventario (stock) son específicos de cada libro, por lo que se almacenan directamente en el documento de books.

Cardinalidad: La relación entre books y authors es muchos a muchos (M:M), ya que un libro puede tener varios autores y un autor puede tener varios libros.

3. Roles

```
{
  "_id": 1,
  "name": "Administrador",
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-15" }
}
```


Normalización: Similar al caso de los autores, los roles no se embeben en users, sino que se utiliza una lista de referencias. Cada usuario puede tener múltiples roles (administrador, cliente, etc.), por lo que los roles se definen en una colección independiente para evitar duplicación y permitir una fácil actualización de los permisos asociados a un rol específico.

4. Users

```
{
  "_id": 1,
  "email": "juanperez@gmail.com",
  "password": "12345678",
  "profile": {
    "_id": 1,
    "firstName": "Juan Miguel",
    "lastName": "Pérez",
    "address": "Calle Falsa 123",
    "phones": ["123456789", "987654321"]
  },
  "roles": [1, 2],
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-15" }
}
```

Normalización: Los datos personales del usuario, como su perfil (nombre, dirección, teléfonos), están embebidos dentro del documento users. Estos detalles son únicos para cada usuario y no requieren duplicación en otras partes del sistema, por lo que es más eficiente embeber esta información directamente en el campo profile.

Cardinalidad: La relación entre users y roles es muchos a muchos (M:M). Los roles se manejan mediante referencias, donde cada usuario tiene una lista de identificadores de roles, permitiendo que un usuario tenga múltiples roles sin redundancia de datos.

5. Orders

```
{
  "_id": 1,
  "user_id": 1,
  "books": [
    {
      "book_id": 1,
      "quantity": 2
    },
    {
      "book_id": 2,
      "quantity": 1
    }
  ],
  "total": 111800,
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-15" }
}
```

Normalización: Las órdenes contienen una lista de libros pedidos, pero en lugar de embebir todos los detalles de los libros, solo se almacena el book_id y la cantidad pedida (quantity). Esto evita duplicar la información del libro en cada pedido. Los detalles completos del libro pueden obtenerse mediante una consulta adicional a la colección books, si es necesario.

Cardinalidad: La relación entre orders y books es de uno a muchos, ya que una orden puede contener múltiples libros, pero cada libro en la orden está asociado a una sola orden. La relación entre orders y users también es de uno a muchos, ya que un usuario puede tener múltiples órdenes, pero cada orden está asociada a un solo usuario.

Actualizaciones

- **\$set:** Actualiza los campos especificados en el documento. Si el campo no existe, lo crea.
- **\$setOnInsert:** Solo establece valores si el documento es nuevo (se inserta con upsert).
- **\$push:** Añade un nuevo valor a un arreglo dentro de un documento.

- **\$each:** Se usa con \$push para insertar múltiples valores en un arreglo de una sola vez.
- **upsert:** Si el documento no existe, lo inserta; si existe, lo actualiza. Combina "update" y "insert".

1. Authors

```
db.authors.updateOne(
  { _id: 1 }, // Coincidencia puntual por el ID del autor
  {
    $set: {
      name: "Miguel de Cervantes",
      country: "España",
      updatedAt: new Date() // Actualizamos la fecha de modificación
    },
    $setOnInsert: {
      createdAt: new Date() // Solo se establece si el documento se inserta
    }
  },
  { upsert: true } // Si no existe, lo inserta
);
```

- **\$set:** Actualiza el nombre, país y la fecha de modificación.
- **\$setOnInsert:** Establece la fecha de creación solo si se inserta un nuevo documento.
- **upsert:** Si el autor no existe, lo inserta.

```
//Antes:

{
  "_id": 1,
  "name": "Cervantes",
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-15" }
}

// Despues:

{
  "_id": 1,
  "name": "Miguel de Cervantes",
  "country": "España",
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-17" }
}
```

2. Books

```
db.books.updateOne(
  { _id: 1 }, // Coincidencia puntual por el ID del libro
  {
    $set: {
      price: 69900,
      stock: 30,
      updatedAt: new Date() // Fecha de modificación
    },
  },
);
```

- Actualiza el precio y el stock del libro con la fecha de modificación.
- No usa upsert ya que se espera que el libro ya exista.

```
//Antes:

{
  "_id": 1,
  "title": "El Quijote",
  "isbn": "978-3-16-148410-0",
  "genre": "Ficción",
  "price": 55000,
  "stock": 25,
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-15" }
}

// Despues:

{
  "_id": 1,
  "title": "El Quijote",
  "isbn": "978-3-16-148410-0",
  "genre": "Ficción",
  "price": 59900,
  "stock": 30,
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-17" }
}
```

3. Orders

```
db.orders.updateOne(
  { _id: 1 }, // Coincidencia puntual por el ID de la orden
  {
    $set: {
      total: 111800, // Actualizamos el total
      updatedAt: new Date() // Fecha de modificación
    },
    $push: {
      books: {
        $each: [
          { book_id: 3, quantity: 1 }, // Nuevo libro agregado a la orden
          { book_id: 4, quantity: 2 } // Otro libro
        ]
      }
    }
  }
);
```

- \$set: Actualiza el total y la fecha de modificación.
- \$push: Usa \$each para agregar varios libros nuevos al arreglo de books.

```
//Antes:
{
  "_id": 1,
  "user_id": 1,
  "books": [
    { "book_id": 1, "quantity": 2 },
    { "book_id": 2, "quantity": 1 }
  ],
  "total": 95000,
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-15" }
}

// Despues:
{
  "_id": 1,
  "user_id": 1,
  "books": [
    { "book_id": 1, "quantity": 2 },
    { "book_id": 2, "quantity": 1 },
    { "book_id": 3, "quantity": 1 },
    { "book_id": 4, "quantity": 2 }
  ],
  "total": 111800,
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-17" }
}
```

4. Roles

```
db.roles.updateOne(
  { _id: 1 }, // Coincidencia puntual por el ID del rol
  {
    $set: {
      name: "Administrador", // Actualizamos el nombre
      updatedAt: new Date() // Fecha de modificación
    },
    $setOnInsert: {
      createdAt: new Date() // Fecha de creación si es insertado
    }
  },
  { upsert: true } // Inserta si no existe
);
```

- \$set: Actualiza el nombre del rol y la fecha de modificación.
- \$setOnInsert: Establece la fecha de creación si se inserta un nuevo rol.

```
//Antes:

{
  "_id": 1,
  "name": "User",
  "createdAt": { "$date": "2024-10-" },
  "updatedAt": { "$date": "2024-10-" }
}

// Despues:

{
  "_id": 1,
  "name": "Administrador",
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-17" }
}
```

5. Users

```
db.users.updateOne(
  { _id: 1 }, // Coincidencia puntual por el ID del usuario
  {
    $set: {
      "profile.firstName": "Juan Miguel", // Actualizamos el nombre
      updatedAt: new Date() // Fecha de modificación
    },
    $push: {
      "profile.phones": {
        $each: ["456789123", "321987654"] // Añadimos varios teléfonos
      },
      "roles": {
        $each: [1, 2] // Agregamos nuevos roles
      }
    }
  }
);
```

- \$set: Actualiza el primer nombre y la fecha de modificación.
- \$push con \$each: Añade múltiples teléfonos y roles al perfil del usuario.

```
//Antes:
{
  "_id": 1,
  "email": "juanperez@gmail.com",
  "password": "12345678",
  "profile": {
    "firstName": "Juan",
    "lastName": "Pérez",
    "phones": ["123456789"]
  },
  "roles": [1],
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-15" }
}

// Despues:
{
  "_id": 1,
  "email": "juanperez@gmail.com",
  "password": "12345678",
  "profile": {
    "firstName": "Juan Miguel",
    "lastName": "Pérez",
    "phones": ["123456789", "456789123",
"321987654"]
  },
  "roles": [1, 3, 4],
  "createdAt": { "$date": "2024-10-15" },
  "updatedAt": { "$date": "2024-10-17" }
}
```

Consultas NoSQL Taller

1. Operaciones de Lectura (READ)

1.1 Encontrar documentos por student_id

Encuentra todos los documentos en la colección grades donde el student_id es

2.

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

```
▼ {
  "_id" : "50b59cd75bed76f46522c35e",
  "student_id" : NumberInt(2),
  "class_id" : NumberInt(25),
  "scores" : [
    { ... },
    { ... },
    { ... },
    { ... },
    { ... }
  ]
}
```


2.1 Encontrar documentos con puntajes específicos

Encuentra documentos donde el primer score (scores.0) es menor o igual a 10.

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

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

3.1 Encontrar documentos por el quinto score

Encuentra documentos donde el quinto score (scores.4) es menor o igual a 10.

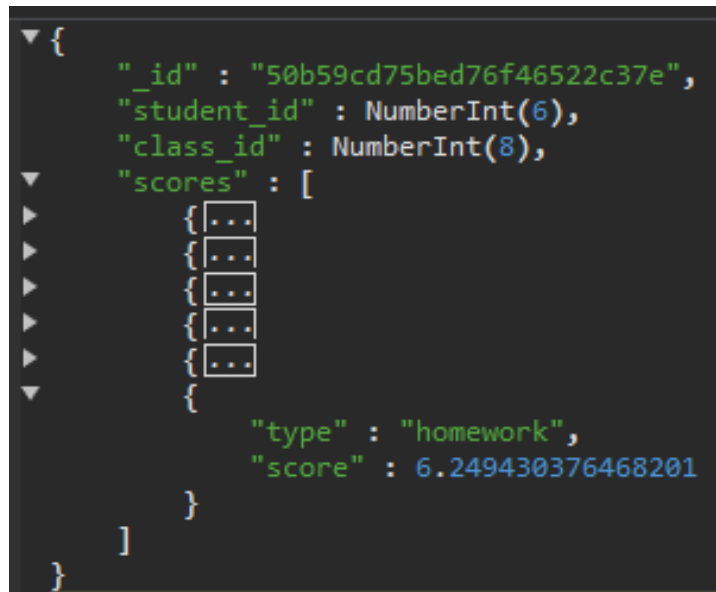
```
db.grades.find( { "scores.4.score": { $lte: 10 } } )
```

```
{
  "_id" : "50b59cd75bed76f46522c350",
  "student_id" : NumberInt(0),
  "class_id" : NumberInt(5),
  "scores" : [
    { ... },
    { ... },
    { ... },
    { ... },
    {
      "type" : "homework",
      "score" : 1.23735944117882
    },
    { ... }
  ]
}
```

4.1 Encontrar documentos por el sexto score

Encuentra documentos donde el sexto score (scores.5) es menor o igual a 10.

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



5.1 Encontrar documentos por el séptimo score

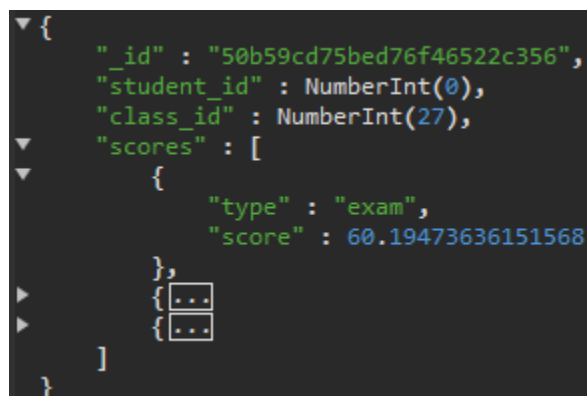
Encuentra documentos donde el séptimo score (scores.6) es menor o igual a 10.

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

6.1 Rango de scores

Encuentra documentos donde el primer score (scores.0) está entre 60 y 61 (inclusive).

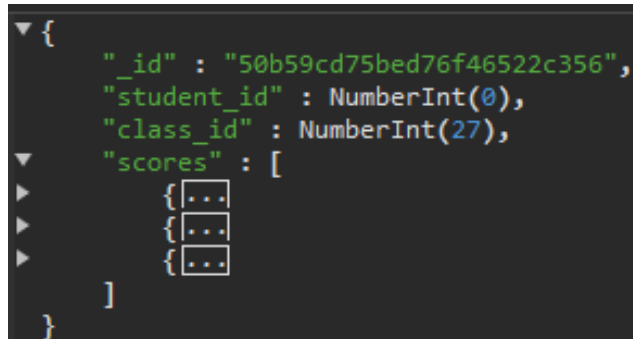
```
db.grades.find( { "scores.0.score": { $gte: 60, $lte: 61 } } )
```



7.1 Rango de scores y ordenación

Encuentra documentos donde el primer score (scores.0) está entre 60 y 61 y ordena los resultados por student_id en orden ascendente.

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

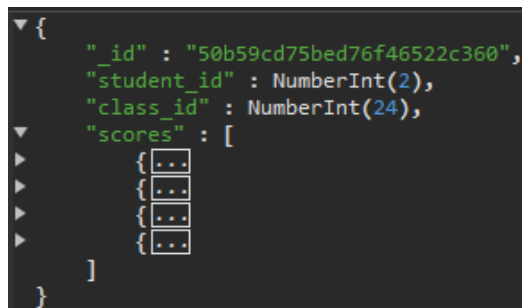


```
{
  "_id" : "50b59cd75bed76f46522c356",
  "student_id" : NumberInt(0),
  "class_id" : NumberInt(27),
  "scores" : [
    { "score": ... },
    { "score": ... },
    { "score": ... }
  ]
}
```

8.1 Buscar documentos con múltiples criterios

Encuentra documentos donde el student_id es 2 y el class_id es 24.

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



```
{
  "_id" : "50b59cd75bed76f46522c360",
  "student_id" : NumberInt(2),
  "class_id" : NumberInt(24),
  "scores" : [
    { "score": ... },
    { "score": ... },
    { "score": ... },
    { "score": ... }
  ]
}
```

9.1 Buscar por rango de scores en una clase

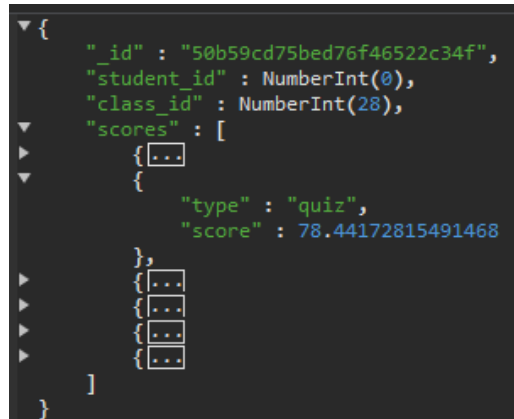
Encuentra documentos con class_id 20 donde el primer score (scores.0) está entre 15 y 30 (inclusive).

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

10.1 Buscar por tipo de score 'quiz'

Encuentra documentos que tienen al menos un score tipo 'quiz' con un score mayor o igual a 50.

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

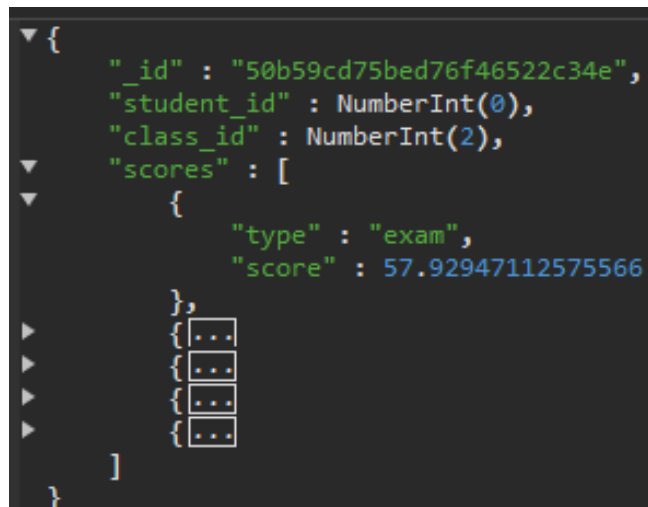


```
{
  "_id" : "50b59cd75bed76f46522c34f",
  "student_id" : NumberInt(0),
  "class_id" : NumberInt(28),
  "scores" : [
    {
      "type" : "quiz",
      "score" : 78.44172815491468
    },
    {
      "type" : "exam",
      "score" : 57.92947112575566
    },
    {
      "type" : "exam",
      "score" : 57.92947112575566
    },
    {
      "type" : "exam",
      "score" : 57.92947112575566
    }
  ]
}
```

11.1 Buscar por tipo de score 'exam'

Encuentra documentos que tienen al menos un score tipo 'exam' con un score mayor o igual a 50.

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



```
{
  "_id" : "50b59cd75bed76f46522c34e",
  "student_id" : NumberInt(0),
  "class_id" : NumberInt(2),
  "scores" : [
    {
      "type" : "exam",
      "score" : 57.92947112575566
    },
    {
      "type" : "exam",
      "score" : 57.92947112575566
    },
    {
      "type" : "exam",
      "score" : 57.92947112575566
    },
    {
      "type" : "exam",
      "score" : 57.92947112575566
    }
  ]
}
```

12.1 Filtrar por runtime en la colección "Narcos"

Encuentra documentos en la colección Narcos donde el runtime es mayor o igual a 55 y solo devuelve name, season, number y runtime (excluye _id).

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

```
{
  "name" : "Deutschland 93",
  "season" : NumberInt(2),
  "number" : NumberInt(7),
  "runtime" : NumberInt(57)
}
```

13.1 Ordenar por season y number

Encuentra documentos donde el runtime es mayor o igual a 15, devuelve season, number y runtime, y ordena por season ascendente y number descendente.

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

```
{
  "season" : NumberInt(1),
  "number" : NumberInt(10),
  "runtime" : NumberInt(45)
}
```

14.1 Filtrar por tipo de dato

Encuentra documentos donde el campo season es de tipo número.

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

```

{
  "_id" : ObjectId("67088361f79c973326a4297b"),
  "id" : NumberInt(208981),
  "url" : "https://www.tvmaze.com/episodes/208981/n",
  "name" : "There Will Be a Future",
  "season" : NumberInt(1),
  "number" : NumberInt(5),
  "type" : "regular",
  "airdate" : "2015-08-28",
  "airtime" : "",
  "airstamp" : "2015-08-28T12:00:00+00:00",
  "runtime" : NumberInt(55),
  "rating" : {...},
  "image" : {...},
  "summary" : "<p>Pablo's extreme methods is almost",
  "_links" : {
    "self" : {...},
    "show" : {...}
  }
}

```

15.1 Existencia de un campo

Encuentra documentos que tienen el campo rating.

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

```

{
  "_id" : ObjectId("67088361f79c973326a4297b"),
  "id" : NumberInt(208981),
  "url" : "https://www.tvmaze.com/episodes/208981/n",
  "name" : "There Will Be a Future",
  "season" : NumberInt(1),
  "number" : NumberInt(5),
  "type" : "regular",
  "airdate" : "2015-08-28",
  "airtime" : "",
  "airstamp" : "2015-08-28T12:00:00+00:00",
  "runtime" : NumberInt(55),
  "rating" : {
    "average" : 8.9
  },
  "image" : {...},
  "summary" : "<p>Pablo's extreme methods is almos",
  "_links" : {...}
}

```

16.1 Filtrar por tipo de dato de rating

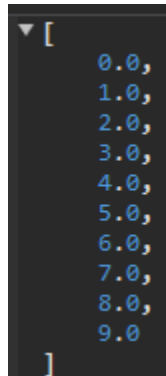
Encuentra documentos que tienen el campo rating y que es de tipo string.

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

17.1 Valores únicos de student_id

Encuentra valores únicos del campo student_id en la colección grades.

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



```
[
  0.0,
  1.0,
  2.0,
  3.0,
  4.0,
  5.0,
  6.0,
  7.0,
  8.0,
  9.0
]
```

18.1 Contar documentos en grades

Cuenta el número total de documentos en la colección grades.

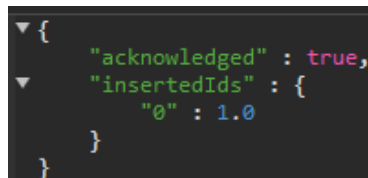
```
db.grades.countDocuments(): 65
```

2. Operaciones de Creación (CREATE)

1.1 Inserción de un documento usando insert

Inserta un documento usando el método insert (método obsoleto).

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



```
{
  "acknowledged" : true,
  "insertedIds" : {
    "0" : 1.0
  }
}
```

2.1 Inserción de un documento usando insertOne

Inserta un documento usando insertOne.

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

```
{
  "acknowledged" : true,
  "insertedId" : 2.0
}
```

3.1 Inserción de múltiples documentos

Inserta múltiples documentos usando insertMany.

```
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: 6, name: "joselito", phone: 1254896, class: [55, 458, 236, 20, 22, 10, 15] }])
```

```
{
  "acknowledged" : true,
  "insertedIds" : {
    "0" : 3.0,
    "1" : 4.0,
    "2" : 5.0,
    "3" : 6.0
  }
}
```

4.1 Consultar documentos con un valor en un array

Consulta documentos donde el array class contiene el valor 10.

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

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


5.1 Inserción de un documento simple

Inserta un documento simple con solo el campo name.

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

```
▼ {
  "acknowledged" : true,
  "insertedId" : ObjectId("6710141cd23b82644c072f8a")
}
```

6.1 Inserción de documento con subdocumento

Inserta otro documento con más campos, incluyendo un subdocumento en information.

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

```
▼ {
  "acknowledged" : true,
  "insertedId" : ObjectId("6710143fd23b82644c072f8b")
}
```

7.1 Mostrar todos los documentos en LabItp02

Muestra todos los documentos en la colección LabItp02. db.LabItp02.find()

```
▼ {
  "_id" : ObjectId("6710141cd23b82644c072f8a"),
  "name" : "carolita"
}
▼ {
  "_id" : ObjectId("6710143fd23b82644c072f8b"),
  "name" : "carolita",
  "information" : {
    "classroom" : "room_01",
    "locker" : NumberInt(12)
  },
  "age" : NumberInt(25)
}
```

3. Operaciones de Actualización (UPDATE)

1.1 Actualizar un documento

Actualiza el campo virtues del documento con _id: 7.

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

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

```
{  
  "_id" : NumberInt(7),  
  "name" : "maña",  
  "phone" : NumberInt(654789),  
  "class" : [  
    NumberInt(10),  
    NumberInt(12),  
    NumberInt(15)  
  ],  
  "virtues" : [  
    "cheerful",  
    "funny",  
    "comprehensive",  
    "sociable",  
    "respectful"  
  ]  
}
```

2.1 Actualizar múltiples campos

Actualiza los campos information y age del documento con _id: 7.

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

```
{  
  "_id" : NumberInt(7),  
  "name" : "maña",  
  "phone" : NumberInt(654789),  
  "class" : [  
    NumberInt(10),  
    NumberInt(12),  
    NumberInt(15)  
  ],  
  "virtues" : [  
    "cheerful",  
    "funny",  
    "comprehensive",  
    "sociable",  
    "respectful"  
  ],  
  "age" : NumberInt(18),  
  "information" : {  
    "classroom" : "room_A",  
    "locker" : NumberInt(15)  
  }  
}
```

3.1 Actualizar con fecha de modificación

Actualiza virtues y añade/modifica la fecha lastModified a la fecha actual para el documento con _id: 7.

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

```
{  
  "_id" : NumberInt(7),  
  "name" : "maña",  
  "phone" : NumberInt(654789),  
  "class" : [  
    NumberInt(10),  
    NumberInt(12),  
    NumberInt(15)  
  ],  
  "virtues" : [  
    "cheerful",  
    "funny",  
    "comprehensive",  
    "sociable",  
    "respectful"  
  ],  
  "age" : NumberInt(18),  
  "information" : {  
    "classroom" : "room_A",  
    "locker" : NumberInt(15)  
  },  
  "lastModified" : ISODate("2024-10-16T19:40:44.517+0000")  
}
```

4.1 Actualización con upsert

Si no existe un documento con _id: 10, lo inserta; de lo contrario, lo actualiza con los nuevos datos.

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

```
{  
  "_id" : NumberInt(10),  
  "age" : NumberInt(19),  
  "information" : {  
  },  
  "lastModified" : ISODate("2024-10-16T19:41:56.812+0000"),  
  "name" : "Joan",  
  "virtues" : [  
  ]  
}
```

5.1 Actualización de múltiples documentos

Actualiza los documentos con `_id` de 1 a 6, agregando el campo `virtues` con un único valor.

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

```
{
  "_id" : NumberInt(1),
  "name" : "pepe",
  "phone" : NumberInt(123456),
  "class" : [
    NumberInt(20),
    NumberInt(22),
    NumberInt(25)
  ]
}
{
  "_id" : NumberInt(2),
  "name" : "juanito",
  "phone" : NumberInt(654789),
  "class" : [
    NumberInt(10),
    NumberInt(12),
    NumberInt(15)
  ]
}
```

```
{
  "_id" : NumberInt(1),
  "name" : "pepe",
  "phone" : NumberInt(123456),
  "class" : [
    NumberInt(20),
    NumberInt(22),
    NumberInt(25)
  ],
  "virtues" : [
    "cheerful"
  ]
}
```

6.1 Actualizar todos los documentos

Actualiza todos los documentos de la colección añadiendo el campo `status` con valor 'A'.

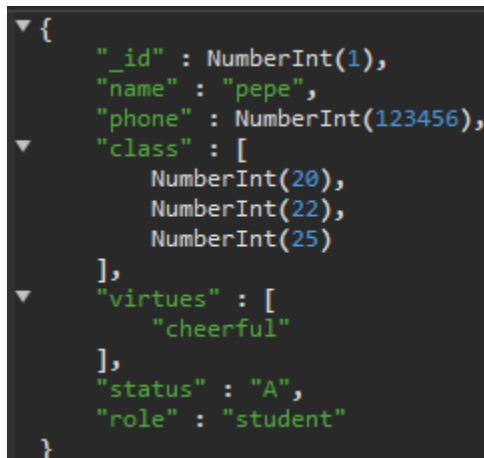
```
db.LabItp01.updateMany( {}, { $set: { status: 'A' } })
```

```
{
  "_id" : NumberInt(1),
  "name" : "pepe",
  "phone" : NumberInt(123456),
  "class" : [
    NumberInt(20),
    NumberInt(22),
    NumberInt(25)
  ],
  "virtues" : [
    "cheerful"
  ],
  "status" : "A"
}
```

7.1 Actualización por nombre

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

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



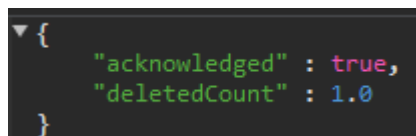
```
{
  "_id" : NumberInt(1),
  "name" : "pepe",
  "phone" : NumberInt(123456),
  "class" : [
    NumberInt(20),
    NumberInt(22),
    NumberInt(25)
  ],
  "virtues" : [
    "cheerful"
  ],
  "status" : "A",
  "role" : "student"
}
```

4. Operaciones de Eliminación (DELETE)

1.1 Eliminar un documento

Elimina un documento en la colección LabItp01 donde el campo name es igual a "carlito".

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



```
{
  "acknowledged" : true,
  "deletedCount" : 1.0
}
```

2.1 Eliminar un documento en grades

Elimina el primer documento en la colección grades donde el campo student_id es igual a 0.

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

```
{
  "acknowledged" : true,
  "deletedCount" : 1.0
}
```

3.1 Eliminar múltiples documentos

Elimina todos los documentos en la colección grades donde el campo student_id es igual a 0.

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

```
{
  "acknowledged" : true,
  "deletedCount" : 1.0
}
```

4.1 Remove un solo documento

Elimina un solo documento en la colección grades donde el campo student_id es igual a 1 (similar a deleteOne).

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

```
{
  "_id" : "50b59cd75bed76f46522c359",
  "student_id" : NumberInt(1),
  "class_id" : NumberInt(18),
  "scores" : [...]
}
{
  "_id" : "50b59cd75bed76f46522c35a",
  "student_id" : NumberInt(1),
  "class_id" : NumberInt(22),
  "scores" : [...]
}
{
  "_id" : "50b59cd75bed76f46522c35b",
  "student_id" : NumberInt(1),
  "class_id" : NumberInt(28),
  "scores" : [...]
}
{
  "_id" : "50b59cd75bed76f46522c35a",
  "student_id" : NumberInt(1),
  "class_id" : NumberInt(22),
  "scores" : [...]
}
{
  "_id" : "50b59cd75bed76f46522c35b",
  "student_id" : NumberInt(1),
  "class_id" : NumberInt(28),
  "scores" : [...]
}
```

5.1 Remover todos los documentos

Elimina todos los documentos en la colección grades donde el campo student_id es igual a 1 (similar a deleteMany).

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

```
{  
  "acknowledged" : true,  
  "deletedCount" : 4.0  
}
```

6.1 Vaciar la colección

Elimina todos los documentos en la colección grades sin condición, es decir, vacía la colección.

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

```
▼ {  
  "acknowledged" : true,  
  "deletedCount" : 49.0  
}
```

7.1 Eliminar una colección

Elimina completamente la colección grades, incluyendo sus índices y metadatos.

```
db.grades.drop(): true
```

Análisis y Discusión

El diseño de la base de datos NoSQL para la librería se centra en la eficiencia mediante el uso de referencias cruzadas y embeber información cuando es necesario. Las relaciones uno a muchos y muchos a muchos se maneja con referencias, reduciendo la duplicación de datos. Las consultas son rápidas gracias al embebido en perfiles y listas dentro de pedidos, optimizando el rendimiento sin sacrificar flexibilidad.

Conclusiones

- **Normalización eficiente:** El uso de referencias minimiza la duplicación y facilita actualizaciones.
- **Escalabilidad:** El diseño permite crecimiento sin reestructuración.
- **Consultas rápidas:** Las consultas comunes son eficientes, gracias a la estructura adecuada.
- **Reducción de redundancia:** Evita errores y facilita el mantenimiento.

Referencias

MongoDB, Inc. (2024). MongoDB documentation. Retrieved from <https://docs.mongodb.com/>

Guerrero, M. C. L. (2024). Librería NoSQL [GitHub repository]. Retrieved from <https://github.com/mclguerrero/LibreriaNoSQL.git>