**BASES DE DATOS NoSQL**

| **Información general** | |
| --- | --- |
| **Duración estimada en minutos:** | 60 |
| **Docente:** | Carlos Andrés Flórez Villarraga |
| **Guía no.** | 05 |

| **Información de la Guía** |
| --- |

**OBJETIVO**

Conocer los conceptos básicos de las bases de datos NoSQL.

**CONCEPTOS BÁSICOS**

Lógica de programación, JSON, línea de comandos.

**CONTEXTUALIZACIÓN TEÓRICA**

SQL (Structured Query Language) y NoSQL (Not Only SQL) son dos enfoques diferentes para el manejo y almacenamiento de datos en sistemas de bases de datos. Aquí hay una explicación breve de cada uno:

**SQL (Structured Query Language):**

* Tipo de Datos: SQL es un lenguaje de consulta estructurado diseñado para gestionar y manipular datos en bases de datos relacionales.
* Estructura: Utiliza un esquema predefinido y tablas con relaciones entre ellas. Cada fila en una tabla representa un registro y cada columna representa un atributo.
* Ejemplos de Bases de Datos SQL: MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, SQLite.

**NoSQL (Not Only SQL):**

* Tipo de Datos: NoSQL es un término genérico que se refiere a cualquier sistema de gestión de bases de datos que no sigue el modelo relacional de SQL.
* Estructura: Puede tener una estructura más flexible, permitiendo almacenar y recuperar datos de formas no tabulares, como documentos, grafos o clave-valor.
* Escalabilidad: Muchas bases de datos NoSQL están diseñadas para ser altamente escalables y distribuibles en entornos de gran escala.
* Ejemplos de Bases de Datos NoSQL: MongoDB (document-oriented), Cassandra (wide-column store), Redis (key-value store), Neo4j (graph database).

**Diferencias clave:**

* Modelo de Datos: SQL utiliza un modelo relacional con tablas y relaciones, mientras que NoSQL ofrece varios modelos, como documentos, grafos, clave-valor o columnares.
* Estructura de Esquema: SQL tiene un esquema fijo y riguroso que debe definirse antes de insertar datos. NoSQL es más flexible y puede adaptarse a cambios en la estructura de datos sin modificar el esquema.
* Escalabilidad: Muchas bases de datos NoSQL están diseñadas para ser fácilmente escalables horizontalmente, permitiendo crecer en tamaño distribuyendo la carga en varios nodos.
* Transacciones: SQL suele ofrecer transacciones ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), que garantizan la integridad de los datos. Algunas bases de datos NoSQL sacrifican ciertos aspectos de ACID para lograr una mayor escalabilidad y rendimiento en entornos distribuidos.

La elección entre SQL y NoSQL depende de los requisitos específicos del proyecto, como la naturaleza de los datos, el volumen de datos, la complejidad de las consultas y la escalabilidad deseada. Ambos enfoques tienen sus pros y contras, y la elección adecuada dependerá de las necesidades y objetivos específicos de cada aplicación o sistema.

**MongoDB**

MongoDB es una base de datos NoSQL orientada a documentos que utiliza un formato de almacenamiento de datos llamado BSON (Binary JSON). En MongoDB, los datos se almacenan en documentos BSON, que son estructuras de datos similares a JSON, pero binarias.

| **Procedimiento** |
| --- |

Una tienda requiere un sistema de información que le permita gestionar la información de sus productos, clientes y transacciones, de cada producto se debe almacenar su código, nombre, precio, unidades y tipo; de cada transacción se debe registrar su código, el cliente que realiza la compra, el (o los) productos que compra, fecha de compra y pago; del pago se debe registrar su fecha, total pagado, estado del pago y método de pago; por último, del cliente se requiere su cédula, nombre, email y teléfono.

Asegúrese que el servidor de MongoDB (mongod) está en ejecución. Abra MongoDB Compass, conéctese al servidor local, y en la consola de mongosh ejecute los siguientes comandos:

1. Vamos a crear la base de datos tienda, así:

| use tienda |
| --- |

1. Agregue dos productos, para eso escriba lo siguiente:

| db.productos.insertOne({ codigo: "P001", nombre: "Portátil ASUS", precio: 1780000, unidades: 20, tipo: "Electrónica" }) |
| --- |

| db.productos.insertOne({ codigo: "P002", nombre: "Camiseta Adidas M", precio: 96000, unidades: 10, tipo: "Ropa" }) |
| --- |

Observe que los datos se guardan por medio de la función insertOne en la colección productos, las colecciones son el equivalente a las tablas en SQL. Cada producto guardado en dicha colección se le conoce como documento. Cada documento es un producto individual, sus propiedades se definen por medio de clave-valor separados por coma. Este formato se conoce como JSON. También se pueden guardar varios productos haciendo uso de la función insertMany.

Para más información:

* <https://www.mongodb.com/docs/manual/reference/method/db.collection.insertOne/>
* <https://www.mongodb.com/docs/manual/reference/method/db.collection.insertMany/>

1. Consulte todos los productos que existen en la colección. Así:

| db.productos.find() |
| --- |

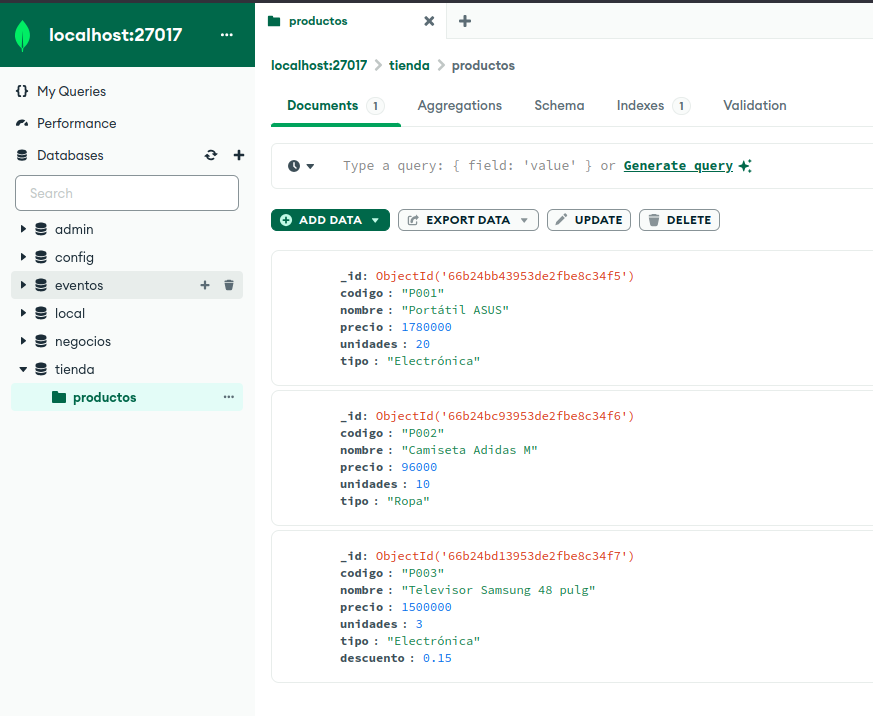
Cada documento tiene una propiedad que se agrega automáticamente \_id, la cual representa un identificador único. Las colecciones son muy flexibles, por ejemplo, si agregamos un nuevo producto con un campo que los otros productos no tienen, simplemente sería hacer algo así:

| db.productos.insertOne({ codigo: "P003", nombre: "Televisor Samsung 48 pulg", precio: 1500000, unidades: 3, tipo: "Electrónica", descuento: 0.15 }) |
| --- |

Consulte todos los productos nuevamente. Observe que el esquema se adapta a los datos de cada documento.

| db.productos.find() |
| --- |

1. En MongoDB Compass actualice las bases de datos (con el botón refresh), observe que se creó la base de datos **tienda** con la colección productos y allí están los tres productos agregados.



1. Para actualizar los productos que ya están guardados, podemos hacer uso de la función updateOne así:

| db.productos.updateOne({ codigo: "P001" }, { $set: { nombre: "Nuevo nombre" } }) |
| --- |

En este ejemplo:

db.productos: Hace referencia a la colección "productos".

updateOne: Indica que estamos actualizando un solo documento.

{ codigo: "P001" }: Es el filtro que identifica el documento que se desea actualizar. En este caso, se selecciona el producto con el código "P001", aunque lo ideal es usar el \_id.

{ $set: { nombre: "Nuevo nombre" } }: Indica la actualización que se desea realizar. En este caso, se utiliza el operador $set para establecer el nuevo valor del campo nombre como "Nuevo Nombre".

Si se desea actualizar más de un documento, podemos utilizar updateMany con un filtro adecuado.

Para más información:

* <https://www.mongodb.com/docs/manual/reference/method/db.collection.updateOne/>
* <https://www.mongodb.com/docs/manual/reference/method/db.collection.updateMany/>

1. Para eliminar un producto podemos utilizar el siguiente comando:

| db.productos.deleteOne({ codigo: "P001" }) |
| --- |

En este ejemplo:

db.productos: Hace referencia a la colección "productos".

deleteOne: Indica que estamos eliminando un solo documento.

{ codigo: "P001" }: Es el filtro que identifica el documento que deseas eliminar. En este caso, se selecciona el producto con el código "P001". Aunque lo ideal es usar el \_id.

Si queremos eliminar más de un documento que cumple con ciertos criterios, podemos usar deleteMany. Por ejemplo, para eliminar todos los productos con un precio menor a 1000:

| db.productos.deleteMany({ precio: { $gt: 1000 } }) |
| --- |

Se debe tener en cuenta que estos comandos son irreversibles, y la eliminación de documentos es una operación importante. Debemos asegurarnos de tener una copia de seguridad o confirmar que estamos eliminando los documentos correctos antes de ejecutar estos comandos en un entorno de producción.

Para más información:

* <https://www.mongodb.com/docs/manual/reference/method/db.collection.deleteOne/>
* <https://www.mongodb.com/docs/manual/reference/method/db.collection.deleteMany/>

1. Agregue tres clientes en la colección clientes. Haga que uno de los clientes se llame Juan Perez y tenga la cédula 123456789.
2. Ahora, agregaremos una transacción, podría quedar así:

| db.transacciones.insertOne({  codigo: "T001",  cedulaCliente: "123456789",  productos: [  {  codigo: "P002",  cantidad: 2,  precio: 1200  }  // Podemos agregar más productos a la transacción separándolos por coma  ],  fechaCompra: new Date("2024-01-09T12:00:00Z"),  pago: {  fecha: new Date("2024-01-09T12:30:00Z"),  totalPagado: 2400.00,  estado: "Pagado",  metodoPago: "Tarjeta de crédito"  }  }) |
| --- |

La transacción contiene la información clave del cliente y una lista de productos, se sabe que es una lista porque se hace uso de corchetes. Adicionalmente, la transacción tiene un pago con los datos relacionados a él. En este escenario los documentos son independientes entre sí. Observe que la “relación” entre la transacción y el cliente se da por medio de la cédula (o el \_id), los datos están desacoplados, no como en SQL que las relaciones son más estrictas.

**NOTA:** Las inserciones, actualizaciones y eliminación de documentos y colecciones se pueden hacer también desde la GUI de MongoDB Compass.

1. Agregue otra transacción con dos productos.
2. Para realizar consultas a los datos, podemos hacer uso de findOne(), find(), count(), etc. Estas funciones tienen filtros para buscar documentos dados los atributos que se requieran. Por ejemplo, obtener todos los datos de un cliente dada su cédula:

| db.clientes.findOne({ cedula: "123456789" }); |
| --- |

Para obtener un producto dado su \_id (se debe usar ObjectId), cambie el id por uno de su colección de productos:

| db.productos.findOne( { \_id: ObjectId('66b972a6f6bbb37e8ce7c79e') } ) |
| --- |

Por ejemplo, obtener todos los productos que tengan el nombre Televisor, el nombre del producto debe ser exactamente "Televisor":

| db.productos.find({ nombre: "Televisor" }); |
| --- |

Para obtener todos los productos que contengan el nombre Televisor, el nombre del producto debe contener la palabra "Televisor", el valor i en $options se refiere a que es indiferente a las mayúsculas y minúsculas:

| db.productos.find({ nombre: { $regex: "televisor", $options: 'i' } }); |
| --- |

Para obtener todos los productos cuyo precio sea mayor a $980000:

| db.productos.find({ precio: { $gt: 980000 } }); |
| --- |

Para conocer los operadores más importantes en MongoDB, los invito a leer lo siguiente:

* <https://kinsta.com/es/blog/operadores-mongodb/>
* <https://www.mongodb.com/docs/manual/reference/operator/query/>

Para conocer más funciones que tiene MongoDB para consultar y manipular datos:

* <https://www.mongodb.com/docs/manual/reference/method/js-collection/>

1. Para realizar consultas más complejas en MongoDB, se debe utilizar la función aggregate. Su funcionamiento se basa en pipelines, que son secuencias de etapas que procesan los documentos. Cada etapa en el pipeline realiza una operación específica, como filtrar, agrupar o ordenar datos, y el resultado de una etapa se pasa a la siguiente. Esto permite construir consultas muy sofisticadas y obtener resultados detallados y personalizados, optimizando así la eficiencia en la manipulación y análisis de grandes volúmenes de datos.

Por ejemplo, en esta consulta obtenemos todos los productos cuyo precio sea menor a $1500000, luego los ordenamos alfabéticamente por el nombre (el 1 es ascendente) y por último quitamos el \_id y las unidades de la respuesta.

| db.productos.aggregate(  { $match: { precio: { $lte: 1500000 } } },  { $sort: { nombre: 1 } },  { $project: { \_id: 0, unidades: 0 } }  ); |
| --- |

Por ejemplo, en esta otra consulta obtenemos todas las transacciones junto con la información personal del cliente, pero observe que se están ignorando algunos campos, entre ellos el \_id y cédula del cliente, y el método de pago.

| db.transacciones.aggregate(  { $lookup: { from: "clientes", localField: "cedulaCliente", foreignField: "cedula", as: "infoCliente" } },  { $project : { "infoCliente.\_id": 0, "infoCliente.cedula": 0, "pago.metodoPago": 0 } }  ); |
| --- |

Se recomienda leer la siguiente documentación:

* <https://www.mongodb.com/docs/manual/aggregation/>
* <https://www.mongodb.com/docs/manual/reference/operator/aggregation-pipeline/#std-label-aggregation-pipeline-operator-reference>

| **Tarea** |
| --- |

1. Cree una base de datos en MongoDB para el siguiente ejercicio: Una biblioteca requiere un sistema de información que le permita gestionar la información de sus libros y préstamos, de cada libro se debe almacenar la información de su autor (o autores), isbn, nombre, género, año, unidades; de cada préstamo se debe registrar el id de la persona que pide el préstamo, los id de los libros que presta, fecha del préstamo, fecha de devolución; por último, de la persona se requiere la cédula, nombre, email y teléfono, y del autor su nombre y nacionalidad.
2. Use MongoDB Compass para ir mirando cómo se van creando las colecciones y documentos en la base de datos.