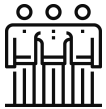
MongoDB

Marco Robles Pulido

Academy Code of Conduct

Seamos respetuosos, no existen preguntas malas



Seamos pacientes

Cuidemos nuestro lenguaje

Objetivos de la sesión

Al final de la sesión seremos capaces de:

- Levantar una instancia de MongoDB
- Definir modelos y esquemas
- Realizar operaciones CRUD

Tabla de contenido

SQL

SQL, es el lenguaje utilizado en los sistemas de gestión de bases de **datos relacionales** (**RDBMS**) desde la década de 1970.

Es utilizado para consultar **bases de datos relacionales**, en las que los datos se **almacenan** en **filas y tablas** vinculadas de diversas maneras.

NoSQL

NoSQL son **bases de datos no relacionales**, lo que significa que permiten **estructuras diferentes** a las de una base de datos **SQL** (no filas y columnas) y más flexibilidad para utilizar el formato que mejor se adapte a los datos.

No significa que los sistemas no utilicen SQL, ya que las bases de datos NoSQL a veces admiten algunos comandos SQL. Más exactamente, "NoSQL" se define a veces como "no sólo SQL".

	SQL	NoSQL
Cómo funciona	Manejan **datos estructurados**, o datos que tienen **relaciones** entre sus **variables y entidades**.	Permiten **diferentes estructuras de datos**. Hay menos necesidad de planificar y organizar previamente los datos, y es más fácil hacer modificaciones.
Escalabilidad	Pueden escalar ***verticalmente*** migrando a un servidor más grande. También pueden escalar *horizontalmente* a través de la lógica de fragmentación o partición.	Escalan mejor ***horizontalmente***, lo que significa que se pueden añadir servidores o nodos adicionales según sea necesario para aumentar la carga.
Estructura	Organiza los datos de forma **relacional y tabular**, utilizando tablas con columnas o atributos y filas de registros.	Generalmente se dividen en uno de los cuatro tipos de estructuras: ● Orientado a columnas ● Tiendas de clave-valor ● Grafos ● **Almacén de Documentos**

	SQL	NoSQL
Propiedades	Deben presentar **cuatro propiedades**, conocidas por el acrónimo ACID, con el fin de garantizar que las transacciones se procesen con éxito y que la base de datos tenga un alto nivel de fiabilidad. • (A) Atomicidad • (C) Consistencia • (I) Aislamiento • (D) Durabilidad	NoSQL sigue la teoría **CAP** que dice que los sistemas de datos distribuidos permiten una compensación que puede garantizar sólo dos de las siguientes tres propiedades: • (C) Consistencia • (A) Disponibilidad • (P) Tolerancia a la partición

Tabla de contenido

Documentos

MongoDB almacena los registros de datos como documentos BSON.

BSON es una representación binaria de los documentos JSON, aunque contiene más tipos de datos que JSON. Los documentos se componen de pares atributo-valor y tienen la siguiente estructura:

Documentos

El valor de un campo puede ser cualquiera de los tipos de datos de BSON[1], incluyendo otros documentos, matrices y matrices de documentos. Por ejemplo, el siguiente documento contiene valores de distintos tipos:

```
{
   _id: ObjectId("5099803df3f4948bd2f98391"),
   name: { first: "Alan", last: "Turing" },
   birth: new Date('Jun 23, 1912'),
   death: new Date('Jun 07, 1954'),
   contribs: [ "Turing machine", "Turing test", "Turingery" ],
   views : NumberLong(1250000)
}
```

[1] https://www.mongodb.com/docs/manual/reference/bson-types/

Colecciones

MongoDB almacena los documentos en colecciones, las cuales se encuentran a su vez almacenadas en bases de datos.

Una base de datos puede tener una o más colecciones.

Tabla de contenido

Instalación directa

Linux	macOS	Windows
• `sudo apt-get update` • `sudo apt-get install gnupg` • `sudo apt-get install -y mongodb-org` • `sudo apt-get install -y mongodb-mongosh`	Ejecutar los siguientes comandos: • `xcode-select --install` • `brew tap mongodb/brew` • `brew update` • `brew install mongodb-community@6.0` • `brew install mongosh`	Descargar el instalador del cliente desde https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-windows-unattended/ Descargar el instalador del shell desde https://www.mongodb.com/try/download/shell?jmp=docs

Instalación con Docker

- docker run --name mongodb -d mongo

```
❯ docker run --name mongodb -d mongo
Unable to find image 'mongo:latest' locally
latest: Pulling from library/mongo
675920708c8b: Pull complete
6f9c8c301e0f: Pull complete
73738965c4ce: Pull complete
7fd6635b9ddf: Pull complete
73a471eaa4ad: Pull complete
bcf274af89b0: Pull complete
04fc489f2a3b: Pull complete
6eff8a505292: Pull complete
a5ef4431fce7: Pull complete
Digest: sha256:2b527cb8eafb4a97a896eb05a2e88c75f0033d4be4bf94fad205299659147c13
Status: Downloaded newer image for mongo:latest
ecfab3dc92c40fcdd5a7c9db8d0cbc35bbd4d17e4190fed3c3ab2024e58a3a00
```

- docker exec -it mongodb mongosh

Bases de datos

- Por defecto, mongodb crea tres bases de datos
- El comando *show dbs* muestra todas las bases de datos en el servidor

```
test> show dbs
admin   40.00 KiB
config  60.00 KiB
local   72.00 KiB
```

- Para comenzar a trabajar con una base de datos se puede utilizar el comando *use*

El comando *USE*

El comando creará una nueva base de datos si no existe, de lo contrario devolverá la base de datos existente.

```
test> use wizeline_baz_db
switched to db wizeline_baz_db
wizeline_baz_db> show dbs
admin   40.00 KiB
config  92.00 KiB
local   72.00 KiB
wizeline_baz_db> db
wizeline_baz_db
wizeline_baz_db>
```

Para comprobar la base de datos seleccionada actualmente existe el comando *db*

¿Y la nueva base de datos?

La nueva base de datos (*wizeline_baz_db*) no está presente en la lista al usar el comando *show dbs*.

Para mostrar la base de datos, necesita insertar al menos un documento en ella.

```
wizeline_baz_db> db.proyecto.insertOne({"nombre":"workshop"})
{
  acknowledged: true,
  insertedId: ObjectId("63159357ad587135ee4a0538")
}
wizeline_baz_db> show dbs
admin            40.00 KiB
config           92.00 KiB
local            72.00 KiB
wizeline_baz_db   8.00 KiB
wizeline_baz_db>
```

Tabla de contenido

Insertar documentos

Insertar un solo documento

```
wizeline_baz_db> db.proyectos.insertOne(
... {
..... nombre: "single insert",
..... tipo: "single",
..... cantidad: 1
..... }
... )
{
  acknowledged: true,
  insertedId: ObjectId("631595faad587135ee4a053a")
}
wizeline_baz_db>
```

Insertar varios documentos

```
wizeline_baz_db> db.proyectos.insertMany([
... {
..... nombre: "Proyecto multiple",
..... responsables: ["CTO", "TL"],
..... tipo: "multiple"
..... },
... {
..... nombre: "Proyecto cancelado"
..... }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("631596c5ad587135ee4a053b"),
    '1': ObjectId("631596c5ad587135ee4a053c")
  }
}
wizeline_baz_db>
```

Consulta de documentos

El comando *find* permite consultar documentos en una colección.

- db.proyectos.find()
- db.proyectos.find({“cantidad”: 1})

```
wizeline_baz_db> db.proyectos.find()
[
  {
    _id: ObjectId("631595faad587135ee4a053a"),
    nombre: 'single insert',
    tipo: 'single',
    cantidad: 1
  },
  {
    _id: ObjectId("631596c5ad587135ee4a053b"),
    nombre: 'Proyecto multiple',
    responsables: [ 'CTO', 'TL' ],
    tipo: 'multiple'
  },
  {
    _id: ObjectId("631596c5ad587135ee4a053c"),
    nombre: 'Proyecto cancelado'
  }
]
```

Operadores de selección:

https://www.mongodb.com/docs/manual/reference/operator/query/#query-selectors

Consulta de documentos *(AND / OR)*

Una consulta compuesta puede especificar condiciones para más de un campo de los documentos de la colección.

- AND:
  - db.proyectos.find({"cantidad": 1, "tipo": "single"})
- OR:
  - db.proyectos.find({$or: [{"cantidad": 1}, {"tipo": "multiple"}]})

Actualizar documentos

El shell de MongoDB proporciona los siguientes métodos para actualizar los documentos de una colección:

- Para actualizar un solo documento: *db.collection.updateOne()*
- Para actualizar varios documentos: *db.collection.updateMany()*
- Para reemplazar un documento: *db.collection.replaceOne()*

Actualizar un solo documento

Para actualizar el primer documento en la colección “proyectos” donde la cantidad sea igual a 1, se utiliza el comando *updateOne*, y para indicar el campo o campos a modificar, se utiliza el operador *set*.

```
wizeline_baz_db> db.proyectos.updateOne({cantidad: 1},
... {
..... $set: {
....... tipo: "prueba"
....... }
..... })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
wizeline_baz_db>
```

Actualizar varios documentos

El comando *db.collection.updateMany()* actualiza todos los documentos que coincidan con el filtro especificado.

```
wizeline_baz_db> db.proyectos.updateMany(
... {presupuesto: {$ne: 100} },
... {$set: {presupuesto: 100} })
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 3,
  modifiedCount: 3,
  upsertedCount: 0
}
wizeline_baz_db>
```

El operador $*ne* permite filtrar por resultados con *presupuesto* diferente de 100, al no existir documentos con *presupuesto*, a todos los documentos se les agrega.

Reemplazar un documento

Para reemplazar todo el contenido de un documento excepto el campo *_id*, hay que pasar un documento completamente nuevo como segundo argumento al comando *db.collection.replaceOne()*

```
wizeline_baz_db> db.proyectos.replaceOne(
... {nombre: 'Proyecto cancelado'},
... {nombre: 'CANCELADO', fecha: '12-09-2016'})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
wizeline_baz_db>
```

Eliminar documentos

El shell de MongoDB proporciona los siguientes métodos para eliminar documentos de una colección:

- Para eliminar varios documentos, utilice db.collection.deleteMany()
- Para eliminar un solo documento, utilice db.collection.deleteOne()

```
wizeline_baz_db> db.proyectos.deleteMany({})
{ acknowledged: true, deletedCount: 3 }
wizeline_baz_db> db.proyectos.find()

wizeline_baz_db>
```

Datos de ejemplo

Sakila es un esquema de ejemplo de MySQL que se publicó hace algunos años. Se basa en un sistema de alquiler de DVD.

En el repositorio se agregaron los datos publicados en esa base de datos.

Para importar estos datos se pueden utilizar los siguientes comandos:

- *docker cp sakila/<nombre-del-archivo>.json mongodb:/tmp/<nombre-del-archivo>.json*
- *docker exec mongodb mongoimport -d <nombre-de-db> -c <nombre-de-la-coleccion> --file /tmp/<nombre-del-archivo>.json*

Tabla de contenido

Modelos

El reto clave en el modelado de datos es equilibrar las necesidades de la aplicación, las características de rendimiento del motor de la base de datos y los patrones de recuperación de datos.

Al diseñar los modelos de datos, siempre hay que tener en cuenta el uso que la aplicación hace de los datos, así como la estructura inherente de los propios datos.

MongoDB proporciona dos tipos de modelos de datos: - Modelo de datos embebido y Modelo de datos normalizado.

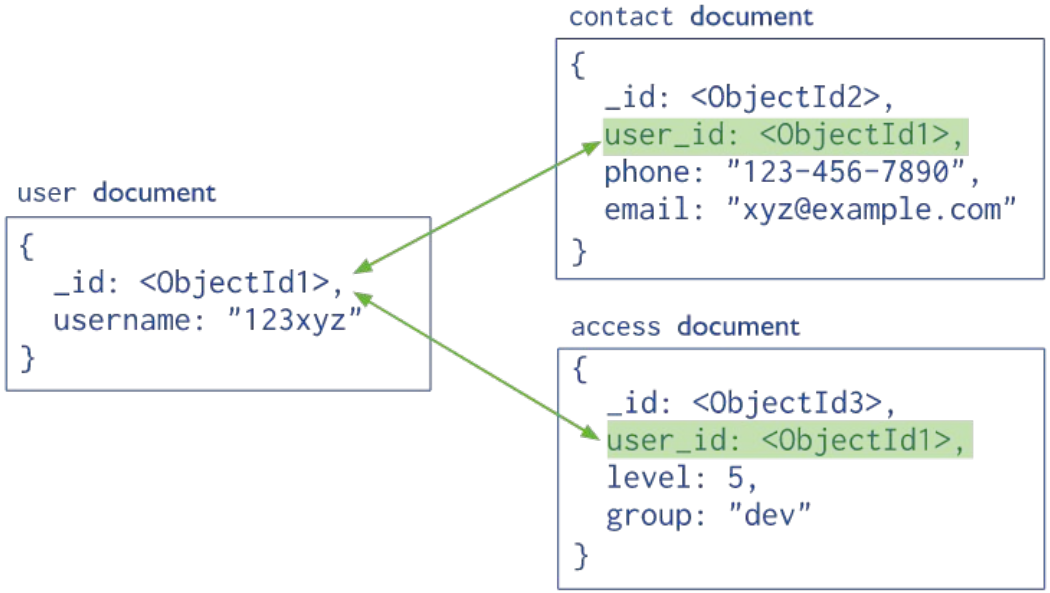
Modelo de datos embebido

Con MongoDB, puede incrustar datos relacionados en una única estructura o documento. Estos esquemas se conocen generalmente como modelos "desnormalizados".

Permiten a las aplicaciones almacenar información relacionada en el mismo registro de la base de datos.

Modelo de datos normalizado

Los modelos de datos normalizados describen las relaciones mediante referencias entre documentos.



Modelos

Embebido	Normalizado
● Proporciona un mejor rendimiento para las operaciones de lectura ● Capacidad de solicitar y recuperar datos relacionados en una sola operación ● Permiten actualizar los datos relacionados en una única operación de escritura atómica	● Usar cuando la incrustación daría lugar a la duplicación de datos, pero no proporcionaría suficientes ventajas de rendimiento de lectura para compensar las implicaciones de la duplicación. ● Para modelar grandes conjuntos de datos jerárquicos.

Esquema

Un esquema es un objeto JSON que define la estructura y el contenido de sus datos.

Los esquemas representan tipos de datos en lugar de valores específicos. Estos incluyen tipos de datos que puedes combinar para crear esquemas que representen tipos de objetos personalizados.

Esquema

```
{
  "title": "car",
  "required": [
     "_id",
     "year",
     "make",
     "model",
     "miles"
  ],
  "properties": {
    "_id": { "bsonType": "objectId" },
    "year": { "bsonType": "string" },
    "make": { "bsonType": "string" },
    "model": { "bsonType": "string" },
    "miles": { "bsonType": "number" }
  }
}
```

Tiempo de ejercicio

Datos de ejemplo

```
❯ docker cp sakila/customers.json mongodb:/tmp/customers.json
❯ docker cp sakila/films.json mongodb:/tmp/films.json
❯ docker cp sakila/stores.json mongodb:/tmp/stores.json
❯ docker exec mongodb mongoimport -d sakila -c customers --file /tmp/customers.json
2022-09-06T22:11:58.505+0000    connected to: mongodb://localhost/
2022-09-06T22:11:58.753+0000    599 document(s) imported successfully. 0 document(s) failed to import.
❯ docker exec mongodb mongoimport -d sakila -c films --file /tmp/films.json
2022-09-06T22:12:18.663+0000    connected to: mongodb://localhost/
2022-09-06T22:12:18.723+0000    1000 document(s) imported successfully. 0 document(s) failed to import.
❯ docker exec mongodb mongoimport -d stores -c films --file /tmp/stores.json
2022-09-06T22:12:36.381+0000    connected to: mongodb://localhost/
2022-09-06T22:12:36.413+0000    2 document(s) imported successfully. 0 document(s) failed to import.
```

docker exec mongodb mongoexport --db sakila --collection customers --type=json --out=/tmp/<nombre_coleccion_export>.json

docker cp mongodb:/tmp/<nombre_coleccion_export>.json ./sakila/

***Entregables**: **3 impresiones de pantalla** (find con query selector, update con query selector y un insert de las 3 colecciones (usar 1 colección por cada **CRUD**) y los **exports de las colecciones modificadas** (customers, films, stores)*

Feedback Form

Let us know your feedback!

https://forms.gle/WKtc8wZeSxWnjGo8A