

BASE DE DATOS NOSQL

¿Qué es MongoDB?

MongoDB es un sistema **gestor de bases de datos no relacionales**, esto es, un sistema para implementar persistencia de información a largo plazo que no se basa en el modelo relacional ni utiliza SQL como lenguaje de consultas.

¿Cuándo necesitamos bases de datos NoSQL?

Una vez que decidimos utilizar una base de datos, la siguiente decisión radica en el **tipo de motor**

de base de datos. Actualmente en el mercado tenemos dos grandes alternativas:

1. **Motores de bases de datos SQL:** siguen un mismo estándar común, tienen estructuras de datos estáticas e invierten recursos en restricciones. Aquí tenemos, entre otros, a MySQL, SQL Server, PostgreSQL, SQLite.

2. **Motores de bases de datos NoSQL:** no siguen un estándar común definido, se basan en estructuras dinámicas y maleables, e invierten la mínima cantidad posible de recursos en restricciones. Aquí tenemos, entre otros, a MongoDB, Redis, Cassandra.

El primer criterio a analizar es el uso de un **lenguaje de consultas común**.

SQL	NoSQL
<ul style="list-style-type: none">• Estos motores de bases de datos comparten un lenguaje común: SQL.• Al compartir un lenguaje común son más fácilmente intercambiables entre sí.• Puedo disponer de un grupo que desarrolle en MySQL y poder migrar con relativa facilidad a otro motor de bases de datos SQL como PostgreSQL.	<ul style="list-style-type: none">• Cada motor de base de datos tiene su propia sintaxis y su propio lenguaje de consulta.• Por ejemplo: MongoDB se basa en JavaScript mientras que Cassandra se basa en una adaptación de SQL.• Al no haber un estándar común, se vuelve más difícil migrar de un motor a otro.

El segundo criterio a analizar es si usa **estructuras de datos fijas o dinámicas**.

SQL	NoSQL
<ul style="list-style-type: none">• Se basa en estructuras de datos fijas, esto es, reglas estrictas sobre los datos que contiene determinada estructura.• Una tabla SQL es una estructura que tiene un conjunto fijo de campos.• No se admiten registros con más campos que los definidos.• Si hay un registro que use menos campos que los definidos, éstos se rellenan con <i>NULL</i> (espacio perdido).	<ul style="list-style-type: none">• Se basa en estructuras de datos dinámicas y maleables.• Desde la perspectiva NoSQL se elimina toda lógica de conjunto. Es decir, ya no hay reglas para los conjuntos de datos. Se trata a cada dato de forma individual.• Al tratar a cada dato de forma individual, no hay reglas sobre su estructura. Puedo tener un registro de 3 campos y otro de 2 sin que se desperdicie espacio.

El tercer criterio a analizar es si invierte o no recursos en restricciones.

SQL	NoSQL
<ul style="list-style-type: none">• Invierte recursos en restricciones. Esto significa que, tanto el motor de bases de datos como el lenguaje, dispone de herramientas que aumentan el control y la rigidez de las estructuras de datos (Claves Foráneas, por ejemplo).• Ocupa espacio en disco y recursos del procesamiento en muchas reglas y controles, lo que vuelve al mantenimiento y a la operación más lentos, cuando se trabaja con un gran volumen de datos.	<ul style="list-style-type: none">• Intenta invertir la mínima cantidad de recursos en reglas de conjunto y restricciones. No existen claves foráneas como tales. No existen restricciones de campo ni de tipos de dato por tabla o colección. Las agrupaciones son simples depósitos de datos.• Al no ocupar tanto en restricciones, su procesamiento y mantenimiento se vuelve mucho más rápido para grandes volúmenes de datos.

Esquema flexible

A diferencia de los motores SQL, en donde se debe definir la estructura de las tablas antes de agregar información, las colecciones de MongoDB, por defecto, no requieren que sus documentos tengan la misma estructura.

Esto significa que:

- Los documentos pertenecientes a la misma colección **no necesitan tener el mismo conjunto de campos y los tipos de datos por campo pueden variar** entre documentos dentro de la misma colección.
- Para **cambiar la estructura** de documentos en una colección, como agregar nuevos campos, quitar existentes o cambiar los tipos de datos; se ejecutan **actualizaciones directas a los documentos**.

Estructura de documentos

La clave en el diseño de modelos de datos para aplicaciones MongoDB gira en torno a la estructura de los documentos y cómo representar las relaciones entre la información.

MongoDB permite dos estrategias:

- **Documentos embebidos:** sirve para representar relaciones de árbol, en donde una información es parte de otra.
- **Referencias:** sirve para representar relaciones de grafos, mediante referencias al ObjectId de otros documentos

Documentos embebidos

```
{  
    nombre: "Carlos",  
    apellido: "Lopez",  
    posts: [  
        {  
            titulo: "Introducción a MongoDB",  
            contenido: "..."  
        }  
    ]  
}
```

Colecciones

Referencias

```
{  
    nombre: "Carlos",  
    apellido: "Lopez",  
    posts: [  
        ObjectId('...'),  
        ObjectId('...')  
    ]  
}
```

¿Cuándo usar uno u otro?

La diferencia fundamental está en la forma de organizar la información:

- Al organizarla en forma de **árbol con documentos embebidos**, se puede acceder más rápido a las distintas partes. Sin embargo, cada rama del árbol debe ser única, por lo que no sirve para la información que deba estar presente en varios documentos a la vez. Es muy útil para representar especificidades o composiciones (un ejemplo es la dirección de un cliente).
- Al organizar la información en forma de **grafo puro con relaciones**, el acceso es más complejo, pues deben relacionarse varias colecciones entre sí. Sin embargo, es ideal para información que debe repetirse en varios lugares (un ejemplo son los artículos que deben estar presentes tanto en ventas como en compras).

Base de datos de documentos

Un registro de una base de datos MongoDB se denomina **documento**, que es una estructura de datos compuesta por pares **clave-valor**.

Los Documentos de MongoDB son similares a los objetos JSON de JavaScript.

```
{  
    nombre: "Carla",  
    apellido: "Lorenzo",  
    edad: 12  
}
```

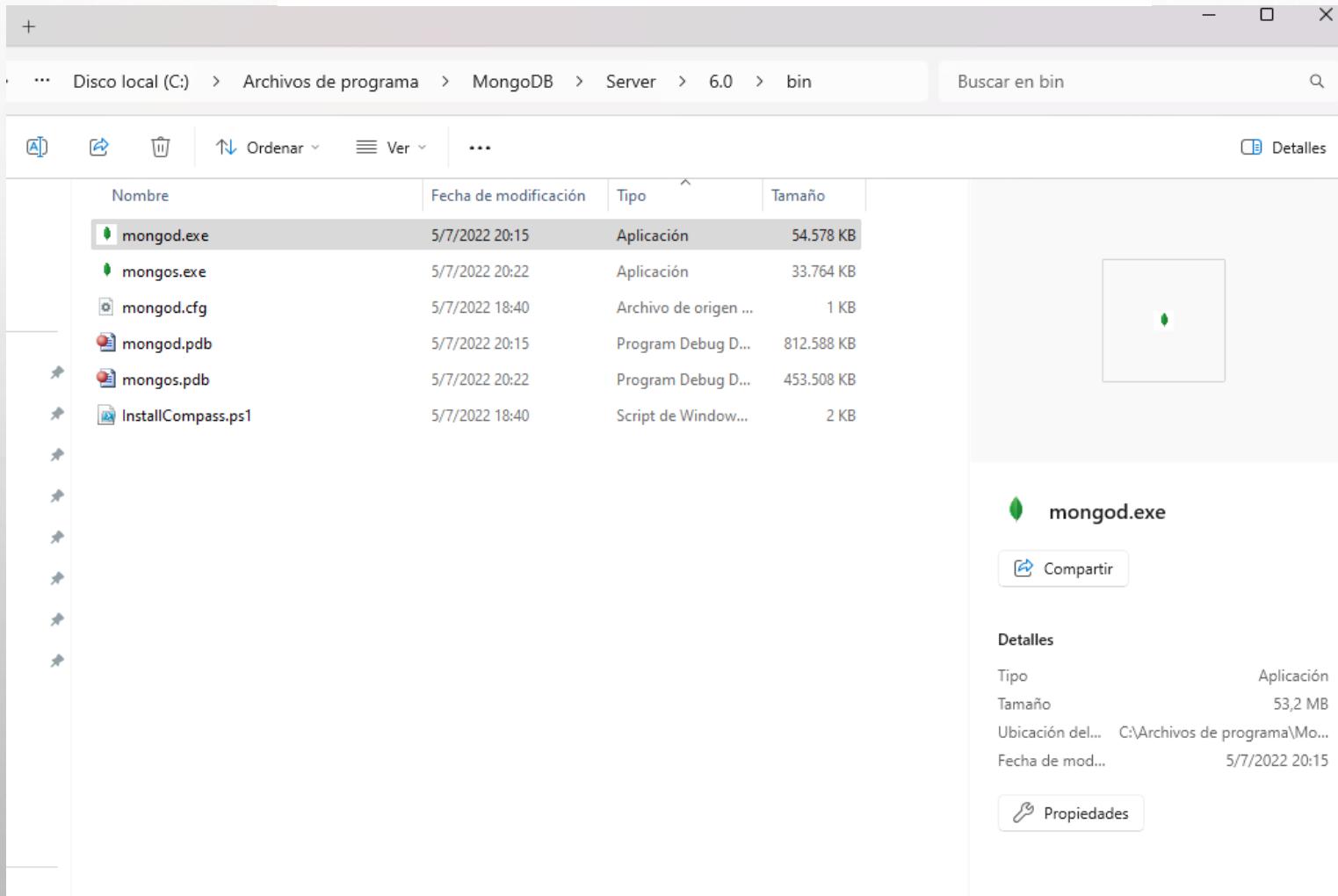
Ventajas de usar Documentos:

- Los documentos y sus tipos de dato tienen **correspondencia directa** con los tipos de dato nativos en muchos lenguajes de programación.
- Podemos **incrustar documentos dentro de documentos y crear arrays**, lo que reduce las consultas JOIN, demasiado costosas en términos de rendimientos.
- El esquema es **dinámico**, esto es, que los documentos pueden variar su estructura; lo que permite implementar **polimorfismo**.

Colecciones

MongoDB almacena los documentos en **colecciones**. Pensemos en las colecciones de MongoDB como un depósito de documentos, o como las tablas de una base de datos relacional.

Operaciones básicas con Mongo Shell



Operaciones básicas con Mongo Shell

```
C:\Program Files\MongoDB\S X + ▾ - □ X
{"t":{"$date":"2024-07-30T17:02:04.308-03:00"},"s":"W", "c":"CONTROL", "id":22140, "ctx":"initandlisten","msg":"This server is bound to localhost. Remote systems will be unable to connect to this server. Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to disable this warning","tags":["startupWarning"]},
{"t":{"$date":"2024-07-30T17:02:04.311-03:00"},"s":"I", "c":"NETWORK", "id":4915702, "ctx":"initandlisten","msg":"Updated wire specification","attr":{"oldSpec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":17}, "incomingInternalClient":{"minWireVersion":0,"maxWireVersion":17}, "outgoing":{"minWireVersion":6,"maxWireVersion":17}, "isInternalClient":true}, "newSpec":{"incomingExternalClient":{"minWireVersion":0,"maxWireVersion":17}, "incomingInternalClient":{"minWireVersion":17,"maxWireVersion":17}, "outgoing":{"minWireVersion":17,"maxWireVersion":17}, "isInternalClient":true}}},
 {"t":{"$date":"2024-07-30T17:02:04.312-03:00"},"s":"I", "c":"REPL", "id":5853300, "ctx":"initandlisten","msg":"current featureCompatibilityVersion value","attr":{"featureCompatibilityVersion":"6.0", "context":"startup"}},
 {"t":{"$date":"2024-07-30T17:02:04.312-03:00"},"s":"I", "c":"STORAGE", "id":5071100, "ctx":"initandlisten","msg":"Clearing temp directory"},
 {"t":{"$date":"2024-07-30T17:02:04.313-03:00"},"s":"I", "c":"CONTROL", "id":20536, "ctx":"initandlisten","msg":"Flow Control is enabled on this deployment"},
 {"t":{"$date":"2024-07-30T17:02:04.658-03:00"},"s":"W", "c":"FTDC", "id":23718, "ctx":"initandlisten","msg":"Failed to initialize Performance Counters for FTDC","attr":{"error":{"code":179,"codeName":"WindowsPdhError","errmsg":"PdhAddEnglishCounterW failed with 'El objeto especificado no se encontró en el equipo.'"}}},
 {"t":{"$date":"2024-07-30T17:02:04.659-03:00"},"s":"I", "c":"FTDC", "id":20625, "ctx":"initandlisten","msg":"Initializing full-time diagnostic data capture","attr":{"dataDirectory":"C:/data/db/diagnostic.data"}},
 {"t":{"$date":"2024-07-30T17:02:04.663-03:00"},"s":"I", "c":"REPL", "id":6015317, "ctx":"initandlisten","msg":"Setting new configuration state","attr":{"newState":"ConfigReplicationDisabled", "oldState":"ConfigPreStart"}},
 {"t":{"$date":"2024-07-30T17:02:04.664-03:00"},"s":"I", "c":"STORAGE", "id":22262, "ctx":"initandlisten","msg":"Timestamp monitor starting"},
 {"t":{"$date":"2024-07-30T17:02:04.666-03:00"},"s":"I", "c":"NETWORK", "id":23015, "ctx":"listener","msg":"Listening on","attr":{"address":"127.0.0.1"}},
 {"t":{"$date":"2024-07-30T17:02:04.666-03:00"},"s":"I", "c":"NETWORK", "id":23016, "ctx":"listener","msg":"Waiting for connections","attr":{"port":27017, "ssl":"off"}}
```

Operaciones básicas con Mongo Shell

The screenshot shows the MongoDB Compass application interface. On the left, there's a sidebar with a green header containing the title "Compass". Below it are sections for "Saved connections" (listing recent connections to localhost:27017) and "Recents" (listing older connections to localhost:27017). A "New connection +" button is located at the top of the sidebar. The main area is titled "New Connection" and contains a "URI" input field with the value "mongodb://localhost:27017/" and an "Edit Connection String" toggle switch. To the right of the URI is a "FAVORITE" button with a star icon. At the bottom of the main panel are "Save", "Save & Connect", and "Connect" buttons. A green callout box in the bottom-left corner of the main panel says "New to Compass and don't have a cluster? If you don't already have a cluster, you can create one for free using MongoDB Atlas" with a "CREATE FREE CLUSTER" button.

Operaciones básicas con Mongo Shell

The screenshot shows the MongoDB Compass application interface. At the top, the title bar reads "MongoDB Compass - localhost:27017/My Queries". Below the title bar is a menu bar with "Connect", "Edit", "View", and "Help". The main window has a header with "localhost:27017" and a "My Queries" tab. The left sidebar contains sections for "Performance", "Databases" (with a search bar), and a list of databases: "UCP", "admin", "config", and "local". The central area displays a message: "No saved queries yet." followed by "Start saving your aggregations and find queries, you'll see them here." Below this message is a link "Not sure where to start? Visit our Docs →". At the bottom of the screen is a mongo shell terminal window with the prompt ">_MONGOSH" and the command "test>".

Operaciones básicas con Mongo Shell

Listar las bases de datos

Comando: **show databases**

Se debe ejecutar: en Mongo Shell.

Sirve para: listar las bases de datos existentes.

```
> show databases
admin          0.000GB
alumnos        0.000GB
config          0.000GB
cotizaciones   0.000GB
example         0.000GB
historicos      0.001GB
investing       0.000GB
local           0.000GB
mi_blog          0.000GB
mongo_c2         0.001GB
nueva_base       0.001GB
pruebas_mongo_2 0.001GB
test            0.000GB
titanic          0.000GB
> █
```

Operaciones básicas con Mongo Shell

Usar una base de datos

Comando: **use BASE DE DATOS**

Se debe ejecutar: en Mongo Shell.

Sirve para: seleccionar una base de datos existente para operar. Si no existe, se crea.

```
> use example
switched to db example
> █
```



Operaciones básicas con Mongo Shell

Comando: **db.createCollection("COLECCION")**

Se debe ejecutar: en Mongo Shell.

Sirve para: crear una nueva colección en la base de datos activa.

```
> db.createCollection("miColeccion")
{ "ok" : 1 }
> █
```

Operaciones básicas con Mongo Shell

Comando: **show collections**

Se debe ejecutar: en Mongo Shell.

Sirve para: listar las colecciones disponibles en la base de datos activa.

```
> show collections
miColección
user
> □
```



Inserción, actualización y eliminación

Insertar un único documento

El siguiente ejemplo insertará un nuevo documento en la colección. Si la colección no existe se intentará crear. Si no se pasa un campo `_id`, se llenará con un nuevo `ObjectId`.

```
db.articulos.insertOne(  
    { nombre: "Celular", stock: 10 }  
)
```

Inserción, actualización y eliminación

Insertar varios documentos

MongoDB permite insertar varios documentos pasando un array de documentos.

Podemos pensar en esta operación como un `insertOne` aplicado a cada elemento del array suministrado.

```
db.articulos.insertMany([
    { nombre: "Mouse", stock: 10 },
    { nombre: "Teclado", stock: 5 }
])
```

Inserción, actualización y eliminación

Actualizar un solo documento

El siguiente ejemplo realiza una búsqueda de un solo elemento. Si el resultado del filtrado son varios documentos, se toma el primero. Luego de buscar el documento, se aplican los cambios mediante los operadores de filtrado.

```
db.usuarios.updateOne(  
  { id: { $eq: 10 } },  
  { $set: { active: false } }  
)
```

Inserción, actualización y eliminación

Actualizar varios documentos

Se comporta exactamente igual que updateOne, con la salvedad de que se aplica a todos los documentos del resultado del filtrado (en vez de aplicarse a uno solo o al primero).

```
db.usuarios.updateMany(  
  { must_delete: { $eq: 1 } },  
  { $set: { active: false } }  
)
```



Inserción, actualización y eliminación

Reemplazar un solo documento

Se comporta prácticamente igual que updateOne, con la salvedad de que, en vez de aplicar actualizaciones específicas, borra el contenido del documento y lo reemplaza completamente por el suministrado.

```
db.usuarios.replaceOne(  
  { id: { $eq: 10 } },  
  { usuario: "CarlosLopez", email: "clopez@gmail.com" }  
)
```

Inserción, actualización y eliminación

Eliminar un solo documento

El siguiente ejemplo realiza una búsqueda de un solo elemento. Si el resultado del filtrado son varios documentos, se toma el primero. Luego de buscar el documento, se procede a eliminarlo.

```
db.usuarios.deleteOne(  
  { id: { $eq: 10 } }  
)
```

Inserción, actualización y eliminación

Eliminar varios documentos

Esta operación extiende `deleteOne` a varios documentos. En este ejemplo, se borrarán todos los documentos que coincidan con el criterio de búsqueda:

```
db.usuarios.deleteMany({ must_delete: { $eq: 1 } })
```

También se puede aplicar `deleteMany` sin un criterio específico, lo que resultará en la eliminación de todos los documentos de la colección dada:

```
db.usuarios.deleteMany({ })
```

Introducción a operaciones CRUD

Crear

Las operaciones de crear o insertar agregan nuevos documentos a una colección. Si la colección no existe, se intenta crear.

MongoDB provee los siguientes métodos para crear información.

- **db.<colección>.insertOne({})**
Inserta un solo documento en una colección.
- **db.<colección>.insertMany([{},{}])**
Inserta varios documentos en una colección.

```
db.usuarios.insertOne(  
{  
  nombre: "Carlos",  
  apellido: "Lopez"  
}  
)
```

Introducción a operaciones CRUD

Leer

Las operaciones de lectura consisten en obtener documentos de una colección.

MongoDB provee los siguientes métodos:

- **db.<colección>.find(<filtro>)**

Devuelve una vista filtrada de una colección.

```
db.usuarios.find(  
{  
  rol: { $in: ['admin', 'gerente'] }  
},  
{ nombre: 1 }  
)
```

Introducción a operaciones CRUD

Actualizar

Las operaciones de actualización consisten en la modificación de documentos ya existentes en una colección.

MongoDB provee los siguientes métodos:

- **db.collection.updateOne()**
Cambia algunas porciones de un solo documento.
- **db.collection.updateMany()**
Cambia algunas porciones de muchos documentos.
- **db.collection.replaceOne()**
Reemplaza totalmente un documento.

```
db.usuarios.updateOne(  
{  
  id: { $eq: 12 }  
},  
{ $set: { active: false } }  
)
```

Introducción a operaciones CRUD

Borrar

Las operaciones de borrado consisten en la eliminación de documentos ya existentes en una colección.

MongoDB provee los siguientes métodos:

- **db.collection.deleteOne()**
Elimina un solo documento.
- **db.collection.deleteMany()**
Elimina varios documentos.

```
db.usuarios.deleteMany(  
 {  
 active: { $eq: false }  
 }  
)
```

Mapeo SQL - MongoDB

Términos y conceptos

Concepto SQL	Concepto MongoDB
Base de datos	Base de datos
Tabla	Colección
Fila	Documento
Columna	Campo
Índice	Índice

Mapeo SQL - MongoDB

CREATE TABLE

SQL	MongoDB
<pre>CREATE TABLE usuarios (id INT NOT NULL AUTO_INCREMENT, edad INT, nombre VARCHAR(50), email VARCHAR(50), clave TEXT, PRIMARY KEY(id))</pre>	<p>Se crea implícitamente en el primer insert. La clave primaria <code>_id</code> se crea de forma automática.</p> <pre>db.usuarios.insertOne({ edad: 45, nombre: "Carlos", email: "lopez@gmail.com", clave: "1234" })</pre>

Mapeo SQL - MongoDB

ALTER TABLE

SQL	MongoDB
<pre>ALTER TABLE usuarios ADD created_at TIMESTAMP DEFAULT current_timestamp()</pre>	<p>Las colecciones no especifican estructura, por lo que la actualización de la colección ocurre con la actualización del documento.</p> <pre>db.usuarios.updateMany({}, { \$currentDate: { created_at: { \$type: "timestamp" } } })</pre>

Mapeo SQL - MongoDB

DROP TABLE

	SQL	MongoDB
	DROP TABLE usuarios	db.usuarios.drop()

Mapeo SQL - MongoDB

INSERT

SQL	MongoDB
<pre>INSERT INTO usuarios (edad, nombre, email, clave) VALUES (30, "Carlos Lopez", "clopez@gmial.com", "1234")</pre>	<pre>db.usuarios.insertOne({ edad: 30, nombre: "Carlos Lopez", email: "clopez@gmail.com", clave: "1234" })</pre>

Mapeo SQL - MongoDB

SELECT

SQL	MongoDB
<pre>SELECT nombre FROM usuarios WHERE id = 1</pre>	<pre>db.usuarios.find({ id: { \$eq: 1 } }, { nombre: 1 })</pre>

Mapeo SQL - MongoDB

UPDATE

SQL	MongoDB
UPDATE nombre SET active = 0 WHERE id = 1	db.usuarios.updateOne({ id: { \$eq: 1 } }, { \$set: { active: 0 } })

Operadores de filtrado y actualización comunes

Filtrado / Comparar por igualdad - no igual

Operador `$eq`

Define una condición de **igualdad**. Este operador filtra documentos que tengan un campo cuyo valor coincide con el especificado.

```
db.usuarios.find({ active: { $eq: true } })
```

Operadores de filtrado y actualización comunes

Filtrado / Comparar mayor - mayor o igual

Operador **\$gt** / **\$gte**

Este operador filtra documentos que tengan campo cuyo valor sea **mayor (\$gt)** o **mayor o igual (\$gte)** al valor especificado.

```
db.articulos.find({ precio: { $gt: 1000 } })
```

Operadores de filtrado y actualización comunes

Filtrado / Comparar menor - menor o igual

Operador **\$lt** / **\$lte**

Este operador filtra documentos que tengan un campo cuyo valor sea **menor (\$lt)** o **menor o igual (\$lte)** que el valor especificado.

```
db.articulos.find({ precio: { $lte: 1000 } })
```

Operadores de filtrado y actualización comunes

Filtrado / Buscar en array

Operador **\$lt** / **\$lte**

Este operador filtra documentos que tengan **un campo cuyo valor se encuentre entre los elementos del array** definido.

```
db.usuarios.find({ role: { $in: ['admin', 'gerente'] } })
```

Operadores de filtrado y actualización comunes

Actualización / Asignar

Operador \$set

Este operador **reemplaza el valor de un campo** por el valor dado. Si el campo no existe, se crea uno con ese valor.

```
db.articulos.updateOne(  
  { id: { $eq: 10 } },  
  { $set: { precio: 1500 } }  
)
```

Operadores de filtrado y actualización comunes

Actualización / Incrementar

Operador \$inc

Este operador realiza la **suma** del valor de un campo y el valor dado. Si el valor dado es negativo, se realiza una **resta**. Se pueden especificar varios campos con sus respectivos incrementos.

```
db.usuarios.updateOne(  
  { id: { $eq: 10 } },  
  { $inc: { inicio_sesion: 1, dias_disponibles: -1 } }  
)
```

Operadores de filtrado y actualización comunes

Actualización / Asignar fecha actual

Operador **\$currentDate**

Almacena la **fecha actual** en un campo dado.

Se puede especificar el tipo de dato de fecha para cada campo ("timestamp" o "date").

Si se pasa un valor booleano true, se almacena la fecha con formato Date.

```
db.usuarios.updateOne(  
  { id: { $eq: 10 } },  
  { $currentDate: { lastLogIn: { $type: "timestamp" } }  
}
```

Operadores de filtrado y actualización comunes

Actualización / Multiplicar

Operador **\$mul**

Multiplica el valor actual de un campo por el valor dado.

```
db.ventas.updateOne(  
  { id: { $eq: 10 } },  
  { $mul: { amount: 10 } }  
)
```

Ejercicio 1: Crear una base de datos y una colección

Objetivo: Crear una base de datos llamada `biblioteca` y una colección llamada `libros`.

Instrucciones:

1. Conéctate a MongoDB usando el cliente de línea de comandos (``mongo``).
2. Usa el comando `use` para crear y cambiar a la base de datos `biblioteca`.
3. Inserta un documento en la colección `libros`.

Mongo

use biblioteca

```
db.libros.insertOne({ titulo: "Cien Años de Soledad", autor: "Gabriel García Márquez", año: 1967 })
```

```
< {  
    _id: ObjectId('66a95ebd0cf32d63f4167b4a'),  
    titulo: 'Cien Años de Soledad',  
    autor: 'Gabriel García Márquez',  
    'año': 1967  
}
```

Ejercicio 2: Insertar múltiples documentos

Objetivo: Insertar múltiples documentos en la colección `libros`.

Instrucciones:

1. Inserta los siguientes documentos en la colección `libros`:

json

 Copiar código

```
{ "titulo": "1984", "autor": "George Orwell", "año": 1949 }
{ "titulo": "To Kill a Mockingbird", "autor": "Harper Lee", "año": 1960 }
{ "titulo": "The Great Gatsby", "autor": "F. Scott Fitzgerald", "año": 1925 }
```

```
db.libros.insertMany([
{ titulo: "1984", autor: "George Orwell", año: 1949 },
{ titulo: "To Kill a Mockingbird", autor: "Harper Lee", año: 1960 },
{ titulo: "The Great Gatsby", autor: "F. Scott Fitzgerald", año: 1925 }
])
```

```
< {
  _id: ObjectId('66a95ebd0cf32d63f4167b4a'),
  titulo: 'Cien Años de Soledad',
  autor: 'Gabriel García Márquez',
  'año': 1967
}
{
  _id: ObjectId('66a95f360cf32d63f4167b4b'),
  titulo: '1984',
  autor: 'George Orwell',
  'año': 1949
}
{
  _id: ObjectId('66a95f360cf32d63f4167b4c'),
  titulo: 'To Kill a Mockingbird',
  autor: 'Harper Lee',
  'año': 1960
}
{
  _id: ObjectId('66a95f360cf32d63f4167b4d'),
  titulo: 'The Great Gatsby',
  autor: 'F. Scott Fitzgerald',
  'año': 1925
}
```

Ejercicio 3: Consultar documentos

Objetivo: Consultar todos los documentos en la colección `libros`.

Instrucciones:

1. Usa el método `find` para consultar todos los documentos en la colección `libros`.

```
db.libros.find().pretty()
```

```
< [
    {
        _id: ObjectId('66a95ebd0cf32d63f4167b4a'),
        titulo: 'Cien Años de Soledad',
        autor: 'Gabriel García Márquez',
        'año': 1967
    },
    {
        _id: ObjectId('66a95f360cf32d63f4167b4b'),
        titulo: '1984',
        autor: 'George Orwell',
        'año': 1949
    },
    {
        _id: ObjectId('66a95f360cf32d63f4167b4c'),
        titulo: 'To Kill a Mockingbird',
        autor: 'Harper Lee',
        'año': 1960
    },
    {
        _id: ObjectId('66a95f360cf32d63f4167b4d'),
        titulo: 'The Great Gatsby',
        autor: 'F. Scott Fitzgerald',
        'año': 1925
    }
]
```

Ejercicio 4: Actualizar documentos

Objetivo: Actualizar el año de publicación de "1984" a 1950.

Instrucciones:

1. Usa el método `updateOne` para actualizar el documento.

```
db.libros.updateOne(  
  { titulo: "1984" },  
  { $set: { año: 1950 } }  
)
```

```
{  
  _id: ObjectId('66a95f360cf32d63f4167b4b'),  
  titulo: '1984',  
  autor: 'George Orwell',  
  'año': 1949  
}
```

```
{  
  _id: ObjectId('66a95f360cf32d63f4167b4b'),  
  titulo: '1984',  
  autor: 'George Orwell',  
  'año': 1950  
}
```

Ejercicio 5: Eliminar documentos

Objetivo: Eliminar el libro "The Great Gatsby" de la colección.

Instrucciones:

1. Usa el método `'deleteOne'` para eliminar el documento.

```
db.libros.deleteOne({ titulo: "The Great Gatsby" })
```

```
> db.libros.find().pretty();
< [
  {
    _id: ObjectId('66a95ebd0cf32d63f4167b4a'),
    titulo: 'Cien Años de Soledad',
    autor: 'Gabriel García Márquez',
    'año': 1967
  },
  {
    _id: ObjectId('66a95f360cf32d63f4167b4b'),
    titulo: '1984',
    autor: 'George Orwell',
    'año': 1950
  },
  {
    _id: ObjectId('66a95f360cf32d63f4167b4c'),
    titulo: 'To Kill a Mockingbird',
    autor: 'Harper Lee',
    'año': 1960
  }
]
```

Ejercicio 6: Buscar documentos con criterios

Objetivo: Encontrar todos los libros publicados después de 1950.

Instrucciones:

1. Usa el método `find` con un criterio de búsqueda.

```
db.libros.find({ año: { $gt: 1950 } }).pretty()
```

```
< {
  _id: ObjectId('66a95ebd0cf32d63f4167b4a'),
  titulo: 'Cien Años de Soledad',
  autor: 'Gabriel García Márquez',
  'año': 1967
}
{
  _id: ObjectId('66a95f360cf32d63f4167b4c'),
  titulo: 'To Kill a Mockingbird',
  autor: 'Harper Lee',
  'año': 1960
}
```

Ejercicio 7: Indexación

Objetivo: Crear un índice en el campo `autor` para mejorar la eficiencia de las consultas.

Instrucciones:

1. Usa el método `createIndex` para crear un índice en el campo `autor`.

```
db.libros.createIndex({ autor: 1 })
```

```
> db.libros.createIndex({ autor: 1 });
< autor_1

> db.libros.find({ autor: "Harper Lee" }).toArray();
< [
  {
    _id: ObjectId('66a95f360cf32d63f4167b4c'),
    titulo: 'To Kill a Mockingbird',
    autor: 'Harper Lee',
    'año': 1960
  }
]
```

Ejercicio 8: Agregación

Objetivo: Obtener la cantidad de libros por autor.

Instrucciones:

1. Usa el método `aggregate` para agrupar los libros por autor y contar el número de libros por autor.

```
db.libros.aggregate([  
  { $group: { _id: "$autor", total: { $sum: 1 } } }  
])
```

```
< [  
    {  
        _id: 'George Orwell',  
        total: 1  
    }  
    {  
        _id: 'Harper Lee',  
        total: 1  
    }  
    {  
        _id: 'Gabriel García Márquez',  
        total: 1  
    }  
]
```

Ejercicio 9: Consulta con proyección

Objetivo: Consultar todos los libros mostrando solo el título y el autor, sin el campo `año`.

Instrucciones:

1. Usa el método `find` con proyección para excluir el campo `año`.

```
db.libros.find({}, { titulo: 1, autor: 1, _id: 0 }).pretty()
```

```
< [
    {
        titulo: 'Cien Años de Soledad',
        autor: 'Gabriel García Márquez'
    },
    {
        titulo: '1984',
        autor: 'George Orwell'
    },
    {
        titulo: 'To Kill a Mockingbird',
        autor: 'Harper Lee'
    }
]
```

Ejercicio 10: Borrado de la colección

Objetivo: Eliminar todos los documentos de la colección `libros`.

Instrucciones:

1. Usa el método `deleteMany` para eliminar todos los documentos de la colección.

```
db.libros.deleteMany({})
```

```
> db.libros.find();
```

```
<
```

```
db.libros.insertMany([ { titulo: "Brave New World", autor: "Aldous Huxley", año: 1932, generos: ["Ciencia Ficción", "Distopía"] }, { titulo: "Fahrenheit 451", autor: "Ray Bradbury", año: 1953, generos: ["Ciencia Ficción", "Distopía"] }, { titulo: "Animal Farm", autor: "George Orwell", año: 1945, generos: ["Satírica", "Política"] }, { titulo: "The Catcher in the Rye", autor: "J.D. Salinger", año: 1951, generos: ["Ficción", "Literatura Americana"] }, { titulo: "The Lord of the Rings", autor: "J.R.R. Tolkien", año: 1954, generos: ["Fantasía", "Aventura"] }, { titulo: "Pride and Prejudice", autor: "Jane Austen", año: 1813, generos: ["Romance", "Clásico"] }, { titulo: "The Hobbit", autor: "J.R.R. Tolkien", año: 1937, generos: ["Fantasía", "Aventura"] }, { titulo: "Moby Dick", autor: "Herman Melville", año: 1851, generos: ["Aventura", "Clásico"] }, { titulo: "War and Peace", autor: "Leo Tolstoy", año: 1869, generos: ["Histórica", "Clásico"] }, { titulo: "The Odyssey", autor: "Homer", año: -800, generos: ["Épica", "Aventura"] }, { titulo: "Crime and Punishment", autor: "Fyodor Dostoevsky", año: 1866, generos: ["Ficción", "Psicológica"] }, { titulo: "The Brothers Karamazov", autor: "Fyodor Dostoevsky", año: 1880, generos: ["Ficción", "Filosófica"] }, { titulo: "Wuthering Heights", autor: "Emily Brontë", año: 1847, generos: ["Romance", "Gótico"] }, { titulo: "Jane Eyre", autor: "Charlotte Brontë", año: 1847, generos: ["Romance", "Clásico"] }, { titulo: "The Divine Comedy", autor: "Dante Alighieri", año: 1320, generos: ["Épica", "Filosófica"] }, { titulo: "Don Quixote", autor: "Miguel de Cervantes", año: 1605, generos: ["Aventura", "Clásico"] }, { titulo: "The Iliad", autor: "Homer", año: -750, generos: ["Épica", "Aventura"] }, { titulo: "Madame Bovary", autor: "Gustave Flaubert", año: 1856, generos: ["Ficción", "Clásico"] }, { titulo: "The Great Expectations", autor: "Charles Dickens", año: 1861, generos: ["Ficción", "Clásico"] }, { titulo: "Les Misérables", autor: "Victor Hugo", año: 1862, generos: ["Histórica", "Ficción"] }])
```

Ejercicio 1: Uso del operador `\\$set`

Objetivo: Actualizar un campo existente en los documentos.

Instrucciones:

1. Actualiza el campo `año` del libro "1984" para que sea 1950 usando el operador `\\$set`.

```
db.libros.updateOne( { titulo: "1984" }, { $set: { año: 1950 } })
```

```
> db.libros.find();
< [
    {
        _id: ObjectId('66a96b970cf32d63f4167b4e'),
        titulo: '1984',
        autor: 'George Orwell',
        'año': 1950
    }
]
```

Ejercicio 2: Uso del operador `\\$unset`

Objetivo: Eliminar un campo de un documento.

Instrucciones:

1. Elimina el campo `año` del libro "1984" usando el operador `\\$unset`.

```
db.libros.updateOne( { titulo: "1984" }, { $unset: { año: "" } })
```

```
> db.libros.find();
< {
    _id: ObjectId('66a96b970cf32d63f4167b4e'),
    titulo: '1984',
    autor: 'George Orwell'
}
```

Ejercicio 3: Uso del operador `\\$inc`

Objetivo: Incrementar el valor de un campo numérico.

Instrucciones:

1. Incrementa el campo `año` de todos los libros en 1 año usando el operador `\\$inc`.

```
db.libros.updateMany( {}, { $inc: { año: 1 } })
```

```
> db.libros.find();
< [
    {
        _id: ObjectId('66a96b970cf32d63f4167b4e'),
        titulo: '1984',
        autor: 'George Orwell',
        'año': 1
    },
    {
        _id: ObjectId('66a96b970cf32d63f4167b4f'),
        titulo: 'To Kill a Mockingbird',
        autor: 'Harper Lee',
        'año': 1961
    }
]
```

Ejercicio 4: Uso del operador `\\$push`

Objetivo: Agregar un elemento a un arreglo.

Instrucciones:

1. Agrega el género "Clásico" al arreglo `generos` del libro "1984" usando el operador `\\$push`.

```
db.libros.updateOne( { titulo: "1984" }, { $push: { generos: "Clásico" } })
```

```
> db.libros.find();
< {
  _id: ObjectId('66a96b970cf32d63f4167b4e'),
  titulo: '1984',
  autor: 'George Orwell',
  'año': 1,
  generos: [
    'Clásico'
  ]
}
```

Ejercicio 5: Uso del operador `'\$addToSet'

Objetivo: Agregar un elemento a un arreglo solo si no existe.

Instrucciones:

1. Agrega el género "Distopía" al arreglo `generos` del libro "1984" usando el operador `'\$addToSet'`.

```
db.libros.updateOne( { titulo: "1984" }, { $addToSet: { generos: "Distopía" } })
```

```
> db.libros.find();
< [
    {
        _id: ObjectId('66a96b970cf32d63f4167b4e'),
        titulo: '1984',
        autor: 'George Orwell',
        'año': 1,
        generos: [
            'Clásico',
            'Distopía'
        ]
    }
]
```

Ejercicio 6: Uso del operador `\\$pull`

Objetivo: Eliminar un elemento de un arreglo.

Instrucciones:

1. Elimina el género "Distopía" del arreglo `generos` del libro "1984" usando el operador `\\$pull`.

```
db.libros.updateOne( { titulo: "1984" }, { $pull: { generos: "Distopía" } })
```

```
> db.libros.find({ titulo: "1984" });
< [
  {
    _id: ObjectId('66a96b970cf32d63f4167b4e'),
    titulo: '1984',
    autor: 'George Orwell',
    'año': 1,
    generos: [
      'Clásico'
    ]
  }
]
```

Ejercicio 7: Uso del operador `\$gte` y `\$lte`

Objetivo: Encontrar documentos que cumplan con criterios de rango.

Instrucciones:

1. Encuentra todos los libros publicados entre 1950 y 2000 inclusive, usando los operadores `\$gte` y `\$lte`.

```
db.libros.find( { año: { $gte: 1950, $lte: 2000 } }).pretty()
```

```
< {
    _id: ObjectId('66a96b970cf32d63f4167b4f'),
    titulo: 'To Kill a Mockingbird',
    autor: 'Harper Lee',
    'año': 1961
}
{
    _id: ObjectId('66a96bd90cf32d63f4167b52'),
    titulo: 'Fahrenheit 451',
    autor: 'Ray Bradbury',
    'año': 1954,
    generos: [
        'Ciencia Ficción',
        'Distopía'
    ]
}
{
    _id: ObjectId('66a96bd90cf32d63f4167b54'),
    titulo: 'The Catcher in the Rye',
    autor: 'J.D. Salinger',
    'año': 1952,
    generos: [
        'Ficción',
        'Literatura Americana'
}
```

Ejercicio 8: Uso del operador `\\$in`

Objetivo: Encontrar documentos donde un campo esté en un conjunto de valores.

Instrucciones:

1. Encuentra todos los libros escritos por "George Orwell" o "Harper Lee" usando el operador `\\$in`.

```
db.libros.find( { autor: { $in: ["George Orwell",  
"Harper Lee"] } }).pretty()
```

```
< {  
    _id: ObjectId('66a96b970cf32d63f4167b4e'),  
    titulo: '1984',  
    autor: 'George Orwell',  
    'año': 1,  
    generos: [  
        'Clásico'  
    ]  
}  
{  
    _id: ObjectId('66a96bd90cf32d63f4167b53'),  
    titulo: 'Animal Farm',  
    autor: 'George Orwell',  
    'año': 1946,  
    generos: [  
        'Satírica',  
        'Política'  
    ]  
}  
{  
    _id: ObjectId('66a96b970cf32d63f4167b4f'),  
    titulo: 'To Kill a Mockingbird',  
    autor: 'Harper Lee',  
    'año': 1961  
}
```

Ejercicio 9: Uso del operador `\$or`

Objetivo: Encontrar documentos que cumplan con al menos una de las condiciones.

Instrucciones:

1. Encuentra todos los libros escritos por "George Orwell" o publicados después de 1950 usando el operador `\\$or` .

```
db.libros.find( { $or: [{ autor: "George Orwell" },  
{ año: { $gt: 1950 } }] }).pretty()
```

```
< {  
    _id: ObjectId('66a96b970cf32d63f4167b4e'),  
    titulo: '1984',  
    autor: 'George Orwell',  
    'año': 1,  
    generos: [  
        'Clásico'  
    ]  
}  
{  
    _id: ObjectId('66a96b970cf32d63f4167b4f'),  
    titulo: 'To Kill a Mockingbird',  
    autor: 'Harper Lee',  
    'año': 1961  
}  
{  
    _id: ObjectId('66a96bd90cf32d63f4167b52'),  
    titulo: 'Fahrenheit 451',  
    autor: 'Ray Bradbury',  
    'año': 1954,  
    generos: [  
        'Ciencia Ficción',  
        'Distopía'
```

Ejercicio 10: Uso del operador `\\$regex`

Objetivo: Buscar documentos que coincidan con una expresión regular.

Instrucciones:

1. Encuentra todos los libros cuyo título contiene la palabra "kill" (sin importar mayúsculas/minúsculas) usando el operador `\\$regex`.

```
db.libros.find( { titulo: { $regex: "kill", $options: "i" } }).pretty()
```

```
< {  
    _id: ObjectId('66a96b970cf32d63f4167b4f'),  
    titulo: 'To Kill a Mockingbird',  
    autor: 'Harper Lee',  
    'año': 1961  
}
```

Ejercicio 11: Uso del operador `\$exists`

Objetivo: Encontrar documentos donde un campo específico exista o no.

Instrucciones:

1. Encuentra todos los libros que tienen el campo `año` usando el operador `"\$exists`".

```
db.libros.find( { año: { $exists: true } }).pretty()
```

```
< {
    _id: ObjectId('66a96b970cf32d63f4167b4e'),
    titulo: '1984',
    autor: 'George Orwell',
    'año': 1,
    generos: [
        'Clásico'
    ]
}
{
    _id: ObjectId('66a96b970cf32d63f4167b4f'),
    titulo: 'To Kill a Mockingbird',
    autor: 'Harper Lee',
    'año': 1961
}
{
    _id: ObjectId('66a96b970cf32d63f4167b50'),
    titulo: 'The Great Gatsby',
    autor: 'F. Scott Fitzgerald',
    'año': 1926
}
{
```

Ejercicio 12: Uso del operador `\\$elemMatch`

Objetivo: Encontrar documentos donde al menos un elemento en un arreglo cumpla con una condición.

Instrucciones:

1. Encuentra todos los libros donde al menos un género en el arreglo `generos` sea "Distopía" usando el operador `\\$elemMatch`.

```
db.libros.find( { generos: { $elemMatch: { $eq: "Distopía" } } }).pretty()
```

```
< {
  _id: ObjectId('66a96bd90cf32d63f4167b51'),
  titulo: 'Brave New World',
  autor: 'Aldous Huxley',
  'año': 1933,
  generos: [
    'Ciencia Ficción',
    'Distopía'
  ]
}
{
  _id: ObjectId('66a96bd90cf32d63f4167b52'),
  titulo: 'Fahrenheit 451',
  autor: 'Ray Bradbury',
  'año': 1954,
  generos: [
    'Ciencia Ficción',
    'Distopía'
  ]
}
```

Ejercicio 13: Uso del operador `\\$all`

Objetivo: Encontrar documentos donde un arreglo contenga todos los valores especificados.

Instrucciones:

1. Encuentra todos los libros cuyos géneros incluyan tanto "Fantasía" como "Aventura" usando el operador `\\$all`.

```
db.libros.find( { generos: { $all: ["Fantasía", "Aventura"] } }).pretty()
```

```
< [
    {
        _id: ObjectId('66a96bd90cf32d63f4167b55'),
        titulo: 'The Lord of the Rings',
        autor: 'J.R.R. Tolkien',
        'año': 1955,
        generos: [
            'Fantasía',
            'Aventura'
        ]
    }
    {
        _id: ObjectId('66a96bd90cf32d63f4167b57'),
        titulo: 'The Hobbit',
        autor: 'J.R.R. Tolkien',
        'año': 1938,
        generos: [
            'Fantasía',
            'Aventura'
        ]
    }
]
```

Ejercicio 14: Uso del operador `\\$size`

Objetivo: Encontrar documentos donde el tamaño de un arreglo sea igual a un valor específico.

Instrucciones:

1. Encuentra todos los libros que tienen exactamente dos géneros usando el operador `\\$size`.

```
db.libros.find( { generos: { $size: 2 } }).pretty()
```

```
{  
  _id: ObjectId('66a96bd90cf32d63f4167b51'),  
  titulo: 'Brave New World',  
  autor: 'Aldous Huxley',  
  'año': 1933,  
  generos: [  
    'Ciencia Ficción',  
    'Distopía'  
  ]  
}  
{  
  _id: ObjectId('66a96bd90cf32d63f4167b52'),  
  titulo: 'Fahrenheit 451',  
  autor: 'Ray Bradbury',  
  'año': 1954,  
  generos: [  
    'Ciencia Ficción',  
    'Distopía'  
  ]  
}  
{  
  _id: ObjectId('66a96bd90cf32d63f4167b53'),  
  titulo: 'Animal Farm',
```

Ejercicio 15: Uso del operador `\\$type`

Objetivo: Encontrar documentos donde un campo sea de un tipo específico.

Instrucciones:

1. Encuentra todos los libros donde el campo `año` sea de tipo `int` usando el operador `\\$type`.

```
db.libros.find( { año: { $type: "int" } }).pretty()
```

```
< {
    _id: ObjectId('66a96b970cf32d63f4167b4e'),
    titulo: '1984',
    autor: 'George Orwell',
    'año': 1,
    generos: [
        'Clásico'
    ]
}
{
    _id: ObjectId('66a96b970cf32d63f4167b4f'),
    titulo: 'To Kill a Mockingbird',
    autor: 'Harper Lee',
    'año': 1961
}
{
    _id: ObjectId('66a96b970cf32d63f4167b50'),
    titulo: 'The Great Gatsby',
    autor: 'F. Scott Fitzgerald',
    'año': 1926
}
{
    _id: ObjectId('66a96bd90cf32d63f4167b51'),
    titulo: 'Brave New World',
    autor: 'Aldous Huxley',
    'año': 1933,
```

Ejercicio 16: Uso del operador '\$and'

Objetivo: Encontrar documentos que cumplan con todas las condiciones especificadas.

Instrucciones:

1. Encuentra todos los libros escritos por "George Orwell" y publicados antes de 1950 usando el operador '\$and'.

```
db.libros.find( { $and: [{ autor: "George Orwell" }, { año: { $lt: 1950 } }] }).pretty()
```

```
< {
    _id: ObjectId('66a96b970cf32d63f4167b4e'),
    titulo: '1984',
    autor: 'George Orwell',
    'año': 1,
    generos: [
        'Clásico'
    ]
}
{
    _id: ObjectId('66a96bd90cf32d63f4167b53'),
    titulo: 'Animal Farm',
    autor: 'George Orwell',
    'año': 1946,
    generos: [
        'Satírica',
        'Política'
    ]
}
```

Ejercicio 17: Uso del operador `\\$not`

Objetivo: Excluir documentos que cumplan con una condición especificada.

Instrucciones:

1. Encuentra todos los libros que no fueron escritos por "George Orwell" usando el operador `\\$not`.

```
db.libros.find( { autor: { $not: { $eq: "George Orwell" } } }).pretty()
```

```
< {
  _id: ObjectId('66a96bd90cf32d63f4167b51'),
  titulo: 'Brave New World',
  autor: 'Aldous Huxley',
  'año': 1933,
  generos: [
    'Ciencia Ficción',
    'Distopía'
  ]
}
{
  _id: ObjectId('66a96bd90cf32d63f4167b63'),
  titulo: 'The Great Expectations',
  autor: 'Charles Dickens',
  'año': 1862,
  generos: [
    'Ficción',
    'Clásico'
  ]
}
{
  _id: ObjectId('66a96bd90cf32d63f4167b5e'),
  titulo: 'Jane Eyre',
```

Ejercicio 18: Uso del operador `\\$nor`

Objetivo: Excluir documentos que cumplan con alguna de las condiciones especificadas.

Instrucciones:

1. Encuentra todos los libros que no fueron escritos por "George Orwell" ni publicados antes de 1950 usando el operador `\\$nor`.

```
db.libros.find( { $nor: [{ autor: "George Orwell" },  
{ año: { $lt: 1950 } }] }).pretty()
```

```
< {  
    _id: ObjectId('66a96b970cf32d63f4167b4f'),  
    titulo: 'To Kill a Mockingbird',  
    autor: 'Harper Lee',  
    'año': 1961  
}  
{  
    _id: ObjectId('66a96bd90cf32d63f4167b52'),  
    titulo: 'Fahrenheit 451',  
    autor: 'Ray Bradbury',  
    'año': 1954,  
    generos: [  
        'Ciencia Ficción',  
        'Distopía'  
    ]  
}  
{  
    _id: ObjectId('66a96bd90cf32d63f4167b54'),  
    titulo: 'The Catcher in the Rye',  
    autor: 'J.D. Salinger',  
    'año': 1952,  
    generos: [  
        'Ficción',  
        'Literatura Americana'  
]
```

Ejercicio 19: Uso del operador `\\$expr`

Objetivo: Usar expresiones agregadas en consultas.

Instrucciones:

1. Encuentra todos los libros cuyo título y autor sean iguales usando el operador `\\$expr`.

```
db.libros.find( { $expr: { $eq: ["$titulo", "$autor"] } }).pretty()
```

```
> db.libros.find(  
    { $expr: { $eq: ["$titulo", "$autor"] } }  
).pretty()  
;  
<  
biblioteca>
```

Ejercicio 20: Uso del operador `\\$mod`

Objetivo: Encontrar documentos donde el valor de un campo dividido por un divisor tenga un residuo específico.

Instrucciones:

1. Encuentra todos los libros cuyo `año` sea divisible por 5 usando el operador `\\$mod`.

```
db.libros.find( { año: { $mod: [5, 0] } }).pretty()
```

```
< {
  _id: ObjectId('66a96bd90cf32d63f4167b55'),
  titulo: 'The Lord of the Rings',
  autor: 'J.R.R. Tolkien',
  'año': 1955,
  generos: [
    'Fantasía',
    'Aventura'
  ]
}
{
  _id: ObjectId('66a96bd90cf32d63f4167b59'),
  titulo: 'War and Peace',
  autor: 'Leo Tolstoy',
  'año': 1870,
  generos: [
    'Histórica',
    'Clásico'
  ]
}
```

Ejercicio 1: Modelado de datos para una biblioteca

Objetivo: Diseñar un esquema para una colección `libros` que almacene información sobre libros, autores y géneros.

Instrucciones:

1. Diseña una colección `libros` que contenga información sobre el título, el autor (incluyendo el nombre y la fecha de nacimiento), el año de publicación y los géneros del libro.
2. Inserta un documento de ejemplo en la colección `libros`.

json

```
{  
    "titulo": "1984",  
    "autor": {  
        "nombre": "George Orwell",  
        "fecha_nacimiento": "1903-06-25"  
    },  
    "año": 1949,  
    "generos": ["Distopía", "Ciencia Ficción"]  
}
```

```
sh
```

```
db.libros.insertOne({  
    "titulo": "1984",  
    "autor": {  
        "nombre": "George Orwell",  
        "fecha_nacimiento": ISODate("1903-06-25T00:00:00Z")  
    },  
    "año": 1949,  
    "generos": ["Distopía", "Ciencia Ficción"]  
})
```

Ejercicio 2: Modelado de datos para una tienda en línea

Objetivo: Diseñar un esquema para una colección `productos` que almacene información sobre productos, categorías y precios.

Instrucciones:

1. Diseña una colección `productos` que contenga información sobre el nombre del producto, la categoría, el precio y una lista de proveedores (nombre y contacto).
2. Inserta un documento de ejemplo en la colección `productos`.

json

```
{  
    "nombre": "Laptop",  
    "categoria": "Electrónica",  
    "precio": 999.99,  
    "proveedores": [  
        {  
            "nombre": "Proveedor A",  
            "contacto": "contactoA@example.com"  
        },  
        {  
            "nombre": "Proveedor B",  
            "contacto": "contactoB@example.com"  
        }  
    ]  
}
```

```
sh

db.productos.insertOne({
  "nombre": "Laptop",
  "categoria": "Electrónica",
  "precio": 999.99,
  "proveedores": [
    {
      "nombre": "Proveedor A",
      "contacto": "contactoA@example.com"
    },
    {
      "nombre": "Proveedor B",
      "contacto": "contactoB@example.com"
    }
  ]
})
```

Ejercicio 3: Modelado de datos para una red social

Objetivo: Diseñar un esquema para una colección `usuarios` que almacene información sobre usuarios, publicaciones y amigos.

Instrucciones:

1. Diseña una colección `usuarios` que contenga información sobre el nombre del usuario, la fecha de nacimiento, las publicaciones (texto y fecha) y una lista de amigos (nombre y fecha de amistad).
2. Inserta un documento de ejemplo en la colección `usuarios`.

json

```
{  
    "nombre": "Juan Pérez",  
    "fecha_nacimiento": "1990-01-15",  
    "publicaciones": [  
        {  
            "texto": "¡Hola, mundo!",  
            "fecha": "2023-07-01"  
        },  
        {  
            "texto": "Este es mi segundo post.",  
            "fecha": "2023-07-02"  
        }  
    ],  
    "amigos": [  
        {  
            "nombre": "Ana Gómez",  
            "fecha_amistad": "2019-05-20"  
        },  
        {  
            "nombre": "Luis Ramírez",  
            "fecha_amistad": "2020-10-15"  
        }  
    ]  
}
```

```
sh
```

```
db.usuarios.insertOne({  
    "nombre": "Juan Pérez",  
    "fecha_nacimiento": ISODate("1990-01-15T00:00:00Z"),  
    "publicaciones": [  
        {  
            "texto": "¡Hola, mundo!",  
            "fecha": ISODate("2023-07-01T00:00:00Z")  
        },  
        {  
            "texto": "Este es mi segundo post.",  
            "fecha": ISODate("2023-07-02T00:00:00Z")  
        }  
    ],  
    "amigos": [  
        {  
            "nombre": "Ana Gómez",  
            "fecha_amistad": ISODate("2019-05-20T00:00:00Z")  
        },  
        {  
            "nombre": "Luis Ramírez",  
            "fecha_amistad": ISODate("2020-10-15T00:00:00Z")  
        }  
    ]  
})
```

Ejercicio 4: Modelado de datos para un sistema de reservas de hoteles

Objetivo: Diseñar un esquema para una colección `reservas` que almacene información sobre reservas de habitaciones de hotel.

Instrucciones:

1. Diseña una colección `reservas` que contenga información sobre el nombre del cliente, la fecha de la reserva, la habitación (número y tipo) y las fechas de check-in y check-out.
2. Inserta un documento de ejemplo en la colección `reservas`.

json

```
{  
    "nombre_cliente": "Maria Garcia",  
    "fecha_reserva": "2023-06-01",  
    "habitacion": {  
        "numero": 101,  
        "tipo": "Suite"  
    },  
    "check_in": "2023-06-15",  
    "check_out": "2023-06-20"  
}
```

```
sh

db.reservas.insertOne({
  "nombre_cliente": "María García",
  "fecha_reserva": ISODate("2023-06-01T00:00:00Z"),
  "habitacion": {
    "numero": 101,
    "tipo": "Suite"
  },
  "check_in": ISODate("2023-06-15T00:00:00Z"),
  "check_out": ISODate("2023-06-20T00:00:00Z")
})
```

Ejercicio 5: Modelado de datos para un sistema de gestión de cursos

Objetivo: Diseñar un esquema para una colección `cursos` que almacene información sobre cursos, instructores y estudiantes.

Instrucciones:

1. Diseña una colección `cursos` que contenga información sobre el nombre del curso, el instructor (nombre y especialidad), y una lista de estudiantes (nombre y fecha de inscripción).
2. Inserta un documento de ejemplo en la colección `cursos`.

```
json

{
  "nombre_curso": "Introducción a MongoDB",
  "instructor": {
    "nombre": "Carlos López",
    "especialidad": "Bases de Datos NoSQL"
  },
  "estudiantes": [
    {
      "nombre": "Luisa Fernández",
      "fecha_inscripcion": "2023-01-10"
    },
    {
      "nombre": "Pedro Sánchez",
      "fecha_inscripcion": "2023-01-15"
    }
  ]
}
```

```
sh

db.cursos.insertOne({
  "nombre_curso": "Introducción a MongoDB",
  "instructor": {
    "nombre": "Carlos López",
    "especialidad": "Bases de Datos NoSQL"
  },
  "estudiantes": [
    {
      "nombre": "Luisa Fernández",
      "fecha_inscripcion": ISODate("2023-01-10T00:00:00Z")
    },
    {
      "nombre": "Pedro Sánchez",
      "fecha_inscripcion": ISODate("2023-01-15T00:00:00Z")
    }
  ]
})
```

Ejercicio 6: Modelado de datos para un sistema de pedidos de comida

Objetivo: Diseñar un esquema para una colección `pedidos` que almacene información sobre pedidos de comida, clientes y artículos.

Instrucciones:

1. Diseña una colección `pedidos` que contenga información sobre el nombre del cliente, la fecha del pedido, y una lista de artículos (nombre, cantidad y precio).
2. Inserta un documento de ejemplo en la colección `pedidos`.

json

```
{  
    "nombre_cliente": "José Martínez",  
    "fecha_pedido": "2023-07-15",  
    "articulos": [  
        {  
            "nombre": "Pizza Margherita",  
            "cantidad": 2,  
            "precio": 8.99  
        },  
        {  
            "nombre": "Ensalada César",  
            "cantidad": 1,  
            "precio": 5.99  
        }  
    ]  
}
```

```
sh

db_pedidos.insertOne({
  "nombre_cliente": "José Martínez",
  "fecha_pedido": ISODate("2023-07-15T00:00:00Z"),
  "articulos": [
    {
      "nombre": "Pizza Margherita",
      "cantidad": 2,
      "precio": 8.99
    },
    {
      "nombre": "Ensalada César",
      "cantidad": 1,
      "precio": 5.99
    }
  ]
})
```

Ejercicio 7: Modelado de datos para un sistema de gestión de empleados

Objetivo: Diseñar un esquema para una colección `empleados` que almacene información sobre empleados, departamentos y salarios.

Instrucciones:

1. Diseña una colección `empleados` que contenga información sobre el nombre del empleado, la fecha de contratación, el departamento (nombre y ubicación) y el salario.
2. Inserta un documento de ejemplo en la colección `empleados`.

json

```
{  
    "nombre": "Laura Rodríguez",  
    "fecha_contratacion": "2022-03-01",  
    "departamento": {  
        "nombre": "Recursos Humanos",  
        "ubicacion": "Edificio A"  
    },  
    "salario": 45000  
}
```

```
sh

db.empleados.insertOne({
  "nombre": "Laura Rodríguez",
  "fecha_contratacion": ISODate("2022-03-01T00:00:00Z"),
  "departamento": {
    "nombre": "Recursos Humanos",
    "ubicacion": "Edificio A"
  },
  "salario": 45000
})
```

Ejercicio 8: Modelado de datos para un sistema de gestión de proyectos

Objetivo: Diseñar un esquema para una colección `proyectos` que almacene información sobre proyectos, empleados y tareas.

Instrucciones:

1. Diseña una colección `proyectos` que contenga información sobre el nombre del proyecto, el líder del proyecto (nombre y departamento), y una lista de tareas (nombre, responsable y fecha límite).
2. Inserta un documento de ejemplo en la colección `proyectos`.

```
json

{
    "nombre_proyecto": "Desarrollo de Nueva Página Web",
    "lider_proyecto": {
        "nombre": "Miguel Torres",
        "departamento": "Tecnología"
    },
    "tareas": [
        {
            "nombre": "Diseño de la Interfaz",
            "responsable": "Ana Gómez",
            "fecha_limite": "2023-08-01"
        },
        {
            "nombre": "Desarrollo del Backend",
            "responsable": "Luis Ramirez",
            "fecha_limite": "2023-09-15"
        }
    ]
}
```

```
sh

db.proyectos.insertOne({
    "nombre_proyecto": "Desarrollo de Nueva Página Web",
    "lider_proyecto": {
        "nombre": "Miguel Torres",
        "departamento": "Tecnología"
    },
    "tareas": [
        {
            "nombre": "Diseño de la Interfaz",
            "responsable": "Ana Gómez",
            "fecha_limite": ISODate("2023-08-01T00:00:00Z")
        },
        {
            "nombre": "Desarrollo del Backend",
            "responsable": "Luis Ramírez",
            "fecha_limite": ISODate("2023-09-15T00:00:00Z")
        }
    ]
})
```

Ejercicio 9: Modelado de datos para un sistema de seguimiento de inventario

Objetivo: Diseñar un esquema para una colección `inventario` que almacene información sobre productos, proveedores y cantidades.

Instrucciones:

1. Diseña una colección `inventario` que contenga información sobre el nombre del producto, el proveedor (nombre y contacto), la cantidad disponible y la ubicación en el almacén.
2. Inserta un documento de ejemplo en la colección `inventario`.

json

```
{  
    "nombre_producto": "Laptop",  
    "proveedor": {  
        "nombre": "Proveedor A",  
        "contacto": "contactoA@example.com"  
    },  
    "cantidad": 50,  
    "ubicacion": "Estante 3B"  
}
```

```
sh

db.inventario.insertOne({
  "nombre_producto": "Laptop",
  "proveedor": {
    "nombre": "Proveedor A",
    "contacto": "contactoA@example.com"
  },
  "cantidad": 50,
  "ubicacion": "Estante 3B"
})
```

Ejercicio 10: Modelado de datos para un sistema de gestión de eventos

Objetivo: Diseñar un esquema para una colección `eventos` que almacene información sobre eventos, organizadores y asistentes.

Instrucciones:

1. Diseña una colección `eventos` que contenga información sobre el nombre del evento, el organizador (nombre y contacto), la fecha y lugar del evento, y una lista de asistentes (nombre y confirmación de asistencia).
2. Inserta un documento de ejemplo en la colección `eventos`.

```
json
```

```
{  
    "nombre_evento": "Conferencia de Tecnología",  
    "organizador": {  
        "nombre": "Juan Pérez",  
        "contacto": "juanperez@example.com"  
    },  
    "fecha": "2023-09-25",  
    "lugar": "Centro de Convenciones",  
    "asistentes": [  
        {  
            "nombre": "Ana Gómez",  
            "confirmacion_asistencia": true  
        },  
        {  
            "nombre": "Luis Ramírez",  
            "confirmacion_asistencia": false  
        }  
    ]  
}
```

```
sh
```

```
db.eventos.insertOne({  
    "nombre_evento": "Conferencia de Tecnología",  
    "organizador": {  
        "nombre": "Juan Pérez",  
        "contacto": "juanperez@example.com"  
    },  
    "fecha": ISODate("2023-09-25T00:00:00Z"),  
    "lugar": "Centro de Convenciones",  
    "asistentes": [  
        {  
            "nombre": "Ana Gómez",  
            "confirmacion_asistencia": true  
        },  
        {  
            "nombre": "Luis Ramírez",  
            "confirmacion_asistencia": false  
        }  
    ]  
})
```

JavaScript en MongoDB

Utilizar JavaScript en MongoDB Shell

El shell de MongoDB es una **interfaz interactiva de Javascript**. Como tal, brinda la capacidad de usar código JavaScript directamente en el shell o ejecutarlo como un archivo independiente. En los temas subsiguientes - que tratan sobre el uso del shell para acceder a la base de datos y crear y manipular colecciones y documentos - se proporcionan ejemplos que están escritos en JavaScript. Para seguirlos, es necesario **comprender al menos algunos de los aspectos fundamentales del lenguaje**.

Se verán algunos de los conceptos básicos, como **variables, funciones y objetos**. Este recorrido no pretende ser una guía completa de JavaScript, sino una sinopsis de la sintaxis y los modismos importantes. Será de gran utilidad para los alumnos que no estén familiarizados con el lenguaje, ya que obtendrán información para comprender los mecanismos de consulta e interacción hacia la base de datos, desde un script con JavaScript.

JavaScript en MongoDB

Sintaxis de la definición de una variable: se utiliza la palabra clave **var** y luego se escribe el nombre que se le dará, como en este ejemplo:

```
var myData;
```

También puede **asignar un valor a la variable en la misma línea**. Por ejemplo, el siguiente código crea una variable denominada **myString** y le asigna el valor de "Some Text":

```
var myString = "Un texto";
```

El código anterior funciona tan bien como éste:

```
var myString;  
myString = "Un texto";
```

Después de haber declarado la variable, **se puede usar su nombre para asignarle un valor y para acceder a ese valor**. Por ejemplo, el siguiente código almacena una cadena en la variable **myString** y luego la usa al asignar el valor a la variable **newString**:

```
var myString = "Un texto";  
var newString = myString + "Más texto";
```

JavaScript en MongoDB

String

Esta variable almacena **cadenas de caracteres**. Los datos de caracteres se especifican mediante comillas simples o dobles. Todos los datos contenidos en las comillas se asignan a la variable de cadena.

Por ejemplo:

```
var myString = 'Un texto';
var anotherString = 'Más texto';
```

Number

Los datos se almacenan como **valor numérico**. Los números son útiles en conteos, cálculos y comparaciones.

Algunos ejemplos:

```
var myInteger = 1;
var cost = 1.33;
```

JavaScript en MongoDB

Boolean

Esta variable almacena **un solo bit que es true o false**. Los booleanos se utilizan a menudo para las banderas. Por ejemplo, se puede establecer una variable como `false` al comienzo de algún código y luego verificarla al finalizar para ver si la ejecución llegó a un punto determinado. A continuación vemos un ejemplo de definición de una variable `true` y una variable `false`:

```
var yes = true;  
var no = false;
```

Array

Un array o matriz indexada es **una serie de elementos de datos distintos**, almacenados bajo un solo nombre de variable. Se puede acceder a cada elemento de la matriz mediante su índice de base cero: `array[index]`. El siguiente es un ejemplo de cómo crear una matriz simple y luego acceder al primer elemento, que se encuentra en el índice 0.

```
var arr = ["uno", "dos", "tres"];  
var first = arr[0];
```

JavaScript en MongoDB

Objeto literal

JavaScript admite la capacidad de crear y utilizar objetos literales. Cuando usa un objeto literal, puede acceder a valores y funciones en el objeto usando la sintaxis `object.property`.

El siguiente ejemplo muestra cómo crear y acceder a propiedades con un objeto literal:

```
var obj = {"nombre":"Carlos", "ocupacion":"Médico", "edad", "Desconocida"};
var nombre = obj.nombre;
```

JavaScript en MongoDB

```
>_MONGOSH
> var obj = {"nombre":"Carlos", "ocupacion":"Médico", "edad": "Desconocida"};
> obj.ocupacion
< Médico
> obj
< { nombre: 'Carlos', ocupacion: 'Médico', edad: 'Desconocida' }
test>
```

JavaScript en MongoDB

Nulo

A veces no hay un valor para almacenar en una variable porque no se ha creado o ya no la está usando. En este momento, puede establecer la variable en `null`. **Usar null es mejor que asignar a la variable un valor de 0 o una cadena vacía ""** porque esos pueden ser valores válidos para la variable.

Asignar la variable `null` le permite no asignar ningún valor y verificar `null` dentro de su código.

```
var newVar = null;
```

JavaScript en MongoDB

Salida de datos en un script de shell de MongoDB

Se utilizan cuatro formas para generar datos desde el script de shell de MongoDB:

```
print(data, ...);  
printjson(object);  
print(tojson(object));  
print(JSON.stringify(object));
```

JavaScript en MongoDB

```
>_MONGOSH

> var obj = {"nombre":"Carlos", "ocupacion":"Médico", "edad": "Desconocida"};
> obj.ocupacion
< Médico
> obj
< { nombre: 'Carlos', ocupacion: 'Médico', edad: 'Desconocida' }
> printjson(obj)
< { nombre: 'Carlos', ocupacion: 'Médico', edad: 'Desconocida' }
> print(JSON.stringify(obj));
< {"nombre":"Carlos","ocupacion":"Médico","edad":"Desconocida"}
```

JavaScript en MongoDB

El método **print()** simplemente imprime los datos que se le pasan como argumento. Por ejemplo, la declaración:

```
print("Hola desde Mongo");
```

da como resultado esta salida:

```
Hola desde Mongo
```



Si se proporcionan varios elementos de datos, se imprimen juntos. Por ejemplo, la declaración:

```
print("Hola", "desde", "Mongo");
```

también genera esto:

```
Hola desde Mongo
```

El método **printjson()** imprime una bonita forma del objeto JavaScript.

Los métodos **print(JSON.stringify(object))** y **print(tojson(object))** también imprimen una forma de cadena muy compacta de un objeto JavaScript.

JavaScript en MongoDB

Operadores aritméticos

Se utilizan para realizar operaciones entre valores variables y directos. Esta tabla enumera las operaciones aritméticas, junto con los resultados que se aplican.

Operador	Descripción	Ejemplo	Resultado en x
+	Suma	<code>x=y+5</code> <code>x=y+"5"</code> <code>x="Four"+y+"4"</code>	9 "49" "Four44"
-	Resta	<code>x=y-2</code>	2
++	Incremento	<code>x=y++</code> <code>x=++y</code>	4 5
--	Decremento	<code>x=y--</code> <code>x=- - y</code>	4 3
*	Multiplicación	<code>x=y*4</code>	16
/	División	<code>x=10/y</code>	2.5
%	Módulo (resto de la división)	<code>x=y%3</code>	1

Operadores aritméticos de JavaScript:
resultados basados en un valor inicial y=4.

JavaScript en MongoDB

Operadores de asignación

Asignan un valor a una variable. Además del operador `=`, varios formularios permiten manipular los datos a medida que se asigna el valor. Esta tabla enumera las operaciones de asignación, junto con los resultados que se aplican.



Operador	Ejemplo	Operaciones aritméticas equivalentes	Resultado en x
<code>=</code>	<code>x=5</code>	<code>x=5</code>	5
<code>+=</code>	<code>x+=5</code>	<code>x=x+5</code>	15
<code>-=</code>	<code>x-=5</code>	<code>x=x-5</code>	5
<code>*=</code>	<code>x*=5</code>	<code>x=x*5</code>	50
<code>/=</code>	<code>x/=5</code>	<code>x=x/5</code>	2
<code>%=</code>	<code>x%=5</code>	<code>x=x%5</code>	0

Operadores de asignación de JavaScript:
resultados basados en el valor inicial `x=10`.

JavaScript en MongoDB

Operadores de comparación

Un operador de comparación evalúa dos datos y **devuelve true si la evaluación es correcta o false si la evaluación no es correcta.**

Los operadores de comparación comparan el valor a la izquierda del operador con el valor a la derecha. La forma más sencilla de comprender la sintaxis de comparación de JavaScript es ver una lista con algunos posibles casos. La siguiente tabla enumera los operadores de comparación, junto con algunos ejemplos.

Operador	Descripción	Ejemplo	Resultado
<code>==</code>	Es igual a (sólo valor)	<code>x==8 x==10</code>	<code>false true</code>
<code>===</code>	Igualdad de valor y tipo	<code>x === 10 x === "10"</code>	<code>true false</code>
<code>!=</code>	No es igual	<code>x != 5</code>	<code>true</code>
<code>!==</code>	Desigualdad de valor y tipo	<code>x !== "10" x !== 10</code>	<code>true false</code>
<code>></code>	Es mayor a	<code>x > 5</code>	<code>true</code>
<code>>=</code>	Es mayor o igual a	<code>x >= 10</code>	<code>true</code>
<code><</code>	Es menor a	<code>x < 5</code>	<code>false</code>
<code><=</code>	Es menor o igual a	<code>x <= 10</code>	<code>true</code>

Operadores de comparación de JavaScript:
resultados basados en un valor inicial de `x=10`.

JavaScript en MongoDB

Podemos encadenar múltiples comparaciones usando operadores lógicos y paréntesis estándar. La tabla por debajo enumera los operadores lógicos y explica cómo usarlos para encadenar comparaciones.

Operador	Descripción	Ejemplo	Resultado
<code>&&</code>	y	<code>(x==10 && y==5)</code> <code>(x==10 && y>x)</code>	<code>true</code> <code>false</code>
<code> </code>	o	<code>(x>=10 y>x)</code> <code>(x<10 && y>x)</code>	<code>true</code> <code>false</code>
<code>!</code>	no combinados	<code>! (x==y)</code> <code>! (x>y)</code> <code>(x>=10 && y<x x==y)</code> <code>((x<y x>=10) && y>=5)</code> <code>(! (x==y) && y>=10)</code>	<code>true</code> <code>false</code> <code>true</code> <code>true</code> <code>false</code>
<code>&&</code>	y	<code>(x==10 && y==5)</code> <code>(x==10 && y>x)</code>	<code>true</code> <code>false</code>

Operadores de comparación de JavaScript: resultados basados en valores iniciales de $x=10$ e $y=5$.

JavaScript en MongoDB

```
>_MONGOSH  
> x=10  
< 10  
> x+=20  
< 30  
test>
```

```
>_MONGOSH  
> y = 5  
< 5  
> x = 10  
< 10  
> (x==10 && y==5)  
< true  
> y = 6  
< 6  
> (x==10 && y==5)  
< false  
test>|
```

```
>_MONGOSH  
> edad = 30  
< 30  
> edad == 30  
< true  
> edad == '30'  
< true  
> edad === '30'  
< false  
test>
```

JavaScript en MongoDB

Uso de sentencias if

La sentencia **if** permite **separar la ejecución del código en función de la evaluación de una comparación**. Las siguientes líneas de código muestran la sintaxis. Los operadores condicionales se escriben entre paréntesis, y el código a ejecutar, si el condicional se evalúa como true, se indica entre corchetes (**{}**):

```
if(x==5){  
    hacer_algo();  
}
```

Además de ejecutar el código que se encuentra dentro del bloque **if** de instrucciones, se puede especificar un bloque **else** que se ejecute solo si la condición es false.

Por ejemplo:

```
if(x==5){  
    hacer_algo();  
} else {  
    hacer_algo_else();  
}
```

JavaScript en MongoDB

También es posible encadenar ó anidar sentencias **if**.
Para hacer esto, se agrega una declaración condicional
junto con una declaración **else**.

Por ejemplo:

```
if(x<5){  
    hacer_algo();  
} else if(x<10) {  
    hacer_algo_else();  
} else {  
    hacer_algo();  
}
```

JavaScript en MongoDB

Implementación de sentencias switch

Otro tipo de lógica condicional es la sentencia **switch**.

Permite evaluar una expresión una vez y luego, según el valor, ejecutar una de las muchas secciones diferentes de código. La sintaxis es la siguiente:

```
switch(expresión){  
    case value1:  
        < código a ejecutar>  
        break;  
    case value2:  
        < código a ejecutar>  
        break;  
    default:  
        < código a ejecutar si no se cumplen value1 o value2>  
}
```

JavaScript en MongoDB

Bucles while

Es el bucle más básico en JavaScript. Un ciclo **while** prueba una expresión y continúa ejecutando el código contenido entre {} corchetes hasta que la expresión se evalúe como false.

Por ejemplo, el siguiente bucle while se ejecuta hasta que el valor de **i** es igual a 5:

```
var i = 1;
while (i<5){
    print("Iteración" + i + "\n");
    i++;
}
```

La salida resultante que aparecerá en la consola:

```
Iteración 1
Iteración 2
Iteración 3
Iteración 4
```

JavaScript en MongoDB

Bucles do / while

Otro tipo de bucle **while** es el bucle **do/while**. Se usa cuando se desea ejecutar el código contenido dentro del ciclo, al menos, una vez. Es decir, la expresión no se puede probar hasta que el código se haya ejecutado al menos una vez.

Por ejemplo, el siguiente bucle **do/while** se ejecuta hasta que el valor de day es igual a miércoles.

```
var dias = ["lunes", "martes", "miércoles",
            "jueves", "viernes"];
var i=0;
do{
    var dia=dias[i++];
    print("Es" + dia + "\n");
} while (dia != "miércoles");
```

La salida resultante por consola es:

```
Es lunes
Es martes
Es miércoles
```

JavaScript en MongoDB

Bucle for

Permite ejecutar código una cantidad específica de veces mediante una declaración **for** que combina tres declaraciones en un solo bloque de ejecución utilizando la siguiente sintaxis:

```
for (asignación; condición; actualización;){  
    código a ejecutar;  
}
```

La sentencia **for** utiliza esas tres declaraciones, al ejecutar el ciclo, como se muestra a la derecha:

- **Asignación:** se ejecuta una única vez, antes de que comience el bucle. De esta manera se inicializan las variables utilizadas en el ciclo como condicionales.
- **Condición:** se evalúa antes de cada iteración del ciclo. Si la expresión se evalúa como true, se ejecuta el ciclo; de lo contrario, **for** finaliza la ejecución del bucle.
- **Actualización:** ejecutó cada iteración después de que se haya ejecutado el código en el ciclo. Su uso típico es incrementar un contador usado en la instrucción 2.

JavaScript en MongoDB

Este ejemplo no solo ilustra un bucle básico **for**, sino que también muestra la capacidad de anidar un bucle dentro de otro:

La salida resultante por consola es:

```
for (var x=1; x<=3; x++){
  for (var y=1; y<=3; y++){
    print(x + " X " + y + " = " + (x*y) + "\n");
  }
}
```

```
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
```

JavaScript en MongoDB

Bucle for in

Otro tipo de bucle **for** es el bucle **for/ in**, que se ejecuta en cualquier tipo de datos que se pueda iterar. En su mayor parte, se usa el bucle **for/ in** en matrices y objetos. El siguiente ejemplo ilustra la sintaxis y el comportamiento del ciclo **for/ in** en una matriz simple:

```
var dias = ["lunes", "martes", "miércoles",
            "jueves", "viernes"];
for (var idx in dias){
    print("Es " + dias[idx] + "\n");
}
```

La variable **idx** se ajusta en cada iteración a través del bucle desde el índice de matriz inicial hasta el último. La salida resultante es:

```
Es lunes
Es martes
Es miércoles
Es jueves
Es viernes
```

JavaScript en MongoDB

Bucle con método **foreach**

Este método entra dentro del grupo de Iterables sin devolver una nueva matriz, lo que hace el **forEach** es ejecutar una función por cada elemento del arreglo. En cada iteración se tendrá acceso a 3 variables: valor (del elemento), índice (del elemento) y arreglo (que se está recorriendo).

Este método es muy útil cuando solo se necesita ejecutar una función a través de cada elemento del arreglo, sin necesidad de obtener un retorno.

```
const cars = ['Ferrari 250 GT Berlinetta.', 'Tesla S', 'Génesis G90', 'Porsche Boxster'];
//Con una función de devolución ES5
cars.forEach(function (element) {
  console.log(element);
});
```

```
//output Ferrari 250 GT Berlinetta
//output Tesla S
//output Génesis G90
//output Porsche Boxster
```

JavaScript en MongoDB

Definición de funciones

Las funciones se definen mediante la palabra clave **function** seguida de un nombre que describe el uso de la función, una lista de cero o más argumentos entre paréntesis y un bloque de una o más declaraciones de código entre **{}** corchetes.



Por ejemplo, la siguiente es una definición de función que escribe "Hola Mundo" en la consola.

```
function myFunction(){  
    print("Hola Mundo");  
}
```

Para ejecutar el código incluido en `myFunction()`, todo lo que se necesita hacer es agregar la siguiente línea al JavaScript principal o dentro de otra función:

```
myFunction();
```

JavaScript en MongoDB

Pasar variables a funciones

Con frecuencia, se deben pasar valores específicos a las funciones que se utilizarán. Los valores se pasan delimitados por comas. La definición de la función necesita una lista de nombres de variables entre paréntesis () que coincidan con el número que se pasa. Por ejemplo, esta función acepta dos argumentos, **nombre** y **ciudad**, y los usa para construir la cadena de salida:

```
function saludo(nombre, ciudad){  
    print("Hola " + nombre);  
    print("Cómo está el clima en " + ciudad);  
}
```

Para llamar a la función `saludo()`, se debe pasar el valor **nombre** y un valor para **ciudad**. El valor puede ser directo o una variable previamente definida. Para ilustrar esto, el siguiente código ejecuta la función `saludo()` con una variable **nombre** y una cadena directa para **ciudad**:

```
var nombre = "Pablo";  
greeting(nombre, "Florencia");
```

JavaScript en MongoDB

Devolver valores de funciones

Con frecuencia, las funciones necesitan devolver un valor al código de llamada. Agregar la palabra clave **return** seguida de una variable o valor devuelve ese valor de la función.

Por ejemplo, el código de la derecha llama a una función para formatear una cadena, asigna el valor devuelto por la función a una variable y luego escribe el valor en la consola:

```
function formatoSaludo(nombre, ciudad){  
    var retStr = "";  
    retStr += "Hola " + nombre + "\n";  
    retStr += "Bienvenido a " + ciudad + "!";  
    return retStr;  
}  
var saludo = formatoSaludo("Carlos", "Roma");  
print(saludo);
```

JavaScript en MongoDB

Se puede incluir más de una instrucción **return** en la función. Cuando la función encuentra una declaración **return**, la ejecución del código de la función se detiene inmediatamente. Si la declaración de devolución contiene un valor para devolver, se devuelve ese valor.

El siguiente ejemplo muestra una función que prueba la entrada y regresa inmediatamente si es cero.

```
function miFunc(valor){  
    if (valor == 0)  
        return valor;  
    < código a ejecutar si el valor es distinto de cero>  
    return valor;  
}
```

JavaScript en MongoDB

```
>_MONGOSH
> function formatoSaludo(nombre, ciudad){
    var retStr = "";
    retStr += "Hola " + nombre + "\n";
    retStr += "Bienvenido a " + ciudad + "!";
    return retStr;
}
< [Function: formatoSaludo]
> var saludo = formatoSaludo("Carlos", "Roma");
> print(saludo);
< Hola Carlos
    Bienvenido a Roma!
test>
```

JavaScript en MongoDB

Manipulación de cadenas

El objeto **String** es el más utilizado en JavaScript.
JavaScript crea automáticamente un objeto String cada vez que se define una variable del tipo de datos de cadena. Por ejemplo:

```
var myStr = "Aprende NoSQL con MongoDB en 24 horas";
```

Al crear una cadena, varios caracteres especiales no se pueden agregar directamente. Para estos caracteres, JavaScript proporciona un conjunto de códigos de escape, veamos la tabla de la siguiente slide.

JavaScript en MongoDB

Escape	Descripción	Ejemplo	Cadena de salida
\'	Comilla simple	"couldn\t be"	couldn't be
\\"	Comillas dobles	"Yo \ "pienso\\" Yo \"soy\\""	Yo "pienso" Yo "soy"
\\"\\	Barra invertida	"uno\\dos\\tres"	uno\dos\tres
\n	Nueva línea	"Yo soy\nYo dije"	Yo soy Yo dije
\r	Retorno de carro	"ser\rro no ser"	Ser o no ser
\t	Tabulación	"uno\todos\ttres"	uno dos tres
\b	Backspace	"correctoin\b\b\bion"	correction
\f	Form feed	"Title A\fTitle B"	Title A then Title B

JavaScript en MongoDB

Para obtener la longitud de la cadena, se puede usar la propiedad **length** del objeto **String**.

Por ejemplo:

```
var numOfChars = myStr.length;
```

El objeto **String** tiene varias funciones que le permiten acceder y manipular la cadena de varias maneras. La tabla que se muestra a continuación describe los métodos de manipulación de cadenas.

JavaScript en MongoDB

Método	Description
concat (arr1, arr2, ...)	Devuelve una copia concatenada del array y los arrays pasados como argumentos.
indexOf(value)	Devuelve el índice de un valor del array o -1 si no se encuentra el elemento.
join(separator)	Une todos los elementos de un array, que están separados por el separador indicado, en una sola cadena. Si no se especifica ningún separador, se utiliza una coma.
lastIndexOf(value)	Devuelve el último índice del valor en el array o -1 si no se encuentra el valor.
pop()	Elimina el último elemento de la matriz y devuelve ese elemento.
push(item1, item2, ...)	Agrega uno o más elementos nuevos al final de una matriz y devuelve la nueva longitud.
reverse()	Invierte el orden de todos los elementos del array.
shift()	Elimina el primer elemento de un array y devuelve ese elemento.
slice(start, end)	Devuelve los elementos entre el índice inicial y final.
sort(sortFunction)	Ordena los elementos del array. La función sortFunction es opcional.
splice(index, count, item1, item2...)	En el índice especificado, se elimina el número de elementos que indica count. Luego, los elementos opcionales que se pasan como argumentos se insertan en el índice.
toString()	Devuelve el array como cadena.
unshift()	Agrega nuevos elementos al comienzo de un array y devuelve la nueva longitud.
valueOf()	Devuelve el valor primitivo de un arreglo de objetos.

JavaScript en MongoDB

Combinación de cadenas

Se pueden combinar varias cadenas usando el signo `+` o la función `concat()` en la primera cadena. Por ejemplo, en el siguiente código, `sentencia1` y `sentencia2` harán lo mismo:

```
var palabra1 = "Hoy ";
var palabra2 = "es ";
var palabra3 = "mañana\' ";
var palabra4 = "ayer.";
var sentencia1 = palabra1 + palabra2 + palabra3 + palabra4;
var sentencia2 = palabra1.concat(palabra2, palabra3, palabra4);
```

JavaScript en MongoDB

Trabajar con matrices

El objeto **array** proporciona un medio para almacenar y manejar un conjunto de otros objetos. Las matrices pueden almacenar números, cadenas u otros objetos de JavaScript. Existen varios métodos para crear matrices de JavaScript.

Por ejemplo, las siguientes declaraciones crean tres versiones idénticas de la misma matriz:

```
var arr = ["uno", "dos", "tres"];  
  
var arr2 = new Array();  
arr2[0] = "uno";  
arr2[1] = "dos";  
arr2[2] = "tres";  
  
var arr3 = new Array();  
arr3.push("uno");  
arr3.push("dos");  
arr3.push("tres");
```

JavaScript en MongoDB

El primer método define **arr** y establece el contenido en una sola declaración usando **[]**.

El segundo método crea el objeto **arr2** y luego le agrega elementos mediante la asignación de índice directo.

El tercer método crea el objeto **arr3** y luego usa la mejor opción para extender matrices: usa el método **push()** para insertar elementos en la matriz.

Para obtener la cantidad de elementos en la matriz, usa la propiedad **length** del objeto Array. Veamos un ejemplo:

```
var numDeItems = arr.length;
```

Las matrices son un índice basado en cero, lo que significa que el primer elemento está en el índice 0, y así sucesivamente. En este ejemplo, el valor de la variable **primero** es **lunes** y el valor de la variable **ultimo** es **viernes**:

```
var semana = ["lunes", "martes",  
"miércoles", "jueves", "viernes"];  
var primero = semana[0];  
var ultimo = semana[semana.length-1];
```

El objeto Array tiene varias funciones integradas que le permiten acceder y manipular la matriz de varias maneras. La siguiente tabla describe los métodos adjuntos al objeto Array que le permiten manipular el contenido de la matriz.

JavaScript en MongoDB

Método	Description
concat (arr1, arr2, ...)	Devuelve una copia concatenada del array y los arrays pasados como argumentos.
indexOf(value)	Devuelve el índice de un valor del array o -1 si no se encuentra el elemento.
join(separator)	Une todos los elementos de un array, que están separados por el separador indicado, en una sola cadena. Si no se especifica ningún separador, se utiliza una coma.
lastIndexOf(value)	Devuelve el último índice del valor en el array o -1 si no se encuentra el valor.
pop()	Elimina el último elemento de la matriz y devuelve ese elemento.
push(item1, item2, ...)	Agrega uno o más elementos nuevos al final de una matriz y devuelve la nueva longitud.
reverse()	Invierte el orden de todos los elementos del array.
shift()	Elimina el primer elemento de un array y devuelve ese elemento.
slice(start, end)	Devuelve los elementos entre el índice inicial y final.
sort(sortFunction)	Ordena los elementos del array. La función sortFunction es opcional.
splice(index, count, item1, item2...)	En el índice especificado, se elimina el número de elementos que indica count. Luego, los elementos opcionales que se pasan como argumentos se insertan en el índice.
toString()	Devuelve el array como cadena.
unshift()	Agrega nuevos elementos al comienzo de un array y devuelve la nueva longitud.
valueOf()	Devuelve el valor primitivo de un arreglo de objetos.

JavaScript en MongoDB

Combinación de matrices

Es posible combinar matrices de la misma manera que se combinan objetos String, usando el signo **+**, instrucciones o el método **concat()**.

En el siguiente código, arr3 hace lo mismo que arr4:

```
var arr1 = [1,2,3];
var arr2 = ["tres", "cuatro", "cinco"]
var arr3 = arr1 + arr2;
var arr4 = arr1.concat(arr2);
```

JavaScript en MongoDB

Iterar a través de matrices

Es posible iterar a través de una matriz usando **for** o **for/in loop**. El siguiente código muestra la iteración a través de cada elemento de la matriz utilizando cada método:

```
var semana = ["lunes", "martes", "miércoles", "jueves", "viernes"];
for (var i=0; i<semana.length; i++){
    print(semana[i] + "\n");
}
for (dayIndex in semana){
    print(semana[dayIndex] + "\n");
}
```

JavaScript en MongoDB

Sintaxis de objetos

Para usar objetos en JavaScript de manera efectiva, se debe comprender su estructura y sintaxis. Un objeto es realmente solo un contenedor para agrupar múltiples valores y, en algunos casos, funciones. Los valores de un objeto se llaman **propiedades** y las funciones se denominan **métodos**.

Para usar un objeto de JavaScript, primero se debe crear una instancia de ese objeto. Las instancias de objetos se crean utilizando la palabra clave **new** con el nombre del constructor del objeto.

Por ejemplo, para crear un objeto **Numero**, se utiliza la siguiente línea de código:

```
var x = new Numero("5");
```

La sintaxis de objetos es sencilla: el nombre del objeto, luego un punto y luego el nombre de la propiedad o del método. Por ejemplo, estas líneas de código obtienen y establecen la propiedad **name** de un objeto denominado **myObj**:

```
var s = myObj.name;  
myObj.name = "Nuevo nombre";
```

JavaScript en MongoDB

También puede obtener y establecer métodos de un objeto de la misma manera. Por ejemplo, el código de la derecha llama al `getName()` método y luego cambian la función del método en un objeto llamado `myObj`.

También puede crear objetos en forma literal y asignar variables y funciones directamente usando la sintaxis `{}`. Por ejemplo, el código a la derecha, define un nuevo objeto y asigna valores y una función de método.

```
var name = myObj.getName();
myObj.getName = function() { return this.name;
};
```

```
var obj = {
  name: "My Object",
  value: 7,
  getValue: function() { return this.value; }
};
```

JavaScript en MongoDB

También se puede acceder a los miembros de un objeto de JavaScript utilizando la sintaxis `object[propertyName]`. Esto es útil cuando se utilizan nombres de propiedades dinámicas o si el nombre de la propiedad debe incluir caracteres que JavaScript no admite.

Los siguientes ejemplos acceden a las propiedades "Nombre de usuario" y "Otro nombre" de un objeto llamado `myObj`:

```
var propName = "Nombre de usuario";
var val1 = myObj[propName];
var val2 = myObj["Otro nombre"];
```

JavaScript en MongoDB

Creación de objetos definidos personalizados

Los objetos integrados de JavaScript tienen varias ventajas. A medida que se comience a escribir código que utilice más datos, se querrá crear **objetos personalizados** con propiedades y métodos específicos.

El objeto de JavaScript se puede definir de un par de maneras diferentes. El más simple es el **método sobre la marcha**, lo que significa que se crea un objeto genérico y luego se le agregan propiedades a medida que se necesitan.

Por ejemplo, el siguiente código crea un objeto de usuario, le asigna un nombre y apellido y define una función para devolver el nombre completo:

```
var user = new Object();
user.first="Carlos";
user.last="Pérez";
user.getName = function( ) { return
this.first + " " + this.last; }
```

JavaScript en MongoDB

Se lograría el mismo efecto a través de una **asignación directa** usando la siguiente sintaxis.
El objeto está entre {} corchetes y las propiedades se definen mediante property:value:

```
var user = {  
    first: 'Carlos',  
    last: 'Pérez',  
    getName: function( ) { return this.first + " " + this.last; }  
};
```

Estas dos primeras opciones funcionan bien para objetos simples que no se necesitan reutilizar más adelante.

JavaScript en MongoDB

Un mejor método para los objetos reutilizables es **encerrar el objeto dentro de su propio bloque de funciones**. Tiene la ventaja de permitirle mantener todo el código perteneciente al objeto local para el propio objeto.

Veamos:

```
function User(first, last){  
    this.first = first;  
    this.last = last;  
    this.getName = function( ) { return this.first + " " + this.last; }  
};  
var user = new User("Carlos", "Pérez");
```

El resultado final de estos métodos es esencialmente el mismo. Se tiene un objeto con propiedades a las que se puede hacer referencia mediante la sintaxis de puntos:

```
print(user.getName());
```

JavaScript en MongoDB

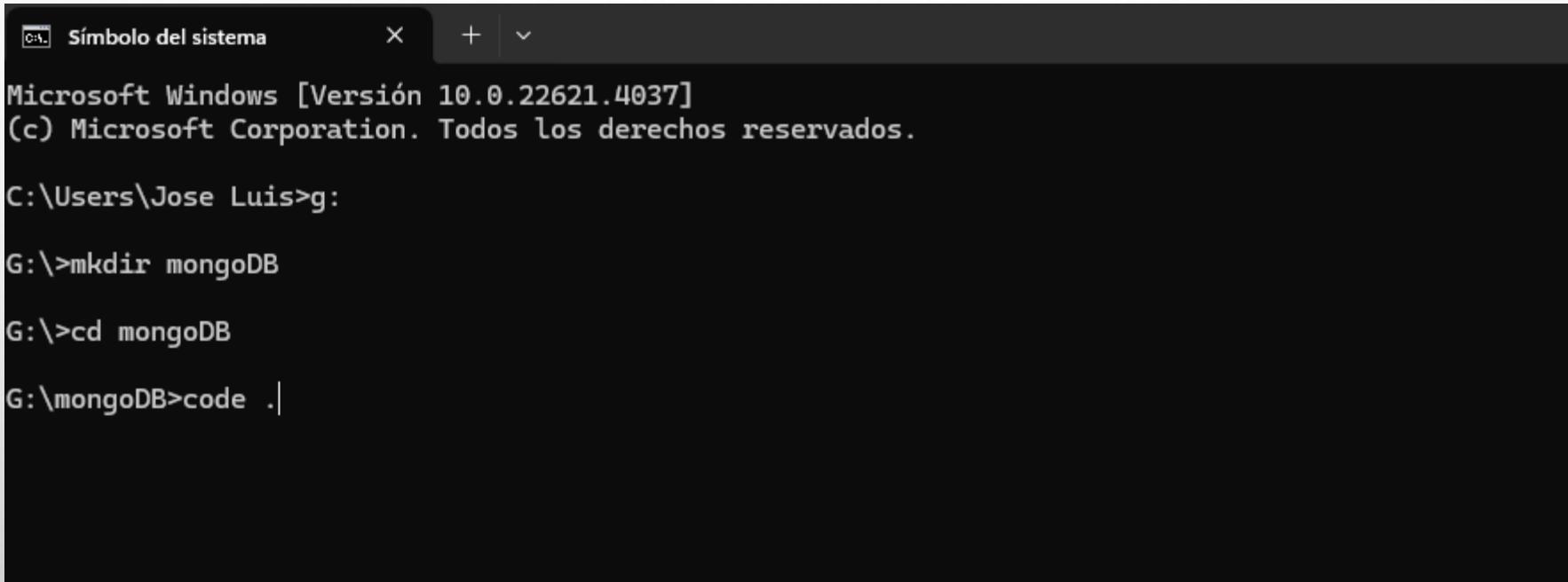
Uso de JavaScript en Mongo Shell

Cargar y ejecutar un archivo *.js

Desde la aplicación de consola de MongoDB (shell), se pueden **ejecutar comandos JavaScript en línea que pueden estar en un archivo *.js** y se pueden cargar, simplemente, mediante un comando. Para implementar un script de mediana o gran complejidad, lo más adecuado es **utilizar un editor de texto** y grabarlo en un archivo *.js

Por ejemplo, se crea la carpeta *scripts* dentro de la carpeta actual de trabajo y dentro de ella un archivo llamado '*creacion.js*'. Luego, con el comando "**code .**", se lanzará la ejecución del editor de código Visual Studio Code (VSC).

JavaScript en MongoDB

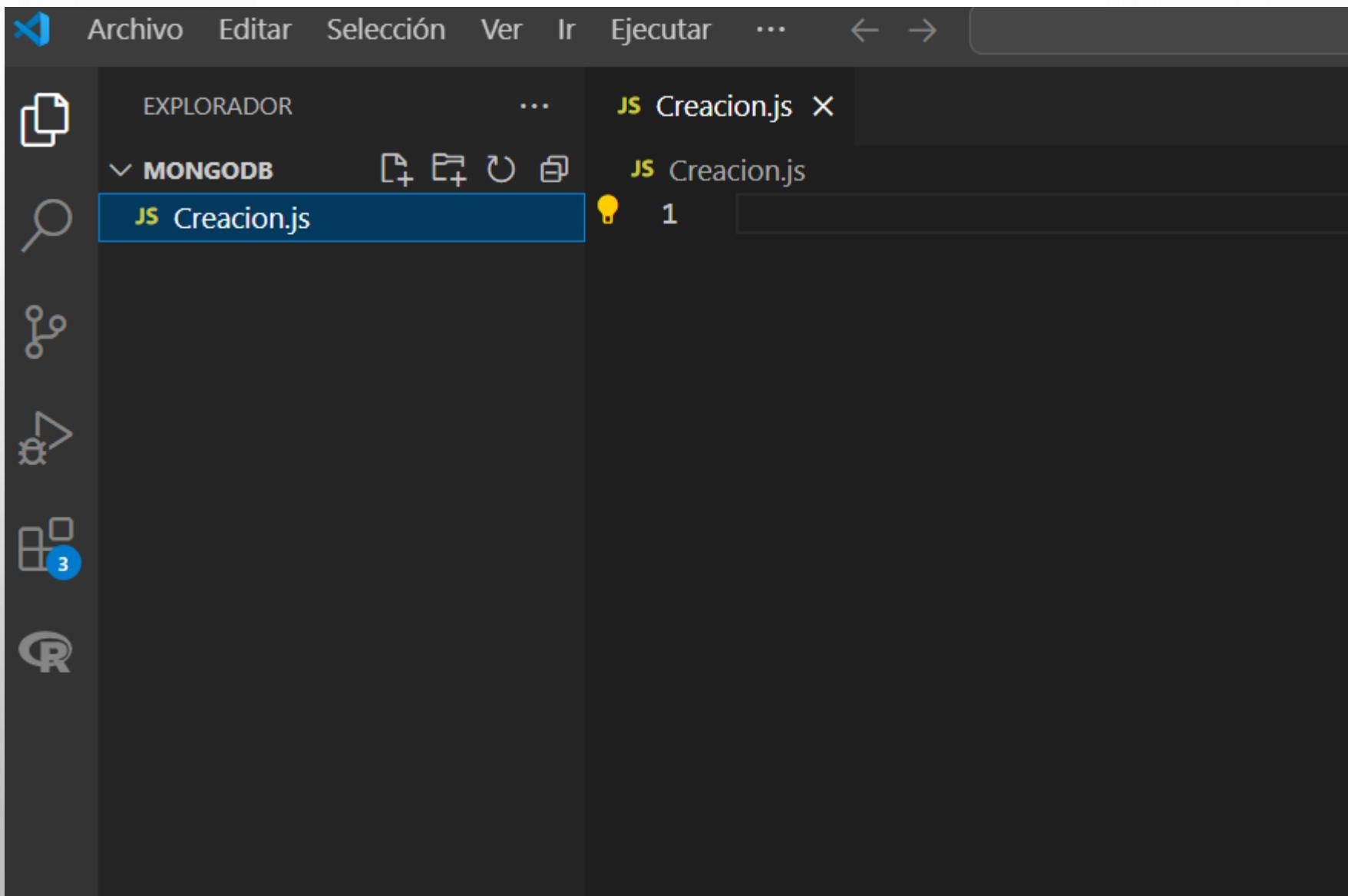


A screenshot of a Windows command prompt window titled "Símbolo del sistema". The window shows the following text:

```
Microsoft Windows [Versión 10.0.22621.4037]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Jose Luis>g:
G:\>mkdir mongoDB
G:\>cd mongoDB
G:\mongoDB>code .|
```

JavaScript en MongoDB



JavaScript en MongoDB

The screenshot shows a dark-themed code editor interface. On the left, there's a vertical toolbar with icons for file operations, search, and other development tools. The main area has a tab bar with 'JS Creacion.js X' and 'JS Creacion.js > ...'. Below the tabs is a code editor with the following content:

```
1 db.articulos.drop()
2 for(i = 1; i <= 10; i++) {
3   db.articulos.insertOne(
4     {
5       _id: i,
6       nombre: 'nombre'+i
7     }
8   )
9 }
```

Below the code editor, there are several tabs: PROBLEMAS, SALIDA, CONSOLA DE DEPURACIÓN, TERMINAL, and PUERTOS. The TERMINAL tab is selected, showing the output of a mongosh command:

```
PS C:\Users\Jose Luis\AppData\Local\Programs\mongosh> .\mongosh.exe
Current Mongosh Log ID: 66c26f8208cb33e2232710bb
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.3.0
Using MongoDB:      7.0.12
For mongosh info see: https://www.mongodb.com/docs/mongodb-shell/

To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy).
You can opt-out by running the disableTelemetry() command.

-----
The server generated these startup warnings when booting
2024-08-18T18:35:48.386-03:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
-----
```

At the bottom of the terminal, there's a prompt: 'test> []'.

JavaScript en MongoDB

```
PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS mongosh + ▾  
To help improve our products, anonymous usage data is collected and sent to MongoDB periodically (https://www.mongodb.com/legal/privacy-policy). You can opt-out by running the disableTelemetry() command.  
----  
The server generated these startup warnings when booting  
2024-08-18T18:35:48.386-03:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted  
----  
test> use pruebas  
switched to db pruebas  
pruebas> show colecciones  
MongoshInvalidInputError: [COMMON-10001] 'colecciones' is not a valid argument for "show".  
pruebas> show colección  
MongoshInvalidInputError: [COMMON-10001] 'colección' is not a valid argument for "show".  
pruebas> load('Creacion.js')  
Error: ENOENT: no such file or directory, open 'C:\Users\Jose Luis\AppData\Local\Programs\mongosh\Creacion.js'  
pruebas> load('g:\mongodb\Creacion.js')  
Error: ENOENT: no such file or directory, open 'g:\mongodb\Creacion.js'  
pruebas> load('g:/mongodb/Creacion.js')  
true  
pruebas> █
```

JavaScript en MongoDB

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS

```
MongoshInvalidInputError: [COMMON-10001] 'colection' is not a valid argument for "show".
pruebas> load('Creacion.js')
Error: ENOENT: no such file or directory, open 'C:\Users\Jose Luis\AppData\Local\Programs\mongosh\Creacion.js'
pruebas> load('g:\mongodb\Creacion.js')
Error: ENOENT: no such file or directory, open 'g:\mongodb\Creacion.js'
pruebas> load('g:/mongodb/Creacion.js')
true
pruebas> db.articulos.find()
[
  { _id: 1, nombre: 'nombre1' },
  { _id: 2, nombre: 'nombre2' },
  { _id: 3, nombre: 'nombre3' },
  { _id: 4, nombre: 'nombre4' },
  { _id: 5, nombre: 'nombre5' },
  { _id: 6, nombre: 'nombre6' },
  { _id: 7, nombre: 'nombre7' },
  { _id: 8, nombre: 'nombre8' },
  { _id: 9, nombre: 'nombre9' },
  { _id: 10, nombre: 'nombre10' }
]
pruebas> █
```

JavaScript en MongoDB

Comandos de MongoDB en un archivo *.js

En un script no se pueden utilizar directamente los comandos **show dbs**, **use**, **show collections**, etc. pero se pueden sustituir llamando a métodos:

use base1	db = db.getSiblingDB('base1')
show dbs, show databases	db.adminCommand('listDatabases')
show collections	db.getCollectionNames()
show users	db.getUsers()
show roles	db.getRoles({showBuiltInRoles: true})
show log	db.adminCommand({ 'getLog' : '' })
show logs	db.adminCommand({ 'getLog' : '*' })
it	cursor = db.collection.find(); while (cursor.hasNext()) { printjson(cursor.next()); }

JavaScript en MongoDB

En un script, siempre que se necesiten hacer salidas por pantalla debemos utilizar la función **print** y **printjson**, por ejemplo:

```
printjson(db.adminCommand('listDatabases'))
```

Se modificará el archivo '**creacion.js**' con el siguiente código y luego se volverá a cargar mediante la función '**load**':

```
printjson(db.adminCommand('listDatabases'))  
  
db = db.getSiblingDB('base1')  
print(db.getCollectionNames())  
  
db.articulos.drop()  
for (i = 1; i <= 10; i++) {  
    db.articulos.insertOne(  
    {  
        _id: i,  
        nombre: 'nombre' + i  
    })  
}  
cursor = db.articulos.find();  
while (cursor.hasNext()) {  
    printjson(cursor.next());  
}
```

JavaScript en MongoDB

JavaScript en MongoDB

JavaScript en MongoDB

