Introducción

MongoDB es una base de datos NoSQL de código abierto, desarrollada por la empresa MongoDB Inc. Fue diseñada para manejar grandes volúmenes de datos con una arquitectura distribuida, y ofrece una forma flexible y escalable de gestionar la información.

Los datos se almacenan en documentos BSON (una forma binaria de JSON). Los documentos se agrupan en colecciones, que vendrían a ser el equivalente a las "tablas" en bases de datos relacionales, y las colecciones son agrupadas en bases de datos, tal y como en las relacionales. No requiere esquemas rígidos, los documentos de una misma colección pueden tener estructuras distintas, lo que brinda una gran flexibilidad.

```
{
  "_id": "1",
  "nombre": "Juan",
  "edad": 30,
  "email": "juan@example.com",
  "direccion": {
    "ciudad": "Madrid",
    "pais": "España"
  }
}
```

Este sería un ejemplo de documento, puede tener distintos niveles de anidación.

Ventajas

1. Modelo de datos flexible (esquema dinámico)
    - No es necesario definir un esquema fijo para los datos.
    - Puedes tener documentos en la misma colección con estructuras distintas.
    - Al tener esquema dinámico la responsabilidad de la consistencia de datos recae en el programador

2. Alto rendimiento
    - Rápido para operaciones de lectura y escritura.
    - Ideal para aplicaciones que manejan grandes volúmenes de datos en

tiempo real.

3. Escalabilidad horizontal
    - Diseñado para escalar fácilmente mediante sharding (distribuir datos entre múltiples servidores).
    - Excelente para aplicaciones que crecen rápidamente y requieren manejar grandes cantidades de información.

4. Alta disponibilidad y replicación
    - Usa replica sets, que son grupos de servidores que mantienen las mismas copias de datos, asegurando tolerancia a fallos.
    - Si un nodo falla, otro toma el control automáticamente.

5. Consultas potentes
    - Soporta consultas complejas, filtrado, proyecciones, agregaciones y operaciones geoespaciales.
    - Incorpora el Aggregation Framework, una poderosa herramienta para transformar y combinar datos.

6. Fácil de usar y desarrollar
    - Su estructura similar a JSON facilita la integración con lenguajes como JavaScript, Python, Node.js, etc.
    - La curva de aprendizaje suele ser más baja que con bases de datos relacionales.

Casos de uso

- Aplicaciones web modernas con estructuras de datos dinámicas.
- Sistemas de gestión de contenidos (CMS).
- Aplicaciones móviles y de IoT.
- Plataformas de análisis de datos en tiempo real.
- Sistemas de recomendación o búsqueda.

Instalación

Para realizar la instalación de mongodb se utilziará Docker, más concretamente un docker compose que levantará un contenedor de mongodb con un volumen para que los datos persistan si el contenedor se detiene o se elimina.

El docker compose esta en el material de la actividad. Detente unos minutos a leer el fichero y sus comentarios.

El atributo ports es un listado donde cada elemento indica un puerto del contenedor mapeado a un puerto del host/anfitrión. La sintaxis es la siguiente:

ports:

- “[host]:[container]”

Hay dos formas de gestionar los volúmenes de persistencia de datos.

La primera es la que esta configurada en el docker-compose actual, y consiste en indicar a Docker que cree y gestione un volumen y que lo asigne a un servicio en concreto. De modo que todo el trabajo de mantenimiento y gestion del volumen, como expandir espacio, mover a otros discos duros… lo realiza el propio Docker. La forma de indicar esto en docker compose es definiendo un volumen:

volumes:

    mongodb_data:

y posteriormente asignarlo al servicio en concreto:

services:

  mongodb:

    volumes:

      - mongodb_data:/data/db

De este modo todo el contenido del directorio /data/db del sistema de ficheros del contenedor estará almacenado en el volumen mongodb_data que gestiona Docker.

La otra forma de gestionar un volumen de persistencia de datos es mapear un directorio interno al contenedor con uno externo, como si fueran carpetas compartidas:

```
services:
  mongodb:
      volumes:
        - ./data:/data/db
```

de este modo todo el contenido del directorio interno /data/db se almacenará en el directorio data del host.

Para levantar los servicios de docker compose se debe ejecutar el siguiente comando desde la consola situada en el directorio del fichero:

```
$ docker compose up -d
```

El parámetro -d indica que la ejecución es detached/desatendida, de modo que si se cierra la terminal de comandos el contenedor no se detendrá.

Una vez levantado el contenedor se debe acceder para validar que todo va bien y poder utilizar mongodb. Para ello se ejecuta el comando:

```
$ docker exec -it mongodb mongosh -u admin -p admin123
```

Este comando accede al contenedor mongodb, abriendo una terminal interactiva(parámetro -it) y ejecuta el comando mongosh -u admin -p admin123.

La pantalla debería mostrar un prompt como:

```
test>
```

Operaciones básicas

Gestión de Bases de Datos

1. Ver la base de datos actual
test> db

Devuelve el nombre de la base de datos actualmente seleccionada.

2. Mostrar todas las bases de datos disponibles
test> show dbs

Una base de datos recién creada no aparece en la lista hasta que contiene al menos un documento.

3. Crear o seleccionar una base de datos
test> use escuela

Si escuela no existe, se creará automáticamente cuando insertes datos.
Ahora db apunta a la base escuela.

4. Ver la base de datos activa
test> db

5. Eliminar una base de datos
db.dropDatabase()

Esto borra la base de datos activa. Asegúrate de haber hecho use antes.

Gestión de Colecciones

1. Mostrar todas las colecciones
test> show collections

2. Crear una colección (opcional)
MongoDB crea colecciones automáticamente cuando insertas datos. Pero si quieres hacerlo explícitamente:

```
test> db.createCollection("students")
```

3. Eliminar una colección

```
db.students.drop()
```

4. Insertar documentos

4.1. Insertar un documento

```
test> db.students.insertOne({
 name: "Alice",
 age: 22,
 major: "Engineering",
 courses: ["Math", "Physics"]
})
```

el resultado debería ser similar a:

```
{
  acknowledged: true,
  insertedId: ObjectId('68e2b620e6bade9e70fe6911')
}
```

4.2. Insertar múltiples documentos

```
test> db.students.insertMany([
  {
    name: "Bob",
    age: 24,
    major: "Law",
    courses: ["History", "Criminal Law"]
  },
  {
    name: "Carol",
    age: 21,
    major: "Medicine",
```

```
    courses: ["Anatomy", "Biology"]
  }
])
```

5. Mostrar todos los documentos
test> db.students.find()

6. Mostrar con formato legible
test> db.students.find().pretty()

7. Mostrar un solo documento
test> db.students.findOne()

8. Filtrar por campo
test> db.students.find({ major: "Engineering" })

si el filtro es un objeto vacío, se mostrarán todos los documentos
test> db.students.find({})

9. Seleccionar los campos a mostrar(proyecciones)

Las proyecciones se definen mediante un objeto JSON, cada atributo de la proyección representa un campo de los documentos resultado. Las proyecciones pueden ser de inclusión o exclusión.

En las proyecciones de inclusión el objecto proyección contiene los atributos que se desean incluir en el resultado.

test> db.students.find({},{name:1})

En las proyecciones de exclusión, el objeto contiene los atributos que no deben aparecer en el resultado

test> db.students.find({},{age:0})

Mongo no permite utilizar los dos tipos de proyecciones a excepción del campo _id.

El siguiente comando produce error:

test> db.students.find({},{age:0, name:1})

Este comando no, aunque no tiene demasiado sentido ya que el campo _id se muestra por defecto:

test> db.students.find({},{_id:1, age:0})

Estos comandos tampoco producen error, y no muestran el campo _id:

```
test> db.students.find({},{_id:0, age:1})
```

```
test> db.students.find({},{_id:0, age:0})
```

10. Consultar con operadores

```
test> db.students.find({ age: { $gt: 22 } })
```

```
test> db.students.find({
 $or: [
 { major: "Engineering" },
 { major: "Medicine" }
 ]
})
```

11. Actualizar un solo documento

```
test> db.students.updateOne(
  { name: "Alice" },
  { $set: { age: 23 } }
)
```

12. Actualizar múltiples documentos

```
test> db.students.updateMany(
  { major: "Engineering" },
  { $set: { faculty: "Exact Sciences" } }
)
```

13. Incrementar un valor numérico

```
test> db.students.updateMany(
  {},
  { $inc: { age: 1 } }
)
```

14. Agregar un valor a un arreglo

```
test> db.students.updateOne(
  { name: "Alice" },
  { $push: { courses: "Programming" } }
)
```

15. Eliminar un documento
test> db.students.deleteOne({ name: "Bob" })

16. Eliminar varios documentos
test> db.students.deleteMany({ major: "Medicine" })

Agregaciones

En MongoDB, las agregaciones son operaciones que permiten procesar documentos de una colección para devolver resultados transformados, resumidos o agregados (por ejemplo, sumas, promedios, agrupaciones).

El mecanismo principal es el framework de agregación (Aggregation Framework), que trabaja con una tubería (pipeline) de etapas secuenciales.

Una operación de agregación toma los documentos de entrada, los pasa por cada etapa del pipeline, y el resultado final se entrega como salida.

Es similar al concepto GROUP BY en SQL, pero en MongoDB puedes hacer transformaciones intermedias más complejas, trabajar con arrays, descomponer documentos anidados, etc.

La sintaxis base es:

```
db.collection.aggregate([
  { stage1 },
  { stage2 },
  // ...
  { stageN }
], options)
```

Donde cada stage es una etapa que opera sobre el conjunto de documentos (o el flujo de documentos) proveniente de la etapa anterior.

Opcionalmente, puedes pasar options como { allowDiskUse: true }, cursor, etc.

Etapas (stages) comunes en un pipeline de agregación

Para los siguientes ejemplos es necesario insertar los siguientes documentos:

```
db.orders.insertMany([
  { customerId: "A", amount: 120, date: ISODate("2024-01-15T00:00:00Z") },
  { customerId: "B", amount: 200, date: ISODate("2024-01-20T00:00:00Z") },
  { customerId: "A", amount: 80,  date: ISODate("2024-02-05T00:00:00Z") },
  { customerId: "C", amount: 300, date: ISODate("2024-02-10T00:00:00Z") },
  { customerId: "A", amount: 150, date: ISODate("2024-02-20T00:00:00Z") }
])
```

$match

Función: filtrar documentos que cumplan ciertas condiciones (similar a find() o la cláusula WHERE en SQL).

Recomendación: siempre que puedas, coloca $match al principio del pipeline para reducir cuanto antes el volumen de datos.

En MongoDB, $match acepta operadores de consulta (por ejemplo $gt, $lt, $in, $or, $and, expresiones de comparación, etc.).

Ejemplo:

Supongamos que tienes una colección orders. Filtras solo los pedidos con monto mayor que 100:

```
db.orders.aggregate([
  { $match: { amount: { $gt: 100 } } }
])
```

$project

Función: remodelar cada documento de entrada (añadir, eliminar, renombrar o calcular nuevos campos).

Permite proyectar (seleccionar) campos específicos y hacer transformaciones sobre ellos.

Puede usarse para incluir campos computados (por ejemplo, sumas, concatenaciones, condicionales, operaciones en fechas).

Ejemplo:

En orders, queremos proyectar solo customerId, amount, y crear un nuevo campo year sacado de la fecha:

```
db.orders.aggregate([
  {
    $project: {
      customerId: 1,
      amount: 1,
      year: { $year: "$date" }  // extrae el año de la fecha
    }
  }
])
```

$group

Función: agrupar documentos según alguna clave (_id) y aplicar acumuladores para generar resultados agregados (suma, promedio, conteo, máximo, mínimo, arrays, etc.).

Los acumuladores comunes son: $sum, $avg, $min, $max, $first, $last

El campo _id indica el valor de agrupación (por ejemplo, agrupar por país, por usuario, por mes).

Ejemplo:

Quieres sumar el monto total de pedidos por cliente:

```
db.orders.aggregate([
  {
    $group: {
```

```
      _id: "$customerId",
      totalAmount: { $sum: "$amount" },
      avgAmount: { $avg: "$amount" },
      countOrders: { $sum: 1 }
    }
  }
])
```

En este ejemplo:
_id: "$customerId" agrupa por cliente.
totalAmount acumula la suma de amount.
avgAmount calcula promedio.
countOrders cuenta cuántos documentos hay en cada grupo.

$sort

Función: ordenar los documentos que salen de la etapa anterior, en base a uno o más campos, en orden ascendente (1) o descendente (-1).

Si combinas $sort con $match u otras etapas, conviene hacer $match antes para que el ordenamiento trabaje con menor conjunto.

Ejemplo:

Tras agrupar por cliente (como en el ejemplo anterior), ordenar por totalAmount descendente:

```
db.orders.aggregate([
  { $group: { _id: "$customerId", totalAmount: { $sum: "$amount" } } },
  { $sort: { totalAmount: -1 } }
])
```

$limit y $skip

$limit: limita el número de documentos que pasan adelante.

$skip: omite los primeros N documentos.

Combinados, sirven para paginación o para tomar “top N”.

Ejemplo:

Tomar los 5 clientes con mayor monto:

```
db.orders.aggregate([
  { $group: { _id: "$customerId", totalAmount: { $sum: "$amount" } } },
  { $sort: { totalAmount: -1 } },
  { $limit: 5 }
])
```

Si quisieras saltarte los dos primeros y luego tomar 3:

```
db.orders.aggregate([
  { $group: { _id: "$customerId", totalAmount: { $sum: "$amount" } } },
  { $sort: { totalAmount: -1 } },
  { $skip: 2 },
  { $limit: 3 }
])
```

$unwind

Función: descomponer un campo que es un arreglo (array) en múltiples documentos, uno por elemento del arreglo.

Útil cuando tus documentos tienen arrays y quieres “explosionarlos” para analizarlos individualmente.

Puedes opcionalmente usar preserveNullAndEmptyArrays: true para que documentos que no tienen el arreglo o tienen arreglo vacío también salgan (sin elementos).

Ejemplo:

Supongamos que una colección users tiene documentos así:

```
db.users.insertMany([
  { name: "Alice", interests: ["music", "travel"] },
  { name: "Bob", interests: ["sports"] },
  { name: "Carol", interests: ["music", "sports", "cooking"] }
])
```

Si queremos analizar cuántas veces aparece cada interés entre todos los usuarios:

```
db.users.aggregate([
  { $unwind: "$interests" },
  { $group: { _id: "$interests", count: { $sum: 1 } } },
  { $sort: { count: -1 } }
])
```

La secuencia:
$unwind produce documentos como { _id:1, name:"Alice", interests:"gaming" }, luego "music", etc.
$group agrupa por interés y cuenta cuántas veces aparece.

Relaciones

MongoDB es una base de datos NoSQL orientada a documentos. Aunque no utiliza tablas ni relaciones en el sentido clásico como en las bases de datos relacionales, sí permite establecer relaciones entre documentos.

Estas relaciones se pueden implementar de dos formas principales:

- Documentos incrustados (Embedding)
- Referencias entre documentos (Referencing)

Cada una tiene ventajas y limitaciones, y su elección depende del tipo de relación, del volumen de los datos y del uso que se les dé.

Documentos Incrustados (Embedding)

Consiste en incluir un documento dentro de otro. Es ideal para representar relaciones estrechas donde los datos relacionados siempre se consultan juntos. Es común en relaciones uno a uno y uno a muchos.

Ventajas

- Consulta rápida: todos los datos están en un solo documento.
- Menor necesidad de operaciones de unión ($lookup).
- Simplicidad de diseño en algunos casos.

Desventajas

- El documento puede crecer demasiado (límite de 16MB por documento en MongoDB).
- Puede haber redundancia si los datos incrustados se repiten en varios documentos.
- Dificultad para actualizar elementos individuales anidados.
-

Ejemplo: Un usuario con múltiples direcciones

```
{
  "_id": ObjectId("653a1f95c1f1b1a7d8d88a77"),
  "name": "John Smith",
  "email": "john.smith@example.com",
  "addresses": [
    {
      "type": "home",
      "street": "123 Main St",
      "city": "New York",
      "country": "USA"
    },
    {
      "type": "work",
      "street": "456 Market St",
      "city": "San Francisco",
      "country": "USA"
```

```
    }
  ]
}
```

Referencias entre Documentos (Referencing)

En este enfoque, un documento hace referencia a otro a través de una clave, generalmente el campo _id.

Es útil para relaciones más complejas, especialmente cuando los documentos relacionados son grandes, se consultan por separado o están compartidos entre múltiples documentos.

Ventajas

- Evita la duplicación de datos.
- Más flexible y escalable.
- Permite relaciones más complejas, como muchos-a-muchos.

Desventajas

- Requiere múltiples consultas o agregaciones para obtener los datos completos.
- Menor rendimiento en lecturas complejas si no se diseñan correctamente.

Ejemplo: Publicaciones y autores
Documento en la colección authors

```
{
  "_id": ObjectId("653a20a2c1f1b1a7d8d88a80"),
  "name": "Alice Johnson",
  "email": "alice@example.com"
}
```

Documento en la colección posts

```
{
  "_id": ObjectId("653a20e4c1f1b1a7d8d88a81"),
  "title": "Working with MongoDB",
  "content": "This is an introduction to MongoDB...",
  "author_id": ObjectId("653a20a2c1f1b1a7d8d88a80")
```

}

Joins con $lookup

MongoDB no tiene joins como SQL, pero permite realizar uniones entre colecciones usando el operador `$lookup` dentro de una agregación.

Realizar un join de las colecciones authors y posts:

```
db.authors.insertOne({
  _id: ObjectId("653a20a2c1f1b1a7d8d88a80"),
  name: "Alice Johnson",
  email: "alice@example.com"
});

db.posts.insertOne({
  _id: ObjectId("653a20e4c1f1b1a7d8d88a81"),
  title: "Working with MongoDB",
  content: "This is an introduction to MongoDB...",
  author_id: ObjectId("653a20a2c1f1b1a7d8d88a80")
});

db.posts.aggregate([
  {
    $lookup: {
      from: "authors",
      localField: "author_id",
      foreignField: "_id",
      as: "author_info"
    }
  }
])
```

Supuesto práctico

Entrega

Se deberá entregar un documento con capturas de pantalla de la ejecución de cada punto y con una descripción del objetivo de cada sentencia. Debe quedar clara la comprensión de la sentencia, en caso de que se tengan dudas, se deben dejar reflejadas también en el documento. Se resolverán en clase.

Enunciado

Una pequeña librería online necesita almacenar y gestionar información sobre sus libros, autores y ventas. Como desarrollador/a, se te ha encargado crear y consultar la base de datos en MongoDB.

Se requiere:

Una colección de libros (books), con información como título, autor, género, precio y stock disponible.

Una colección de autores (authors), con nombre, país y fecha de nacimiento.

Una colección de ventas (sales), con fecha de venta, libro vendido, cantidad y total pagado.

1.Inserción de Datos

Colección: authors

```
db.authors.insertMany([
  { _id: 1, name: "Gabriel García Márquez", country: "Colombia", birthdate: ISODate("1927-03-06") },
  { _id: 2, name: "Isabel Allende", country: "Chile", birthdate: ISODate("1942-08-02") },
  { _id: 3, name: "Haruki Murakami", country: "Japón", birthdate: ISODate("1949-01-12") }
])
```

Colección: books

```
db.books.insertMany([
  { _id: 101, title: "Cien Años de Soledad", author_id: 1, genre: "Realismo Mágico", price: 20, stock: 10 },
  { _id: 102, title: "El Amor en los Tiempos del Cólera", author_id: 1, genre: "Romance", price: 18, stock: 5 },
  { _id: 103, title: "La Casa de los Espíritus", author_id: 2, genre: "Realismo
```

```
Mágico", price: 22, stock: 8 },
  { _id: 104, title: "Tokio Blues", author_id: 3, genre: "Drama", price: 15, stock:
12 }
])
```

📓Colección: sales

```
db.sales.insertMany([
  { book_id: 101, date: ISODate("2025-10-01"), quantity: 2, total: 40 },
  { book_id: 104, date: ISODate("2025-10-02"), quantity: 1, total: 15 },
  { book_id: 103, date: ISODate("2025-10-03"), quantity: 3, total: 66 },
  { book_id: 101, date: ISODate("2025-10-04"), quantity: 1, total: 20 }
])
```

2. CRUD

2.1. Listar todos los libros disponibles:

```
db.books.find()
```

2.2. Buscar libros del género "Realismo Mágico":

```
db.books.find({ genre: "Realismo Mágico" })
```

2.3. Agregar un nuevo libro:

```
db.books.insertOne({
  _id: 105,
  title: "Kafka en la orilla",
  author_id: 3,
  genre: "Ficción",
  price: 19,
  stock: 7
})
```

2.4. Actualizar el stock del libro con ID 104:

```
db.books.updateOne(
  { _id: 104 },
  { $inc: { stock: -1 } }  // Se vendió un ejemplar
)
```

2.5. Eliminar un libro que ya no se vende:

```
db.books.deleteOne({ _id: 102 })
```

3. Agregaciones
3.1. Total de ventas por libro

```
db.sales.aggregate([
  {
    $group: {
      _id: "$book_id",
      total_quantity: { $sum: "$quantity" },
      total_earnings: { $sum: "$total" }
    }
  }
])
```

3.2. Listar libros con información del autor (join con $lookup)

```
db.books.aggregate([
  {
    $lookup: {
      from: "authors",
      localField: "author_id",
      foreignField: "_id",
      as: "author_info"
    }
  },
  { $unwind: "$author_info" },
  {
    $project: {
      title: 1,
      genre: 1,
      price: 1,
      "author_info.name": 1,
      "author_info.country": 1
    }
  }
])
```

3.3. Géneros más vendidos (usando ventas y libros)

```
db.sales.aggregate([
 {
 $lookup: {
 from: "books",
 localField: "book_id",
 foreignField: "_id",
 as: "book"
 }
 },
 { $unwind: "$book" },
 {
 $group: {
 _id: "$book.genre",
 total_sales: { $sum: "$total" }
 }
 },
 { $sort: { total_sales: -1 } }
])
```

Ejercicio

Entrega

Este ejercicio consiste en resolver las necesidades expuestas en un supuesto práctico en dos partes.

La primera parte se compone de sentencias parecidas a las anteriores que deben ejecutarse en el cliente de consola, como las anteriores, y guardar una captura de pantalla en un documento word, haciendo referencia a la sentencia en cuestión. También se pueden añadir comentarios, en caso de que se quiera plantear alguna duda o aclaración sobre la solución empleada.

La segunda parte consiste en ejecutar todas las sentencias pero esta vez mediante algún lenguaje de programación, puede ser cualquiera. En este caso no se exige un proyecto completo, sino unos archivos que contengan las funciones necesarias para conectarse a la base de datos y ejecutar las sentencias anteriores.

Supuesto práctico

Se debe construir una base de datos para gestionar la información académica de una institución educativa. Esta base de datos almacenará información sobre:

- Estudiantes
- Asignaturas
- Inscripciones de estudiantes en asignaturas

La base de datos permitirá registrar datos personales de los estudiantes, las asignaturas ofrecidas y las notas obtenidas en cada curso.

Estructura de las colecciones
students

name, age, email, city

courses
name, code, credits, instructor

enrollments
student_id, course_id, year, grade

Sentencias

1. Insertar al menos 3 documentos de cada colección
2. Buscar un curso por id
3. Buscar un alumno por nombre
4. Actualizar la ciudad de un estudiante
5. Eliminar una inscripción
6. Agregación. Mostrar las inscripciones con nombres de estudiantes y asignaturas. Es decir, cada documento resultante debe contener todos los campos de una inscripción además del nombre del alumno y de la asignatura.
7. Agregación. Calcular el promedio de notas por estudiante
8. Agregación. Contar cuántos estudiantes hay inscritos por asignatura.