

MongoDB: Fundamentos y Mongoose

1. Conceptos Básicos de MongoDB

MongoDB es una base de datos NoSQL orientada a documentos. A diferencia de las bases de datos relacionales, almacena datos en documentos flexibles similares a JSON.

Conceptos clave:

- Base de datos: contenedor de colecciones
- Colección: grupo de documentos (similar a una tabla)
- Documento: registro individual en formato BSON (similar a un objeto JSON)

Estructura de un documento:

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "nombre": "María García",
  "email": "maria@ejemplo.com",
  "edad": 28,
  "direccion": {
    "calle": "Gran Vía 123",
    "ciudad": "Madrid",
    "pais": "España"
  },
  "intereses": ["programación", "música", "viajes"],
  "createdAt": ISODate("2024-01-15T10:30:00Z")
}
```

El campo `_id` es obligatorio y único. Si no lo proporcionas, MongoDB genera automáticamente un `ObjectId`.

Ventajas de MongoDB:

- Esquema flexible: cada documento puede tener estructura diferente
- Escalabilidad horizontal mediante sharding
- Alto rendimiento en lecturas y escrituras
- Documentos anidados reducen necesidad de joins
- Índices potentes incluyendo texto completo y geoespaciales

2. Operaciones CRUD

CRUD significa Create, Read, Update, Delete - las operaciones básicas de base de datos.

CREATE - Insertar documentos:

```
// Insertar uno
db.usuarios.insertOne({
  nombre: "Juan",
  email: "juan@ejemplo.com"
});
```

```
// Insertar varios
db.usuarios.insertMany([
  { nombre: "Ana", email: "ana@ejemplo.com" },
  { nombre: "Pedro", email: "pedro@ejemplo.com" }
]);

READ - Consultar documentos:
// Encontrar todos
db.usuarios.find();

// Encontrar con filtro
db.usuarios.find({ edad: { $gte: 18 } });

// Encontrar uno
db.usuarios.findOne({ email: "juan@ejemplo.com" });

// Proyección (seleccionar campos)
db.usuarios.find({}, { nombre: 1, email: 1, _id: 0 });

UPDATE - Actualizar documentos:
// Actualizar uno
db.usuarios.updateOne(
  { email: "juan@ejemplo.com" },
  { $set: { edad: 30 } }
);

// Actualizar varios
db.usuarios.updateMany(
  { activo: false },
  { $set: { activo: true } }
);

// Reemplazar documento completo
db.usuarios.replaceOne(
  { _id: ObjectId("...") },
  { nombre: "Juan Nuevo", email: "nuevo@ejemplo.com" }
);

DELETE - Eliminar documentos:
db.usuarios.deleteOne({ email: "juan@ejemplo.com" });
db.usuarios.deleteMany({ activo: false });
```

3. Operadores de Consulta

MongoDB proporciona operadores potentes para consultas complejas.

Operadores de comparación:

\$eq: igual a

\$ne: no igual a

\$gt: mayor que

\$gte: mayor o igual que

\$lt: menor que

\$lte: menor o igual que

\$in: valor está en array
\$nin: valor no está en array

Ejemplos:

```
db.productos.find({ precio: { $gte: 100, $lte: 500 } });  
db.usuarios.find({ rol: { $in: ["admin", "moderador"] } });
```

Operadores lógicos:

\$and: todas las condiciones deben cumplirse
\$or: al menos una condición debe cumplirse
\$not: niega la condición
\$nor: ninguna condición debe cumplirse

Ejemplos:

```
db.productos.find({  
  $and: [  
    { precio: { $lt: 100 } },  
    { stock: { $gt: 0 } }  
  ]  
});
```

```
db.usuarios.find({  
  $or: [  
    { edad: { $lt: 18 } },  
    { edad: { $gt: 65 } }  
  ]  
});
```

Operadores de elementos:

\$exists: el campo existe
\$type: el campo es de cierto tipo

```
db.usuarios.find({ telefono: { $exists: true } });
```

Operadores de arrays:

\$all: array contiene todos los valores
\$elemMatch: elemento cumple todas las condiciones
\$size: array tiene tamaño específico

```
db.usuarios.find({ intereses: { $all: ["música", "deportes"] } });  
db.pedidos.find({ items: { $elemMatch: { cantidad: { $gt: 5 } } } });
```

4. Aggregation Pipeline

El Aggregation Pipeline es el framework de MongoDB para transformar y analizar datos. Procesa documentos a través de etapas (stages) secuenciales.

Etapas principales:

\$match: filtra documentos (como find)
\$group: agrupa y calcula agregados
\$sort: ordena documentos
\$project: selecciona/transforma campos
\$limit/\$skip: paginación

\$lookup: join con otra colección

\$unwind: descompone arrays

Ejemplo - Ventas por categoría:

```
db.ventas.aggregate([
  // Filtrar ventas del último mes
  { $match: {
    fecha: { $gte: ISODate("2024-01-01") }
  } },
  // Agrupar por categoría
  { $group: {
    _id: "$categoria",
    totalVentas: { $sum: "$monto" },
    cantidadVentas: { $count: {} },
    promedioVenta: { $avg: "$monto" }
  } },
  // Ordenar por total descendente
  { $sort: { totalVentas: -1 } },
  // Limitar a top 5
  { $limit: 5 }
]);
```

Ejemplo - Join con \$lookup:

```
db.pedidos.aggregate([
  { $lookup: {
    from: "usuarios",
    localField: "usuarioid",
    foreignField: "_id",
    as: "usuario"
  } },
  { $unwind: "$usuario" },
  { $project: {
    numeroPedido: 1,
    total: 1,
    "usuario.nombre": 1,
    "usuario.email": 1
  } }
]);
```

\$unwind convierte un documento con array en múltiples documentos, uno por cada elemento del array.

5. Índices

Los índices mejoran drásticamente el rendimiento de las consultas al evitar escaneos completos de colección.

Crear índices:

// Índice simple ascendente

```
db.usuarios.createIndex({ email: 1 });
```

// Índice descendente

```
db.usuarios.createIndex({ createdAt: -1 });
```

```
// Índice compuesto
db.productos.createIndex({ categoria: 1, precio: -1 });

// Índice único
db.usuarios.createIndex({ email: 1 }, { unique: true });

// Índice de texto para búsquedas
db.articulos.createIndex({ titulo: "text", contenido: "text" });
```

Tipos de índices:

- Single field: un campo
- Compound: múltiples campos
- Multikey: para arrays
- Text: búsqueda de texto completo
- Geospatial: datos geográficos (2d, 2dsphere)
- Hashed: para sharding

Ver índices existentes:

```
db.usuarios.getIndexes();
```

Analizar uso de índices con explain():

```
db.usuarios.find({ email: "test@ejemplo.com" }).explain("executionStats");
```

Revisar el campo "stage":

- COLLSCAN: escaneo completo (malo)
- IXSCAN: uso de índice (bueno)
- FETCH: recuperar documentos

Buenas prácticas:

- Crear índices para campos frecuentemente consultados
- Índices compuestos siguen orden de izquierda a derecha
- Los índices ocupan memoria y ralentizan escrituras
- Usar explain() para verificar uso de índices

6. Mongoose con Node.js

Mongoose es una librería ODM (Object Document Mapper) que facilita trabajar con MongoDB desde Node.js.

Conexión:

```
import mongoose from 'mongoose';

await mongoose.connect('mongodb://localhost:27017/miapp');
```

Definir esquema y modelo:

```
const usuarioSchema = new mongoose.Schema({
  nombre: { type: String, required: true },
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true
  },
});
```

```
edad: { type: Number, min: 0, max: 150 },
rol: {
  type: String,
  enum: ['usuario', 'admin'],
  default: 'usuario'
},
activo: { type: Boolean, default: true },
createdAt: { type: Date, default: Date.now }
});
```

```
const Usuario = mongoose.model('Usuario', usuarioSchema);
```

Operaciones CRUD con Mongoose:

// Crear

```
const usuario = await Usuario.create({ nombre: "Ana", email: "ana@ej.com" });
```

// Leer

```
const usuarios = await Usuario.find({ activo: true });
```

```
const usuario = await Usuario.findById(id);
```

```
const usuario = await Usuario.findOne({ email: "ana@ej.com" });
```

// Actualizar

```
await Usuario.findByIdAndUpdate(id, { nombre: "Ana María" });
```

```
await Usuario.updateMany({ rol: "usuario" }, { $set: { verificado: false } });
```

// Eliminar

```
await Usuario.findByIdAndDelete(id);
```

```
await Usuario.deleteMany({ activo: false });
```

Middleware (hooks):

```
usuarioSchema.pre('save', async function(next) {
  if (this.isModified('password')) {
    this.password = await bcrypt.hash(this.password, 10);
  }
  next();
});
```

Métodos personalizados:

```
usuarioSchema.methods.compararPassword = async function(password) {
  return bcrypt.compare(password, this.password);
};
```