



# Resumen-UT4.pdf



kriseren125



Acceso a datos



1º Desarrollo de Aplicaciones Multiplataforma



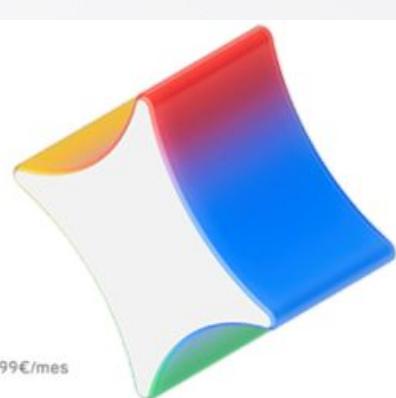
POETA PACO MOLLÀ

**Google Gemini:  
Plan Pro a 0€ durante 1 año.  
Tu ventaja por ser estudiante.**

Oferta válida hasta el 9 de diciembre de 2025

[Consigue la oferta](#)

Después 21,99€/mes



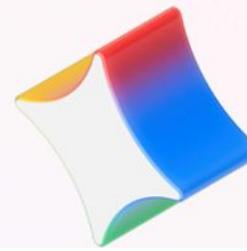
Google Gemini: Plan Pro a 0€ durante 1 año.

Tu ventaja por ser estudiante.

Oferta válida hasta el 9 de diciembre de 2025

[Consigue la oferta](#)

Después 21,99€/mes



### Acceso a datos

Diego Calatayud Muñoz  
2º DAM  
IES Poeta Paco Mollá

## Resumen UT4: MongoDB



### ¿QUÉ ES MONGODB?

3

### USO DE LA LÍNEA DE COMANDOS DE MONGODB

3

CONEXIÓN

3

VISUALIZACIÓN DE BASES DE DATOS EXISTENTES

3

CONECTAR CON UNA BASE DE DATOS

3

VISUALIZAR COLECCIONES

4

ELIMINAR BASES DE DATOS

4

### OPERACIONES CRUD DESDE JAVA

4

AGREGAR LA DEPENDENCIA DE MONGODB

4

CONEXIÓN A UNA BASE DE DATOS

5

OPERACIONES DE INSERCIÓN

5

OPERACIONES DE LECTURA (BÚSQUEDAS)

6

OPERACIONES DE ACTUALIZACIÓN

6

OPERACIONES DE ELIMINACIÓN

7

### OPERADORES

8

\$EQ

8

\$GT y \$GTE

8

\$LT y \$LTE	8
\$INC	8
\$AND	9
\$OR	9
\$SORT	10
<b>PROYECCIONES</b>	<b>10</b>
<b>MAPEO DE CLASES</b>	<b>10</b>

## ¿QUÉ ES MONGODB?

MongoDB es una de las principales representantes de bases de datos NoSQL (Not Only SQL). Su estructura está formada por los siguientes componentes:

- ❖ **Documentos:** Son la unidad básica de MongoDB y el equivalente a las filas dentro de las tablas de las bases de datos relacionales. La información se almacena en archivos con formato BSON (Binary JSON).
- ❖ **Colecciones:** Son agrupaciones de documentos y las equivalentes a las tablas en las bases de datos relacionales. No obstante, las colecciones no imponen una estructura rígida, por lo que los documentos pueden ser distintos entre sí.
- ❖ **Bases de datos:** Son los contenedores físicos para almacenar las colecciones. Cada colección tiene un propio archivo en la base de datos.
- ❖ **Clusters:** Son conjuntos de bases de datos.

## USO DE LA LÍNEA DE COMANDOS DE MONGODB

El programa de MongoDB ofrece una interfaz de línea de comandos para realizar la gestión de las bases de datos.

Algunos de los comandos más útiles son los siguientes.

### CONEXIÓN

Para conectarnos al servidor de MongoDB, deberemos ejecutar el comando siguiente.

```
mongo --host localhost --port 27017 -u usuario -p password
```

### VISUALIZACIÓN DE BASES DE DATOS EXISTENTES

Para visualizar las bases de datos que tenemos creadas en nuestro servidor de MongoDB, podremos ejecutar el siguiente comando.

```
show dbs
```

```
> show dbs
admin      0.000GB
biblioteca 0.000GB
config     0.000GB
local      0.000GB
ut4        0.000GB
> █
```

### CONECTAR CON UNA BASE DE DATOS

Para conectar o hacer uso de una base de datos, ejecutaremos el siguiente comando.

```
use nombreBaseDatos
```

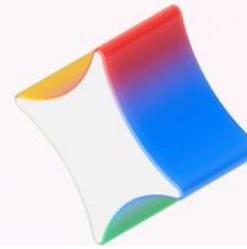
# Google Gemini: Plan Pro a 0€ durante 1 año.

Tu ventaja por ser estudiante.

Oferta válida hasta el 9 de diciembre de 2025

[Consigue la oferta](#)

Después 21,99€/mes



Acceso a datos

Diego Calatayud Muñoz

## VISUALIZAR COLECCIONES

Para visualizar las colecciones dentro de una base de datos, deberemos ejecutar el siguiente comando.

```
show collections
```

```
> show collections
artistas
```

## ELIMINAR BASES DE DATOS

Para eliminar una base de datos, deberemos primeramente posicionarnos en ella y ejecutar el siguiente comando.

```
use prueba
db.dropDatabase()
```

```
> use prueba
switched to db prueba
> db.dropDatabase()
{ "dropped" : "prueba", "ok" : 1 }
```

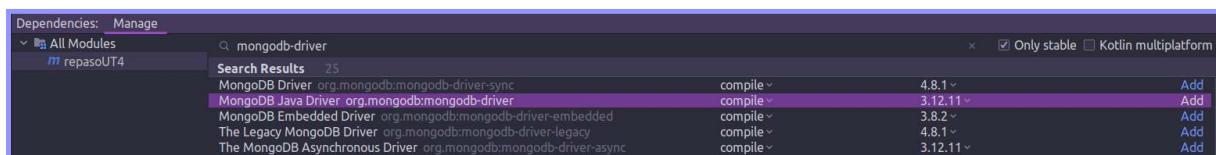
## OPERACIONES CRUD DESDE JAVA

Para la explicación de los ejemplos siguientes se hará uso de una clase Alumno, la cual tendrá 4 campos: ID, nombre, edad y nota media.

```
private Long id;
private String nombre;
private int edad;
private double notaMedia;
```

## AGREGAR LA DEPENDENCIA DE MONGODB

Para poder trabajar con una base de datos de MongoDB desde Java, deberemos agregar la respectiva dependencia al proyecto. Así pues, agregaremos la siguiente dependencia.



Así pues, el archivo “pom.xml” deberá quedar tal que así.

```
<dependencies>
    <dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongodb-driver</artifactId>
        <version>3.12.11</version>
    </dependency>
</dependencies>
```

## CONEXIÓN A UNA BASE DE DATOS

Antes de poder realizar ninguna operación en una base de datos de MongoDB, debemos conectarnos a ella. Bastará con crear un objeto MongoClient y un objeto MongoDataBase.

```
//Creamos el objeto MongoClient especificando el host y el puerto.  
MongoClient mc = new MongoClient("localhost",27017);  
//Obtenemos el objeto MongoDatabase para conectarnos a una base de datos específica.  
MongoDatabase mdb = mc.getDatabase("repasoUT4");  
//Creamos el objeto MongoCollection para conectarnos a una colección específica.  
MongoCollection mco = mdb.getCollection("alumnos");
```

## OPERACIONES DE INSERCIÓN

Para insertar un objeto Alumno, deberemos implementar un código como el siguiente.

```
//Creamos los objetos necesarios para obtener la conexión a la colección.  
MongoClient mc = new MongoClient("localhost",27017);  
MongoDatabase mdb = mc.getDatabase("repasoUT4");  
MongoCollection mco = mdb.getCollection("alumnos");  
//Creamos el objeto Alumno con el identificador 1 (L indica que es un Long).  
Alumno a = new Alumno(1L,"Pedro",24,7.8);  
//Creamos un documento con los datos del alumno.  
Document document = new Document("_id",a.getId())  
    .append("nombre",a.getNombre())  
    .append("edad",a.getEdad())  
    .append("nota_media",a.getNotaMedia());  
//Insertamos al alumno.  
mco.insertOne(document);
```

En el caso de querer insertar más de un alumno, el código será similar, puesto que deberemos insertar una lista de objetos document. El código es el siguiente.

```
//Creamos los objetos necesarios para obtener la conexión a la colección.  
MongoClient mc = new MongoClient("localhost",27017);  
MongoDatabase mdb = mc.getDatabase("repasoUT4");  
MongoCollection mco = mdb.getCollection("alumnos");  
//Creamos los objetos alumno (L indica que es un Long).  
Alumno a1 = new Alumno(1L,"Rubén",21,6.7);  
Alumno a2 = new Alumno(1L,"Marcos",19,9.4);  
//Creamos una lista de documentos.  
List<Document> documents = new ArrayList<>();  
//Agregamos a la lista los documentos de los alumnos a1 y a2.  
documents.add(new Document("_id",2
```

BUENA VISTA INTERNATIONAL  
Y MEDITERRÁNEO  
PRESENTAN

UNA PELÍCULA DIRIGIDA POR

SAMANTHA LÓPEZ SPERANZA



# TODOS LOS LADOS DE LA CAMA

14 DE NOVIEMBRE SOLO EN CINES



ERNESTO ALTERIO

PILAR CASTRO

JAN BUXADERAS

LUCÍA CARABALLO

CINEMA

WEEKEND

MALIA

LIGHTBOX

TIK TOK

MEDIANET

NETFLIX

TRAILER

RE

MOGAMBO

GO 10

NETFLIX

MEDITERRÁNEO

BUENA VISTA  
INTERNATIONAL

NETFLIX

```
.append("nombre",a1.getNombre())
.append("edad",a1.getEdad())
.append("nota_media",a1.getNotaMedia())
);
documents.add(new Document("_id",3)
.append("nombre",a2.getNombre())
.append("edad",a2.getEdad())
.append("nota_media",a2.getNotaMedia())
);
//Insertamos los alumnos.
mco.insertMany(documents);
```

## OPERACIONES DE LECTURA (BÚSQUEDAS)

Las operaciones de lectura se realizan usando documentos de filtrado. Un ejemplo de código que obtiene todos los objetos alumno es el siguiente.

```
//Creamos los objetos necesarios para obtener la conexión a la colección.
MongoClient mc = new MongoClient("localhost",27017);
MongoDatabase mdb = mc.getDatabase("repasoUT4");
MongoCollection mco = mdb.getCollection("alumnos");
//Creamos un objeto FindIterable sobre una consulta vacía.
FindIterable fit = mco.find();
//Obtenemos un mongoCursor.
MongoCursor<Alumno> cursor = fit.iterator();
//Por cada elemento del cursor mostramos el artista.
while (cursor.hasNext())
    System.out.println(cursor.next());
```

## OPERACIONES DE ACTUALIZACIÓN

Imaginemos que queremos modificar el nombre de un alumno. Para ello, deberemos implementar dos documentos, uno de filtrado y otro de actualización. El código será similar al siguiente.

```
//Creamos los objetos necesarios para obtener la conexión a la colección.
MongoClient mc = new MongoClient("localhost",27017);
MongoDatabase mdb = mc.getDatabase("repasoUT4");
MongoCollection mco = mdb.getCollection("alumnos");
//Creamos un documento de filtrado con el id del alumno a modificar.
Document filtrado = new Document("_id",1);
//Creamos un documento de actualización para definir los nuevos valores.
Document actualizacion = new Document("$set",new Document("nombre","Ramón"));
mco.updateOne(filtrado,actualizacion);
```

# Google Gemini: Plan Pro a 0€ durante 1 año. Tu ventaja por ser estudiante.



Oferta válida hasta el 9 de diciembre de 2025

Consigue la oferta

Después 21,99€/mes

Sintetiza horas de investigación en minutos.

Necesito estudiar a fondo el comportamiento de la fotosíntesis según el tipo de planta y el

Un momento...

Fotosíntesis: Tipos, Entorno e Impacto  
Iniciando búsqueda...

+ Deep Research Canvas

Acceso a datos

Diego Calatayud Muñoz

De igual manera, si quisiéramos actualizar varios alumnos, y aumentar su nota media en un punto, podríamos implementar el siguiente código.

```
//Creamos los objetos necesarios para obtener la conexión a la colección.
MongoClient mc = new MongoClient("localhost",27017);
MongoDatabase mdb = mc.getDatabase("repasoUT4");
MongoCollection mco = mdb.getCollection("alumnos");
//Creamos un documento de filtrado vacío para modificar todos los alumnos.
Document filtrado = new Document();
//Creamos un documento de actualización para definir los nuevos valores.
Document actualizacion = new Document("$inc",new Document("nota_media",1));
//Actualizamos los alumnos.
mco.updateMany(new Document(),actualizacion);
```

## OPERACIONES DE ELIMINACIÓN

Las operaciones de eliminación son sencillas, puesto que para eliminar un documento, bastará con aportar un documento de filtrado. Para eliminar el alumno “Ramón”, deberemos implementar el siguiente código.

```
//Creamos los objetos necesarios para obtener la conexión a la colección.
MongoClient mc = new MongoClient("localhost",27017);
MongoDatabase mdb = mc.getDatabase("repasoUT4");
MongoCollection mco = mdb.getCollection("alumnos");
//Creamos un documento de filtrado.
Document filtrado = new Document("nombre","Ramón");
//Actualizamos los alumnos.
mco.deleteOne(filtrado);
```

En el caso de querer eliminar todos los alumnos con identificador mayor a 2, deberemos implementar el siguiente código.

```
//Creamos los objetos necesarios para obtener la conexión a la colección.
MongoClient mc = new MongoClient("localhost",27017);
MongoDatabase mdb = mc.getDatabase("repasoUT4");
MongoCollection mco = mdb.getCollection("alumnos");
//Creamos un documento de filtrado.
Document filtrado = new Document("_id",new Document("$gt",2));
//Actualizamos los alumnos.
mco.deleteOne(filtrado);
```

## OPERADORES

Los operadores permiten realizar filtros sobre los documentos de una colección con el fin de realizar operaciones de lectura, borrado o actualización. Los operadores más comunes son los mencionados a continuación.

### \$EQ

El operador \$eq hace referencia a “equals” y permite identificar si un campo es igual a otro. Un ejemplo de uso sería el siguiente.

```
//Documento de filtrado que comprueba que el identificador sea igual a 3.  
Document filtro = new Document("_id",new Document("$eq",3));
```

### \$GT y \$GTE

Los operadores \$gt y \$gte hacen referencia a “Greater Than” y a “Greater Than or Equal”, y permiten identificar si el valor de un campo es mayor o mayor o igual a cierto valor. Un ejemplo de su uso es el siguiente.

```
//Documento de filtrado que comprueba que el identificador sea mayor a 3.  
Document filtro = new Document("_id",new Document("$gt",3));
```

### \$LT y \$LTE

Los operadores \$lt y \$lte hacen referencia a “Lower Than” y a “Lower Than or Equal”, y permiten identificar si el valor de un campo es menor o mayor o igual a cierto valor. Un ejemplo de su uso es el siguiente.

```
//Documento de filtrado que comprueba que el identificador sea menor o igual a 3.  
Document filtro = new Document("_id",new Document("$lte",3));
```

### \$INC

El operador \$inc hace referencia a “Increment” o “Increase” y permite aumentar el valor de un campo. Un ejemplo típico de su uso se da en las operaciones de actualización.

```
//Documento de actualización que incrementa la nota media en 1.  
Document actualizacion = new Document("$inc",new Document("nota_media",1));
```

## \$AND

El operador \$and permite comprobar que varias condiciones lógicas sean ciertas para realizar filtros más complejos. A continuación se muestra un ejemplo de ello.

```
//Documento de filtrado que comprueba que el id sea mayor de 3 y menor de 10.  
//Creamos una lista con las condiciones del and.  
List<Document> condicionesAnd = new ArrayList<>();  
Document condicion1 = new Document("_id",new Document("$gt",3));  
Document condicion2 = new Document("_id",new Document("$lt",10));  
condicionesAnd.add(condicion1);  
condicionesAnd.add(condicion2);  
//Creamos un documento de filtrado con las dos condiciones.  
Document filtro = new Document("$and",condicionesAnd);  
FindIterable<Document> fit = mco.find(filtro);
```

De igual forma, podemos implementar los filtros haciendo uso de la clase Filters.

```
import static com.mongodb.client.model.Filters.*;  
...  
//Documento de filtrado que comprueba que el id sea mayor de 3 y menor de 10.  
FindIterable<Document> fit = mco.find(and(gt("_id",3),lt("_id",10)));
```

## \$OR

Al igual que en el caso anterior, el operador \$or permite comprobar que alguna de las condiciones lógicas sean ciertas. Un ejemplo de su uso es el siguiente.

```
//Documento de filtrado que comprueba que la nota media sea mayor de 6 o el nombre sea  
Marcos.  
//Creamos una lista con las condiciones del or.  
List<Document> docs = new ArrayList<>();  
Document condicion1 = new Document("nota_media",new Document("$gt",6));  
Document condicion2 = new Document("nombre",new Document("$eq","Marcos"));  
docs.add(condicion1);  
docs.add(condicion2);  
//Creamos un documento de filtrado con las dos condiciones.  
FindIterable<Document> fit = mco.find(new Document("$or",docs));
```

De igual manera, podemos implementar el código anterior mediante la clase Filters.

```
FindIterable<Document> fit = mco.find(or(gt("nota_media",6),eq("nombre","Marcos")));
```

# Google Gemini: Plan Pro a 0€ durante 1 año. **Tu ventaja por ser estudiante.**



Oferta válida hasta el 9 de diciembre de 2025

[Consigue la oferta](#)

Después 21,99€/mes

Domina cualquier tema con el Aprendizaje Guiado.

Puedes explicarme como se crea un eclipse lunar completo y

¡Claro vamos paso a paso para que lo entiendas a la perfección!          



Aprendizaje Guiado

6

## Acceso a datos

Diego Calatayud Muñoz

**\$SORT**

El operador sort permite ordenar el resultado de una consulta en función de un campo. El código a implementar es el siguiente. En el caso de este operador, únicamente se puede utilizar mediante la clase Filters.

```
FindIterable<Document> fit = mco.find(new Document(filtro)).sort(Sorts.Ascending("nombre"));
```

## PROYECCIONES

Las proyecciones son el equivalente a la selección de campos a la hora de consultar una tabla en SQL. Si quisieramos obtener únicamente los nombres de los alumnos en SQL haríamos algo parecido “SELECT nombre FROM ALUMNOS”

En el caso de MongoDB, el código Java a implementar es el siguiente.

En este código, excluimos el identificador e incluimos el campo nombre.

## MAPEO DE CLASES

El mapeado automático de POJOs se puede implementar en Java haciendo uso de la clase `CodecRegistry`. Así pues, para poder mapear objetos `alumno`, deberemos modificar la clase para agregar la notación `bson` correspondiente.

Por tanto los atributos de la clase Alumno deberemos definirlos de la siguiente manera.

```
@BsonId  
private Long id;  
private String nombre;  
private int edad;  
private double notaMedia;
```

Una vez definida la clase, la implementación del código de consulta es muy similar.

```
ConnectionString connectionString = new ConnectionString("mongodb://localhost");
CodecRegistry pojoCodecRegistry =
fromProviders(PojoCodecProvider.builder().automatic(true).build());
CodecRegistry codecRegistry =
fromRegistries(MongoClientSettings.getDefaultCodecRegistry(),
    pojoCodecRegistry);
MongoClientSettings clientSettings = MongoClientSettings.builder()
    .applyConnectionString(connectionString)
    .codecRegistry(codecRegistry)
    .build();
try(MongoClient mc = (MongoClient) MongoClients.create(clientSettings))
{
    MongoDB mdb = mc.getDatabase("repasoUT4");
    MongoCollection<Alumno> mco = mdb.getCollection("alumnos", Alumno.class);
    FindIterable<Alumno> fit = mco.find();
    MongoCursor<Alumno> cursor = fit.iterator();
    while (cursor.hasNext())
    {
        System.out.println(cursor.next());
    }
}
catch(MongoException e)
{
    e.printStackTrace();
}
```