

# Trabajo Práctico – Introducción a MongoDB

## Consigna

El objetivo de este trabajo es familiarizarse con MongoDB, utilizando Mongo Shell (mongosh) y Mongo Compass. Se deberán crear múltiples bases de datos, colecciones y documentos, además de aplicar operaciones básicas (insertar, consultar, actualizar y eliminar).

El alumno deberá entregar:

- Un documento con capturas de pantalla de los comandos ejecutados y sus resultados.

## Parte 1 – Preguntas de repaso

### 1. Conceptos generales

#### a. ¿Qué es un documento en MongoDB y en qué formato se almacena?

Es la unidad básica de datos en MongoDB (pares clave–valor, anidables).

Se almacena en BSON (Binary JSON). En la consola lo ves como JSON, pero físicamente es BSON. Siempre tiene un campo `_id` único.

#### b. ¿Qué es una colección y en qué se diferencia de una tabla en una base de datos relacional?

Una colección es un conjunto de documentos, posee un esquema flexible (los documentos pueden tener distintos campos) y una tabla son filas con esquema fijo (mismas columnas).

### 2. Comandos básicos

#### a. ¿Qué hace el comando `use <nombreBD>`?

Cambia el contexto a esa base. La BD no se crea realmente hasta que haya una escritura.

#### b. ¿Qué comando se utiliza para listar todas las bases de datos disponibles?

`show dbs`

#### c. ¿Por qué una base de datos recién creada no aparece en la lista de dbs hasta que se inserta un dato?

Porque `use <bd>` solo cambia el contexto: MongoDB crea las bases de forma diferida (lazy) y solo las materializa cuando hay alguna escritura (p. ej., insertar un documento o crear una colección/índice).

`show dbs` lista únicamente las bases con tamaño `> 0` en disco; si no hubo escritura, la BD no tiene datos y no aparece.

### 3. Inserción de datos

**a. Explica con tus palabras la diferencia entre insertOne() e insertMany().**

insertOne() inserta un documento.

insertMany() inserta varios documentos

**b. ¿Qué ventajas tiene insertMany() cuando queremos cargar grandes volúmenes de datos?**

La ventaja de insertMany() es que permite insertar múltiples documentos en una sola operación, lo que lo hace más rápido y eficiente que ejecutar varios insertOne(), reduciendo el número de solicitudes al servidor y mejorando el rendimiento al cargar grandes volúmenes de datos.

### 4. Consultas

**a. ¿Qué diferencia hay entre find() y find().pretty()?**

Ambos devuelven lo mismo; pretty() solo formatea la salida en la consola para que sea legible (sangrías, saltos de línea).

### 5. Eliminaciones

**a. ¿Qué diferencia hay entre deleteOne(), drop() y dropDatabase()?**

deleteOne(filtro): elimina un documento que cumpla el filtro (existe deleteMany para varios).

drop(): elimina la colección completa (y sus índices).

dropDatabase(): elimina toda la base de datos (todas las colecciones e índices).

**b. Si eliminas una colección con drop(), ¿sigue existiendo la base de datos?**

Sí. La BD permanece. Si queda sin colecciones, puede no listarse hasta que vuelvas a insertar algo.

### 6. Colecciones y administración

**a. ¿Qué hace el comando db.createCollection() y en qué caso puede ser útil, si MongoDB crea colecciones automáticamente al insertar datos?**

El comando db.createCollection("nombreColeccion") crea explícitamente una colección en la base de datos actual.

El comando db.createCollection() es útil en casos específicos, como cuando necesitas:

- Definir opciones avanzadas de la colección, por ejemplo:
  - Tamaño máximo de la colección (capped).
  - Límite de documentos.
  - Validación de esquemas (con reglas sobre los campos).
- Crear colecciones vacías pero con restricciones antes de empezar a insertar datos.

## b. ¿Qué sucede si intentamos insertar un documento sin definir campos (ejemplo: {})?

Si insertamos algo como `db.clientes.insertOne({})` MongoDB inserta un documento vacío, pero le asigna automáticamente un campo `_id` (que es obligatorio y único).

El resultado sería algo como: `{ "_id": ObjectId("64e2fbd3a1c3d2...") }`

No es muy útil insertar `{}` porque no guarda información relevante.

## Parte 2 – Conexión y exploración inicial

### 1. Abrir una terminal y ejecutar:

#### Mongosh

```

MongoDB server starting
Loading configuration files
MongoDB server started
test>

```

### 2. Ejecutar los siguientes comandos de exploración:

`show dbs`

`help`

```

test> show dbs
admin 40.00 KiB
config 12.00 KiB
local 40.00 KiB
test>
test> help
Shell Help:
log
  'log.info(msg)': Write a custom info/warn/error/fatal/debug message to the log file
  'log.getPath()': Gets a path to the current log file
use
  Set current database
show
  'show databases'/'show dbs': Print a list of all available databases
  'show collections'/'show tables': Print a list of all collections for current database
  'show profile': Prints system profile information
  'show users': Print a list of all users for current database
  'show roles': Print a list of all roles for current database
  'show log <name>': Display log for current connection, if name is not set uses 'global'
  'show logs': Print all logger names
exit
  Quit the MongoDB shell with exit()/exit().
quit
  Quit the MongoDB shell with quit()/quit().
mongo
  Create a new connection and return the Mongo object. Usage: new Mongo([URI], options [optional])
connect
  Create a new connection and return the Database object. Usage: connect([URI], username [optional], password [optional])
it
  result of the last line evaluated; use to further iterate
version
  Shell version
load
  Loads and runs a JavaScript file into the current shell environment
enableTelemetry
  Enables collection of anonymous usage data to improve the mongosh CLI
disableTelemetry
  Disables collection of anonymous usage data to improve the mongosh CLI
passwordPrompt
  Prompts the user for a password
sleep
  Sleep for the specified number of milliseconds
print
  Prints the contents of an object to the output
printjson
  Alias for print()

```

## db.help()

```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000
isInteractive           Returns whether the shell will enter or has entered interactive mode

For more information on usage: https://mongodb.com/docs/manual/reference/method
test> db.help()

Database Class:

getMongo                Returns the current database connection
getName                 Returns the name of the DB
getCollectionNames      Returns an array containing the names of all collections in the current database.
getCollectionInfos      Returns an array of documents with collection information, i.e. collection name and options, for the current database.
runCommand              Runs an arbitrary command on the database.
adminCommand            Runs an arbitrary command against the admin database.
aggregate               Runs a specified aggregation pipeline which does not require an underlying collection.
getSiblingDB            Returns another database without modifying the db variable in the shell environment.
getCollection            Returns a collection or a view object that is functionally equivalent to using the db.<collectionName>.
dropDatabase            Removes the current database, deleting the associated data files.
createUser              Creates a new user for the database on which the method is run. db.createUser() returns a duplicate user error if the user already
exists on the database.
updateUser              Updates the user's profile on the database on which you run the method. An update to a field completely replaces the previous field
's values. This includes updates to the user's
changePassword          Updates a user's password. Run the method in the database where the user is defined, i.e. the database you created the user.
logout                 Ends the current authentication session. This function has no effect if the current session is not authenticated.
dropUser               Removes the user from the current database.
dropAllUsers           Removes all users from the current database.
with                   Allows a user to authenticate to the database from within the shell.
grantRolesToUser       Grants additional roles to a user.
revokeRolesFromUser    Removes a one or more roles from a user on the current database.
getUsers               Returns user information for a specified user. Run this method on the user's database. The user must exist on the database on which
the method runs.
getUsers               Returns information for all the users in the database.
createCollection        Create new collection
createEncryptedCollection Create a new collection with a list of encrypted fields each with unique and auto-created data encryption keys (DEKs). This is a u
tility function that internally utilises ClientEncryption.createEncryptedCollection.
createView              Create new view
createRole              Creates a new role.
updateRole              Updates the role's profile on the database on which you run the method. An update to a field completely replaces the previous field
's values.
dropRole               Removes the role from the current database.
dropAllRoles           Removes all roles from the current database.
grantRolesToRole       Grants additional roles to a role.
revokeRolesFromRole    Removes a one or more roles from a role on the current database.
grantPrivilegesToRole  Grants additional privileges to a role.
revokePrivilegesFromRole Removes a one or more privileges from a role on the current database.
getRole                Returns role information for a specified role. Run this method on the role's database. The role must exist on the database on which
the method runs.
getRoles               Returns information for all the roles in the database.
currentOp              Runs an aggregation using $currentOp operator. Returns a document that contains information on in-progress operations for the datab
ase instance. For further information, see $currentOp.

```

## Db

## Parte 3 – Creación de bases de datos y colecciones

1. Crear una base de datos llamada Ecommerce:  
use Ecommerce

```

mongosh mongodb://127.0.0.1:27017/?directConnection=
test> use Ecommerce
switched to db Ecommerce
Ecommerce>

```

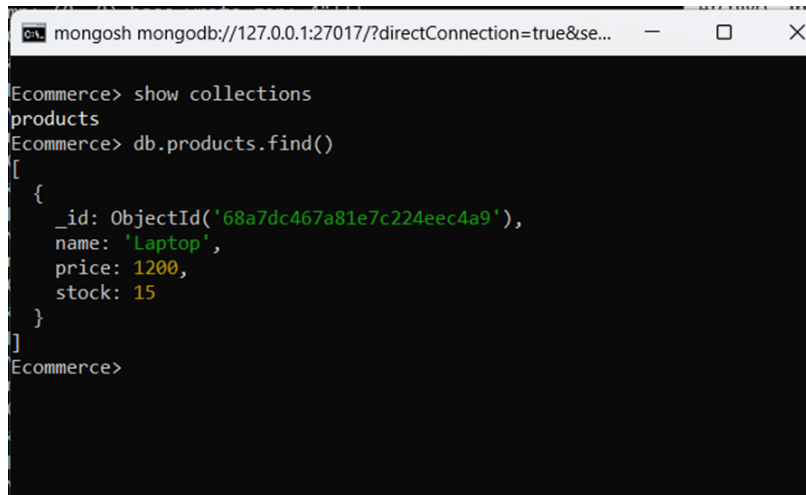
2. Crear una colección products e insertar un documento:  
db.products.insertOne({ name: "Laptop", price: 1200, stock: 15 })

```

Ecommerce> db.products.insertOne({name:"Laptop",price:1200,stock:15})
{
  acknowledged: true,
  insertedId: ObjectId('68a7dc467a81e7c224eec4a9')
}
Ecommerce>

```

3. Verificar con:  
show collections  
db.products.find()



```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...
Ecommerce> show collections
products
Ecommerce> db.products.find()
[
  {
    _id: ObjectId('68a7dc467a81e7c224eec4a9'),
    name: 'Laptop',
    price: 1200,
    stock: 15
  }
]
Ecommerce>

```

### Parte 3 – Inserción de múltiples datos

#### 1. Insertar varios productos:

```

db.products.insertMany([
  { name: "Monitor", price: 400, stock: 25 },
  { name: "Teclado", price: 50, stock: 100 },
  { name: "Mouse", price: 30, stock: 200 }
])

```



```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...
Ecommerce> db.products.insertMany([
...   { name: "Monitor", price: 400, stock: 25 },
...   { name: "Teclado", price: 50, stock: 100 },
...   { name: "Mouse", price: 30, stock: 200 }
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68a7dc8d7a81e7c224eec4aa'),
    '1': ObjectId('68a7dc8d7a81e7c224eec4ab'),
    '2': ObjectId('68a7dc8d7a81e7c224eec4ac')
  }
}
Ecommerce>

```

#### 2. Consultar:

```

db.products.find()

```



```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...
Ecommerce> db.products.find()
[
  {
    _id: ObjectId('68a7dc467a81e7c224eec4a9'),
    name: 'Laptop',
    price: 1200,
    stock: 15
  },
  {
    _id: ObjectId('68a7dc8d7a81e7c224eec4aa'),
    name: 'Monitor',
    price: 400,
    stock: 25
  },
  {
    _id: ObjectId('68a7dc8d7a81e7c224eec4ab'),
    name: 'Teclado',
    price: 50,
    stock: 100
  },
  {
    _id: ObjectId('68a7dc8d7a81e7c224eec4ac'),
    name: 'Mouse',
    price: 30,
    stock: 200
  }
]
Ecommerce>

```

db.products.find({ name: "Monitor" })



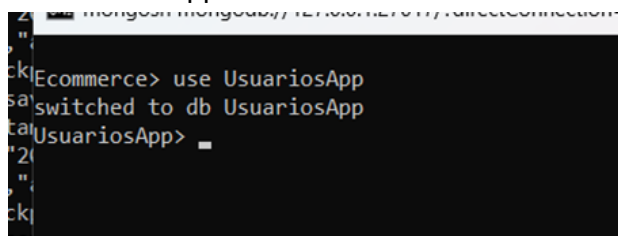
```

mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...
Ecommerce> db.products.find({ name: "Monitor" })
[
  {
    _id: ObjectId('68a7dc8d7a81e7c224eec4aa'),
    name: 'Monitor',
    price: 400,
    stock: 25
  }
]
Ecommerce>

```

## Parte 4 – Otra base de datos con usuarios

1. Crear una base de datos llamada UsuariosApp:  
use UsuariosApp



```

mongosh mongodb://127.0.0.1:27017/?directConnection=...
Ecommerce> use UsuariosApp
switched to db UsuariosApp
UsuariosApp>

```

2. Insertar documentos en una colección users:

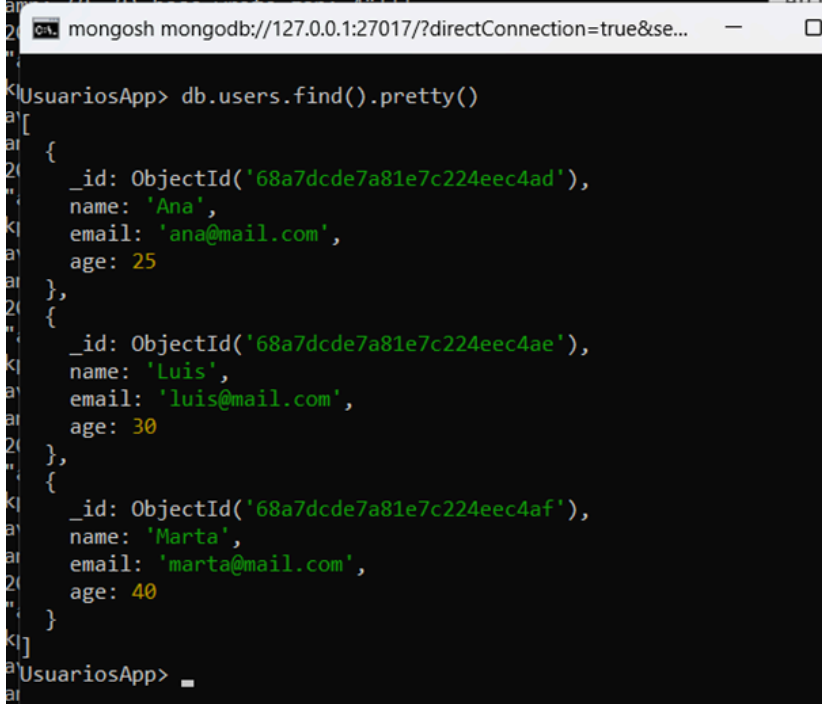
```
db.users.insertMany([
  { name: "Ana", email: "ana@mail.com", age: 25 },
  { name: "Luis", email: "luis@mail.com", age: 30 },
  { name: "Marta", email: "marta@mail.com", age: 40 }
])
```



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...
UsuariosApp> db.users.insertMany([
...   { name: "Ana", email: "ana@mail.com", age: 25 },
...   { name: "Luis", email: "luis@mail.com", age: 30 },
...   { name: "Marta", email: "marta@mail.com", age: 40 }
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68a7dcde7a81e7c224eec4ad'),
    '1': ObjectId('68a7dcde7a81e7c224eec4ae'),
    '2': ObjectId('68a7dcde7a81e7c224eec4af')
  }
}
UsuariosApp>
```

3. Consultar todos los usuarios con:

```
db.users.find().pretty()
```



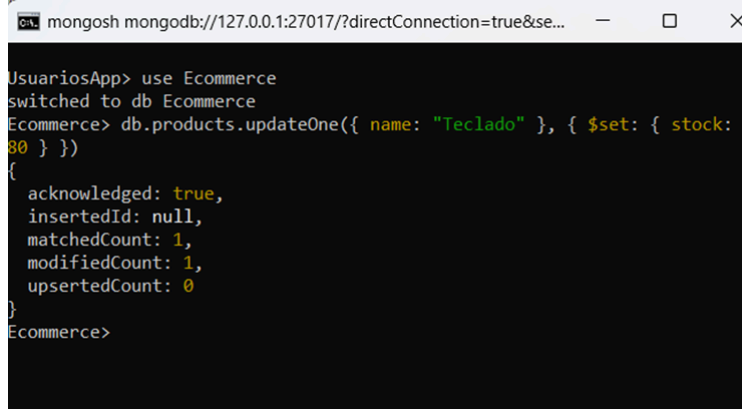
```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...
UsuariosApp> db.users.find().pretty()
[
  {
    _id: ObjectId('68a7dcde7a81e7c224eec4ad'),
    name: 'Ana',
    email: 'ana@mail.com',
    age: 25
  },
  {
    _id: ObjectId('68a7dcde7a81e7c224eec4ae'),
    name: 'Luis',
    email: 'luis@mail.com',
    age: 30
  },
  {
    _id: ObjectId('68a7dcde7a81e7c224eec4af'),
    name: 'Marta',
    email: 'marta@mail.com',
    age: 40
  }
]
UsuariosApp>
```

## Parte 5 – Actualizaciones y eliminaciones

1. Actualizar el stock de un producto:

use Ecommerce

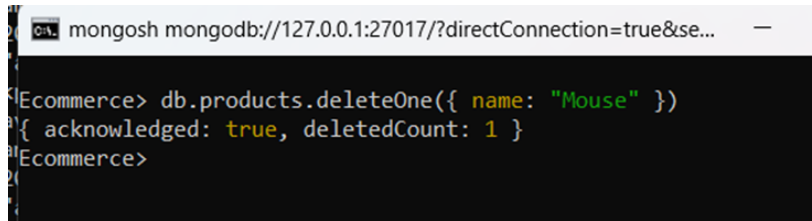
db.products.updateOne({ name: "Teclado" }, { \$set: { stock: 80 } })



```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...  
UsuariosApp> use Ecommerce  
switched to db Ecommerce  
Ecommerce> db.products.updateOne({ name: "Teclado" }, { $set: { stock:  
80 } })  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 1,  
  modifiedCount: 1,  
  upsertedCount: 0  
}  
Ecommerce>
```

2. Eliminar un producto:

db.products.deleteOne({ name: "Mouse" })

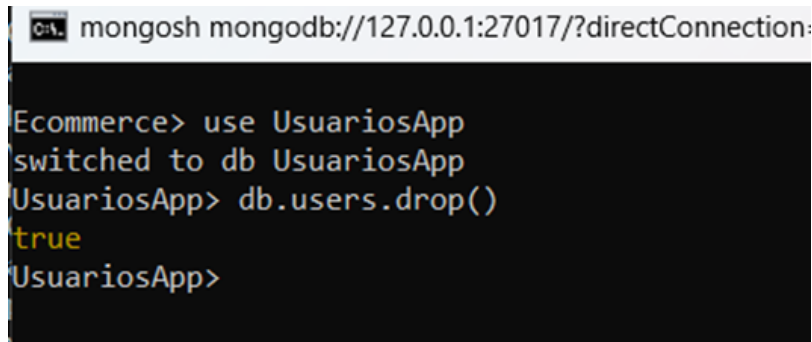


```
mongosh mongodb://127.0.0.1:27017/?directConnection=true&se...  
Ecommerce> db.products.deleteOne({ name: "Mouse" })  
{ acknowledged: true, deletedCount: 1 }  
Ecommerce>
```

3. Eliminar toda la colección users:

use UsuariosApp

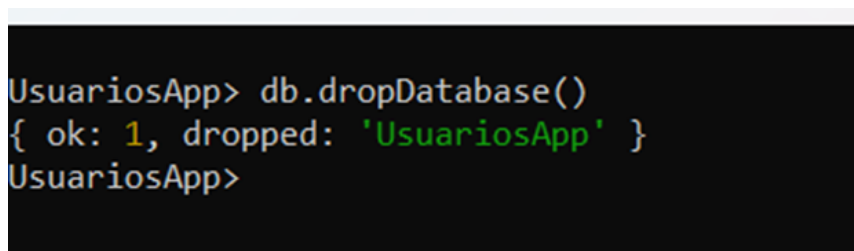
db.users.drop()



```
mongosh mongodb://127.0.0.1:27017/?directConnection:  
Ecommerce> use UsuariosApp  
switched to db UsuariosApp  
UsuariosApp> db.users.drop()  
true  
UsuariosApp>
```

4. Eliminar la base de datos UsuariosApp:

db.dropDatabase()



```
UsuariosApp> db.dropDatabase()  
{ ok: 1, dropped: 'UsuariosApp' }  
UsuariosApp>
```



## Parte 6 – Extra: Crear una colección explícita

1. Crear la colección pedidos en la base Ecommerce:

use Ecommerce

db.createCollection("pedidos")

```
UsuariosApp> use Ecommerce
switched to db Ecommerce
Ecommerce> db.createCollection("pedidos")
{ ok: 1 }
Ecommerce> 
```

2. Insertar algunos pedidos:

```
db.pedidos.insertMany([
  { user: "Ana", product: "Laptop", quantity: 1 },
  { user: "Luis", product: "Monitor", quantity: 2 }
])
```

```
mongosh mongod://127.0.0.1:27017/?directConnection=true&se...
Ecommerce> db.pedidos.insertMany([
...   { user: "Ana", product: "Laptop", quantity: 1 },
...   { user: "Luis", product: "Monitor", quantity: 2 }
... ])
...
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('68a7dd787a81e7c224eec4b0'),
    '1': ObjectId('68a7dd787a81e7c224eec4b1')
  }
}
Ecommerce> 
```

3. Consultar los pedidos con:

db.pedidos.find().pretty()

```
Ecommerce> db.pedidos.find().pretty()
[
  {
    _id: ObjectId('68a7dd787a81e7c224eec4b0'),
    user: 'Ana',
    product: 'Laptop',
    quantity: 1
  },
  {
    _id: ObjectId('68a7dd787a81e7c224eec4b1'),
    user: 'Luis',
    product: 'Monitor',
    quantity: 2
  }
]
Ecommerce> 
```