

## Unidad didáctica 05

### Desarrollo de Aplicaciones con Bases de Datos Documentales en MongoDB usando Python



## Índice

1. Introducción a las Bases de Datos Documentales (NoSQL).....	4
Ventajas:.....	4
Inconvenientes:.....	4
Ejemplo donde el uso de NoSQL es adecuado:.....	4
Ejemplo donde NoSQL no es adecuado:.....	5
Definiciones clave en bases de datos NoSQL:.....	5
Documento:.....	5
Colección:.....	6
Base de datos:.....	6
2. Conexión con la Base de Datos MongoDB usando Python.....	7
Actividad de clase 1:.....	9
3. Gestión de Colecciones en MongoDB.....	10
4. Desarrollo de Aplicaciones para Consultar la Base de Datos.....	11
4.1. Consultas básicas.....	11
4.1.1. Consultar todos los documentos.....	11
4.2. Consultas con condiciones.....	11
4.2.1. Consultar con condiciones simples.....	11
4.2.2. Consultar con múltiples condiciones.....	12
4.3. Consultas con proyección.....	12
4.3.1. Incluir campos específicos.....	13
4.3.2. Excluir campos específicos.....	13
4.4. Consultas con límites y ordenamiento.....	14
4.4.1. Limitar el número de resultados.....	14
4.4.2. Ordenar resultados.....	14
4.5. Consultas con expresiones regulares.....	14
4.5.1. Búsquedas por patrones.....	14
4.6. Contar documentos.....	15
4.6.1. Contar todos los documentos.....	15
4.6.2. Contar documentos con una condición.....	15
Actividad de clase 2:.....	15
5. Desarrollo de Aplicaciones para Añadir, Modificar y Eliminar Documentos.....	15
5.1. Añadir documentos.....	16
5.2. Modificar documentos.....	16
5.2.1. Modificar un solo documento: update_one().....	16
5.2.2. Modificar varios documentos: update_many().....	17
5.3. Eliminar documentos.....	17
5.3.1. Eliminar un solo documento: delete_one().....	17
5.3.2. Eliminar varios documentos: delete_many().....	17

5.4. Actividad de clase 3:.....	18
Anexo A: MongoDB en Ubuntu 24.....	19
A.1. Instalación.....	19
B.2. Creación de una base de datos, una colección y 5 documentos.....	19
A.3. Creación de usuario.....	21

# UD 05: Desarrollo de Aplicaciones con Bases de Datos Documentales en MongoDB usando Python

## 1. Introducción a las Bases de Datos Documentales (NoSQL)

Las bases de datos documentales son un tipo de base de datos NoSQL que almacenan datos en formato de documento, como JSON o BSON (Binary JSON).

Ventajas e inconvenientes de las bases de datos documentales nativas

### Ventajas:

- Flexibilidad en la estructura de los datos: No se necesita un esquema rígido, a diferencia de lo que ocurre con bases de datos SQL.
- Escalabilidad horizontal: Fácil de escalar en clústeres distribuidos.
- Rendimiento en lectura/escritura: Ideal para aplicaciones que manejan grandes volúmenes de datos no estructurados o semi-estructurados.

### Inconvenientes:

- Menor consistencia transaccional comparada con bases de datos relacionales.
- Complejidad en la gestión de relaciones entre documentos, ya que no hay un concepto nativo de claves foráneas como en las bases de datos relacionales.

### Ejemplo donde el uso de NoSQL es adecuado:

Aplicaciones de comercio electrónico (e-commerce): En plataformas como Amazon o eBay, los productos tienen una gran cantidad de variaciones en cuanto a sus atributos (por ejemplo, ropa puede tener tallas y colores, mientras que la electrónica puede tener especificaciones técnicas). NoSQL es ideal porque:

- Se requiere flexibilidad en el esquema para almacenar diferentes tipos de productos con propiedades muy variables.
- La escalabilidad horizontal es esencial debido a la gran cantidad de tráfico y datos generados diariamente, como las reseñas de productos, detalles de los usuarios, y registros de compra.

Ejemplo:

Un documento podría representar un producto con campos como `nombre`, `precio`, `categoría`, `tamaño`, `color`, y `especificaciones`, que varían significativamente entre diferentes tipos de productos.

## Ejemplo donde NoSQL no es adecuado:

Sistemas de gestión de recursos humanos (HR):

Para gestionar empleados, sueldos y nóminas, es importante mantener relaciones y consistencia entre diferentes tablas o entidades (por ejemplo, entre empleados, departamentos y salarios). En este caso, una base de datos relacional es más adecuada porque:

- Los datos están altamente estructurados y requieren relaciones estrictas.
- Es crucial la consistencia para cálculos como la generación de nóminas y actualizaciones de sueldos.

Ejemplo:

Las relaciones entre los empleados, sus departamentos y su historial de nóminas requieren integridad referencial para evitar inconsistencias.

## Definiciones clave en bases de datos NoSQL:

### Documento:

Un documento es la unidad básica de almacenamiento en una base de datos documental NoSQL. Es un registro estructurado en formato JSON o BSON (Binary JSON). Cada documento puede contener varios campos y valores, y lo más importante, los documentos en una misma colección no tienen que seguir un esquema rígido, lo que ofrece mucha flexibilidad.

Ejemplo: mostramos cómo los documentos en bases de datos NoSQL pueden tener diferentes campos es el siguiente: Usuarios en una aplicación de streaming de música:

Usuario 1

```
{
  "nombre": "Carlos",
  "edad": 32,
  "plan": "premium",
  "playlists": ["Rock Clásico", "Jazz", "Pop Latino"]
}
```

Usuario 2:

```
{
  "nombre": "María",
  "edad": 25,
  "plan": "gratis",
}
```

```
"anuncios_vistos": 23,  
"última_canción_escuchada": "Shape of You"  
}
```

En este ejemplo, ambos documentos representan usuarios de una aplicación de música, pero:

- Usuario 1 tiene un campo `playlists` con una lista de sus playlists.
- Usuario 2 al tener un plan gratuito, tiene campos como `anuncios\_vistos` y `última\_canción\_escuchada`, que no están presentes en el documento del Usuario 1.

Este caso ilustra la flexibilidad de las bases de datos NoSQL al manejar diferentes tipos de información en documentos dentro de una misma colección, sin necesidad de un esquema predefinido para todos los usuarios.

## Colección:

Una colección es un grupo de documentos relacionados. En bases de datos relacionales, la colección sería equivalente a una tabla, pero con la ventaja de que cada documento dentro de una colección puede tener una estructura diferente. Las colecciones no imponen un esquema rígido, lo que permite una gran flexibilidad para manejar diferentes tipos de datos.

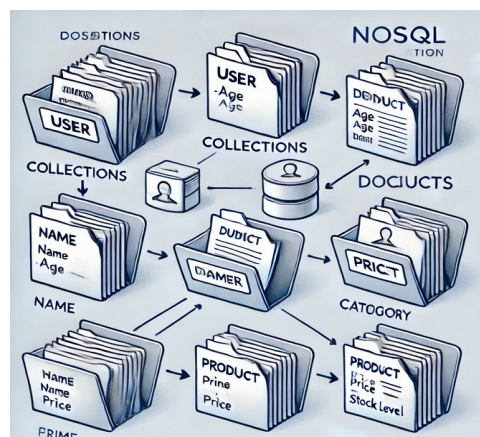
Ejemplo:

Una colección llamada `usuarios` puede contener documentos de usuarios con distintos campos. Algunos usuarios pueden tener el campo `foto\_de\_perfil`, mientras que otros no.

## Base de datos:

En NoSQL, una base de datos es un conjunto de colecciones. Cada base de datos puede contener varias colecciones, y a su vez, cada colección contiene múltiples documentos. Las bases de datos NoSQL permiten manejar grandes volúmenes de información de manera escalable y eficiente.

Ejemplo: na base de datos de una tienda online podría tener varias colecciones como `productos`, `clientes`, y `pedidos`.



## 2. Conexión con la Base de Datos MongoDB usando Python

*[Nota: para la realización de este apartado es imprescindible haber instalado en local la base de datos MongoDB tal y como se indica en el Anexo A]*

Instalación de dependencias: Para trabajar con MongoDB en Python, se utiliza la librería `pymongo``:

```
pip install pymongo
```

Conexión a MongoDB: Ejemplo de código para conectarse a nuestra base de datos MongoDB alojada localmente:

```
from pymongo import MongoClient, errors

# Datos de conexión
usuario = "usuario"
clave = "usuario"
base_datos = "ldam"
host = "localhost"
puerto = 27017

try:
    # Intentar conectarse al servidor MongoDB

    client = MongoClient( f"mongodb://{usuario}:{clave}@{host}:{puerto}/{base_datos}",
serverSelectionTimeoutMS=5000)

    # Seleccionar la base de datos
    db = client[base_datos]

    # Intentar acceder a la base de datos para verificar la conexión
    colecciones = db.list_collection_names()

    print("Conexión exitosa. Colecciones en la base de datos:")
    print(colecciones)

except errors.ServerSelectionTimeoutError as err:
    # Este error ocurre si el servidor no está disponible o no se puede conectar
    print(f"No se pudo conectar a MongoDB: {err}")

except errors.OperationFailure as err:
    # Este error ocurre si las credenciales son incorrectas o no se tienen los permisos necesarios
    print(f"Fallo en la autenticación o permisos insuficientes: {err}")

except Exception as err:
    # Manejar cualquier otro error inesperado
    print(f"Ocurrió un error inesperado: {err}")

finally:
    # Cerrar la conexión si se estableció correctamente
    if 'client' in locals():
        client.close()
        print("Conexión cerrada.")
```

Explicación de las excepciones:

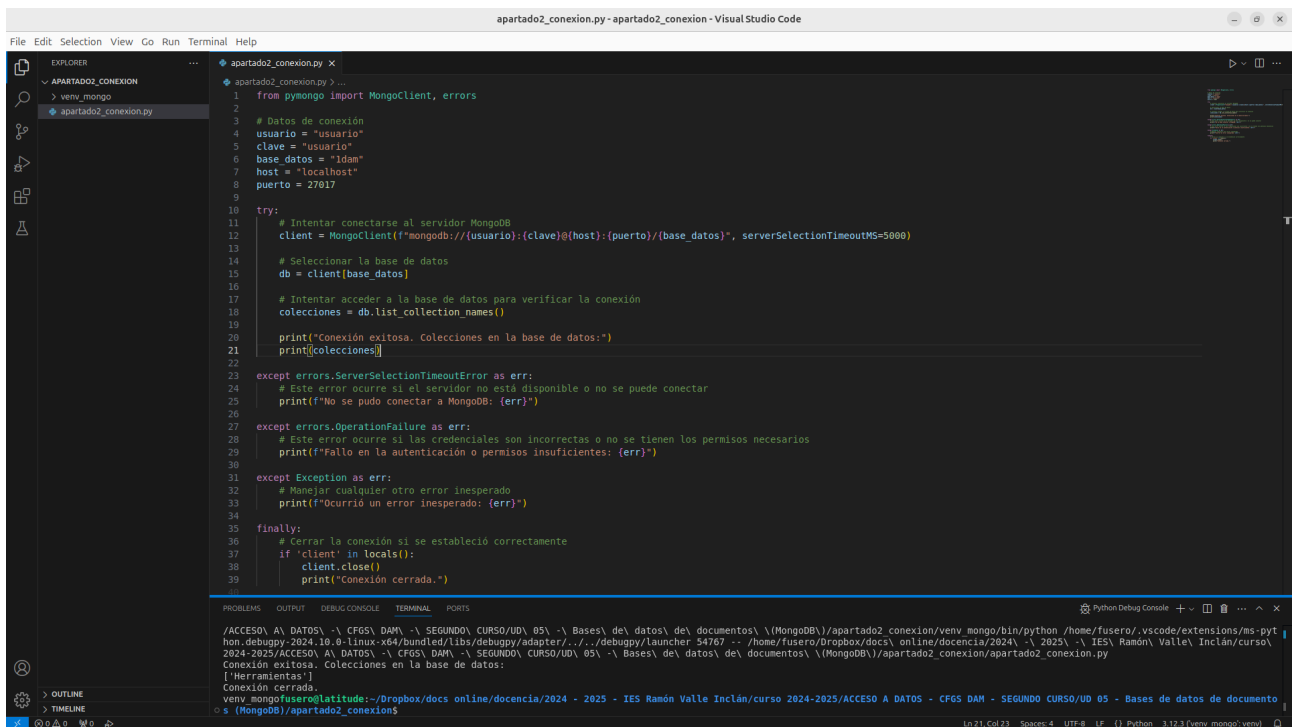
- ``ServerSelectionTimeoutError``: Este error se lanza si el cliente no puede conectarse al servidor MongoDB dentro del tiempo especificado. Esto puede suceder si el servidor está apagado, si el puerto está bloqueado, o si la dirección IP es incorrecta.
- ``OperationFailure``: Este error se produce si hay un fallo en la autenticación o si el usuario no tiene los permisos necesarios para acceder a la base de datos solicitada.
- ``Exception``: Captura cualquier otro error no anticipado que pueda ocurrir durante la ejecución del programa.

### Notas:

- ``serverSelectionTimeoutMS=5000``: Este parámetro establece el tiempo de espera para la selección del servidor en 5 segundos (5000 milisegundos). Si el cliente no puede conectarse dentro de ese tiempo, lanzará una excepción.

Este código maneja adecuadamente posibles errores, como fallos de conexión, credenciales incorrectas o permisos insuficientes. Aquí tienes una captura de cómo se ve en el ordenador:





```
File Edit Selection View Go Run Terminal Help
apartado2_conexion.py - apartado2_conexion - Visual Studio Code

EXPLORER
  APARTADO2_CONEXION
    venv_mongo
    apartado2_conexion.py

apartado2_conexion.py
1 from pymongo import MongoClient, errors
2
3 # Datos de conexión
4 usuario = "usuario"
5 clave = "usuario"
6 base_datos = "Idem"
7 host = "localhost"
8 puerto = 27017
9
10 try:
11     # Intentar conectarse al servidor MongoDB
12     client = MongoClient(f"mongodb://{usuario}:{clave}@{host}:{puerto}/{base_datos}", serverSelectionTimeoutMS=5000)
13
14     # Seleccionar la base de datos
15     db = client[base_datos]
16
17     # Intentar acceder a la base de datos para verificar la conexión
18     colecciones = db.list_collection_names()
19
20     print("Conexión exitosa. Colecciones en la base de datos:")
21     print(colecciones)
22
23 except errors.ServerSelectionTimeoutError as err:
24     # Este error ocurre si el servidor no está disponible o no se puede conectar
25     print(f"No se pudo conectar a MongoDB: {err}")
26
27 except errors.OperationFailure as err:
28     # Este error ocurre si las credenciales son incorrectas o no se tienen los permisos necesarios
29     print(f"Fallo en la autenticación o permisos insuficientes: {err}")
30
31 except Exception as err:
32     # Manejar cualquier otro error inesperado
33     print(f"Ocurrió un error inesperado: {err}")
34
35 finally:
36     # Cerrar la conexión si se estableció correctamente
37     if 'client' in locals():
38         client.close()
39     print("Conexión cerrada.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python Debug Console
/ACCESO/ A DATOS\ -\ CFGS\ DAM\ -\ SEGUNDO\ CURSO\ UD\ 05\ -\ Bases\ de\ datos\ de\ documentos\ \ (MongoDB)\ apartado2_conexion\ venv_mongo\ bin\ python /home/fusero/.vscode/extensions/ms-pyt
hon.debugpy-2024.18.0-linux-x64/bundled/libs/debugpy/adapter/../../debugpy/launcher 54767 -- /home/fusero/dropbox/docs/online/docencia/2024\ -\ 2025\ -\ IES\ Ramón\ Valle\ Inclán\ curso\
2024-2025\ ACCESO\ A DATOS\ -\ CFGS\ DAM\ -\ SEGUNDO\ CURSO\ UD\ 05\ -\ Bases\ de\ datos\ de\ documentos\ \ (MongoDB)\ apartado2_conexion\ apartado2_conexion.py
Conexión exitosa. Colecciones en la base de datos:
['Herramientas']
Conexión cerrada.
venv_mongo\ fusero@latitude:~/Dropbox/docs/online/docencia/2024 - 2025 - IES Ramón Valle Inclán/curso 2024-2025/ACCESO A DATOS - CFGS DAM - SEGUNDO CURSO/UD 05 - Bases de datos de documento
o s (MongoDB)\ apartado2_conexion$
```

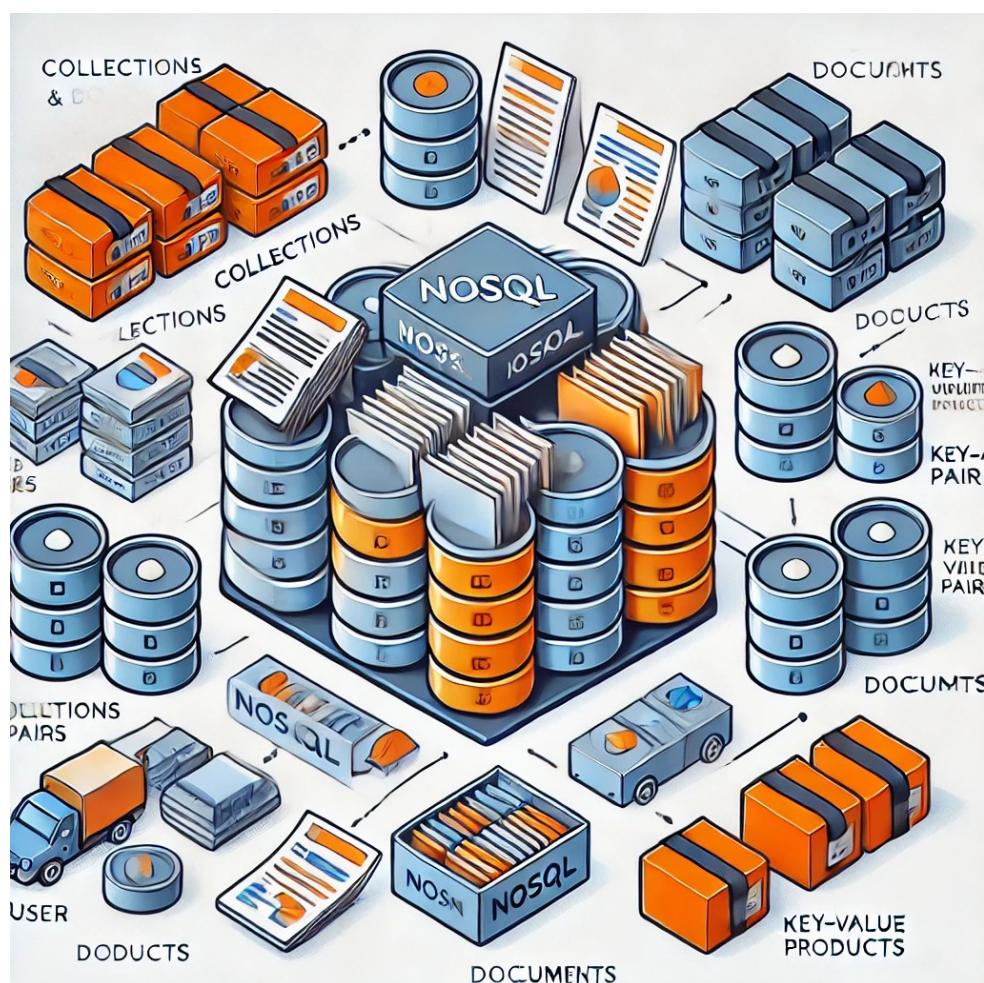
## Actividad de clase 1:

1. Objetivo: El alumnado deberá conectar su entorno de desarrollo Visual Studio Code a su base de datos MongoDB alojada localmente, con control de excepciones y adjuntar una captura de pantalla como la anterior, en la que se vea el código del programa y la traza de la ejecución.

### 3. Gestión de Colecciones en MongoDB

Una colección en MongoDB es similar a una tabla en una base de datos relacional, pero la diferencia principal es que en MongoDB no existe un esquema predefinido para la estructura de los documentos.

Cada documento dentro de una colección puede tener una estructura diferente. Los documentos en MongoDB están en formato BSON (Binary JSON), lo que permite mayor flexibilidad en el tipo de datos almacenados.



MongoDB permite crear colecciones automáticamente cuando se inserta el primer documento, pero también es posible crearlas manualmente.

```
db.create_collection("nueva_coleccion")
```

### Eliminar una colección:

```
db.drop_collection("nueva coleccion")
```

## 4. Desarrollo de Aplicaciones para Consultar la Base de Datos

MongoDB ofrece una gran flexibilidad para realizar consultas, que pueden ir desde simples búsquedas hasta consultas avanzadas con agregaciones. Aquí profundizaremos en las operaciones más comunes y avanzadas que puedes realizar para consultar la base de datos.

### 4.1. Consultas básicas

MongoDB permite realizar consultas muy simples utilizando métodos como `find()` y `find_one()`. El resultado de una consulta será un cursor que puede ser iterado para obtener los documentos.

#### 4.1.1. Consultar todos los documentos

Usamos `find()` sin ningún argumento para obtener todos los documentos de una colección.

```
# Seleccionar la colección
coleccion_usuarios = db["usuarios"]
# Consultar todos los documentos
usuarios = coleccion_usuarios.find()
for usuario in usuarios:
    print(usuario)
```

### 4.2. Consultas con condiciones

MongoDB permite realizar consultas más específicas utilizando operadores como `$eq` (igual a), `$gt` (mayor que), `$lt` (menor que), `$in` (dentro de un conjunto), y muchos más.

#### 4.2.1. Consultar con condiciones simples

MongoDB utiliza una sintaxis específica para las consultas basadas en condiciones. La estructura básica de una consulta es un diccionario donde las claves representan los campos sobre los que queremos filtrar, y los valores pueden ser tanto un valor directo como un operador. Los operadores en MongoDB siempre van precedidos por un signo \$.

Ejemplo de sintaxis:

Igualdad:

```
{"campo": "valor"}
```

Esto buscará los documentos donde el campo sea igual a "valor".

Operadores: Cuando necesitas realizar comparaciones más complejas, como mayor que, menor que, o diferentes, debes usar operadores como `$gt`, `$lt`, `$ne`, entre otros.

Ejemplo con `$gt` (mayor que):

```
{"edad": {"$gt": 30}}
```

Esto buscará todos los documentos donde el campo edad sea mayor que 30.

Combinación de operadores: Puedes usar múltiples operadores dentro de la misma consulta para aplicar diferentes condiciones sobre el mismo campo.

Ejemplo:

```
{"edad": {"$gt": 30, "$lt": 50}}
```

Esto devolverá los documentos donde la edad esté entre 30 y 50 años.

Aquí buscamos los usuarios cuya edad es mayor que 30.

```
consulta = {"edad": {"$gt": 30}}
usuarios = coleccion_usuarios.find(consulta)
for usuario in usuarios:
    print(usuario)
```

### 4.2.2. Consultar con múltiples condiciones

Podemos usar varias condiciones en una consulta combinando operadores lógicos como ``$and`` y ``$or``.

```
# Buscar usuarios mayores de 30 años o cuyo nombre sea 'Alberto'
consulta = {
    "$or": [
        {"edad": {"$gt": 30}},
        {"nombre": "Alberto"}
    ]
}
usuarios = coleccion_usuarios.find(consulta)
for usuario in usuarios:
    print(usuario)
```

## 4.3. Consultas con proyección

La proyección nos permite seleccionar solo ciertos campos de los documentos. Para ello, usamos un segundo argumento en el método ``find()`` que especifica qué campos queremos incluir (o excluir).

### 4.3.1. Incluir campos específicos

En una consulta MongoDB con proyección, si intentas incluir un campo que no existe en algunos documentos, esos documentos **sí serán mostrados**. Sin embargo, el campo que no existe en el documento simplemente no aparecerá en el resultado.

Ejemplo:

Supón que tienes una colección usuarios con algunos documentos que tienen el campo email y otros que no lo tienen. Si haces la siguiente consulta y proyección:

```
consulta = {"edad": {"$gt": 25}}
proyeccion = {"nombre": 1, "edad": 1, "email": 1, "_id": 0}
usuarios = coleccion_usuarios.find(consulta, proyeccion)
for usuario in usuarios:
    print(usuario)
```

Resultado:

Si un documento tiene el campo email, será incluido en la salida. Si un documento no tiene el campo email, MongoDB simplemente omitirá ese campo para ese documento, pero el documento será mostrado de todas formas con los campos disponibles (nombre y edad).

Ejemplo de salida:

```
{"nombre": "Carlos", "edad": 28, "email": "carlos@example.com"}
{"nombre": "Ana", "edad": 35} # El campo 'email' no existe, pero el documento se muestra igual
```

MongoDB no filtrará los documentos solo porque les falta un campo que has especificado en la proyección. Solo muestra los campos que están presentes en cada documento.

### 4.3.2. Excluir campos específicos

En lugar de incluir campos, también podemos excluir ciertos campos. Aquí excluimos el campo `edad`:

```
python
proyeccion = {"edad": 0}
usuarios = coleccion_usuarios.find({}, proyeccion)
for usuario in usuarios:
    print(usuario)
```

## 4.4. Consultas con límites y ordenamiento

Podemos limitar el número de documentos devueltos y también ordenarlos.

### 4.4.1. Limitar el número de resultados

Usamos `limit()` para limitar el número de resultados devueltos. Aquí devolvemos solo los primeros 3 usuarios.

```
usuarios = coleccion_usuarios.find().limit(3)
for usuario in usuarios:
    print(usuario)
```

### 4.4.2. Ordenar resultados

Para ordenar los resultados, usamos `sort()`. Por ejemplo, para ordenar los usuarios por edad en orden descendente:

```
usuarios = coleccion_usuarios.find().sort("edad", -1) # -1 para descendente
for usuario in usuarios:
    print(usuario)
```

## 4.5. Consultas con expresiones regulares

Podemos utilizar expresiones regulares para hacer búsquedas más flexibles, como buscar todos los nombres que comiencen con una letra específica.

### 4.5.1. Búsquedas por patrones

Buscar todos los usuarios cuyo nombre comience con la letra "A":

```
consulta = {"nombre": {"$regex": "^A"}}
usuarios = coleccion_usuarios.find(consulta)
for usuario in usuarios:
    print(usuario)
```

## 4.6. Contar documentos

Si simplemente queremos saber cuántos documentos hay en una colección o cuántos cumplen una condición específica, podemos usar `count_documents()`.

### 4.6.1. Contar todos los documentos

```
total_usuarios = coleccion_usuarios.count_documents({})
print(f"Total de usuarios: {total_usuarios}")
```

### 4.6.2. Contar documentos con una condición

```
total_usuarios_mayores = coleccion_usuarios.count_documents({"edad": {"$gt": 30}})
print(f"Total de usuarios mayores de 30 años: {total_usuarios_mayores}")
```

## Actividad de clase 2:

1. Objetivo: Practicar consultas básicas, proyecciones y límites/ordenamientos en MongoDB usando la tabla **que tenga asignada cada alumno**.

2. Instrucciones:

- **Consultar los objetos que cumplan una determinada condición.** Por ejemplo, con herramientas sería realizar una consulta que busque todas las herramientas cuyo tipo sea "Manual". Mostrar dichos objetos por pantalla.
- **Proyectar campos específicos:** Realiza una proyección sobre la consulta anterior para mostrar solo dos campos, excluyendo todos los demás campos, incluido el `_id`.
- **Limitar y ordenar los resultados:** Realiza la consulta con proyección, pero ahora limita el número de resultados a solo 2 documentos y ordénalos por el campo que quieras en orden alfabético ascendente.

## 5. Desarrollo de Aplicaciones para Añadir, Modificar y Eliminar Documentos

Veamos cómo realizar las distintas operaciones:

## 5.1. Añadir documentos.

Para insertar un documento en una colección:

```
nueva_herramienta = { "nombre": "Cinzel", "tipo": "Manual", "material": "Acero", "marca": "Stanley" }
# Insertar la herramienta en la colección
resultado = coleccion_herramientas.insert_one(nueva_herramienta)
print("ID del documento insertado:", resultado.inserted_id)
```

Para insertar varios documentos:

```
# Definir varios documentos de herramientas
herramientas = [
    {"nombre": "Taladro", "tipo": "Eléctrico", "material": "Plástico", "marca": "Bosch"},
    {"nombre": "Martillo", "tipo": "Manual", "material": "Acero", "marca": "Stanley"},
    {"nombre": "Sierra", "tipo": "Manual", "material": "Acero", "marca": "Truper"},
    {"nombre": "Llave Inglesa", "tipo": "Manual", "material": "Acero", "marca": "Black&Decker"},
    {"nombre": "Destornillador", "tipo": "Manual", "material": "Metal", "marca": "Stanley"}
]

# Insertar múltiples documentos en la colección
resultado = coleccion_herramientas.insert_many(herramientas)

# Imprimir los IDs de los documentos insertados
print("IDs de documentos insertados:", resultado.inserted_ids)
```

## 5.2. Modificar documentos

### 5.2.1. Modificar un solo documento: update\_one()

Si deseas modificar un solo documento, debes especificar un criterio de búsqueda para identificar el documento y luego usar \$set para modificar los campos que deseas. Si el criterio de búsqueda coincide con más de un documento, solo el primer documento que MongoDB encuentre (según el orden en que están almacenados en la base de datos) será actualizado.

No hay forma de predecir qué documento será el primero, ya que MongoDB no garantiza un orden específico de los documentos a menos que se haya especificado explícitamente un criterio de ordenación en las consultas.

Ejemplo para modificar un solo documento: Supongamos que quieres actualizar la marca de una herramienta llamada Martillo:

```
resultado = coleccion_herramientas.update_one(
    {"nombre": "Martillo"}, # Filtro para encontrar el documento
    {"$set": {"marca": "Truper"}} # Campos que deseas modificar
)

# Verificar si el documento fue actualizado
if resultado.modified_count > 0:
    print("Documento modificado con éxito.")
else:
```



```
print("No se encontró el documento o no hubo cambios.")
```

### 5.2.2 Modificar varios documentos: update\_many()

Si deseas modificar varios documentos al mismo tiempo, puedes utilizar update\_many() y proporcionar un criterio de búsqueda más general.

Ejemplo para modificar varios documentos: Supongamos que quieres actualizar todas las herramientas de tipo Manual para cambiar su material a Hierro:

```
# Actualizar varios documentos (cambiar el material de todas las herramientas 'Manual')
resultado = coleccion_herramientas.update_many(
    {"tipo": "Manual"}, # Filtro para encontrar los documentos
    {"$set": {"material": "Hierro"}} # Campos que deseas modificar
)

# Mostrar cuántos documentos fueron modificados
print(f"Se modificaron {resultado.modified_count} documentos.")
```

## 5.3. Eliminar documentos

En MongoDB, para eliminar documentos de una colección, puedes utilizar dos métodos principales en pymongo: delete\_one() para eliminar un solo documento y delete\_many() para eliminar varios documentos que coincidan con un criterio.

### 5.3.1. Eliminar un solo documento: delete\_one()

Este método elimina el primer documento que coincida con el criterio que especifiques. Solo elimina un documento, incluso si hay varios que coinciden con el filtro.

Ejemplo para eliminar un solo documento: Supongamos que quieres eliminar una herramienta llamada Martillo:

```
# Eliminar un solo documento (eliminar el Martillo)
resultado = coleccion_herramientas.delete_one({"nombre": "Martillo"})
# Verificar si el documento fue eliminado
if resultado.deleted_count > 0:
    print("Documento eliminado con éxito.")
else:
    print("No se encontró el documento para eliminar.")
```

### 5.3.2. Eliminar varios documentos: delete\_many()

Este método elimina todos los documentos que coincidan con el criterio de búsqueda. Si no proporcionas ningún filtro, eliminará todos los documentos de la colección (aunque esto es peligroso y debe usarse con precaución).

Ejemplo para eliminar varios documentos: Supongamos que quieres eliminar todas las herramientas de tipo Manual:

```
# Eliminar todos los documentos de tipo 'Manual'
resultado = coleccion_herramientas.delete_many({"tipo": "Manual"})

# Mostrar cuántos documentos fueron eliminados
print(f"Se eliminaron {resultado.deleted_count} documentos.")
```

## 5.4. Actividad de clase 3:

1. Objetivo: Los alumnos deben crear un sistema CRUD básico donde puedan añadir, modificar y eliminar documentos en la colección de objetos que le fue asignada a principios de curso.

2. Instrucciones:

- Añadir tres documentos a la colección.
- Actualizar un campo de un sólo documento.
- Eliminar uno de los documentos.

## Anexo A: MongoDB en Ubuntu 24

### A.1. Instalación

Seguiremos las instrucciones de la documentación oficial de MongoDB:

<https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/>

(Ojo, sólo la parte de instalación)

Una vez instalado, nos aseguramos de que arrancamos el servicio cada vez que arrancamos el ordenador:

```
sudo systemctl enable mongod
```

Podemos iniciar una sesión de consola con el servidor de Mongo tecleando:

```
mongosh
```

```
fusero@latitude:~$ mongosh
Current Mongosh Log ID: 678550b5fac83b0333964032
Connecting to:      mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000&appName=mongosh+2.3.1
Using MongoDB:      8.0.0
Using Mongosh:       2.3.1

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
2024-10-08T17:32:44.801+02:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/prodnotes-filesystem
2024-10-08T17:32:45.240+02:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2024-10-08T17:32:45.240+02:00: For customers running the tcmalloc-google memory allocator, we suggest setting the contents of sysfsFile to 'always'
2024-10-08T17:32:45.240+02:00: For customers running the updated tcmalloc-google memory allocator, we suggest setting the contents of sysfsFile to 'defer+advise'
2024-10-08T17:32:45.240+02:00: We suggest setting the contents of sysfsFile to 0.
2024-10-08T17:32:45.240+02:00: We suggest setting swappiness to 0 or 1, as swapping can cause performance problems.
-----
test> |
```

Si no te funcionara, prueba parando y arrancando el demonio. No reiniciar. Primero parar y después arrancar.

### B.2. Creación de una base de datos, una colección y 5 documentos

Todos el alumnado creará su base de datos “1dam”, con una colección que se corresponda con el objeto que tiene asignado este curso (en el caso del profesor, “Herramientas”) y 5 documentos con campos **diversos** desde la consola de mongosh.

A modo de ejemplo mostraremos como hacerlo en el caso del objeto asignado al profesor, en este caso (herramientas). Se pueden seguir estos pasos:

1. Conectar a MongoDB:

Si ya estás en la consola de `mongosh`, puedes proceder directamente con los siguientes comandos.

2. Crear la base de datos `1dam`: Para seleccionar (o crear) la base de datos `1dam`:

```
use 1dam
```

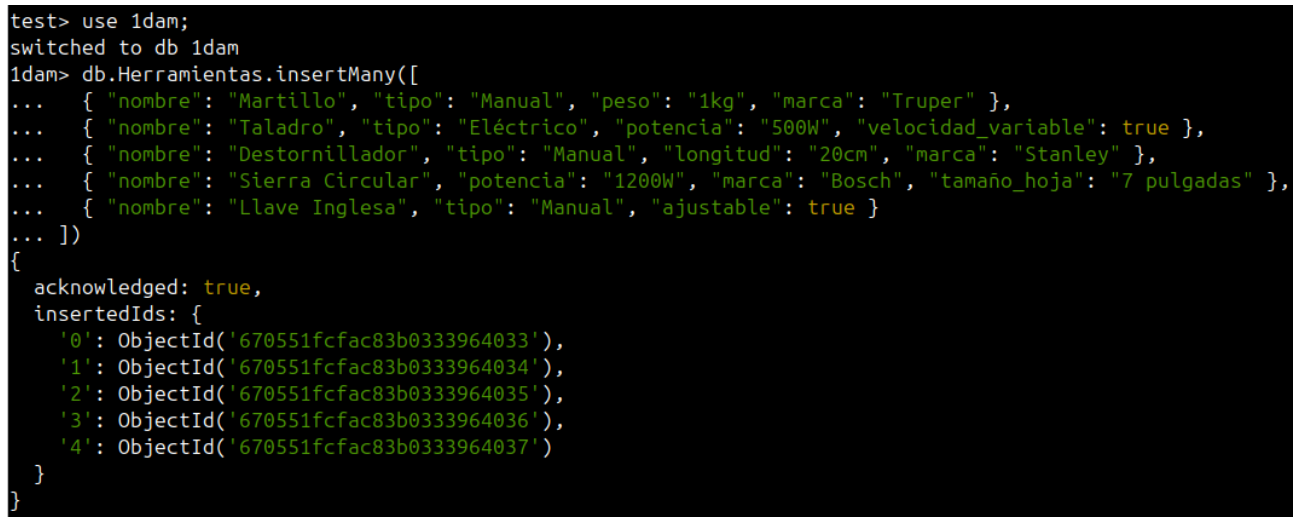
3. Crear la colección `Herramientas`: MongoDB creará la colección automáticamente cuando insertemos el primer documento. Vamos a añadir 5 documentos con campos diversos.

4. Insertar los documentos en la colección `Herramientas`: Ejecuta los siguientes comandos para insertar los documentos:

```
db.Herramientas.insertMany([
  { "nombre": "Martillo", "tipo": "Manual", "peso": "1kg", "marca": "Truper" },
  { "nombre": "Taladro", "tipo": "Eléctrico", "potencia": "500W", "velocidad_variable": true },
  { "nombre": "Destornillador", "tipo": "Manual", "longitud": "20cm", "marca": "Stanley" },
  { "nombre": "Sierra Circular", "potencia": "1200W", "marca": "Bosch", "tamaño_hoja": "7 pulgadas" },
  { "nombre": "Llave Inglesa", "tipo": "Manual", "ajustable": true }
])
```

Aquí hemos añadido 5 herramientas, y puedes ver que algunos documentos tienen campos diferentes. Por ejemplo:

- El Martillo tiene los campos `peso` y `marca`.
- El Taladro tiene los campos `potencia` y `velocidad\_variable`.
- La Llave Inglesa tiene el campo `ajustable`.



```
test> use 1dam;
switched to db 1dam
1dam> db.Herramientas.insertMany([
...   { "nombre": "Martillo", "tipo": "Manual", "peso": "1kg", "marca": "Truper" },
...   { "nombre": "Taladro", "tipo": "Eléctrico", "potencia": "500W", "velocidad_variable": true },
...   { "nombre": "Destornillador", "tipo": "Manual", "longitud": "20cm", "marca": "Stanley" },
...   { "nombre": "Sierra Circular", "potencia": "1200W", "marca": "Bosch", "tamaño_hoja": "7 pulgadas" },
...   { "nombre": "Llave Inglesa", "tipo": "Manual", "ajustable": true }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('670551fcfac83b0333964033'),
    '1': ObjectId('670551fcfac83b0333964034'),
    '2': ObjectId('670551fcfac83b0333964035'),
    '3': ObjectId('670551fcfac83b0333964036'),
    '4': ObjectId('670551fcfac83b0333964037')
  }
}
```

5. Verificar los documentos insertados: para asegurarte de que los documentos se han insertado correctamente, puedes consultar la colección con el siguiente comando:

```
db.Herramientas.find().pretty()
```

Este comando mostrará todos los documentos de la colección `Herramientas` con un formato legible.

```
1dam> db.Herramientas.find().pretty()
[
  {
    _id: ObjectId('670551fcfac83b0333964033'),
    nombre: 'Martillo',
    tipo: 'Manual',
    peso: '1kg',
    marca: 'Truper'
  },
  {
    _id: ObjectId('670551fcfac83b0333964034'),
    nombre: 'Taladro',
    tipo: 'Eléctrico',
    potencia: '500W',
    velocidad_variable: true
  },
  {
    _id: ObjectId('670551fcfac83b0333964035'),
    nombre: 'Destornillador',
    tipo: 'Manual',
    longitud: '20cm',
    marca: 'Stanley'
  },
  {
    _id: ObjectId('670551fcfac83b0333964036'),
    nombre: 'Sierra Circular',
    potencia: '1200W',
    marca: 'Bosch',
    'tamaño_hoja': '7 pulgadas'
  },
  {
    _id: ObjectId('670551fcfac83b0333964037'),
    nombre: 'Llave Inglesa',
    tipo: 'Manual',
    ajustable: true
  }
]
```

Si todo ha funcionado bien, ahora tendrás una base de datos `1dam` con una colección `Herramientas` que contiene 5 documentos con diferentes campos.

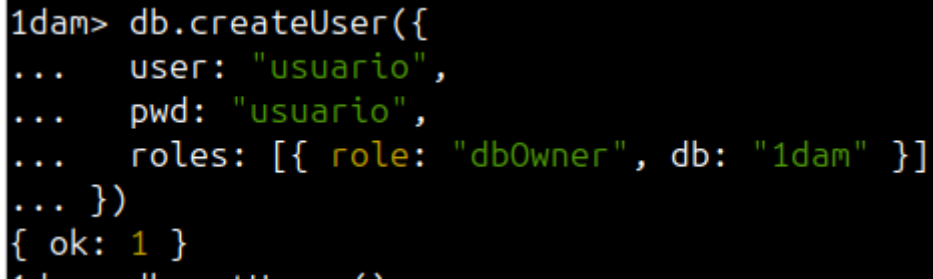
### A.3. Creación de usuario

1. Selecciona la base de datos 1dam: Si aún no lo has hecho, selecciona la base de datos 1dam:

use 1dam

2. Crear el usuario con permisos completos: Para crear un usuario con permisos de lectura y escritura en la base de datos, utiliza el siguiente comando:

```
db.createUser({
  user: "usuario",
  pwd: "usuario",
  roles: [{ role: "dbOwner", db: "1dam" }]
})
```



```
1dam> db.createUser({
...   user: "usuario",
...   pwd: "usuario",
...   roles: [{ role: "dbOwner", db: "1dam" }]
... })
{ ok: 1 }
```

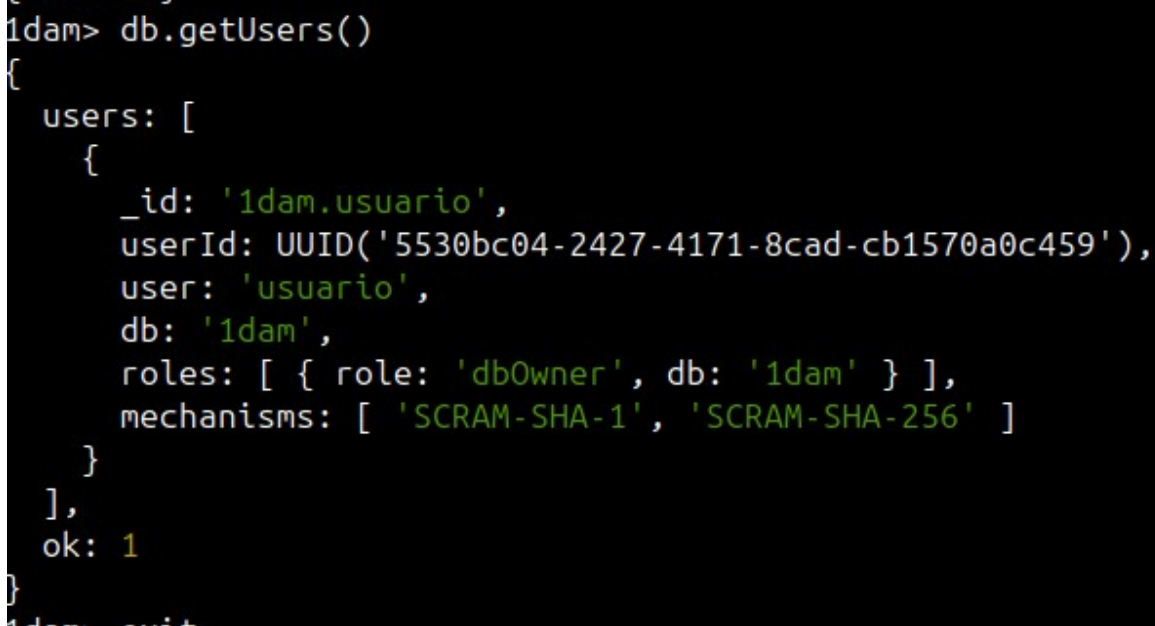
Desglose del comando:

- "user": "usuario" — Nombre del usuario.
- "pwd": "usuario" — Contraseña del usuario.
- "roles": [{ role: "dbOwner", db: "1dam" }] — Se le otorga el rol dbOwner, lo que le da control total sobre la base de datos 1dam.

3. Verificar la creación del usuario: Para asegurarte de que el usuario fue creado correctamente, puedes listar los usuarios de la base de datos con el siguiente comando:

```
db.getUsers()
```

Esto te mostrará los usuarios creados en la base de datos actual, incluyendo al nuevo usuario usuario.



```
1dam> db.getUsers()
{
  users: [
    {
      _id: '1dam.usuario',
      userId: UUID('5530bc04-2427-4171-8cad-cb1570a0c459'),
      user: 'usuario',
      db: '1dam',
      roles: [ { role: 'dbOwner', db: '1dam' } ],
      mechanisms: [ 'SCRAM-SHA-1', 'SCRAM-SHA-256' ]
    }
  ],
  ok: 1
}
```

#### 4. Conectar con el nuevo usuario:

Si deseas probar la conexión utilizando el nuevo usuario, puedes salir de la sesión actual de mongosh y conectarte nuevamente con el usuario usuario de la siguiente manera:

```
mongosh -u usuario -p usuario
```

Este comando usará las credenciales del nuevo usuario para conectarse.