# Acceso a Datos

UT6 – BASES DE DATOS NOSQL

OPERACIONES BÁSICAS EN MONGODB

# 1. Operaciones básicas

Todos los comandos para operar con esta base de datos se escriben en minúscula, los más comunes son los siguientes:

- Listar las bases de datos: show databases

```
> show databases
< admin      41 kB
  config    111 kB
  local    81.9 kB
```

- • Mostrar la base de datos actual: db

```
> db
< local
```

# 1. Operaciones básicas

- Crear y abrir una base de datos: use nombrebasedatos. Una vez escrito el comando nos situaremos en la Base de datos, pero todavía no aparecerá hasta que no añadamos objetos JSON.

```
> use mibasededatos
< 'switched to db mibasededatos'
> show databases
< admin      41 kB
  config    111 kB
  local    81.9 kB
mibasededatos>
```

# 1. Operaciones básicas

- Crear una colección: *db.createCollection("nombre_coleccion").* Acepta las comillas dobles y las simples. Recuerda que las bases de datos en MongoDB constan de colecciones y dentro de las colecciones hay documentos. Por lo tanto, antes de crear documentos es necesario crear previamente la colección dentro de la base de datos deseada.

```
> db.createCollection('amigos')
< { ok: 1 }
mibasededatos>
```

# 1. Operaciones básicas

- Mostrar las colecciones de la base de datos actual: show collections

```
> show collections
< amigos
mibasededatos >
```

- Si queremos saber el número de documentos dentro de las colecciones, utilizaremos la función countDocuments, escribiremos: db.nombre_colección.countDocuments().

```
> db.amigos.countDocuments()
< 0
```

# 2. Crear documentos

Para añadir datos a la base de datos utilizaremos los comandos **`.insertOne`** e **`.insertMany()`** según este formato:

```
db.nombre_colección.insertOne(objeto)
db.nombre_colección.insertMany(array de objetos)
```

Donde **db** es una variable que hace referencia a la base de datos actual, la que estemos usando (la abriremos con **use**) y nombre_ colección es la colección donde se van a añadir los registros, si no existe se crea en ese momento.

# 2. Crear documentos

**Ejemplo:** me sitúo en la base de datos “mibasededatos”, y dentro de ella en la colección amigos inserto un documento con el comando **insertOne():**

```
> db.amigos.insertOne(
          {nombre:'Ana',
           telefono:927175645,
     curso:'1DAM',
     edad:20}
  );
< { acknowledged: true,
    insertedId: ObjectId("63138007073db6dbd583cd64") }
mibasededatos>
```

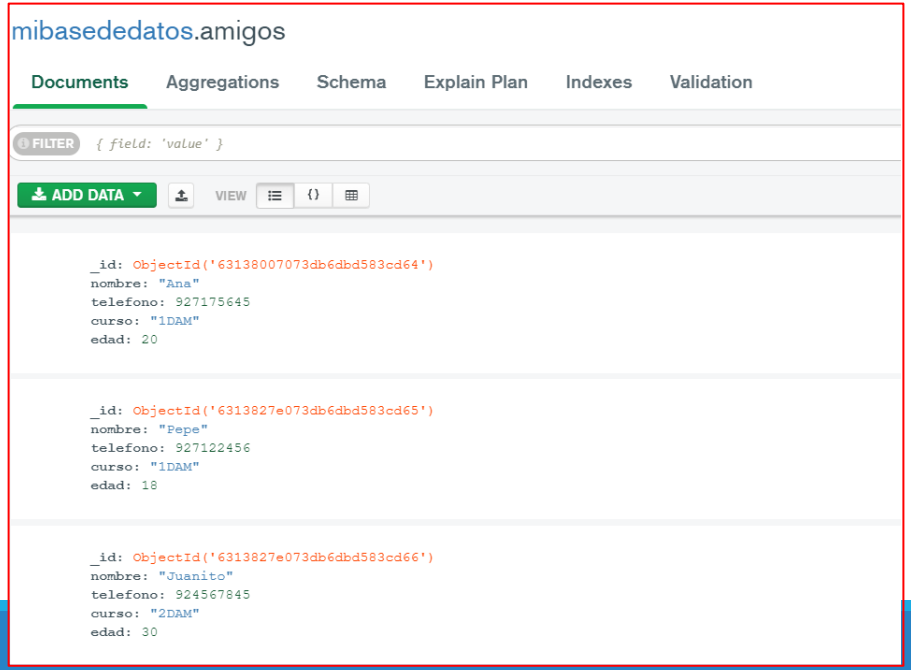
# 2. Crear documentos

Añado más amigos utilizando la orden **insertMany**():

```
> db.amigos.insertMany([
          {nombre:'Pepe',
           telefono:927122456,
           curso:'1DAM',
           edad:18},
          {nombre:'Juanito',
           telefono:924567845,
           curso:'2DAM',
           edad:30}
  ]
  );
< { acknowledged: true,
    insertedIds:
     { '0': ObjectId("6313827e073db6dbd583cd65"),
       '1': ObjectId("6313827e073db6dbd583cd66") } }
mibasededatos>
```

# 2. Crear documentos

Si nos vamos ahora al modo visual de **MongoDB Compass** podemos consultar los documentos insertados:



# 2. Crear documentos

Desde la consola para visualizar los documentos insertados utilizamos el comando **db.amigos.find()** sin pasarle parámetros al método:

```
> db.amigos.find()
< { _id: ObjectId("63138007073db6dbd583cd64"),
    nombre: 'Ana',
    telefono: 927175645,
    curso: '1DAM',
    edad: 20 }
  { _id: ObjectId("6313827e073db6dbd583cd65"),
    nombre: 'Pepe',
    telefono: 927122456,
    curso: '1DAM',
    edad: 18 }
  { _id: ObjectId("6313827e073db6dbd583cd66"),
    nombre: 'Juanito',
    telefono: 924567845,
    curso: '2DAM',
    edad: 30 }
mibasededatos >
```

# 2. Crear documentos

Si por el motivo que sea deseamos eliminar la base de datos deberemos abrirla con **use** y después usar el comando `db.dropDatabase()`

```
> use mibasededatos
< 'switched to db mibasededatos'
> db.dropDatabase()
< { ok: 1, dropped: 'mibasededatos' }
```

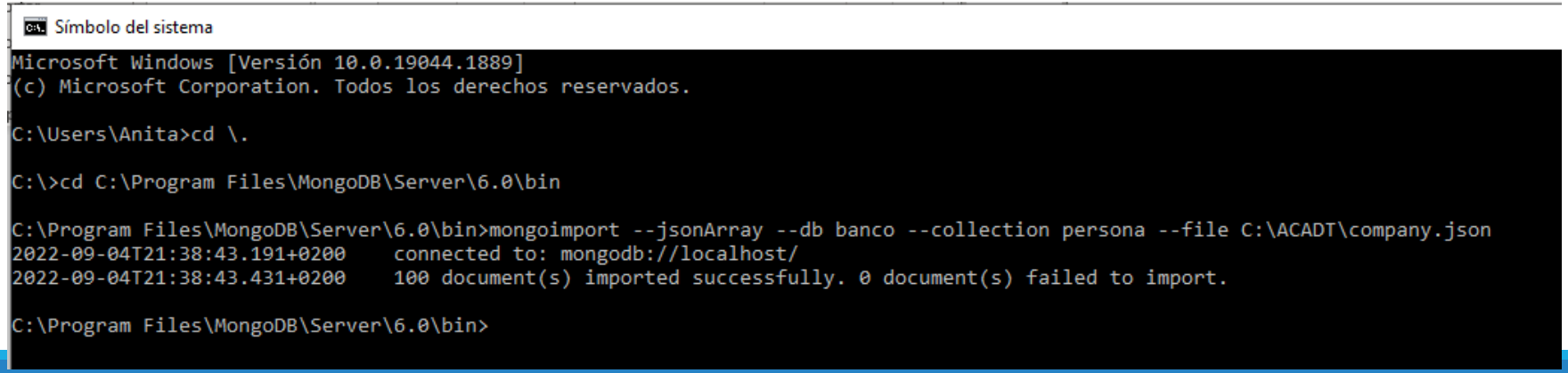
# 2. Crear documentos

Vamos a cargar los datos de una financiera (*company.json*), sería un JSON con 100 registros o documentos. El documento lo hemos obtenido a través de la página https://json-generator.com y os lo podéis descargar del curso de Moodle. Se puede hacer de 2 formas:

# 2. Crear documentos

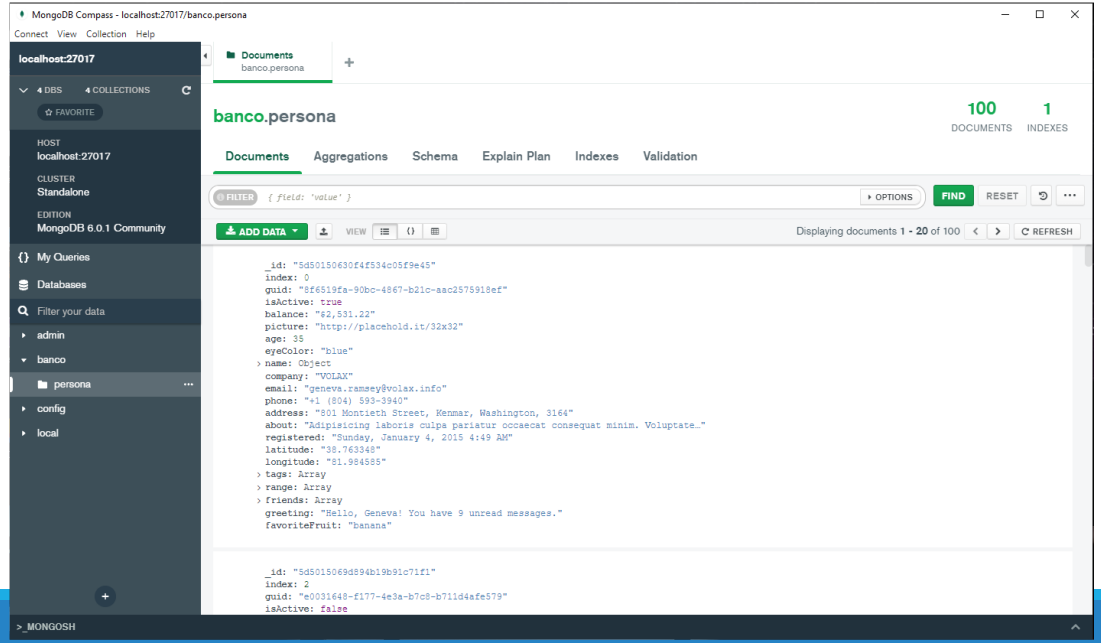
a) La forma más rápida es utilizando el comando **mongoimport** desde la línea de comandos

```
mongoimport --jsonArray --db banco --collection persona --file C:\ACADT\company.json
```



# 2. Crear documentos

Nos vamos a MongoDB Compass y comprobamos que se han cargado los 100 documentos:



# 2. Crear documentos

b) Utilizando el operador insertMany

```
use banco
db.createCollection('persona')
```

Una vez creada la colección utilizamos el comando insertMany y copiamos y pegamos el contenido del archivo *company.json*

```
db.persona.insertMany(todo el contenido del archivo company.json)
```

# 2. Crear documentos

El resultado es el mismo pero es un proceso más lento y quizás más engorroso.

```
>_MONGOSH
> use banco
< 'switched to db banco'
> db.createCollection('persona')
< { ok: 1 }
banco> db.persona.insertMany([
        {
          "_id": "5d50150630f4f534c05f9e45",
          "index": 0,
          "guid": "8f6519fa-90bc-4867-b21c-aac2575918ef",
          "isActive": true,
          "balance": "$2,531.22",
          "picture": "http://placehold.it/32x32",
          "age": 35,
          "eyeColor": "blue",
          "name": {
            "first": "Geneva",
            "last": "Ramsey"
          },
          "company": "VOLAX",
          "email": "geneva.ramsey@volax.info",
```

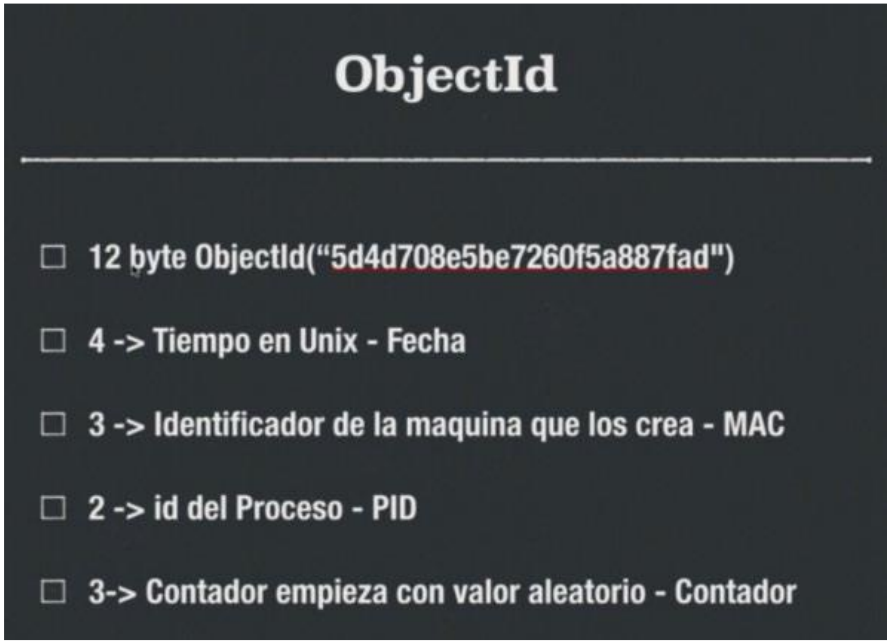
# 3. Identificador de objetos

Los identificadores de cada documento (registro) son únicos. Se asignan automáticamente al crear el documento, se generan de forma rápida y ordenada. También se pueden crear de forma manual.

Es un número hexadecimal que consta de **12 bytes**, los primeros 4 son una marca de tiempo, los tres siguientes la identificación de la máquina, 2 bytes de identificador de proceso y un contador de 3 bytes empezando en un número aleatorio.

El **ObjectId** o **_id**, es como si fuese la clave del documento, no se repetirá en una colección. Si un documento no tiene id MongoDB se lo asignará automáticamente, es lo que ocurre cuando insertamos y no indicamos el identificador.

# 3. Identificador de objetos



# 4. Consultar registros

Para consultar datos de una colección utilizaremos la orden **`find()`**, escribiremos:

```
db.nombre_colección.find()
```

En el ejemplo si queremos ver la colección amigos escribiremos: **`db.amigos.find()`**. Se muestran los identificativos (_id) de cada objeto JSON, únicos por colección, con el resto de campos. Tal y como vimos en el apartado anterior.

# 4. Consultar registros

Si se desea que la salida sea ascendente por uno de los campos, utilizamos el operador **`.sort`**, por ejemplo, para obtener los datos de la colección ordenados por nombre escribimos:

```
db.amigos.find().sort({nombre:-1});
```

El número que acompaña a la orden indica el tipo de ordenación, **1 ascendente y -1 descendente**.

```
> db.amigos.find().sort({nombre:-1})
< { _id: ObjectId("6313827e073db6dbd583cd65"),
    nombre: 'Pepe',
    telefono: 927122456,
    curso: '1DAM',
    edad: 18 }
  { _id: ObjectId("6313827e073db6dbd583cd66"),
    nombre: 'Juanito',
    telefono: 924567845,
    curso: '2DAM',
    edad: 30 }
  { _id: ObjectId("63138007073db6dbd583cd64"),
    nombre: 'Ana',
    telefono: 927175645,
    curso: '1DAM',
    edad: 20 }
```

# 4. Consultar registros

Si queremos limitar el número de resultados a mostrar podemos utilizar el método **limit (num),** que recibe como parámetro el número de elementos a mostrar

```
db.amigos.find().sort({nombre:-1}).limit(2)
```

Si se desean hacer búsquedas de documentos que cumplan una o varias condiciones, utilizamos el siguiente formato:

```
db.nombre_colección.find(filtro, campos)
```

# 4. Consultar registros

- En **filtro** indicamos la condición de búsqueda, podemos añadir los pares *nombre:valor* a buscar. Si omitimos este parámetro devuelve todos los documentos, o pasa un documento vacío ({ }).

- En **campos** se especifican los campos a devolver de los documentos que coinciden con el filtro de la consulta. Para devolver todos los campos de los documentos omitimos este parámetro.

- Si se desean devolver uno o más campos escribiremos {nombre_campo1: 1, nombre_campo2: 1, ...}. Si no se desean que se seleccionen los campos escribimos: {nombre campo1: 0, nombre_campo2: 0, ...} También podemos poner **true** o **false** en lugar de 1 o 0.

# 4. Consultar registros

Por ejemplo, para buscar el amigo con nombre Pepe lo escribimos así:

```
db.amigos.find({nombre : "Pepe"});
```

```
> db.amigos.find({nombre:'Pepe'})
< { _id: ObjectId("6313827e073db6dbd583cd65"),
    nombre: 'Pepe',
    telefono: 927122456,
    curso: '1DAM',
    edad: 18 }
```

# 4. Consultar registros

Si solo deseo mostrar su teléfono escribo:

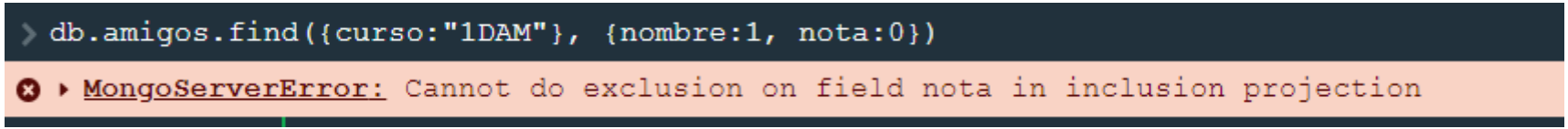
```
db.amigos.find({nombre : "Pepe"},{telefono:1});
```

```
> db.amigos.find({nombre:'Pepe'},{telefono:1})
< { _id: ObjectId("6313827e073db6dbd583cd65"),
    telefono: 927122456 }
```

# 4. Consultar registros

**Importante:** no podemos incluir a la vez los campos a mostrar y los campos a excluir, excepto el campo _id:

```
db.amigos.find({curso:"1DAM"}, {nombre:1, nota:0})
```



# 4. Consultar registros

Si queremos saber el número de registros que devuelve una consulta pondremos **`db.nombre_coleccion.find({filtros}).size()`**, por ejemplo, para saber cuántos son del curso 1DAM escribiremos:

```
> db.amigos.find({curso:"1DAM"}, {nota:0}).size()
< 2
```

# 4. Consultar registros

Igual que en SQL, a partir de una colección, si queremos obtener todos los diferentes valores que existen en un campo, utilizaremos el método **`distinct:`** `db.collection.distinct(field, query, options)`

```
> db.amigos.distinct('curso',{edad:{$gt:18}})
< [ '1DAM', '2DAM' ]
```

Y para saber cuántos documentos distintos hay añadiremos la propiedad **length**:

```
db.amigos.distinct('curso',{edad:{$gt:18}}).length
```

# 4.1. Consultas operador LIKE

Cuando accedemos a los datos de los campos de un documento en MongoDB podemos encontrarnos la necesidad de realizar consultas MongoDB **like**. Es decir, realizar consultas por cadenas similares de texto. Por ejemplo, nombres que empiecen por una letra o letras, palabras que contengan una cierta cadena de caracteres, ...

Las consultas MongoDB like se resuelven mediante expresiones regulares. Lo que realizaremos mediante la siguiente sintaxis:

```
db.coleccion.find({campo:expresión_regular});
```

# 4.1. Consultas operador LIKE

Para los patrones de las expresiones regulares MongoDB utiliza *“Perl Compatible Regular Expressions” (PCRE)*. De esta forma tendremos las siguientes similitudes con los patrones LIKE.

**cadena%** **/^cadena/**

**%cadena%** **/cadena/**

**%cadena** **/cadena$/**

# 4.1. Consultas operador LIKE

De esta forma la sentencia para realizar consultas MongoDB like será la siguiente:

```
db.amigos.find({nombre:/^A/})
```

En este caso hemos realizado un filtro LIKE de amigos cuyo nombre empiecen por A.

A la hora de realizar búsquedas también podemos utilizar el operador **`$regex`** para especificar la expresión regular a utilizar. La sentencia anterior quedaría así:

```
db.amigos.find({"nombre":{$regex:/^A/}})
```

# 4.1. Consultas operador LIKE

Otro ejemplo podrían ser nombres de amigos que contengan una "a":

```
db.amigos.find({nombre:/a/})
```

Si queremos que sea insensible a mayúsculas y minúsculas, añadiremos a la expresión regular la “i”:

```
db.amigos.find({nombre:/a/i})
```

O amigos cuyo nombre acaben en e:

```
db.amigos.find({nombre:/e$/})
```

# 4.2. Selectores de búsqueda de comparación

- **$eq**, igual a un valor. Esta consulta obtiene los documentos con edad= 20:
- `db.amigos.find({edad:{$eq:20}})`
- **$gt**, mayor que y **$gte** mayor o igual que. Esta consulta obtiene los documentos con edad > 20:
- `db.amigos.find({edad:{$gt:20}})`

# 4.2. Selectores de búsqueda de comparación

- **$lt**, menor que y **$lte**, menor o igual que. El ejemplo muestra los amigos de 1DAM con edades entre 20 y 30 incluidas, preguntamos por un intervalo >=20 y <=30:
- `db.amigos.find({curso:"1DAM",edad:{$gte:20,$lte:30}})`
- **$ne**, distinto a un valor. El siguiente ejemplo obtiene los documentos con edad distinta de 30:
- `db.amigos.find({edad:{$ne:30}})`

# 4.2. Selectores de búsqueda de comparación

- **$in**, entre una lista de valores y **$nin**, no está entre la lista de valores. En el ejemplo se obtienen los documentos cuya edad sea uno de estos valores: 18 o25:
- `db.amigos.find({edad:{$in :[18,25]}})`
- Ahora especificamos los campos a devolver, en nuestro caso nombre y el curso:
- `db.amigos.find({ edad: { $in :[18, 25]}}, {nombre:1,curso:1})`

# 4.3. Selectores de búsqueda lógicos

- **$or**. La siguiente orden obtiene los documentos de los cursos 1DAM , o los que tienen una edad de 20:
- `db.amigos.find({ $or :[{edad:{$eq:20}},{curso : "1DAM"} ]})`
- Esta consulta obtiene los amigos con nombre Ana o Pepe:
- `db.amigos.find({$or:[{nombre:"Ana"},{nombre:"Pepe"}]})`

# 4.3. Selectores de búsqueda lógicos

- **$and.** Este operador se maneja implícitamente, no es necesario especificarlo. Las siguientes órdenes hacen lo mismo, obtienen los amigos del curso 1DAM y con edad mayor que 18:

- `db.amigos.find({$and: [{curso:"1DAM"},{edad : {$gt:18}} ]})`

- `db.amigos.find({curso:"1DAM",edad:{$gt:18}})`

# 4.3. Selectores de búsqueda lógicos

- **$not**. Representa la negación, en el ejemplo la consulta visualizará el nombre, el curso de los amigos que no contengan una “a” en su nombre:

- `db.amigos.find({ nombre:{$not:/a/}},{_id:0, nombre :1, curso:1})`

# 4.3. Selectores de búsqueda lógicos

- **$exists**, este operador booleano permite filtrar la búsqueda tomando en cuenta la existencia del campo de la expresión. Este ejemplo obtiene los registros que tengan nota.

- `db.amigos.find({curso:{$exists:true}})`

# Práctica


**Realizar el ejercicio 1 de la práctica 1 de MongoDB (UT6)**

# 5. Actualizar registros


Para actualizar datos utilizaremos los comandos **.updateOne** y **.updateMany**.

# 5.1. updateOne

La sintaxis del método updateOne es:

```
db.nombre_coleccion.updateOne(
    filtro_búsqueda,
    cambios_a_realizar,
    opciones)
```

Donde:

- En **`filtro_búsqueda`**, se indica la condición para localizar los registros o documentos a modificar.
- En **`cambios_a_realizar`**, se especifican los cambios que se desean hacer. Los operadores de modificación a utilizar los veremos en el siguiente apartado.

# 5.1. updateOne


- Y en **`opciones`** tenemos las siguientes posibilidades:
  - **`upsert<boolean>`** si asignamos **true** a este parámetro, se indica que si el filtro de búsqueda no encuentra ningún resultado, entonces, el cambio debe ser insertado como un nuevo registro.
  - **`writeConcern: <document>`** mecanismo para comprobar que la operación de escritura se haya propagado al número de nodos especificado en el *write concern*. El valor por defecto es 1 (Se espera únicamente la confirmación del nodo primario)
  - **`collation: <document>`** Permite al usuario utilizar reglas específicas del idioma para el documento

# 5.1. updateOne

- **`arrayFilters: [ <filterdocument1>, ... ]:`** permite especificar un filtro para indicar que elementos de una matriz son actualizados
- **`hint: <document|string>:`** especificar un índice para realizar la actualización

En caso de que el filtro de búsqueda devuelva más de un resultado, el cambio se realizará en el primero que encuentre, es decir, el que tenga menor identificativo de objeto "_id".

Si tenemos varios registros y queremos actualizar uno en concreto, lo lógico sería pasarle en el como filtro el _id del objeto que es único.

# 5.2. Operadores de modificación

**`$set`**, permite actualizar con nuevas propiedades a un documento. Por ejemplo, actualizo el nombre del amigo Pepe a Pepa:

```
db.amigos.updateOne({nombre:"Pepe"},{ $set: {nombre:
"Pepa"}})
```

```
> db.amigos.updateOne({nombre:'Pepe'},{$set:{ nombre:'Pepa'}});
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
```

# 5.2. Operadores de modificación

Si no existiese coincidencia no se modificaría ningún registro y tampoco se insertaría ninguno nuevo. Si quisíeramos que se insertara un nuevo registro cuando no haya ninguna coincidencia, tendríamos que poner la opción **`upsert`** a **true**.

```
db.amigos.updateOne({nombre:"Juan"},{ $set: {nombre: "Juana"}},{upsert:true} )
```

```
> db.amigos.updateOne({nombre:"Juan"},{ $set: {nombre: "Juana"}},{upsert:true} )
< { acknowledged: true,
    insertedId: ObjectId("6315e11c5a2b91c36d27dad1"),
    matchedCount: 0,
    modifiedCount: 0,
    upsertedCount: 1 }
```

# 5.2. Operadores de modificación

Vemos que se ha insertado un nuevo amigo con nombre Juana, y sin el resto de campos:

```
_id: ObjectId('6315e11c5a2b91c36d27dad1')
nombre: "Juana"
```

# 5.2. Operadores de modificación

- **$unset**, permite eliminar propiedades de un documento. Por ejemplo, borro nombre del registro que tiene ese _id

- `db.amigos.updateOne({_id:ObjectId('6315e11c5a2b91c36d27dad1')},{ $unset:{nombre:""}})`

- **$inc**, incrementa en una cantidad numérica especificada en el valor del campo a incrementar. Por ejemplo sumo 1 a la edad de Ana:

- `db.amigos.updateOne({nombre:"Ana"},{$inc:{edad:1}})`

# 5.2. Operadores de modificación


- **`$rename`,** renombra campos del documento. Por ejemplo, cambiamos los campos nombre y edad de Juana y los ponemos en inglés, si hay algún campo que no exista no lo crea.

- ```
  db.amigos.updateOne({nombre:'Juana'},{
  $rename:{edad:'age', nombre:'name'}})
  ```

**Más info:**
https://www.mongodb.com/docs/manual/reference/method/db.collection.updateOne
  ```

# 5.3. updateMany

La sintaxis del método updateMany es la misma que la de updateOne:

```
db.nombre_coleccion.updateMany(
    filtro_búsqueda,
    cambios_a_realizar,
    opciones)
```

# 5.3. updateMany

La única diferencia es que en caso de que el filtro de búsqueda devuelva más de un resultado, el cambio se realizará en todos los registros, no solamente en el primero.

Por ejemplo, vamos a disminuir en 1 año la edad de todos los alumnos de 1DAM:

```
db.amigos.updateMany({curso:'2DAM'},{$inc:{edad:-1}})
```

```
> db.amigos.updateMany({curso:'1DAM'},{$inc:{edad:-1}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 2,
    modifiedCount: 2,
    upsertedCount: 0 }
```

# 5.3. updateMany

La única diferencia es que en caso de que el filtro de búsqueda devuelva más de un resultado, el cambio se realizará en todos los registros, no solamente en el primero.

Por ejemplo, vamos a disminuir en 1 año la edad de todos los alumnos de 1DAM:

```
db.amigos.updateMany({curso:'2DAM'},{$inc:{edad:-1}})
```

```
> db.amigos.updateMany({curso:'1DAM'},{$inc:{edad:-1}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 2,
    modifiedCount: 2,
    upsertedCount: 0 }
```

# 5.3. updateMany

Si alguno de los registros no tiene el campo edad, lo crearía, y pondría como valor -1.

```
_id: ObjectId('63b49431164a05416f4a92f4')
curso: "2DAM"
telefono: 927453456
edad: -1
```

# Práctica


**Realizar el ejercicio 2 de la práctica 1 de MongoDB (UT6)**

# 6. Operaciones con Arrays


En este apartado vamos a ver cómo realizar consultas en documentos que contienen arrays. Creamos la colección **libros** con tres libros que contienen un array con temas del libro:

```
db.libros.insertMany([{codigo:1,nombre:"Acceso a datos",
pvp: 35, editorial:"Garceta", temas: ["Base de datos",
"Hibernate","Neodatis"]},
{codigo:2,nombre:"Entornos de desarrollo", pvp: 27,
editorial:"Garceta", temas:["UML", "Subversión",
"ERMaster"]},
{codigo:3,nombre:"Programación de Servicios", pvp: 25,
editorial:"Garceta", temas:["SOCKET", "Multihilo"]}])
```

# 6. Operaciones con Arrays

Para consultar los elementos del array escribimos el array y el elemento a consultar.

Ejemplos:

- Libros que tengan el tema UML:
- `db.libros.find({temas:"UML"})`

```
> db.libros.find({temas:'UML'})
< { _id: ObjectId("63172227be9e0b41d402a00e"),
    codigo: 2,
    nombre: 'Entornos de desarrollo',
    pvp: 27,
    editorial: 'Garceta',
    temas: [ 'UML', 'Subversión', 'ERMaster' ] }
```

# 6. Operaciones con Arrays

- Libros que tengan el tema UML o Neodatis:
- Con el operador $or:
- `db.libros.find({$or:[ {temas:"UML"},{temas: "Neodatis"}] } )`

```
> db.libros.find( { $or:[ {temas:"UML"},{temas: "Neodatis"}] } )
< { _id: ObjectId("63172227be9e0b41d402a00d"),
    codigo: 1,
    nombre: 'Acceso a datos',
    pvp: 35,
    editorial: 'Garceta',
    temas: [ 'Base de datos', 'Hibernate', 'Neodatis' ] }
  { _id: ObjectId("63172227be9e0b41d402a00e"),
    codigo: 2,
    nombre: 'Entornos de desarrollo',
    pvp: 27,
    editorial: 'Garceta',
    temas: [ 'UML', 'Subversión', 'ERMaster' ] }
```

# 6. Operaciones con Arrays

- O con el operador $in:
- `db.libros.find( {temas:{$in:[ "UML","Neodatis"] }} )`

# 6. Operaciones con Arrays


- Libros que tengan el tema XML y Hibernate:
- Con el operador **$and**:
- `db.libros.find({$and:[ {temas:"XML"},{temas: "Hibernate"}] } )`
- con el operador **$all**:
- `db.libros.find( {temas:{$all:[ "XML","Hibernate"] }} )`

# 6. Operaciones con Arrays

```
{ _id: ObjectId("63a9b49c8da0e5fecd0428fd"),
  codigo: 1,
  nombre: 'Acceso a datos',
  pvp: 30,
  editorial: 'Garceta',
  temas:
   [ 'Hibernate',
     'Neodatis',
     'MongoDB',
     'Acceso a Datos',
     'XML',
     'SOCKET' ] }
```

# 6. Operaciones con Arrays

- Libros de la editorial Garceta, con pvp > 25 y que tengan el tema UML o Neodatis:

- `db.libros.find({editorial:'Garceta',pvp:{$gt:25},$or:[{temas:'UML'},{temas:'Neodatis'}]})`

```
> db.libros.find({editorial:'Garceta',pvp:{$gt:25},$or:[{temas:'UML'},{temas:'Neodatis'}]})
< { _id: ObjectId("63172227be9e0b41d402a00d"),
    codigo: 1,
    nombre: 'Acceso a datos',
    pvp: 35,
    editorial: 'Garceta',
    temas: [ 'Base de datos', 'Hibernate', 'Neodatis' ] }
  { _id: ObjectId("63172227be9e0b41d402a00e"),
    codigo: 2,
    nombre: 'Entornos de desarrollo',
    pvp: 27,
    editorial: 'Garceta',
    temas: [ 'UML', 'Subversión', 'ERMaster' ] }
```

# 6.1. Operaciones de modificación para Arrays

- **`$push`**: añade un elemento a un array. **`$push`** no tiene en cuenta lo que contiene el array, por tanto, si un elemento ya existe, se repetirá y tendremos duplicados.
- Este ejemplo añade el tema MongoDB al libro con código 1:
- `db.libros.updateOne({codigo:1},{$push:{temas:"MongoDB"}})`

# 6.1. Operaciones de modificación para Arrays

Si quisiéramos añadir un objeto al array sería:

```
db.libros.updateOne(
{codigo:1},{$push:{temas:{_id:1,
                          tema:"Angular",
                          version:12,
                          fecha_desarrollo:new Date("2009-10-
                                                    05")}}})
```

# 6.1. Operaciones de modificación para Arrays

- **`$addToSet`**, agrega elementos a un array solo sí estos no existen. En el ejemplo se añade el tema “Base de datos” a todos los libros de la colección. Como se ve en la captura de pantalla, hay dos coincidencias (matchedCount), pero solamente hay una modificación (modifiesCount)

- `db.libros.updateMany({},{ $addToSet: {temas:'Acceso a datos' }});`

```
> db.libros.updateMany( {},{ $addToSet: {temas:'Acceso a datos' }});
< { acknowledged: true,
    insertedId: null,
    matchedCount: 2,
    modifiedCount: 1,
    upsertedCount: 0 }
```

# 6.1. Operaciones de modificación para Arrays

- **`$each`**, se usa en conjunto con **`$addToSet`** o **`$push`** para indicar que se añaden varios elementos al array.

- `db.libros.updateOne({codigo:1},{$push:{temas:{$each:["JSON","XML"]} } })`

```
> db.libros.updateOne({codigo:1},{$push:{temas:{$each:["JSON","XML"]} } })
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
```

# 6.1. Operaciones de modificación para Arrays

- `db.libros.updateOne({codigo:2},{$addToSet:{temas:{$each: ["Eclipse","Developper" ]}}})`

```
> db.libros.updateOne({codigo:2},{$addToSet:{temas:{$each: ["Eclipse","Developper" ]}}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
```

# 6.1. Operaciones de modificación para Arrays

- **$pop**, elimina el primer o último valor de un array. Con valor -1 borra el primero, con otro valor el último. En el ejemplo se borra el primer tema del libro con código 3:

- `db.libros.updateOne({codigo:3},{$pop:{ temas:-1 } })`

# 6.1. Operaciones de modificación para Arrays

- **`$pull`**, elimina los valores de un array que cumplan con el filtro indicado. En el ejemplo se borran de todos los libros los elementos "Base de datos " y "JSON", si los tienen:

- `db.libros.updateMany({},{$pull:{ temas:{ $in:["Base de datos","JSON" ]}}} )`

```
> db.libros.updateMany({},{$pull:{ temas: { $in:["Base de datos","JSON" ]}}} )
< { acknowledged: true,
    insertedId: null,
    matchedCount: 3,
    modifiedCount: 1,
    upsertedCount: 0 }
```

# 6.1. Operaciones de modificación para Arrays

- **$(update),** El operador posicional $ identifica un elemento en un array para actualizar sin especificar explícitamente la posición del elemento en el array. El formato es:
- `{"<array>.$":value}`

- En el siguiente ejemplo reemplazamos el tema Socket por el tema Routing:

- `db.libros.updateMany({temas:"SOCKET"},{$set:{"temas.$":"Routing"}})`

Para más información de los operadores de actualización:
https://www.mongodb.com/docs/manual/reference/operator/update/#std-label-update-operators

# 7. Borrar registros en MongoDB

Para borrar datos JSON podemos utilizar las ordenes **`.deleteOne`**, **`.deleteMany`** y **`. drop`**. Se puede eliminar los documentos que cumplan una condición, o todos los documentos de la colección o la colección completa.

- Para borrar un documento que cumpla una condición utilizaremos la orden `deleteOne({nombre:valor})`. Por ejemplo, se borra de la colección al amigo Juanito:

- `db.amigos.deleteOne({nombre:'Juanito'}`

```
> db.amigos.deleteOne({nombre:'Juanito'})
< { acknowledged: true, deletedCount: 1 }
```

# 7. Borrar registros en MongoDB

- Para borrar todos los elementos de la colección ponemos:
- `db.amigos.deleteMany({ }) ;`

```
> db.amigos.deleteMany({})
< { acknowledged: true, deletedCount: 2 }
```

- Para borrar la colección escribiremos:
- `db.amigos.drop()`

```
> db.amigos.drop()
< true
```

# Dudas y preguntas

# Práctica


**Realizar el ejercicio 3 de la práctica 1 de MongoDB (UT6)**