

Índice

Comando básicos de MONGO	2
Creación y eliminación de BDs	3
Creación y eliminación de Colecciones	4
Tipos de datos	5
OPERACIONES CON DATOS (CRUD)	9
Insertar	9
Eliminar	10
Actualizar	11
Reemplazar	12
EJERCICIOS	14
Consultar	15
Operaciones de comparación	16
Comparaciones con strings	18
Operaciones según la existencia o el tipo de los elementos	18
Operaciones lógicas	19
EJERCICIOS	27

Referencias:

Curso básico <https://openwebinars.net/accounts/login/?next=/academia/portada/mongodb/>
Tutorial <https://www.tutorialspoint.com/mongodb/index.htm>
Manual <https://docs.mongodb.com/manual/>

Comando básicos de MONGO

- Limpiar consola (Ctrl + L / cls)
- Listar todas las bases de datos (show dbs)
- Cambiarse de base de datos (use <dbname>). **Si no existe la crea.** Aunque una base de datos no se crea realmente hasta que obtiene contenido.
- Mostrar el nombre de la base de datos (db.getName () o db)

```
test> show dbs
admin      41 kB
config    73.7 kB
local      41 kB
sample    55.2 MB
test> use sample
switched to db sample
sample> show collections
grades
inspections
tweets
zips
sample> db.getName()
sample
sample> db
sample
```

- Listar los metadatos sobre una base de datos (db.stats())

```
sample> db.stats()
{
  db: 'sample',
  collections: 4,
  views: 0,
  objects: 261945,
  avgObjSize: 513.7219798049209,
  dataSize: 134566904,
  storageSize: 52400128,
  freeStorageSize: 1224704,
  indexes: 4,
  indexSize: 2789376,
  indexFreeStorageSize: 114688,
  totalSize: 55189504,
  totalFreeStorageSize: 1339392,
  scaleFactor: 1,
  fsUsedSize: 183785254912,
  fsTotalSize: 479458226176,
  ok: 1
}
```

- Solicitar ayuda sobre comandos (db.help())

Creación y eliminación de BDs

- Creación / usar

- Eliminación

```
bash
> show dbs
admin    0.000GB
colegio  0.000GB
config   0.000GB
local    0.000GB
> use universidad
switched to db universidad
> show dbs
admin    0.000GB
colegio  0.000GB
config   0.000GB
local    0.000GB
>
```

Dónde está la base de datos **universidad** que acabo de crear??

Hasta que no insertes al menos un documento en una de sus colecciones, no estará disponible

- Creación / usar

- Eliminación

```
bash
> db.asignaturas.insertOne({'id': '881716', 'nombre': 'Matemáticas avanzadas'})
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e498d8e4b839543f78c14a4")
}
> show dbs
admin    0.000GB
colegio  0.000GB
config   0.000GB
local    0.000GB
universidad 0.000GB
>
```

Una vez que se hace al menos una inserción (y se crea la colección **asignaturas**), la nueva base de datos **universidad** aparece en el listado

Eliminar

- Creación / usar

- Eliminación

```
bash
> show dbs
admin    0.000GB
colegio  0.000GB
config   0.000GB
local    0.000GB
universidad 0.000GB
> db
universidad
> db.dropDatabase()
{ "dropped" : "universidad", "ok" : 1 }
> show dbs
admin    0.000GB
colegio  0.000GB
config   0.000GB
local    0.000GB
>
```

Usar el método `dropDatabase()` supone eliminar datos, índices y metadatos.

Creación y eliminación de Colecciones

- Creación

Hay 2 formas de crear una colección.

Método 1

Puede crear una colección utilizando el método de base de datos ***createCollection()***.

```
db.createCollection(name, options)
```

En el comando, el nombre (es de tipo string) es el nombre de la colección que se creará. Options es un documento y se utiliza para especificar la configuración de la colección.

```
db.createCollection("mycol") ó  
db.createCollection("mycol", { capped : true, autoIndexID : true,  
size : 6142800, max : 10000 } )
```

Método 2

También puede crear una colección durante el proceso ***insertOne()*** o ***insertMany()***.

Ejemplo

Estamos asumiendo aquí object que es un objeto JavaScript válido que contiene datos:

```
db.mycol.insertOne(object)
```

En MongoDB, no es necesario crear una colección. MongoDB crea una colección automáticamente, cuando inserta algún documento.

- Eliminación

```
db.COLLECTION_NAME.drop()
```

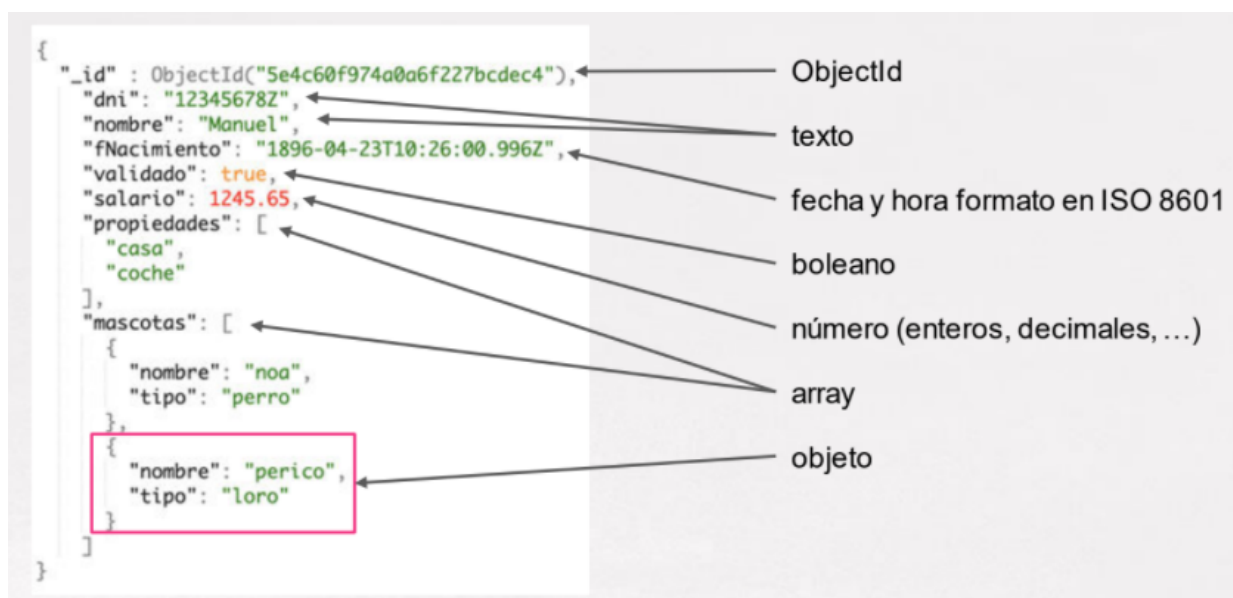
En el comando, el nombre (es de tipo string) es el nombre de la colección que se creará. Options es un documento y se utiliza para especificar la configuración de la colección.

```
db.mycol.drop()
```

Tipos de datos

MongoDB admite muchos tipos de datos. Algunos de ellos son ...

- Cadena : este es el tipo de datos más utilizado para almacenar los datos. La cadena en MongoDB debe ser válida para UTF-8.
- Entero : este tipo se utiliza para almacenar un valor numérico. El número entero puede ser de 32 bits o 64 bits dependiendo de su servidor.
- Booleano : este tipo se utiliza para almacenar un valor booleano (verdadero / falso).
- Doble : este tipo se utiliza para almacenar valores de punto flotante.
- Teclas Min / Max : este tipo se utiliza para comparar un valor con los elementos BSON más bajos y más altos.
- Matrices : este tipo se utiliza para almacenar matrices o listas o valores múltiples en una clave.
- Marca de tiempo : timestamp. Esto puede ser útil para grabar cuando se ha modificado o agregado un documento.
- Objeto : este tipo de datos se utiliza para documentos incrustados.
- Nulo : este tipo se utiliza para almacenar un valor nulo.
- Símbolo : este tipo de datos se utiliza de forma idéntica a una cadena; sin embargo, generalmente está reservado para idiomas que usan un tipo de símbolo específico.
- Fecha : este tipo de datos se utiliza para almacenar la fecha u hora actual en formato de hora UNIX. Puede especificar su propia fecha y hora creando un objeto de Fecha y pasando día, mes, año en él.
- ID de objeto : este tipo de datos se utiliza para almacenar la ID del documento.
- Datos binarios : este tipo de datos se utiliza para almacenar datos binarios.
- Código : este tipo de datos se utiliza para almacenar código JavaScript en el documento.
- Expresión regular : este tipo de datos se utiliza para almacenar expresiones regulares.



Tipo de dato Date en MongoDB

Veremos ahora en detalle cómo almacenar datos de tipo Date.

La fecha es un entero de 64 bits que representa el número de milisegundos desde la época de Unix (1 de enero de 1970). Esto da como resultado un intervalo de fechas representables de aproximadamente 290 millones de años en el pasado y el futuro. El tipo Date almacena el signo, un valor negativo representa una fecha anterior a 1970.

Veamos un ejemplo de cómo podemos almacenar un tipo de dato Date tomando la fecha y hora actual del servidor.

Ejemplo

Una playa de estacionamiento cada vez que ingresa un vehículo crea un documento donde almacena la patente y la fecha y hora de ingreso .

use ejdate1

```
db.autos.insertOne(
  {
    patente : 'aaa111',
    fechahora : new Date()
  }
)
db.autos.insertOne(
  {
    patente : 'bbb222',
    fechahora : new Date()
  }
)
db.autos.insertOne(
  {
    patente : 'ccc333',
    fechahora : new Date()
  }
)
```

```
db.autos.find().pretty()
```

Para almacenar la fecha se utiliza el estándar ISO 8601 que tiene un formato: YYYY-MM-DDTHH:MM:SS

La representación de los datos de tipo Date cuando llamamos al método find es:

```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
>
> db.autos.find().pretty()
{
  "_id" : ObjectId("5c3a60d52c419f7680420488"),
  "patente" : "aaa111",
  "fechahora" : ISODate("2019-01-12T21:49:09.852Z")
}
{
  "_id" : ObjectId("5c3a60d52c419f7680420489"),
  "patente" : "hhh222",
  "fechahora" : ISODate("2019-01-12T21:49:09.873Z")
}
{
  "_id" : ObjectId("5c3a60d52c419f768042048a"),
  "patente" : "ccc333",
  "fechahora" : ISODate("2019-01-12T21:49:09.883Z")
}
>
```

Podemos almacenar una fecha particular cuando creamos el objeto de la clase Date con "new Date(YYYY,M,D)".

Ejemplo

Crear una colección empleados con los datos de ingreso de los empleados: id,nombre y fechaingreso.

use ejdate2

```
db.empleados.insertOne( {
  _id : 20456234,
  nombre : 'Rodriguez Pablo',
  fechaingreso : new Date(2010,0,31)
})
```

```
db.empleados.insertOne( {
  _id : 17488834,
  nombre : 'Gomez Ana',
  fechaingreso : new Date(2001,11,1)
})
```

```
db.empleados.insertOne( {
  _id : 23463564,
  nombre : 'Juarez Carla',
  fechaingreso : new Date(2005,3,14)
})
```

Como el shell de MongoDB está implementado en JavaScript debemos indicar al crear un objeto de la clase Date para el mes es un valor comprendido entre 0 y 11.

Podemos ver las fechas almacenadas en el campo 'fechaingreso', y que la parte de la hora está en cero por no pasarlas cuando creamos el objeto de la clase Date:



```
C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe
>
> db.empleados.find().pretty()
{
  "_id" : 20456234,
  "nombre" : "Rodriguez Pablo",
  "fechaingreso" : ISODate("2010-01-31T03:00:00Z")
}
{
  "_id" : 17488834,
  "nombre" : "Gomez Ana",
  "fechaingreso" : ISODate("2001-12-01T03:00:00Z")
}
{
  "_id" : 23463564,
  "nombre" : "Juarez Carla",
  "fechaingreso" : ISODate("2005-04-14T03:00:00Z")
}
>
```


OPERACIONES CON DATOS (CRUD)

Son las diferentes operaciones que se pueden hacer con los datos.

CRUD son las siglas, en inglés, de las operaciones que permiten crear (insertar), leer (consultar), actualizar y eliminar.

Insertar

Un solo documento:

`db.collectionName.insertOne(<json>);`

`db.collectionName.insertOne(<json>);`

```
{
  "referencia": "C0001",
  "nombre": "Estadística",
  "duracion": 6,
  "activo": true,
  "fInicio": "2020-01-21T09:00:00.000Z"
}
```

```
> show dbs
admin      0.000GB
config     0.000GB
local      0.000GB
universidad 0.000GB
> use universidad
switched to db unviuersidad
> show collections
> db.cursos.insertOne({
...   "referencia": "C0001",
...   "nombre": "Estadística",
...   "duracion": 6,
...   "activo": true,
...   "fInicio": "2020-01-21T09:00:00.000Z"
... });
{
  "acknowledged" : true,
  "insertedId" : ObjectId("5e4e946177a92631239bd288")
}
```

Varios documentos:

`db.collectionName.insertMany(<json>);`

`db.collectionName.insertMany(<json>);`

```
[
  {
    "referencia": "C0002",
    "nombre": "Pintura",
    "duracion": 24,
    "activo": false,
    "fInicio": "2020-02-18T16:30:00.000Z"
  },
  {
    "referencia": "C0003",
    "nombre": "Física",
    "duracion": 46,
    "activo": true,
    "fInicio": "2019-12-13T10:30:00.000Z"
  }
]
```

```
> use universidad
switched to db universidad
> db.cursos.insertMany([
... {
...   "referencia": "C0002",
...   "nombre": "Pintura",
...   "duracion": 24,
...   "activo": false,
...   "fInicio": "2020-02-18T16:30:00.000Z"
... },
... {
...   "referencia": "C0003",
...   "nombre": "Física",
...   "duracion": 46,
...   "activo": true,
...   "fInicio": "2019-12-13T10:30:00.000Z"
... }
... ]);
{
  "acknowledged" : true,
  "insertedIds" : [
    ObjectId("5e4e98be1cd78466ff804f1f"),
    ObjectId("5e4e98be1cd78466ff804f20")
  ]
}
```

Eliminar

Un solo documento:

db.collectionName.deleteOne(<json>);

CRUD - Eliminar- Ejemplos

db.collectionName.deleteOne(<json>);

- donde <json> es un documento con condiciones para eliminar un documento concreto.
- Si más de un documento cumple con la condición, se eliminará el primero de ellos.

```
db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4ea153cd78466ff804f24"),
  "referencia" : "C0001",
  "nombre" : "Estadística",
  "duracion" : 6,
  "activo" : true,
  "finicio" : "2020-01-21T09:00:00.000Z"
}
{
  "_id" : ObjectId("5e4ea153cd78466ff804f25"),
  "referencia" : "C0002",
  "nombre" : "Pintura",
  "duracion" : 24,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
}
{
  "_id" : ObjectId("5e4ea153cd78466ff804f26"),
  "referencia" : "C0003",
  "nombre" : "Física",
  "duracion" : 46,
  "activo" : true,
  "finicio" : "2019-12-13T10:30:00.000Z"
}
}

db.cursos.deleteOne({activo: true});
{
  "acknowledged" : true, "deletedCount" : 1 }
}
db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4ea153cd78466ff804f25"),
  "referencia" : "C0002",
  "nombre" : "Pintura",
  "duracion" : 24,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
}
{
  "_id" : ObjectId("5e4ea153cd78466ff804f26"),
  "referencia" : "C0003",
  "nombre" : "Física",
  "duracion" : 46,
  "activo" : true,
  "finicio" : "2019-12-13T10:30:00.000Z"
}
}
```

Varios documentos:

db.collectionName.deleteMany(<json>);

CRUD - Eliminar- Ejemplos

db.collectionName.deleteMany(<json>);

- donde <json> es un documento con condiciones para eliminar

```
db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4ea153cd78466ff804f24"),
  "referencia" : "C0001",
  "nombre" : "Estadística",
  "duracion" : 6,
  "activo" : true,
  "finicio" : "2020-01-21T09:00:00.000Z"
}
{
  "_id" : ObjectId("5e4ea153cd78466ff804f25"),
  "referencia" : "C0002",
  "nombre" : "Pintura",
  "duracion" : 24,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
}
{
  "_id" : ObjectId("5e4ea153cd78466ff804f26"),
  "referencia" : "C0003",
  "nombre" : "Física",
  "duracion" : 46,
  "activo" : true,
  "finicio" : "2019-12-13T10:30:00.000Z"
}
}

db.cursos.deleteMany({activo: true});
{
  "acknowledged" : true, "deletedCount" : 1 }
}
db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4ea153cd78466ff804f25"),
  "referencia" : "C0002",
  "nombre" : "Pintura",
  "duracion" : 24,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
}
{
  "_id" : ObjectId("5e4ea153cd78466ff804f26"),
  "referencia" : "C0003",
  "nombre" : "Física",
  "duracion" : 46,
  "activo" : true,
  "finicio" : "2019-12-13T10:30:00.000Z"
}
}
```

Actualizar

`db.collection.updateOne(<filter>, <update>)`

`db.collection.updateMany(<filter>, <update>)`

<filter> json con la condición o condiciones por las que se va a filtrar

<update> json con los campos y valores con los que se quiere actualizar (\$set: {key:value})

Un solo documento:

db.collection.updateOne(<filter>, <update>)

The diagram illustrates the `updateOne` operation. On the left, a terminal window shows the initial state of the `db.cursos` collection with three documents. The middle part shows the command `db.cursos.updateOne` with a filter `{referencia: 'C0002'}` and an update `{ $set: { nombre: 'Física elemental', duracion: 16 } }`. Arrows point from the labels 'Filtro' and 'Nuevos valores' to the corresponding parts of the command. On the right, a terminal window shows the state after the update, where the document with `referencia: 'C0002'` has been modified.

```
db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4eac231cd78466ff804f2c"),
  "referencia" : "C0001",
  "nombre" : "Estadística",
  "duracion" : 6,
  "activo" : true,
  "finicio" : "2020-01-21T09:00:00.000Z"
},
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2d"),
  "referencia" : "C0002",
  "nombre" : "Pintura",
  "duracion" : 24,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
},
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2e"),
  "referencia" : "C0003",
  "nombre" : "Física",
  "duracion" : 46,
  "activo" : true,
  "finicio" : "2019-12-13T10:30:00.000Z"
}

db.cursos.updateOne(
  { referencia: "C0002" },
  { $set: {
    nombre: "Física elemental",
    duracion: 16
  } }
);

db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4eac231cd78466ff804f2c"),
  "referencia" : "C0001",
  "nombre" : "Estadística",
  "duracion" : 6,
  "activo" : true,
  "finicio" : "2020-01-21T09:00:00.000Z"
},
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2d"),
  "referencia" : "C0002",
  "nombre" : "Física elemental",
  "duracion" : 16,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
},
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2e"),
  "referencia" : "C0003",
  "nombre" : "Física",
  "duracion" : 46,
  "activo" : true,
  "finicio" : "2019-12-13T10:30:00.000Z"
}
```

Varios documentos:

db.collection.updateMany(<filter>, <update>)

The diagram illustrates the `updateMany` operation. On the left, a terminal window shows the initial state of the `db.cursos` collection with three documents. The middle part shows the command `db.cursos.updateMany` with a filter `{duracion: { $gt: 10 } }` and an update `{ $set: { nombre: 'Física elemental', duracion: 20 } }`. Arrows point from the labels 'Filtro' and 'Nuevos valores' to the corresponding parts of the command. On the right, a terminal window shows the state after the update, where the documents with `duracion` greater than 10 have been modified.

```
db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4eac231cd78466ff804f2c"),
  "referencia" : "C0001",
  "nombre" : "Estadística",
  "duracion" : 6,
  "activo" : true,
  "finicio" : "2020-01-21T09:00:00.000Z"
},
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2d"),
  "referencia" : "C0002",
  "nombre" : "Pintura",
  "duracion" : 24,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
},
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2e"),
  "referencia" : "C0003",
  "nombre" : "Física",
  "duracion" : 46,
  "activo" : true,
  "finicio" : "2019-12-13T10:30:00.000Z"
}

db.cursos.updateMany(
  { duracion: { $gt: 10 } },
  { $set: {
    nombre: "Física elemental",
    duracion: 20
  } }
);

db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4eac231cd78466ff804f2c"),
  "referencia" : "C0001",
  "nombre" : "Estadística",
  "duracion" : 6,
  "activo" : true,
  "finicio" : "2020-01-21T09:00:00.000Z"
},
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2d"),
  "referencia" : "C0002",
  "nombre" : "Física elemental",
  "duracion" : 20,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
},
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2e"),
  "referencia" : "C0003",
  "nombre" : "Física elemental",
  "duracion" : 20,
  "activo" : true,
  "finicio" : "2019-12-13T10:30:00.000Z"
}
```

UPSERT en MongoDB, actualiza el documento, y si no existe, lo crea aprovechando los datos del update como definición del nuevo documento.

`> db.collection.update(<filter>, { <update> }, { upsert : true })`

Reemplazar

`db.collection.replaceOne(<filter>, <update>)`

<filter> json con la condición o condiciones por las que se va a filtrar

<update> json con los campos y valores con los que se quiere reemplazar

Reemplazo de un documento:

`db.collection.replaceOne(<filter>, <update>)`

The diagram illustrates the MongoDB `replaceOne` operation. It shows a 'Filtro' (Filter) and 'Nuevos valores' (New values) being used to update a document in a collection. The diagram includes three terminal screenshots showing the state of a `db.cursos` collection before and after the operation.

Filtro

```
{ "referencia": "C0003" }
```

Nuevos valores

```
{ "nombre": "Física", "duracion": 46, "matriculados": 15, "listaEspera": 6 }
```

Before Operation:

```
> db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4eac231cd78466ff804f2c"),
  "referencia" : "C0001",
  "nombre" : "Estadística",
  "duracion" : 6,
  "activo" : true,
  "finicio" : "2020-01-21T09:00:00.000Z"
}
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2d"),
  "referencia" : "C0002",
  "nombre" : "Física elemental",
  "duracion" : 20,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
}
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2e"),
  "referencia" : "C0003",
  "nombre" : "Física elemental",
  "duracion" : 20,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
}
```

After Operation:

```
> db.cursos.find().pretty();
{
  "_id" : ObjectId("5e4eac231cd78466ff804f2c"),
  "referencia" : "C0001",
  "nombre" : "Estadística",
  "duracion" : 6,
  "activo" : true,
  "finicio" : "2020-01-21T09:00:00.000Z"
}
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2d"),
  "referencia" : "C0002",
  "nombre" : "Física elemental",
  "duracion" : 20,
  "activo" : false,
  "finicio" : "2020-02-18T16:30:00.000Z"
}
{
  "_id" : ObjectId("5e4eac251cd78466ff804f2e"),
  "referencia" : "C0003",
  "nombre" : "Física",
  "duracion" : 46,
  "matriculados" : 15,
  "listaEspera" : 6,
  "activo" : true,
  "finicio" : "2020-12-13T10:30:00.000Z"
}
```

Ejemplo

Creemos una BD llamada `blog` y en ella creamos una colección llamada `posts`, en la que insertamos uno y varios documentos:

```
use blog
```

```
db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

```
db.posts.insertMany([
  {
    title: "Post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 3",
```

```

        body: "Body of post.",
        category: "Technology",
        likes: 3,
        tags: ["news", "events"],
        date: Date()
    },
    {
        title: "Post Title 4",
        body: "Body of post.",
        category: "Event",
        likes: 4,
        tags: ["news", "events"],
        date: Date()
    }
]
])

```

Consulta la información con `db.nameCollection.find().pretty()`:

```
blog> db.posts.find().pretty()
```

Actualizamos el contenido de los documentos la colección posts que cumplan una determinada condición (`category= "Event"`) cambiándole la categoría.

```

db.posts.updateMany(
  {category : "Event"},
  {$set:{category : "Success"}}
)

```

Actualizar el contenido de los documentos la colección posts que cumplan una determinada condición (`category= "Technology"`) incrementando en 1 sus likes.

```

db.posts.updateMany(
  {category : "Technology"},
  {$inc:{likes : 1}}
)

```

NOTA:

Aunque se use `updateMany` si solo hay un documento que lo cumpla, lo actualiza.

Eliminamos el documento la colección posts que cumplan una determinada condición (`title: "Post Title 1"`)

```

db.posts.deleteOne(
  {title: "Post Title 1"}
)

```

EJERCICIOS

EJERCICIO1

1. Partiendo del escritorio de tu ordenador, ejecuta los pasos necesarios para poder usar la consola de mongodb (abrir un terminal o consola, iniciar/parar servicios, ejecutar la shell de mongodb)
2. Crea una base de datos llamada concesionario. Recuerda insertar al menos un documento en una colección que se llame coches. Como propiedades del documento json, puedes usar matrícula, marca, modelo, versiones (sport, confort), kms y fecha de matriculación.
3. Crea una colección que se llame clientes. Como propiedades del documento json, puedes usar nombre, f_nacimiento, e_mail y telefono.
4. Visualiza el listado de todas las colecciones de datos disponibles y su contenido.

EJERCICIO2

- Crea la base de datos Retail.
- Inserta en la colección productos con los siguientes documentos de uno en uno:

```
{
  "referencia": "P0001",
  "tipo": "camisa",
  "paraHombre": true,
  "talla": "XS",
  "precio": 20.99
}
```

```
{
  "referencia": "P0002",
  "tipo": "camisa",
  "paraHombre": true,
  "talla": "XL",
  "precio": 30.25
}
```

```
{
  "referencia": "P0003",
  "tipo": "pantalon",
  "paraMujer": true,
  "talla": "L",
  "precio": 20.99
}
```

- Elimínalos todos.
- Ahora créalos, pero todos a la vez.
- Actualiza todos los documentos con un precio inferior a 25 para que tenga un precio un 10% más caro.
- Reemplaza cada documento que sea para hombre con la misma estructura pero añadiendo una propiedad nueva: "paraMujer": false.
- Reemplaza cada documento que sea para mujer con la misma estructura pero añadiendo una propiedad nueva: "paraHombre": false.
- Actualiza cada documento, filtrando por sus referencia. Las propiedades cambiando el tipo camisa por chaqueta y la talla debe ser igual a M.

Consultar

El método find ()

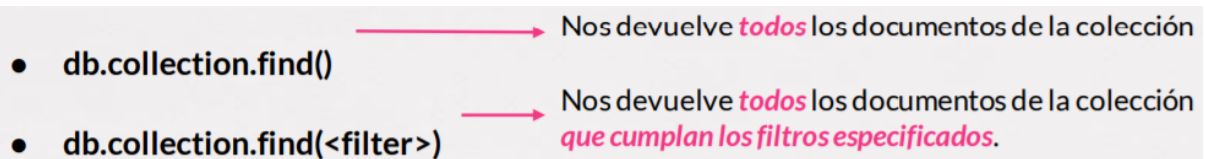
Para consultar datos de la colección de MongoDB, debe utilizar el método find ().

La sintaxis básica del método find () es la siguiente:

```
>db.COLLECTION_NAME.find()
```

Para mostrar los resultados de forma formateada, puede utilizar el método pretty ().

```
>db.COLLECTION_NAME.find().pretty()
```

- 
- **db.collection.find()** → Nos devuelve *todos* los documentos de la colección
 - **db.collection.find(<filter>)** → Nos devuelve *todos* los documentos de la colección *que cumplan los filtros especificados*.

<filter> json con la/s condicion/es que deben cumplir los documentos devueltos por la búsqueda.

Proyecciones

Mostrar solo algunas claves de los resultados de las consultas sobre todos los documentos de la colección

```
>db.COLLECTION_NAME.find({}, {key1:1, key2:1, _id:0})
```

o aplicando algún filtro:

```
>db.COLLECTION_NAME.find(<filter>,{key1:1, key2:1, _id:0})
```

el valor 1 es para mostrar la propiedad y el valor 0 para no mostrarlos

Existen otros métodos asociados a las consultas:

- Método limit () para leer un número determinado de datos, acepta un parámetro numérico que especifica el número de registros leídos.

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

- Método skip () para saltar un número determinado de datos, acepta un parámetro numérico como el número de registros para saltar.

```
>db.COLLECTION_NAME.find().skip(NUMBER).limit(NUMBER)
```

- Método sort () para ordenar los datos, el método sort () para especificar la columna de ordenación por el argumento, y el uso de 1 y -1 para especificar el tipo de camino, con 1 están dispuestas en orden ascendente, y -1 para en orden descendente.

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

Podemos combinar sort con skip o limit, y el orden es relevante, primero se aplica el filtro, luego sort, después skip y finalmente limit.

Operaciones de comparación

En las bases de datos relacionales, es muy típico filtrar los resultados según el valor de un determinado campo. Por ejemplo si $X > 0$, $Y \leq 0$. Pero ¿cómo realizamos esto en **MongoDB**?

Imaginemos que queremos mostrar las personas de la colección *people*, que tienen más de 30 años. Para ello utilizaremos el operador **\$gt** (abreviatura de “greater than” en inglés).

```
> db.people.find({edad:{$gt:30}}, {name:1, edad:1})
```

Como veis la consulta es bastante sencilla. Como primer parámetro del comando find añadimos la consulta y como segunda parte una proyección con los datos que queremos que nos devuelva dicha consulta (en este caso *name* y *edad*). Todo en **MongoDB** se hace con *JSON* así que para buscar los mayores de 30 años añadimos otro documento *JSON* con el operador utilizado y el valor por el que debe filtrar.

Si os fijáis en los resultados, no se devuelve ninguna persona con edad igual a 30. Esto es porque hemos usado **\$gt** y no **\$gte** (“greater than equals” en inglés). Así que si ejecutamos la consulta siguiente, obtendremos todos elementos de la colección *people* con edad mayor o igual a 30 años.

```
> db.people.find({edad:{$gte:30}}, {name:1, edad:1})
```

Lo mismo haremos si queremos obtener las personas menores de 30 años utilizando los comandos **\$lt** (“lower than”) o **\$lte** (“lower than equals”).

```
> db.people.find({edad:{$lt:30}}, {name:1, edad:1})
```

Y si quisieramos extraer todas las personas cuya edad NO es 30 utilizaríamos el operador **\$ne**.

```
> db.people.find({edad:{$ne:30}}, {name:1, edad:1})
```

Imaginemos ahora que queremos extraer las personas con una edad igual a 25, 30 o 35 años. En SQL podríamos utilizar un *WHERE edad IN (25,30,35)*. En **MongoDB** utilizaríamos el operador **\$in** y un array con los datos.

```
> db.people.find({edad:{$in:[25,30,35]}}, {name:1, edad:1})
```

Esta consulta nos devuelve todas las personas cuyas edades son igual a 25, 30 o 35.

Los operador condicionales son:

Operación	Sintaxis	Ejemplo	Equivalente a RDBMS
Igualdad	{<clave>: {\$ p. ej.: <valor>}}	db.mycol.find ({"por": "punto de tutoriales"}). pretty ()	donde by = 'punto de tutoriales'
Menos que	{<clave>: {\$ lt: <valor>}}	db.mycol.find ({"me gusta": {\$ lt: 50}}). pretty ()	donde me gusta <50
Menor que igual	{<clave>: {\$ lte: <valor>}}	db.mycol.find ({"me gusta": {\$ lte: 50}}). pretty ()	donde me gusta <= 50
Mas grande que	{<clave>: {\$ gt: <valor>}}	db.mycol.find ({"me gusta": {\$ gt: 50}}). pretty ()	donde me gusta > 50
Mayor que igual	{<clave>: {\$ gte: <valor>}}	db.mycol.find ({"me gusta": {\$ gte: 50}}). pretty ()	donde me gusta >= 50
No es igual	{<clave>: {\$ ne: <valor>}}	db.mycol.find ({"me gusta": {\$ ne: 50}}). pretty ()	donde me gusta! = 50
Valores en una matriz	{<clave>: {\$ en: [<valor1>, <valor2>, <valorN>]}}	db.mycol.find ({"nombre": {\$ in: ["Raj", "Ram", "Raghu"]}}). bonita ()	Donde el nombre coincide con cualquiera de los valores en: ["Raj", "Ram", "Raghu"]
Valores que no están en una matriz	{<clave>: {\$ nin: <valor>}}	db.mycol.find ({"nombre": {\$ nin: ["Ramu", "Raghav"]}}). bonita ()	Donde los valores de nombre no están en la matriz: ["Ramu", "Raghav"] o, no existe en absoluto

Comparaciones con strings

Los operadores descritos anteriormente son aplicables a los *strings*. Pero hay que tener en cuenta que **MongoDB** distingue entre mayúsculas y minúsculas y que utiliza **el orden lexicográfico**. Esto quiere decir que **MongoDB** ordena los *strings* de la misma manera que un diccionario, aunque diferenciando mayúsculas y minúsculas.

Este es un ejemplo de orden de strings que hace **MongoDB**.

```
{ "_id" : "AAab" }
```

```
{ "_id" : "Abb" }
```

```
{ "_id" : "Abc" }
```

```
{ "_id" : "BCb" }
```

```
{ "_id" : "Bbaab" }
```

```
{ "_id" : "abb" }
```

```
{ "_id" : "abc" }
```

```
{ "_id" : "bcb" }
```

En orden ascendente las mayúsculas van primero y luego se tiene en cuenta el orden lexicográfico de cada letra. Como podéis ver el número de caracteres no se tiene en cuenta.

Buscar una subcadena dentro de un string, podemos usar:

- el carácter comodín "/": { "_id" : /bc/ }
- operador \$regex: { "_id" : {\$regex: "bc"}}

Operaciones según la existencia o el tipo de los elementos

Como ya sabéis **MongoDB** es una base de datos sin esquema, lo que quiere decir que los documentos, aun siendo de la misma colección, pueden tener distintos campos. Incluso estos campos pueden ser de distintos tipos en cada documento.

Así que en ocasiones puede ser útil realizar una consulta que nos devuelva los documentos en los que exista un determinado campo. En la siguiente consulta comprobamos buscamos los documentos en los que existe el campo *company*.

```
>
```

```
db.people.find({company:{$exists:true}}, {name:1,edad:1,company:1  
})
```

Si quisieramos buscar los documentos que no tienen el campo `company`, bastará con cambiar el `true` por un `false` en el **`$exists`**.

MongoDB puede guardar documentos con distintos tipos en el mismo campo. Por ejemplo aunque `edad` es un número para todos los documentos, podríamos insertar un documento nuevo con un `string` en ese campo. Por tanto, también podríamos necesitar filtrar por documentos en los cuales un campo será de un determinado tipo. Esto se hace con el operador **`$type`**.

```
> db.people.find({company:{$type:2}}, {name:1, edad:1, company:1})
```

En este caso estamos buscando los documentos cuyo campo `company` sea de tipo 2, que es el tipo `string`. Podéis encontrar número asignado a cada tipo [en la ayuda del operador `\$type` en la página de MongoDB](#).

Operaciones lógicas

Los operadores lógicos son:

AND

```
>db.COLLECTION_NAME.find({ $and:[{<key1>:<value1>},{ <key2>:<value2>}]})
```

OR

```
>db.COLLECTION_NAME.find({ $or: [{key1: value1},{key2:value2}]})
```

NOR

```
>db.COLLECTION_NAME.find({ $nor:[{key1: value1},{key2:value2}]})
```

NOT

```
>db.COLLECTION_NAME.find({ $NOT: [{key1: value1},{key2:value2}]})
```

En bases de datos relacionales es muy típico añadir operadores *OR* a la cláusula *WHERE*. Por ejemplo *WHERE genero = "femenino" OR edad > 20*. Hasta ahora las consultas que hemos visto buscaban por uno o más campos, pero lo hacían con la lógica de un *AND*, es decir, que todas las condiciones debían cumplirse. Si queremos añadir cláusulas *OR* usaremos **`$or`**.

```
> db.people.find(
  { $or:[{genero:"femenino"}, {edad:{$gt:20}}] },
  {name:1, genero:1, edad:1} )
```

En este caso buscamos los documentos cuyo campo `genero` sea *"femenino"* o cuyo campo `edad` sea mayor que 20. Como vemos basta con especificar un array de condiciones para que el operador **`$or`** realice la consulta.

Lo curioso es que también existe un operador **\$and**. ¿Por qué es curioso? Pensemos en las siguientes consultas

```
> db.people.find( {genero:"femenino", edad:{$gt:20}} ,
{name:1,genero:1,edad:1} )
```

```
> db.people.find( {$and: [{genero:"femenino"},
    {edad:{$gt:20}}] } ,
    {name:1,genero:1,edad:1} )
```

Pues es curioso, porque en realidad, la consulta es la misma. En la primera, **MongoDB** hace un *and implícito* de los parámetros de la consulta, mientras que en la segunda hemos incluido el *and explícitamente*.

¿Entonces por qué usar este operador? Pues puede utilizarse si tenemos que hacer una consulta que incluya dos veces el mismo campo. Si no utilizamos el *and* y lo hacemos de esta manera, podemos obtener resultados erróneos. Por ejemplo las siguientes consultas son iguales, aunque invirtiendo el orden de las condiciones.

```
> db.people.find( {edad:{$gt:30}, edad:{$lt:40}} ,
{name:1,genero:1,edad:1} )

> db.people.find( {edad:{$lt:40},edad:{$gt:30}} ,
{name:1,genero:1,edad:1} )
```

Si las ejecutáis veréis que devuelven resultados distintos. Esto es porque **MongoDB** coge el último valor para realizar la consulta. La consulta correcta sería:

```
> db.people.find(
    { $and: [{edad:{$gt:30}}, {edad:{$lt:40}}] } ,
    {name:1,genero:1,edad:1} )
```

\$and también puede ser útil para filtrar conjuntamente con **\$or**

```
db.people.find(
    {$or: [ {edad:{$lt:30}},
    {$and:[{edad:{$gt:50}}, {genero:"femenino"}]}
    ]
})
```

Este comando nos buscará las personas en `people` cuya edad es mayor que 30, o cuyo género es “femenino” y su edad mayor que 50.

Además de ***\$and*** y ***\$or***, tenemos otros dos operadores lógicos que son ***\$not*** y ***\$nor***. ***\$not*** es bastante sencillo de entender ya que lo que hace es buscar los documentos que no cumplan una determinada condición.

```
> db.people.find( {edad:{$not:{$gt:30}}})
```

Lo importante en este caso, es saber que ***\$not*** solo puede usarse con otros operadores como ***\$gt*** o ***\$lt***. No puede usarse con valores directos o documentos. Para eso ya existe el operador ***\$ne*** que hemos explicado antes. También hay que tener en cuenta que estamos buscando edades que no sean mayores que 30. Esto incluye el 30, ya que está fuera del conjunto de números mayores que 30, y los documentos que no tengan campo `edad`.

También podemos utilizar el operador ***\$nor*** que acepta dos o más valores. Por ejemplo en la siguiente consulta buscamos las personas cuya edad NO sea mayor que 30 y cuyo campo `isActive` NO sea true.

```
>db.people.find({$nor:[{edad:{$gt:30}}, {isActive:true}]},{edad:1,isActive:1})
```

Destacar que al igual que ***\$not***, ***\$nor*** devuelve también los documentos si los campos no existen. Para evitar esto, si es algo que no deseamos, podemos añadir el operador ***\$exists***.

```
db.people.find(  
  {$nor:  
    [  
      {edad:{$gt:30}}, {edad:{$exists:false}},  
      {isActive:true}, {isActive:{$exists:false}}  
    ]  
  })
```

En este caso buscamos los documentos cuya edad NO sea mayor que 30, cuyo campo `isActive` NO sea true y que ambos campos existan.

Consultas sobre Arrays

La estructura de documentos sobre la que se hacen las consultas sería del tipo:

```
{
  key1: value1
  key2: [value_a0, value_a1, value_a2]
}
```

Igualdad de todo el array exactamente (exactamente en un orden concreto)

```
>db.COLLECTION_NAME.find({key2: [value_a0, value_a1, value_a2]})
```

Igualdad de todo el array exactamente (sin importar el orden)

```
>db.COLLECTION_NAME.find({key2: {$all:[value_a1, value_a2,
value_a0]}})
```

Encontrar un elemento dentro del array con un valor concreto

```
>db.COLLECTION_NAME.find({key2: value_a1})
```

Encontrar un elemento dentro del array que cumpla una condición

```
>db.COLLECTION_NAME.find({key2: {$operator: value_a1}})
```

Encontrar un elemento dentro del array que cumpla más de una condición

```
>db.COLLECTION_NAME.find({key2:{$elemMatch:{value_a1,value_a2}}})
```

Búsqueda por posición dentro del array

```
>db.COLLECTION_NAME.find({"key2.pos": value_a1})
```

Búsqueda de los documentos que tengan el array de un tamaño concreto

```
>db.COLLECTION_NAME.find({key2: {$size:tamaño}})
```

Consultas sobre Documentos anidados

La estructura de documentos sobre la que se hacen las consultas sería del tipo:

```
{
  key1: {
    subkey1: val_s1,
    subkey2: val_s2
  }
  key2: value2
}
```

Por condición de alguna propiedad dentro del documento anidado

```
>db.COLLECTION_NAME.find({"key1.subkey1": val_s1})
```

donde value puede llevar por ejemplo un operador condicional

```
>db.COLLECTION_NAME.find({"key1.subkey1": {$operator: val_s1}})
```

donde se puede tener más de una condición sobre una propiedad del documento principal o del embebido

```
>db.COLLECTION_NAME.find({"key1.subkey1":val_s1,"key2":value2})
```

EJERCICIO3

- Utilizando la base de datos Retail.

```
{  
  "referencia": "P0001",  
  "tipo": "camisa",  
  "paraHombre": true,  
  "talla": "XS",  
  "precio": 20.99  
}
```

```
{  
  "referencia": "P0002",  
  "tipo": "camisa",  
  "paraHombre": true,  
  "talla": "XL",  
  "precio": 30.25  
}
```

```
{  
  "referencia": "P0003",  
  "tipo": "pantalon",  
  "paraMujer": true,  
  "talla": "L",  
  "precio": 20.99  
}
```

- Actualiza la colección productos con las siguientes propiedades:
 - P0001, añade las siguientes propiedades:
tienda: ["Jerez", "Sevilla", "Cordoba"]
proveedor: {nombre: "Camiseros SA", nif:"B12345678", contacto:"Jose"}
 - P0002, añade las siguientes propiedades:
tienda: ["Jerez", "Chucena", "Cordoba"]
proveedor: {nombre: "Camiseros SA", nif:"B12345678", contacto:"Juan"}
 - P0003, añade las siguientes propiedades:
tienda: ["Rota", "Sevilla"]
proveedor: {nombre: "Pantaloneros SA", nif:"B87654321", contacto:"Jose"}
- Realiza las siguientes consultas sobre la colección productos:
 - De cada producto mostrar su referencia, las tiendas en las que se vende y el nombre del proveedor que lo suministra.
 - Mostrar la referencia y talla de todos los productos que se venden en la tienda de Jerez.
 - Mostrar los productos que se venden en las tiendas de Jerez y Cordoba.
 - Mostrar la referencia, talla y precio de los productos suministrados por el proveedor cuyo nif es "B12345678".
 - Mostrar los productos cuyo contacto del proveedor sea "Jose".

Modificaciones sobre Arrays

La estructura de documentos sobre la que se hacen las consultas sería del tipo:

```
{
  key1: value1
  key2: [value_a0, value_a1, value_a2]
}
```

Modificar un elemento concreto (por posición) dentro de un array.

```
> db.COLLECTION_NAME.updateOne({ <filter>}, { $set : { "key2.pos" : "new_value" }})
```

Añadir un elemento al array por la derecha

```
> db.COLLECTION_NAME.updateOne({ <filter>}, { $push : { "key2" : "new_value" }})
```

Eliminar un elemento del array por la izquierda (para que fuese por la derecha, habría que sustituir -1 por 1 positivo)

```
> db.COLLECTION_NAME.updateOne({ <filter>}, { $pop : { "key2" : -1 }})
```

Añadir varios los elementos especificados al Array

```
> db.COLLECTION_NAME.update({ <filter>}, { $pushAll : { "key2" : [ "new_value_b0",
"new_value_b1", "new_value_b2" ]}})
```

Eliminar un elemento cualquiera del array especificando su valor.

```
> db.COLLECTION_NAME.update({ <filter> }, { $pull : { "key2" : "value_a1" }})
```

Eliminar todos los elementos coincidentes del Array.

```
> db.COLLECTION_NAME.update({ <filter> }, { $pullAll : { "key2" : [ "value_a0", "value_a1" ]}})
```

Añadir un elemento al array únicamente si no existe ya.

```
> db.COLLECTION_NAME.update({ <filter> }, { $addToSet : { "key2" : "new_value_c1" }})
```


EJERCICIO4

Trabajando con Profesores

```
{
  {
    "nombre": "María",
    "apellidos": "Suárez Manrique",
    "especialidad": [
      "biología"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 51
  },
  {
    "nombre": "Jose Luis",
    "apellidos": "López Pérez",
    "especialidad": [
      "matemáticas",
      "física"
    ],
    "esTitular": false,
    "esAsociado": true,
    "edad": 39
  },
  {
    "nombre": "Antonio",
    "apellidos": "Munguía Arteche",
    "especialidad": [
      "física, química"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 54
  }
}
```

- Obtener todos los profesores.
- Obtener todos los profesores asociados.
- Obtener todos los profesores titulares y mayores de 50.
- Obtener todos los profesores con edades entre 50 y 60.
- Número total de profesores.
- Número total de profesores asociados.
- Obtener solo los 2 primeros resultados.

- Obtener todos los profesores con especialidad física o biología.
- Obtener todos los profesores con especialidad biología o que sea asociado.
- Obtener todos los profesores que imparten exactamente matemáticas y física (exactamente en ese orden).
- Obtener todos los profesores que imparten matemáticas y física (sin orden).
- Número total de profesores que imparten biología.

Trabajando con Profesores y asignaturas

```
{
  {
    "nombre": "María",
    "apellidos": "Suárez Manrique",
    "especialidad": [
      "biología"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 51,
    "asignatura": {
      "id": "A0001",
      "nombre": "Biología molecular",
      "creditos": 6
    }
  },
  {
    "nombre": "Jose Luis",
    "apellidos": "López Pérez",
    "especialidad": [
      "matemáticas",
      "física"
    ],
    "esTitular": false,
    "esAsociado": true,
    "edad": 39,
    "asignatura": {
      "id": "A0002",
      "nombre": "Termodinámica",
      "creditos": 9
    }
  },
  {
    "nombre": "Antonio",
    "apellidos": "Munguía Arteche",
    "especialidad": [
      "física, química"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 54,
    "asignatura": {
      "id": "A0003",
      "nombre": "Termodinámica",
      "creditos": 9
    }
  }
}
```

```
{
  {
    "nombre": "María",
    "apellidos": "Suárez Manrique",
    "especialidad": [
      "biología"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 51,
    "asignatura": {
      "id": "A0001",
      "nombre": "Biología molecular",
      "creditos": 6
    }
  },
  {
    "nombre": "Jose Luis",
    "apellidos": "López Pérez",
    "especialidad": [
      "matemáticas",
      "física"
    ],
    "esTitular": false,
    "esAsociado": true,
    "edad": 39,
    "asignatura": {
      "id": "A0002",
      "nombre": "Termodinámica",
      "creditos": 9
    }
  },
  {
    "nombre": "Antonio",
    "apellidos": "Munguía Arteche",
    "especialidad": [
      "física, química"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 54,
    "asignatura": {
      "id": "A0003",
      "nombre": "Termodinámica",
      "creditos": 9
    }
  }
}
```

```
{
  {
    "nombre": "María",
    "apellidos": "Suárez Manrique",
    "especialidad": [
      "biología"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 51,
    "asignatura": {
      "id": "A0001",
      "nombre": "Biología molecular",
      "creditos": 6
    }
  },
  {
    "nombre": "Jose Luis",
    "apellidos": "López Pérez",
    "especialidad": [
      "matemáticas",
      "física"
    ],
    "esTitular": false,
    "esAsociado": true,
    "edad": 39,
    "asignatura": {
      "id": "A0002",
      "nombre": "Termodinámica",
      "creditos": 9
    }
  },
  {
    "nombre": "Antonio",
    "apellidos": "Munguía Arteche",
    "especialidad": [
      "física, química"
    ],
    "esTitular": true,
    "esAsociado": false,
    "edad": 54,
    "asignatura": {
      "id": "A0003",
      "nombre": "Termodinámica",
      "creditos": 9
    }
  }
}
```

Actualiza la colección Profesores con los datos de las asignaturas que imparten.

Obtener todos los profesores que imparten la asignatura A0002.

Obtener todos los profesores que imparten termodinámica.

Obtener todos los profesores que den asignaturas de más de 6 créditos.

Obtener todos los que den asignaturas de más de 6 créditos y mayores de 40.

Trabajando con Profesores y vehículos

Añadir propiedades (vehículo) y registros a la colección Profesores

```
{ "nombre": "María", "apellidos": "Suárez Manrique", "especialidad": [ "biología" ],  
  "esTitular": true, "esAsociado": false, "vehiculo": { "matricula": "1111ABC", "tipo": "SUV",  
  "marca": "Lexus", "caballos": 190, "uso": 3 } },
```

```
{ "nombre": "Jose Luis", "apellidos": "López Pérez", "especialidad": [ "matemáticas",  
  "física" ], "esTitular": true, "esAsociado": false, "vehiculo": { "matricula": "2222DEF",  
  "tipo": "SUV", "marca": "Toyota", "caballos": 120, "uso": 13 } },
```

```
{ "nombre": "Antonio", "apellidos": "Munguía Arteché", "especialidad": [ "física, química"  
  ], "esTitular": true, "esAsociado": false, "vehiculo": { "matricula": "3333HIJ", "tipo":  
  "turismo", "marca": "volvo", "caballos": 176, "uso": 6 } },
```

```
{ "nombre": "Fernando", "apellidos": "Delgado De La Fuente", "especialidad": [ "física" ],  
  "esTitular": false, "esAsociado": true, "vehiculo": { "matricula": "4444KLM", "tipo": "moto",  
  "marca": "Yamaha", "caballos": 75, "uso": 8 } },
```

```
{ "nombre": "Elena", "apellidos": "Hérendez Serafín", "especialidad": [ "matemáticas",  
  "física" ], "esTitular": true, "esAsociado": false, "vehiculo": { "matricula": "5555NOP",  
  "tipo": "moto", "marca": "honda", "caballos": 170, "uso": 4 } }
```

Consultas a realizar:

- Obtener todos los profesores que tienen un SUV.
- Obtener todos los profesores que tienen una moto y un uso de menos de 5 años.
- Obtener todos los profesores con vehículo de más de 150 caballos y que impartan física.
- Obtener el nº de profesores con más de 50 años, que tienen moto.
- Obtener el nº de profesores titulares que tienen un SUV.