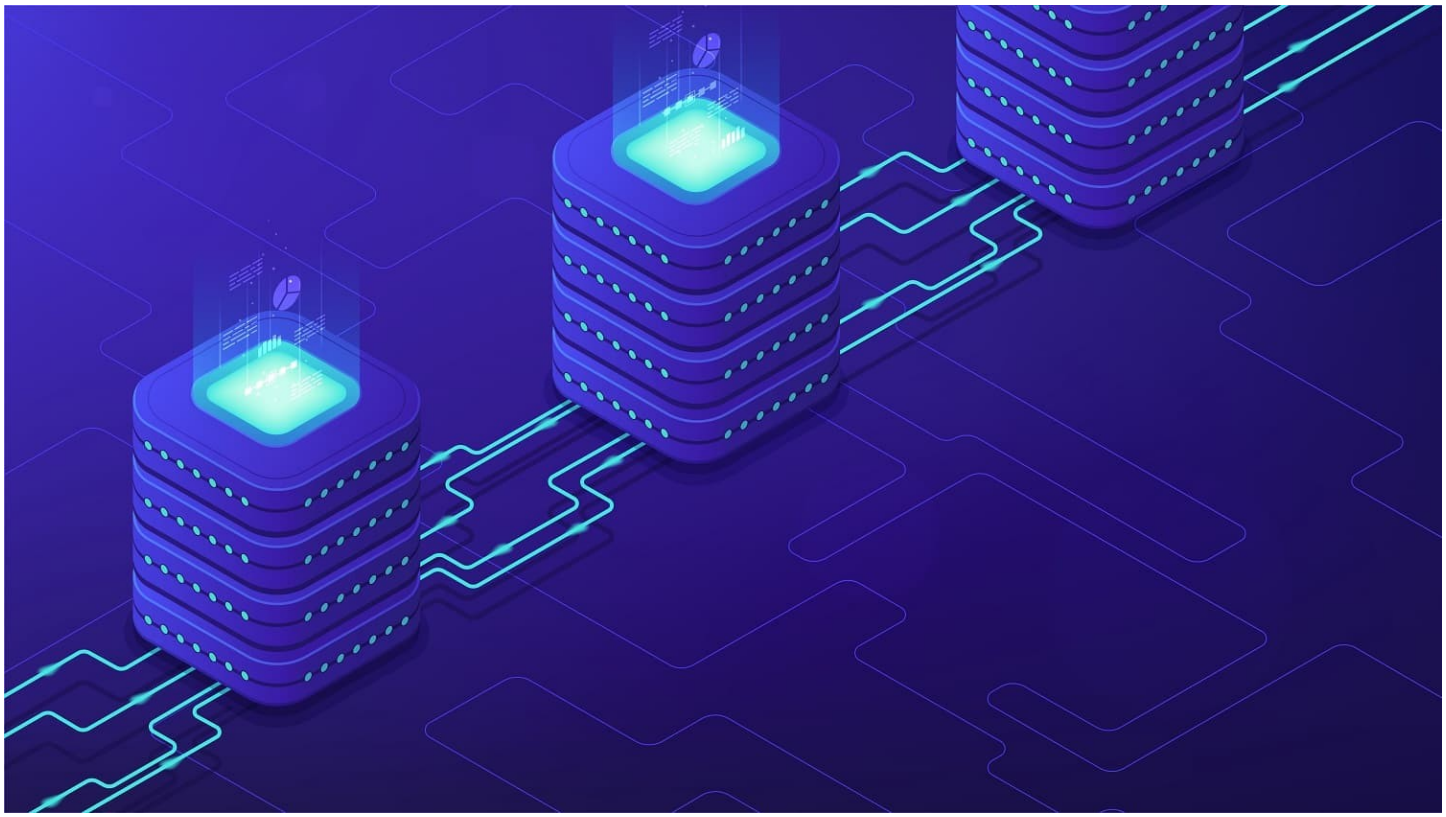
PRIMER PROYECTO DE BASE DE DATOS (MONGO DB)



DIEGO JOSÉ FERNÁNDEZ AUGUSTO 1ºASIR
MÓDULO: GESTIÓN DE BASE DE DATOS

-En este primer proyecto nos hemos enfrentado a la creación de una base de datos completa, en la cuál yo he elegido hacerla sobre los distintos tipos de coches con sus características.

-Después de hacer la base de datos he creado una serie de consultas con los operadores que hemos trabajado en clase y he añadido algunos más como aportación personal.

```
db.coches.find({$and:[
    {CV:{$gte:300}},
    {"consumo.min":{$lt:10}},
    {cambioManual:{$exists:true}},
    {tipoCombustible:{$in:["Gasolina","Diesel"]}},
    {tipo:{$eq: "Coupé"}}
]}).pretty()
```

→ En esta primera consulta he utilizado los operadores:

$and: Realiza una lógica de operación en una matriz de *una o más* expresiones y selecciona los documentos que satisfacen *todas l*as expresiones.

$gte: Selecciona los documentos donde el valor de campo que es mayor o igual a un valor especificado.

$lt: Selecciona los documentos donde el valor de campo que es menor a un valor especificado.

$exists: Cuando un booleano es verdadero, exists coincide con los documentos que contienen el campo, incluidos los documentos en los que se encuentra el valor del campo si el booleano es falso, la consulta devuelve solo los documentos que no contienen el campo.

$in: Selecciona los documentos donde el valor de un campo es igual a cualquier valor en la matriz especificada.

$eq: Selecciona los documentos donde el valor de campo que es igual a un valor especificado.

```
db.coches.find({$or:[
        {"motor.bloque.capacidad":{$lte:2}},
        {"motor.inyección":{$ne:"directa"}}
],
   marca:{$nin:["Audi", "Cupra"]},
   "motor.cilindrada":{$not:{$gt:2000, $lt:1500}}

}).pretty()
```

→ En la según da consulta los operadores usados son:

$or: Realiza una operación lógica en una matriz de *dos o más* expresiones y selecciona los documentos que satisfacen *al menos* uno de los expresiones.

$lte: Selecciona los documentos donde el valor de campo que es menor o igual a un valor especificado.

$ne: Selecciona los documentos donde el valor del campo no es igual al valor especificado. Esto incluye documentos que no contienen el campo.

$nin: Tiene la misma función que el $ne.

$not: Realiza una operación lógica en el especificado campo y selecciona los documentos que *no* coinciden la expresión.

$gt: Selecciona los documentos donde el valor de campo que es mayor a un valor especificado.

```
db.coches.find({$nor:[
    {CV:{$gte:140}},
    {"consumo.max":{$gt:8}}
],
    tipoCombustible:{$all:["Diesel"]},
    tipo:"Sedane",
    precioBasico:{$lte:25000}
}
).pretty()
```

→ En esta consulta he utilizado:

$nor: realiza una lógica en una matriz de una o más expresiones de consulta y selecciona los documentos que fallan en todas las expresiones de consulta de la base de datos.

$all: Selecciona los documentos donde el valor de un campo es una matriz que contiene todos los elementos especificados.

```
db.coches.find({
    tipoCombustible:{$size:2},
    frenos:{$all:[{"$elemMatch":{tipo:"carbocerámicos",grosor:{$lt:320}}}]},
    puertas:{$type:1}
}
).pretty()
```

→ Esta es la 4º consulta, los operadores usados son:

$size: operador hace coincidir cualquier campo con el número de elementos especificados por el argumento

$elemMatch: operador que compara documentos que contienen un campo de array con al menos un elemento que coincide con todos los criterios de consulta especificados.

$type: selecciona documentos donde el de campo una instancia de los tipos BSON especificados.

```
db.coches.find({
    $nor:[{$jsonSchema:{required:["freno"], properties:{tipo:{bsonType:"string"},grodor:{bsonType:"double"}}}}],
    tipoCombustible:["Gasolina"],
    fechaFabricación:{$gte:new Date("2018-01-01")},
    CV:{$gt:400}
}).prettyy()
```

→ Los operadores nuevos son:

$jsonSchema: Operador que coincide con los documentos que satisfacen el esquema JSON especificado.

```
db.coches.find({
    precioBasico:{$lte:15000},
    tipoCombustible:["Diesel"],
    capacidadMaletero:{$gt:500},
    tipoCombustible:{$not:{$regex: /^P./m},}
}).pretty()
```

→ Este es el último operador que vimos en clase:

$regex: Proporciona capacidades de expresión regular para *cadenas de c*oincidencia de patrones en consultas.

```
db.coches.aggregate([{
    $addFields:{tracción:["trasera", "delantera"]},
}]).pretty()
```

```
db.coches.aggregate([
    {$match:
        {$or:
            [
                {CV:{ $gt:95, $lt:150}},
                {"motor.cilindrada":{$lte:3000}}
            ]
        }
    },
    { $group: { _id: null, count: { $sum: 1 } } },
]).pretty()
```

```
db.coches.aggregate([
    {$match:{precioBasico:{$gt:50000}}},
    {$count:"Número de coches por un valor superior a 50000€"}
]).pretty()
```

→ Estos son algunos de los nuevos operadores que he añadido al proyecto:
(son .aggregate)

$addFields: Agrega nuevos campos a los documentos de salida que contienen todos los campos existentes de los documentos de entrada y los campos recién agregados.
$match: Toma un documento que especifica las condiciones de la consulta. La sintaxis de la consulta es idéntica a la sintaxis de la consulta de la operación de lectura es decir match no acepta expresiones de agregación sin procesar.
$group: Agrupa los documentos de entrada por la expresión especificada y para cada agrupación distinta, genera un documento.
$count: Pasa un documento a la siguiente etapa que contiene un recuento del número de documentos ingresados.