# Diseño y Desarrollo de Sistemas de Información

## Tema 3

## Sistemas de Gestión de Bases de Datos no Relacionales

- Introducción a las BD NoSQL
- MongoDB (operaciones CRUD)

# ¿Por qué surge esta necesidad?

- La mayoría de las bases de datos son de tipo relacional, es decir, conjuntos de tablas que se relacionan entre sí para almacenar y gestionar la información de forma eficiente

- Esta información se gestiona usando SQL (*Structured Query Language*), un lenguaje estructurado eficiente para el tratamiento de este tipo de estructura

- En los últimos años, Internet ha crecido de forma espectacular dando lugar a nuevos tipos de aplicaciones que necesitan gestionar grandes volúmenes de datos, que deben ser tratados de forma eficiente y que deben permitir su crecimiento de manera sencilla y poco costosa

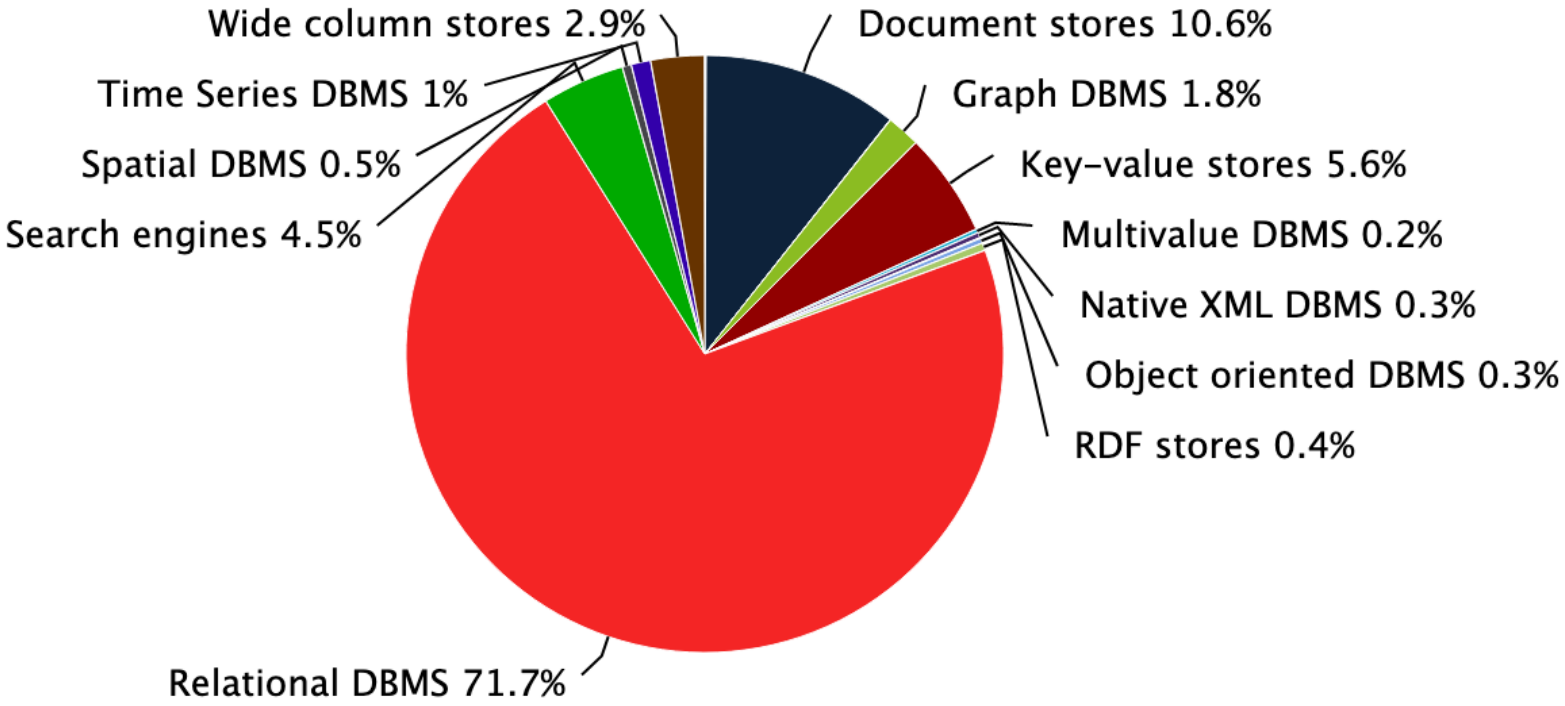
# ¿Por qué surge esta necesidad?

- En estos sistemas es más relevante la flexibilidad, la velocidad y la capacidad de escalado horizontal que otras cuestiones, tradicionalmente cruciales, como la consistencia o disponer de un esquema perfectamente definido

- Existe una necesidad de una nueva definición y uso de bases de datos que, en lugar de almacenar tablas de datos, almacenen estructuras heterogéneas **(Bases de Datos NoSQL)**

- Estas nuevas bases de datos se caracterizan por la enorme flexibilidad de sus esquemas de almacenamiento, simplicidad de uso, alto rendimiento y escalabilidad, facilidad de mantenimiento y capacidad de recuperación

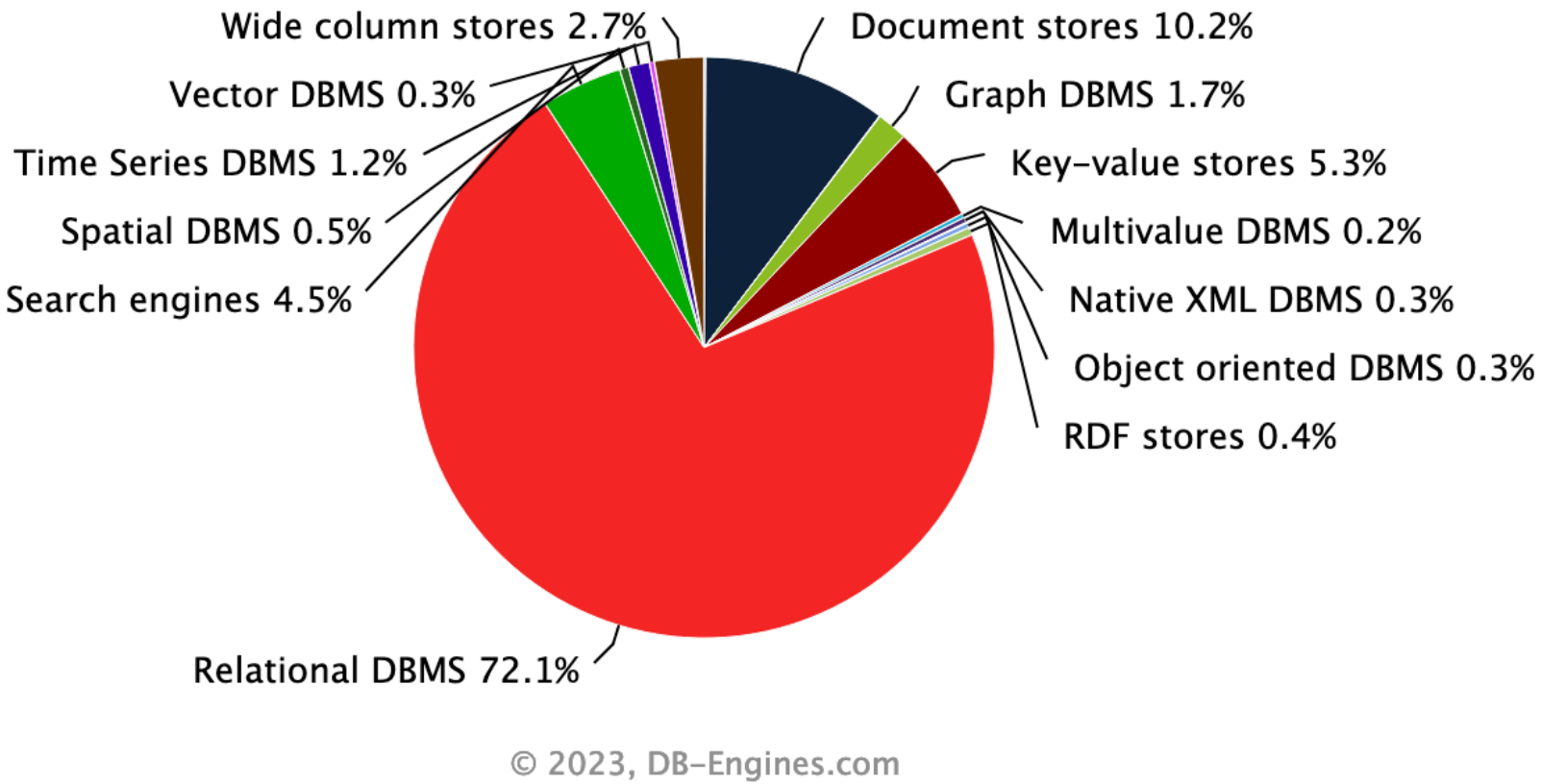
# Panorama actual




2022

https://db-engines.com/en/ranking_categories

# Panorama actual

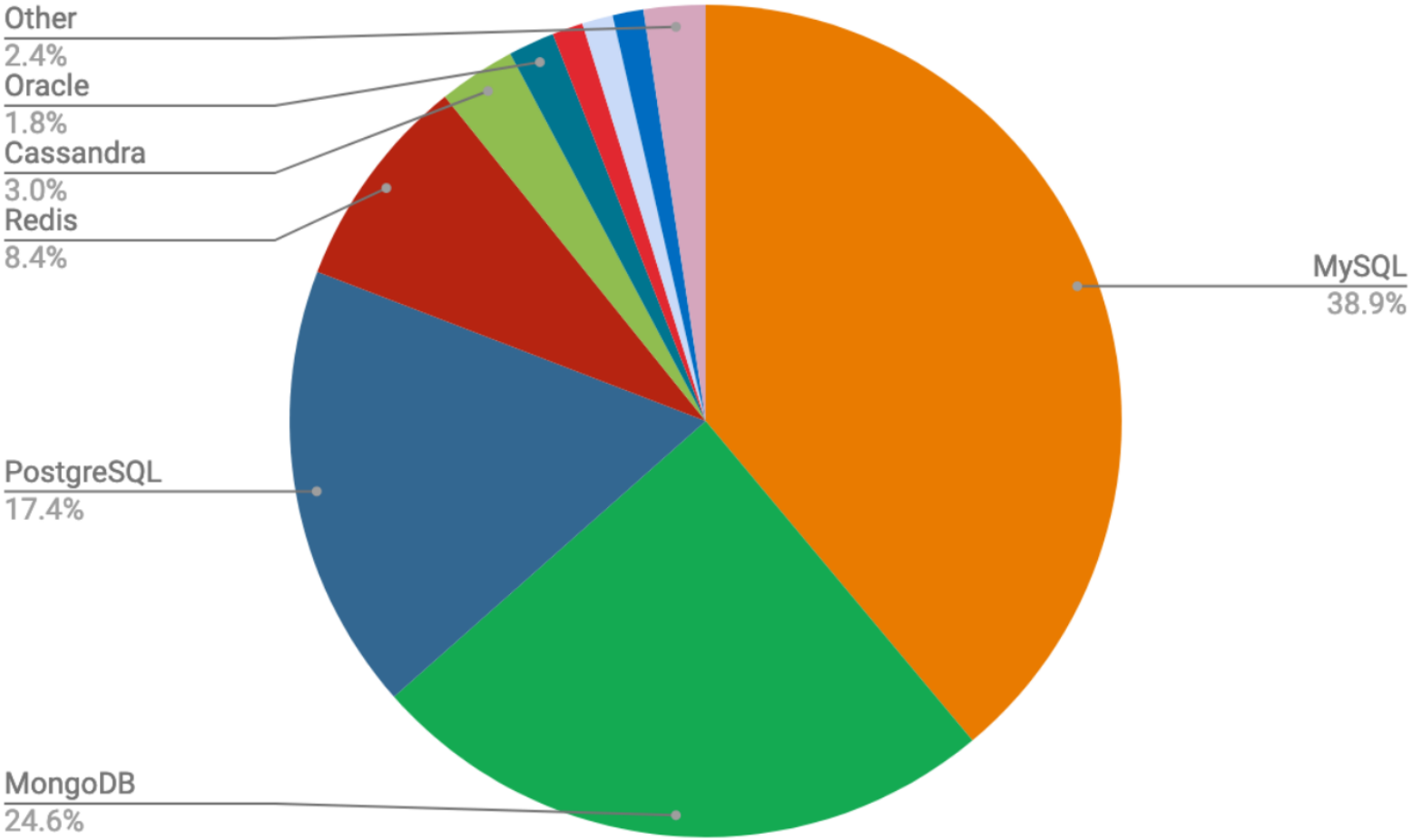



2023



https://db-engines.com/en/ranking_categories

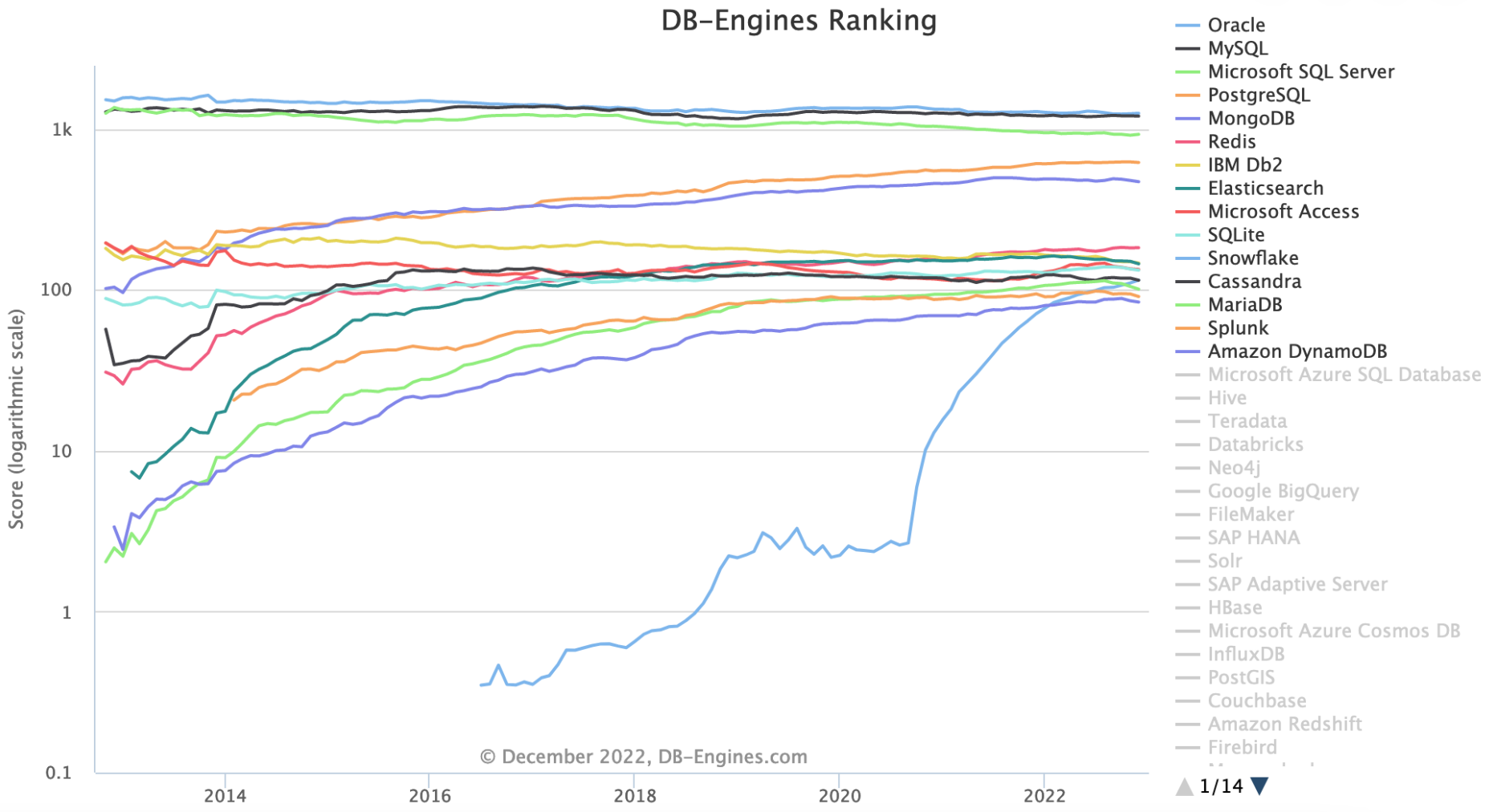
# Panorama actual




https://scalegrid.io/blog/2019-database-trends-sql-vs-nosql-top-databasessingle-vs-multiple-database-use/
Encuesta realizada a desarrolladores, ingenieros, arquitectos de software, equipos de desarrollo y líderes de TI en la DeveloperWeek

# Panorama actual

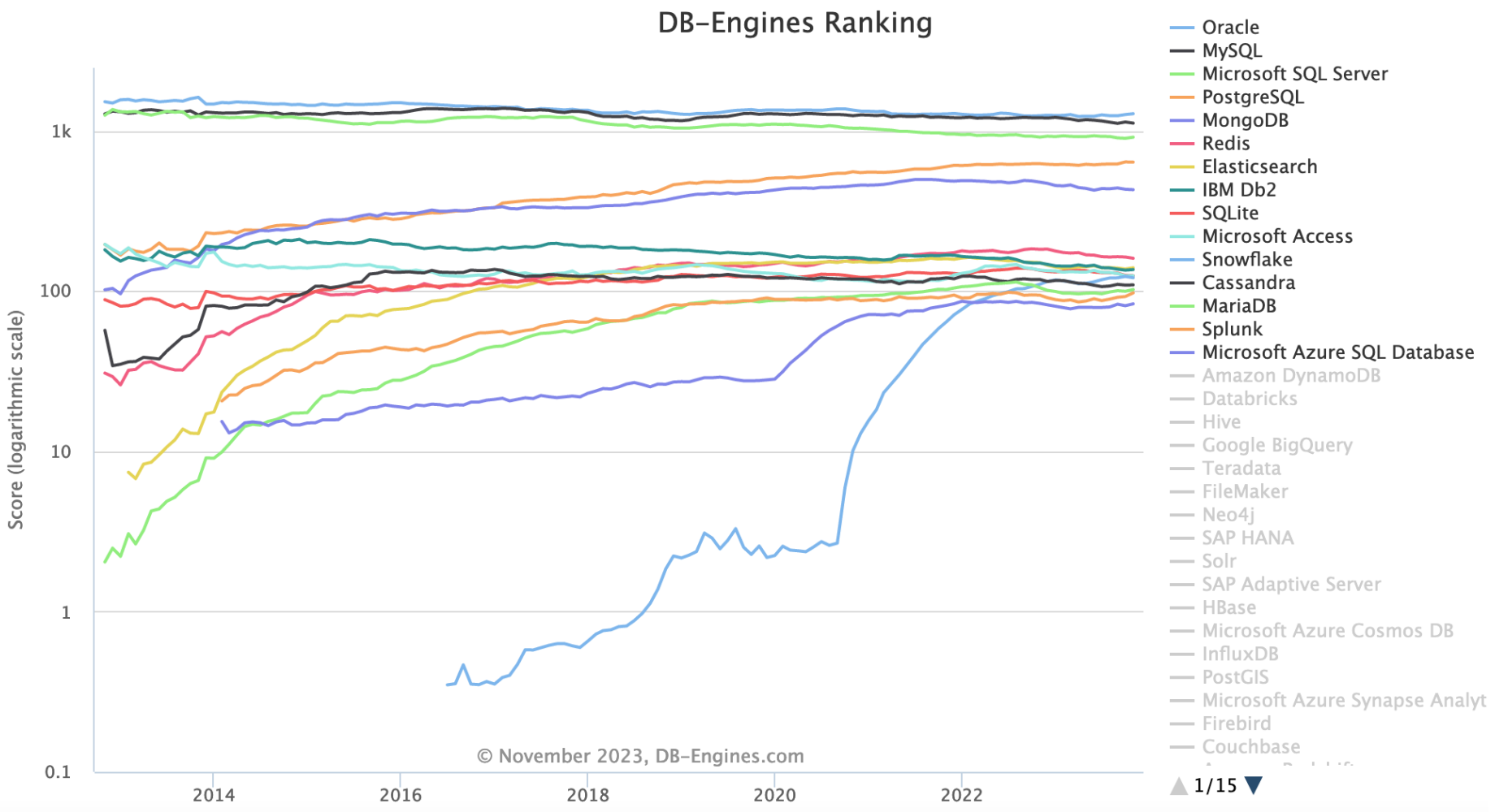
2022




https://db-engines.com/en/ranking_trend

# Panorama actual

2023




https://db-engines.com/en/ranking_trend

# Características de las BD NoSQL

- **Flexibilidad en los esquemas** o ausencia de éstos
  - Cada registro (en este modelo se suele llamar documento) puede contener información con diferente estructura, incluso pudiendo almacenar únicamente los campos o atributos que se necesiten o de los que se tenga información
  - Se pueden almacenar estructuras de datos complejas en un único documento. Por ejemplo, se puede almacenar, en un único registro, la información sobre la entrada de un blog (título, cuerpo de texto, fecha, autor, etiquetas, etc.) junto a los comentarios y respuestas
  - Aumenta la claridad (al tener todos los datos relacionados en un mismo bloque de información) y el rendimiento (no son necesarios los JOIN para obtener los datos relacionados, pues éstos se encuentran directamente en el mismo documento)

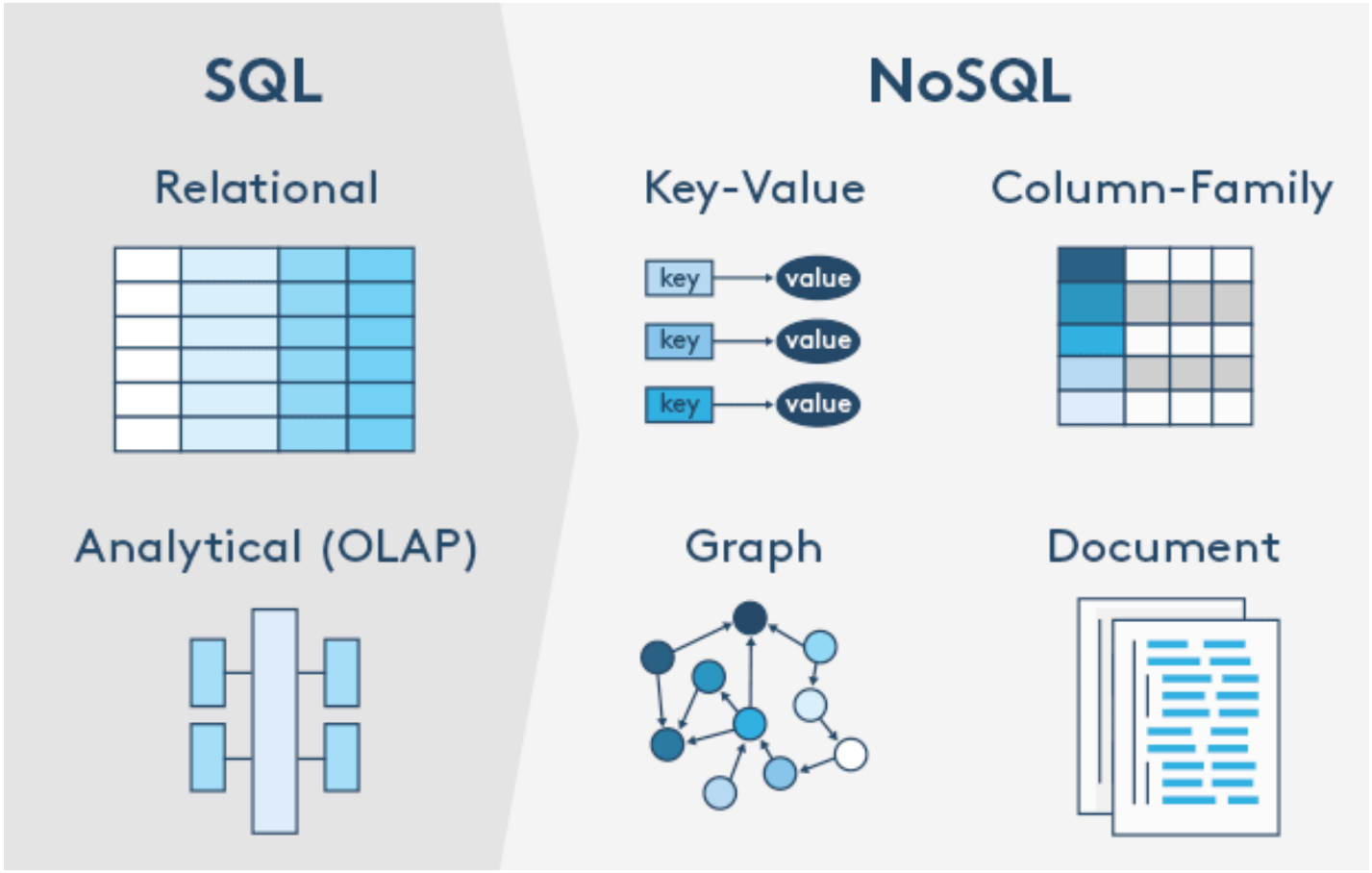
# Características de las BD NoSQL

- **Escalabilidad horizontal** sobre equipos estándar
  - Posibilidad de aumentar el rendimiento del sistema simplemente añadiendo más nodos de una forma muy sencilla
  - *Sharding* (particionado horizontal por clave y distribución)

- Alto rendimiento
  - Actualizaciones intensivas de los datos en entornos distribuidos
  - Uso eficiente de índices distribuidos
  - Almacenamiento orientado a memoria RAM

- Relajación (incluso eliminación) de algunas características ACID
  - Atomicidad, Consistencia, Aislamiento, Durabilidad

# Clasificación



# Clasificación

- Clave - valor:
  - Redis, Tokyo, BerkeleyDB, JBoss cache, Velocity, Amazon Dynamo, Voldemort, Dynomite, SubRecord, . . .
- Orientadas a columnas:
  - Google BigTable, HBase, Facebook Cassandra, HyperTable, . . .
- Orientadas a documentos:
  - CouchDB, MongoDB, Apache JackRabbit, Firestore, ThruDB, . . .
- Orientadas a grafos:
  - Neo4j, VertexDB, Infogrid, Sones, Filament, Allegrograph, HyperGraphDB, . . .

# Algunas Conclusiones

- NoSQL no siempre es la mejor solución
  - Si los datos tienen una naturaleza relacional, es conveniente seguir usando los SGDBR

- NoSQL se basa en 4 principios generales:
  - El control transaccional no es importante
  - Las operaciones de JOIN tampoco son especialmente lo son especialmente relevantes. En especial las complejas y distribuidas. Se persigue la desnormalización
  - Algunos elementos del modelo relacional son necesarios y aconsejables: claves primarias, modelado(keys).
  - Gran capacidad de escalabilidad y de replicación en múltiples servidores.

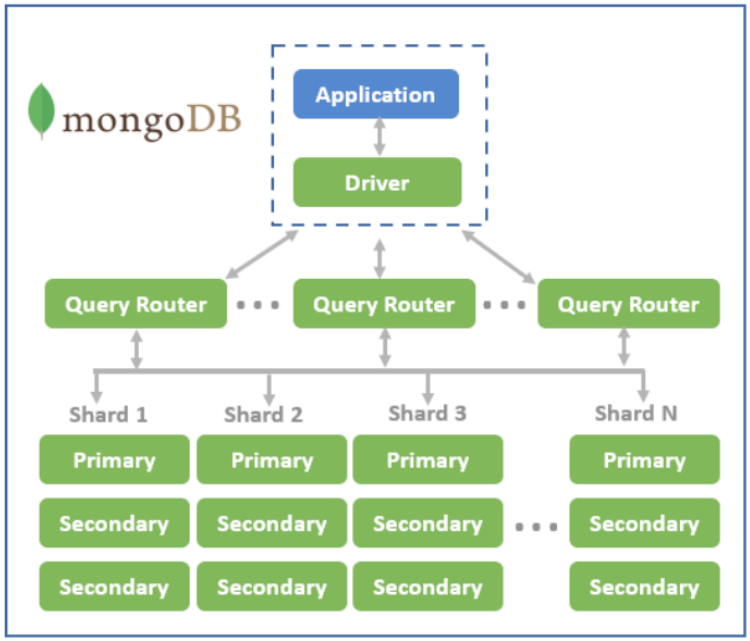
# Algunas Conclusiones

- Su uso es adecuado en situaciones cuando:
  - Es necesario manejar un gran volumen de datos
  - Hay una alta frecuencia de accesos de lectura y escritura
  - Se producen cambios frecuentes en los esquemas de datos
  - No requiere consistencia ACID

- Casos de aplicación:
  - Servicios Web2.0 (redes sociales, blogs, etc.)
  - Aplicaciones IoT
  - Almacenamiento de perfiles sociales
  - Juegos sociales
  - Gestión de contenidos


mongoDB

- Base de datos NoSQL de código abierto orientada a documentos creada por la compañía 10gen en 2007
- Posee varias estrategias de manejo de los datos que la han posicionado donde se encuentra hoy en día:
  - Facilidad para la distribución (réplica) de los datos en múltiples equipos
  - División de documentos de gran tamaño en fragmentos que se almacenan por separado (*sharding*). Al recuperarlo, el propio sistema se encarga de unir automáticamente los documentos



- Los documentos son las unidades básicas de información y se almacenan en formato BSON (*Binary* JSON), que es una versión enriquecida del formato JSON (*JavaScript Object Notation*). Son los equivalentes a las tuplas o filas en el modelo relacional
- Una colección es un conjunto de documentos. Es similar al concepto de TABLA en el modelo relacional

| MongoDB | SQL |
|---|---|
| Base de Datos | Base de Datos |
| Colecciones | Tablas |
| Documentos | Filas (registros) |
| Campos | Columnas |

Comparativa entre SQL y MongoDB
https://docs.mongodb.org/manual/reference/sql-comparison/

- La estructura de almacenamiento es muy flexible

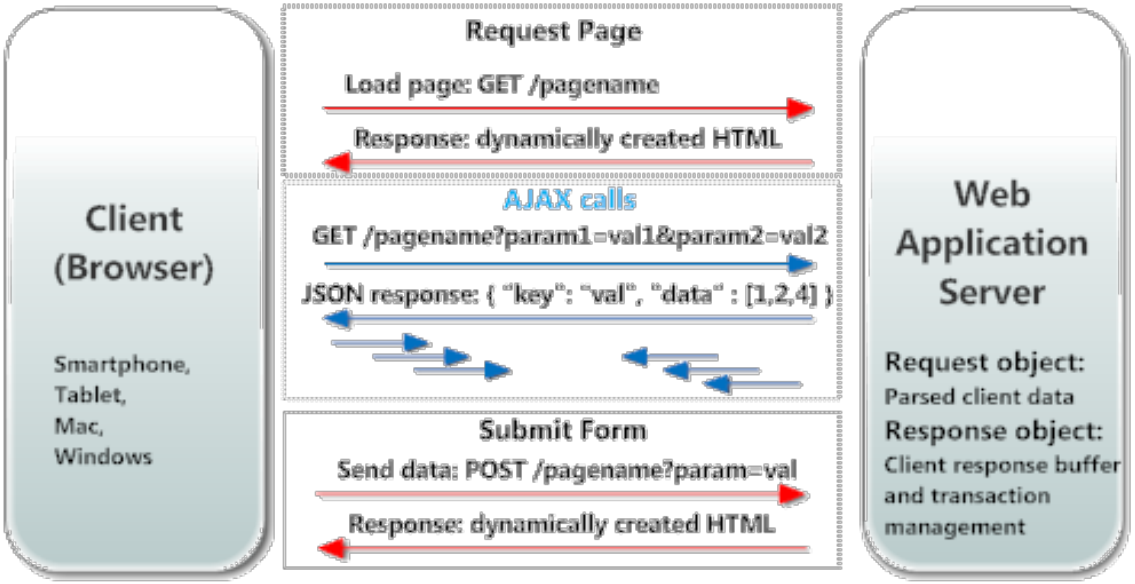
- Los documentos de una misma colección no tienen que tener, obligatoriamente, los mismos campos o estructura

- Dentro de un documento, el valor de un campo puede ser de cualquier tipo admitido por JSON, incluyendo otros documentos, arrays,  arrays de documentos, etc.

```
{
  name: "sue",                          ← field: value
  age: 26,                              ← field: value
  status: "A",                          ← field: value
  groups: [ "news", "sports" ]          ← field: value
}
```

# ¿Qué es? ¿para qué sirve?

- JSON (*JavaScript Object Notation*) es un formato para el intercambio de datos
- Una de las mayores ventajas que tiene el uso de JSON es que puede ser leído e interpretado por cualquier lenguaje de programación
- Se puede usar para el intercambio de información entre distintas tecnologías



# Sintaxis

- Colección desordenada de pares *nombre:valor*
  - Para asignar a un nombre un valor se usa los dos puntos ':'
  - Este separador es el equivalente al símbolo igual ('=') de cualquier lenguaje de programación

- Tipos de valores admitidos en JSON:
  - **número (entero o float)**. {y : 45}, {x : 3,14}
  - **string**. {x : "ejemplo"}
  - **booleano (true o false).** {x : true}
  - **array** (entre corchetes [] ). {x : ["a", "b", "c"]}
  - **objeto** (entre llaves {}). {x : {y : 20} }
  - **null**. {x : null}

# Ejemplo de documento JSON


https://techeplanet.com/json-example/

# ¿Cómo trabajar con MongoDB?

- En su página oficial (https://www.mongodb.com/es) se encuentra toda la información necesaria para trabajar con este SGBD
- Mantiene una documentación muy completa y actualizada

https://docs.mongodb.com/
https://docs.mongodb.com/manual/tutorial/getting-started/

- MongoDB es multiplataforma, así que su instalación dependerá de las características del S.O
- Actualmente (noviembre 2023), la versión estable es la 6.0.12
- Se recomienda instalar la opción *MongoDB Community Server*

https://www.mongodb.com/try/download/community
https://docs.mongodb.com/manual/installation/

# ¿Cómo trabajar con MongoDB?

- Otra opción (y es la que usaremos en este curso) es hacer uso de una versión gratuita en la nube
- MongoDB ofrece Atlas de forma gratuita


https://www.mongodb.com/es/what-is-mongodb


mongoDB®
Atlas

# Conexión una base de datos MongoDB

- Además de disponer de un servidor MongoDB, debemos tener aplicaciones "cliente" para interactuar con él
- La manera más sencilla es a través de su propio shell. Actualmente se llama *mongosh*

Toda la información sobre el shell de MongoDB en https://docs.mongodb.com/mongodb-shell/

- Otra forma de interactuar con un servidor MongoDB es mediante alguna herramienta gráfica. Las más utilizadas son:

  - **MongoDB Compass** (https://docs.mongodb.com/compass/current/)
  - **Studio 3T** (https://studio3t.com/)

# Algunas operaciones útiles del *shell*

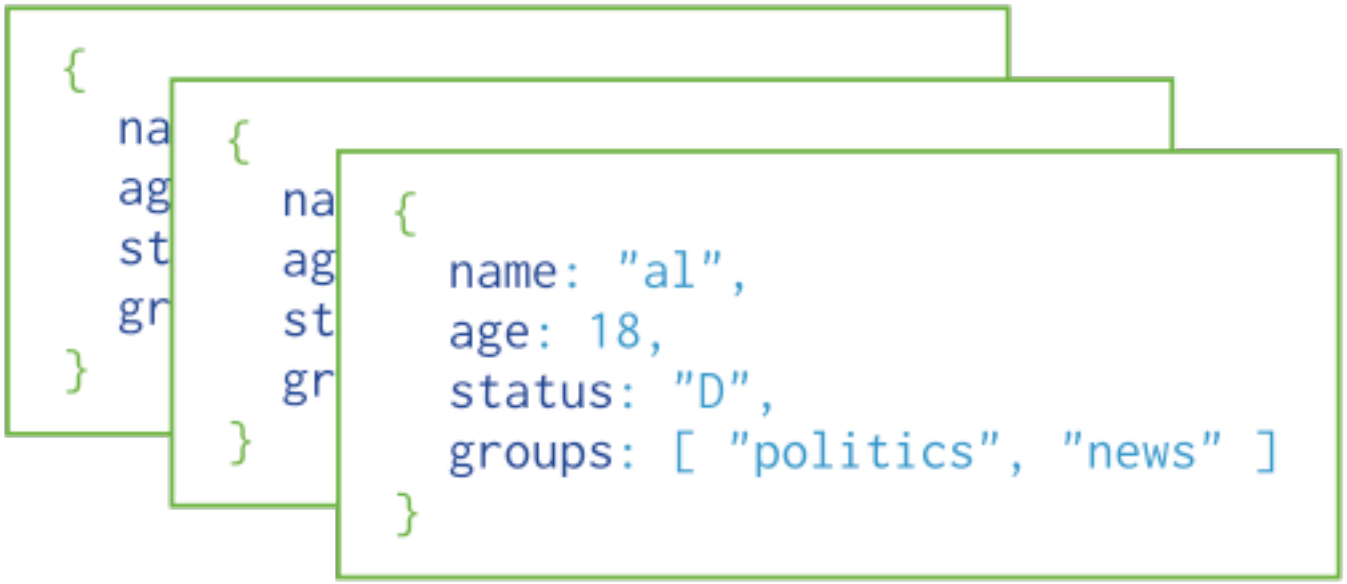
- **use nombreBD.** Cambia de base de datos (también sirve para crear una BD)
- **show dbs.** Muestra las bases de datos que hay en la instancia
- **show collections.** Muestra las colecciones de la BD activa
- **show users.** Muestra los usuarios de la BD activa
- **db.help().** Muestra ayuda de los métodos globales de las bases de datos
- **db.collection.help().** Muestra ayuda de los métodos de las colecciones
- **db.shutdownServer().** Detiene el servidor de MongoDB al que está conectado el shell. Es necesario estar situado en la base de datos **admin** (no lo usaremos en Atlas)

Más información sobre comandos del shell en https://docs.mongodb.com/mongodb-shell/reference/methods/

# Operaciones CRUD

## *Create*, *Read*, *Update* and *Delete*

- Los documentos se almacenan en **colecciones**
- Una colección es un conjunto de documentos relacionados que tienen unos índices en común
- Sería el equivalente al concepto de tabla en el modelo relacional



# Operaciones CRUD

## Operaciones de escritura

- Los métodos db.collection.insertOne() y db.collection.insertMany() añaden nuevos documentos a una colección. Una colección se crea en el momento en el que se ejecuta una sentencia de tipo insert
- La colección se crea en la base de datos en la que estemos situados. Para crear una base de datos, simplemente se utiliza la sentencia use nombreBaseDeDatos
- El método db.createCollection() también crea, de forma explícita, una colección

```
db.users.insertOne(          <-- collection
   {
      name: "sue",           <-- field: value
      age: 26,               <-- field: value
      status: "pending"      <-- field: value
   }                         } document
)
```

```
INSERT INTO users                          <-- table
            ( name, age, status )          <-- columns
VALUES      ( "sue", 26, "A" )             <-- values/row
```

# Operaciones CRUD

## Operaciones de escritura

- Ejemplos de inserción de documentos:

```
db.autores.insertOne ( {
        nombre:   'Andrés',
        apellido:   'Rodríguez',
        secciones:   ['Cocina Fácil' , 'Postres']  } );
```

```
var autorDelPost = {
        nombre: 'Andrés',
        apellido: 'Rodríguez',
        secciones:
          ['Cocina Fácil' , 'Postres']  };

db.autores.insert (autorDelPost);
```

# Operaciones CRUD

## Operaciones de escritura

- El método **insertMany()** inserta varios documentos en una única operación
  - La cláusula **ordered : <boolean>** indica si la inserción de los documentos se realiza de forma ordenada y cómo se actúa en el caso de que falle la inserción de uno de los documentos
    - **True**: se insertan de forma ordenada y si ocurre un fallo en la inserción de un documento, el resto no se insertan
    - **False**: se insertan sin orden y si ocurre un fallo en la inserción de algún documento, se sigue insertando el resto

```
db.autores.insertMany ( [ { nombre: 'Amalia',
      apellido: 'González',
      secciones: ['Entrantes' , 'Cocina Fácil' , 'Peques'],
      administrador:  true  } ,
      { nombre: 'Alberto',
      apellido: 'Ruiz',
      secciones:  'Postres',
      genero:   "M"  } ],

      { ordered : false } );
```

# Operaciones CRUD

## Operaciones de escritura

- Cuando se inserta un nuevo documento, MongoDB añade un campo "_id" y le asigna un valor único (ObjectId), que se utiliza como índice por defecto
- Se puede especificar "manualmente" el valor del campo **_id** cuando se inserta un registro, pero hay que asegurarse de que este valor sea único o se producirá un error de duplicidad

```
db.persona.insertOne({ _id: "P001", nombre: "Pedro", edad: 25 });
db.persona.insertOne({ _id: "P001", nombre: "Cristina", edad: 18 });
WriteError {
      err: {
        index: 0,
        code: 11000,
        errmsg: 'E11000 duplicate key error collection: persona.persona index: _id_ dup key: { _id: "P001" }',
        errInfo: undefined,
        op: { _id: 'P001', nombre: 'Cristina', edad: 18 }
      }
    }
}
```

# Operaciones CRUD

## Operaciones de lectura (consultas)

- El método más genérico para realizar consultas es db.collection.find()
- Acepta criterios de selección y de proyección, y devuelve un cursor con los documentos recuperados

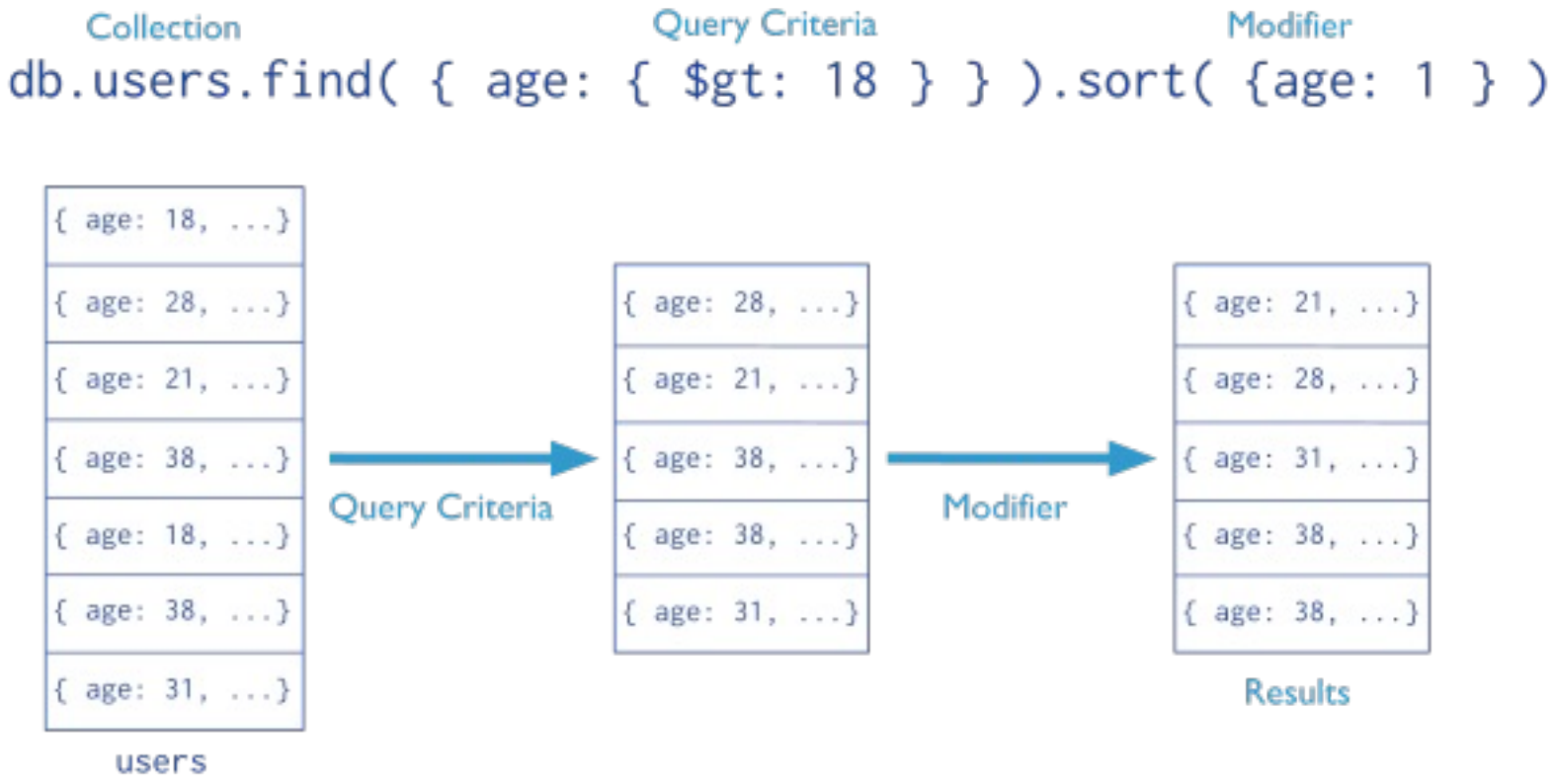
```
db.users.find(                          ← collection
    { age: { $gt: 18 } },               ← query criteria
    { name: 1, address: 1 }             ← projection
).limit(5)                              ← cursor modifier
```

```
SELECT _id, name, address   ← projection
FROM   users                ← table
WHERE  age > 18             ← select criteria
LIMIT  5                    ← cursor modifier
```

# Operaciones CRUD

## Operaciones de escritura

- Criterios de selección:



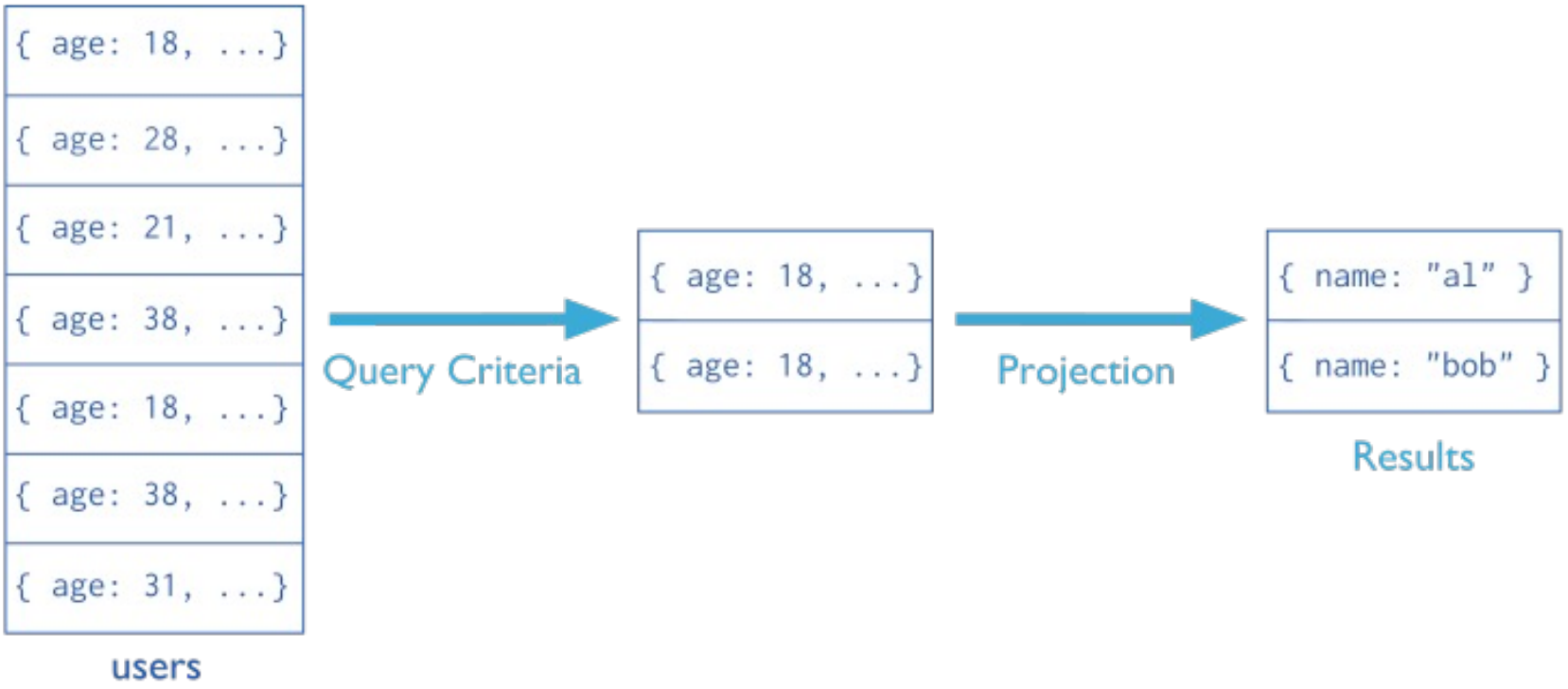
# Operaciones CRUD

## Operaciones de escritura

- Proyección:

Collection Query Criteria Projection

```
db.users.find( { age: 18 }, { name: 1, _id: 0 } )
```



# Operaciones CRUD
## Ejemplos de consultas

- Para practicar las primeras consultas, crearemos la base de datos **etsi** y la colección **estudiantes**
  - Con el comando **insertOne()**, inserta el siguiente estudiante

```
"nombre":"Juan Manuel"
"titulo": "Ingeniería Informática"
"comienzo": 2016
"fin": 2021
```

Para comprobar que se ha insertado correctamente, se puede usar el método de consulta db.estudiantes.find()

  - Con el comando **insertOne()**, inserta el siguiente estudiante

```
"nombre":"Rocío"
"titulo": "Ingeniería Informática"
"comienzo": 2021
```

# Operaciones CRUD

## Ejemplos de consultas

- Vamos a recuperar documentos según diferentes criterios de búsqueda

*En primer lugar, eliminamos la base de datos "etsi" creada anteriormente y creamos una nueva ejecutando el comando que se encuentra en el fichero "estudiantes.txt"*

1. Estudiantes del Grado en "Ingeniería Informática"

```
db.estudiantes.find({"titulo": "Ingeniería Informática"})
```

2. Estudiantes que hayan comenzado sus estudios en 2021

```
db.estudiantes.find({"comienzo": 2021})
```

3. Estudiantes que hayan finalizado sus estudios en 2021

```
db.estudiantes.find({"fin": 2021})
```

# Operaciones CRUD

## Ejemplos de consultas

4. Estudiantes que hayan comenzado sus estudios antes de 2016

```
db.estudiantes.find({"comienzo": {$lt: 2016}})
```

5. Estudiantes que hayan comenzado sus estudios en 2016 o en años anteriores

```
db.estudiantes.find({"comienzo": {$lte: 2016}})
```

6. Estudiantes del Grado en "Ingeniería Informática" que hayan comenzado sus estudios después de 2018

```
db.estudiantes.find({"titulo": "Ingeniería Informática", "comienzo": {$gt: 2018}})
```

7. Estudiantes del Grado en "Ingeniería Mecánica" que hayan comenzado sus estudios entre 2014 y 2020

```
db.estudiantes.find({"titulo": "Ingeniería Mecánica", $and : [{"comienzo": {$gt: 2014}}, {"comienzo": {$lt: 2020}} ] } )
```

# Operaciones CRUD

## Ejemplos de consultas

8. Estudiantes que han concluido sus estudios (parámetro $exists)

```
db.estudiantes.find( { "fin" : { $exists : true } } )
```

9. Todos los estudiantes mostrando, únicamente, su nombre

```
db.estudiantes.find({ }, { _id : 0, "nombre" : 1 } )
```

10. Todos los estudiantes mostrando el nombre y el título

```
db.estudiantes.find({ },{ _id : 0, "nombre": 1, "titulo": 1 } )
```

# Operaciones CRUD

## Ejemplos de consultas

- Resolved estas consultas

11. Estudiantes que hayan empezado sus estudios en 2016 y los hayan terminado en 2021
12. Estudiantes del Grado en "Ingeniería Informática" o del Grado en "Ingeniería Forestal"
13. Estudiantes del Grado en "Ingeniería Mecánica" que no hayan finalizado sus estudios o que los hayan empezado en el año 2015

# Operaciones CRUD

## Ejemplos de consultas

- Una de las características más relevantes en los SGBD NoSQL es la capacidad de almacenar conjuntos de valores. Para ello utiliza el tipo *array*

- Para realizar ejemplos de consultas sobre arrays, desde *Compass* creamos la base de datos Papeleria y la colección productos

- Con la herramienta "import file" de *Compass*, introducimos los 5 documentos en la colección, importando el fichero *productos.json* de la página web

# Operaciones CRUD

## Ejemplos de consultas

1. Productos que sean de color rojo

```
db.productos.find( { colores : "rojo" } )
```

2. La consulta anterior devuelve todos los productos que son de color rojo pero que también pueden ser de otros colores. Si queremos encontrar los productos que solo son de color rojo

```
db.productos.find( { colores : ["rojo"] } )
```

3. Productos que son de color rojo y blanco

```
db.productos.find( { colores : ["rojo", "blanco"] } )
```

4. La consulta anterior solo devuelve los productos que son de color rojo y blanco pero cuyo array de colores esté escrito en ese orden. Para obtener los productos de color rojo y blanco independientemente del orden, hay que usar el operador $all

```
db.productos.find( { colores : { $all : ["rojo", "blanco"] } } )
```

# Operaciones CRUD

## Ejemplos de consultas

5. Productos que tengan, al menos, una dimensión mayor que 26

```
db.productos.find( { dim_cm : { $gt : 26 } } )
```

6. Productos cuyas dimensiones sean o menor que 12 o mayor que 14

```
db.productos.find( { dim_cm : { $lt: 12, $gt: 14 } } )
```

Devuelve el documento "_id: 2" ya que es el único donde todos los elementos del array "dim_cm": [ 10, 15.25 ] cumplen algunas de las condiciones. En este caso, la dimensión "10" cumple una de las condiciones (10 < 12), y la dimensión "15.25" cumple otra de las condiciones (15.25 > 14)

Los documentos con "dim_cm": [ 14, 21 ] no se devuelven porque la dimensión "14" no cumple ninguna de las condiciones

7. Productos que tengan, al menos, una de sus dimensiones mayor que 20 y menor que 22

```
db.productos.find( { dim_cm : { $elemMatch : { $gt: 20, $lt: 22 } } } )
```

El propósito del operador $elemMatch es buscar los documentos en los que, al menos, uno de los valores del array cumpla la condición

# Operaciones CRUD

## Ejemplos de consultas

8. Productos cuya primera dimensión sea mayor que 13

```
db.productos.find( { "dim_cm.0" : { $gt: 13 } } )
```

9. Productos que tengan 3 colores

```
db.productos.find( { "colores" : { $size: 3 } } )
```

10. Productos que solo tengan 1 color

```
db.productos.find( { colores : { $size: 1 } } )
```

11. Productos cuya segunda dimensión sea 21

```
db.productos.find( { "dim_cm.1" : 21 } )
```

12. Productos que tengan, al menos, colores rojo y negro

```
db.productos.find( { colores : { $all : ["rojo", "negro"] } } )
```

# Operaciones CRUD

## Ejemplos de consultas

13. Productos cuyas dimensiones sean exactamente 10 x 15.25

```
db.productos.find( { dim_cm : [10, 15.25] } )
```

14. Productos de color rojo de los que haya más de 60 unidades

```
db.productos.find( { cantidad : { $gt : 60}, colores : "rojo" } )
```

15. Productos de color blanco o de los que haya más de 60 unidades

```
db.productos.find( { $or : [ {cantidad : { $gt : 60 } }, { colores : "blanco" } ] } )
```

16. Productos cuya segunda dimensión esté entre 10 y 18

```
db.productos.find( { "dim_cm.1" : { $gt : 10, $lt : 18 } } )
```

# Operaciones CRUD

## Ejemplos de consultas

- Además de los arrays, los SGBD NoSQL también pueden almacenar documentos completos dentro de un campo
- Este campo se suele denominar "**documento embebido**" (del inglés ***embedded document***), "**documento incrustado**" o "**documento anidado**"
- Dentro de la base de datos **Papeleria**, vamos a crear la colección **productos2** para estudiar algunas consultas sobre este tipo de estructura. Importamos el fichero ***productos2.json*** de la página web


```
{ "_id": 1, "item": "cuaderno", "cantidad": 25, "tam": { "h": 14, "w": 21, "unidad": "cm" }, "estado": "A" },
{ "_id": 2, "item": "tarjeta", "cantidad": 45, "tam": { "h": 10, "w": 15.25, "unidad": "cm" }, "estado": "A" },
{ "_id": 3, "item": "agenda", "cantidad": 75, "tam": { "h": 22.85, "w": 30, "unidad": "cm" }, "estado": "D" },
{ "_id": 4, "item": "bloc", "cantidad": 50, "tam": { "h": 8.5, "w": 11, "unidad": "in" }, "estado": "A" },
{ "_id": 5, "item": "cartulina", "cantidad": 100, "tam": { "h": 8.5, "w": 11, "unidad": "in" }, "estado": "D" }
```

# Operaciones CRUD

## Ejemplos de consultas

- Para especificar una condición de igualdad en un campo de tipo documento embebido, se especifica el filtro { <campo>: <valor> } donde <valor> es el documento embebido que debe coincidir

```
db.productos2.find( { tam: { h: 14, w: 21, unidad: "cm" } } )
```

Devuelve los documentos cuyo campo "tam" sea, exactamente, { h: 14, w: 21, unidad: "cm" }

- Para especificar una condición de consulta en este tipo de campos se utiliza la notación "punto"

```
db.productos2.find( { "tam.unidad": "in" } )
```

Devuelve los documentos cuyo campo "unidad" del campo "tam" esté expresado en pulgadas ("in")

# Operaciones CRUD

## Ejemplos de consultas

- Se pueden utilizar operadores para realizar filtros en las consultas

```
db.productos2.find( { "tam.h": { $lt: 15 } } )
```

Devuelve los documentos cuyo campo "h" del campo "tam" sea menor que 15

- Se pueden combinar condiciones entre campos de distintos niveles de anidamiento

```
db.productos2.find( { "tam.h": { $lt: 15 }, "tam.unidad": "in", estado: "A" } )
```

Devuelve los documentos cuyo campo anidado "h" sea menor que 15, el campo anidado "unidad" sea "in", y el campo "estado" sea "A"

# Operaciones CRUD
## Ejemplos de consultas

- Y, por último, veamos la estructura de **array de documentos**

- Dentro de la base de datos **Papeleria**, vamos a crear la colección **productos3** para estudiar algunas consultas sobre este tipo de estructura. Importamos el fichero *productos3.json* de la página web


```
{ "_id": 1, "item": "cuaderno", "stock": [ { "almacen": "A", "cantidad": 5 }, { "almacen": "C", "cantidad": 15 } ] },
{ "_id": 2, "item": "tarjeta", "stock": [ { "almacen": "C", "cantidad": 5 } ] },
{ "_id": 3, "item": "agenda", "stock": [ { "almacen": "A", "cantidad": 60 }, { "almacen": "B", "cantidad": 15 } ] },
{ "_id": 4, "item": "bloc", "stock": [ { "almacen": "A", "cantidad": 40 }, { "almacen": "B", "cantidad": 5 } ] },
{ "_id": 5, "item": "cartulina", "stock": [ { "almacen": "B", "cantidad": 15 }, { "almacen": "C", "cantidad": 35 } ] }
```

# Operaciones CRUD

## Ejemplos de consultas

```
db.productos3.find( { "stock": { almacen: "A", cantidad: 5 } } )
```

Devuelve los documentos que tengan, exactamente, un documento { almacen: "A", cantidad: 5 } en el array "stock"

```
db.productos3.find( { 'stock.cantidad': { $lte: 20 } } )
```

Devuelve los documentos en cuyo campo "stock" hay, al menos, un documento que contiene un campo "cantidad" menor o igual a 20

```
db.productos3.find( { 'stock.0.cantidad': { $lte: 20 } } )
```

Devuelve los documentos cuyo campo "stock" tiene, como primer elemento, un documento que contiene un campo "cantidad" menor o igual a 20

# Operaciones CRUD

## Ejemplos de consultas

- Cuando se especifican condiciones en más de un campo anidado de un array de documentos, se puede especificar la consulta de forma que un único documento cumpla las condiciones
- Para realizar este tipo de consultas se utiliza el operador **$elemMatch**

```
db.productos3.find( { "stock": { $elemMatch: { cantidad: 5, almacen: "A" } } } )
```

Devuelve los documentos en cuyo campo "stock" hay, al menos, un documento que tiene un campo "cantidad" igual a 5 y un campo "almacen" igual a "A"

```
db.productos3.find( { "stock": { $elemMatch: { cantidad: { $gt: 10, $lte: 20 } } } } )
```

Devuelve los documentos en cuyo campo "stock" hay, al menos, un documento que tiene un campo "cantidad" mayor que 10 y menor o igual que 20

# Operaciones CRUD

## Ejemplos de consultas

- Si las condiciones de la consulta compuesta **sobre un campo de un documento anidado del array** no utilizan el operador **$elemMatch**, la consulta selecciona aquellos documentos cuyo array contenga cualquier combinación de elementos que satisfaga las condiciones

```
db.productos3.find( { "stock.cantidad": { $gt: 10,  $lte: 11 } } )
```

Devuelve los documentos en cuyo campo "stock" hay algún documento con el campo "cantidad" mayor que 10 y algún documento (pero no necesariamente el mismo) con el campo "cantidad" menor o igual que 11

```
db.productos3.find( { "stock.cantidad": 5, "stock.almacen": "A" } )
```

Devuelve los documentos en cuyo campo "stock" hay algún documento con el campo "cantidad" igual a 5 y algún documento (pero no necesariamente el mismo) con el campo "almacen" igual a "A"

# Operaciones CRUD

## Ejemplos de consultas

- También se pueden hacer proyecciones sobre campos anidados

```
db.productos3.find( { _id: { $lt: 4 } }, { _id : 0, item: 1, "stock.almacen" : 1 } )
```

Muestra el campo "item" y, del campo "stock", sólo muestra el almacén de los documentos con "_id" menor que 4

```
db.productos3.find( { _id: { $lt: 4 } }, { _id : 0, "stock.almacen" : 0 } )
```

Muestra todos los campos menos el campo "almacen" del array "stock", de los documentos con "_id" menor que 4

# Ejercicios

- Realizar las siguientes operaciones CRUD sobre la colección "productos2"

1. Productos con tamaño en pulgadas, altura menor a 15 y estado "A" o "D"

```
db.productos2.find({
  "tam.unidad": "in", "tam.h": { $lt: 15 },
  $or: [  { "estado": "A" },  { "estado": "D" }  ] } )
```

2. Productos con tamaño en centímetros y cantidad mayor a 20, o tamaño en pulgadas y cantidad menor o igual a 50

```
db.productos2.find({
  $or: [
    { "tam.unidad": "cm", "cantidad": { $gt: 20 } },
    { "tam.unidad": "in", "cantidad": { $lte: 50 } }
  ]
})
```

# Ejercicios

3. Productos con cantidad mayor a 30 y tamaño en centímetros, proyectando sólo los campos "item" y "cantidad"

```
db.productos2.find({ "cantidad": { $gt: 30 }, "tam.unidad": "cm" }, { "_id": 0, "item": 1, "cantidad": 1 })
```

4. Productos con tamaño en centímetros y estado "A", ordenados por cantidad descendentemente

```
db.productos2.find({ "tam.unidad": "cm", "estado": "A" }).sort({ "cantidad": -1 })
```

5. Productos con estado "A" y cantidad mayor a 30, proyectando sólo los campos "item" y "tam", y ordenados por tamaño de ancho descendentemente

```
db.productos2.find( { "estado": "A", "cantidad": { $gt: 30 } },
  { "_id": 0, "item": 1, "tam": 1 } ).sort({ "tam.w": -1 })
```

# Ejercicios

- Realizar las siguientes operaciones CRUD sobre la colección "productos3"

1. Productos con stock en el almacén "A" y cantidad igual a 40 en ese almacén

```
db.productos3.find( { "stock": { $elemMatch: { "almacen": "A", "cantidad": 40 } } } )
```

2. Documentos que tienen al menos un almacén con una cantidad inferior a 10 para cualquier producto

```
db.productos3.find( { "stock": { $elemMatch: { "cantidad": { $lt: 10 } } } } )
```

3. Documentos que tienen stock en el almacén "C" con una cantidad mayor a 10 para cualquier producto

```
db.productos3.find( { "stock": { $elemMatch: { "almacen": "C", "cantidad": { $gt: 10 } } } } )
```

4. Documentos que tienen stock en los almacenes "A" y "B" para cualquier producto

```
db.productos3.find( {
  $and: [ { "stock": { $elemMatch: { "almacen": "A"} } }, { "stock": { $elemMatch: { "almacen": "B" } } }  ] } )
```

# Operaciones CRUD

## Operaciones de actualización

- Los métodos db.collection.updateOne(), db.collection.updateMany() y db.collection.replaceOne() se utilizan para actualizar documentos

- Se pueden especificar criterios o filtros para identificar los documentos que se quieren actualizar. Estos filtros utilizan la misma sintaxis que las operaciones de lectura

```
db.users.updateMany(                                  <-- collection
   { age: { $lt: 18 } },                              <-- update filter
   { $set: { status: "reject" } }                     <-- update action
)
```

# Operaciones CRUD

## Operaciones de actualización

```
db.coleccion.updateMany (
   <filtro>,
   <actualización>,
   {
      upsert: booleano,
      arrayFilters: [ <filterdocument1>, ... ],
   }
);
```

- **filtro:** la condición de búsqueda del/los documento/s que se van a actualizar
- **actualización:** se especifican los cambios a realizar.

- upsert: (opcional, *false* por defecto). Si es **true** y el filtro no encuentra ningún resultado, se inserta un nuevo documento con la <actualización>

- arrayFilters: (opcional). Se utiliza para especificar qué elementos de un array deben ser actualizados

# Operaciones CRUD

## Operaciones de actualización

- Ejemplo

```
db.autores.insertOne ( {
    nombre:  'Ricardo',
    apellido:   'Sanc'
} );
```

```
db.autores.replaceOne (
    { nombre: 'Ricardo' },
    {
        nombre: 'Ricardo',
        apellido: 'Sánchez',
        secciones: ['Peques', 'Postres'],
        administrador:  true
    }
);
```

- En este caso se ha modificado el documento completo
- La operación de sobrescribir un documento no modifica su identificador único **"_id"**

# Operaciones CRUD

## Operaciones de actualización

- Operadores de actualización
  - Para modificar campos específicos debemos utilizar los operadores de modificación. Algunos de ellos son:
    - **$inc** - incrementa una cantidad en un campo de tipo numérico
    - **$rename** - renombra campos del documento
    - **$set** - permite especificar los campos que van a ser modificados
    - **$unset** - elimina campos del documento

Consultar todos los operadores de actualización en:
https://docs.mongodb.org/manual/reference/operator/update/#id1

# Operaciones CRUD

## Operaciones de actualización

- Ejemplo

```
{
  _id: 1,
  item: "TBD",
  stock: 0,
  info: { publisher: "1111", pages: 430 },
  tags: [ "technology", "computer" ],
  ratings: [ { by: "ijk", rating: 4 }, { by: "lmn", rating: 5 } ],
  reorder: false
}
```

```
db.books.updateOne (
  { _id: 1 },
  {
    $inc: { stock: 5 },
    $set: {
      item: "ABC123",
      "info.publisher": "2222",
      tags: [ "software" ],
      "ratings.1": { by: "xyz", rating: 3 }
    }
  } )
```

```
{
  _id: 1,
  item: 'ABC123',
  stock: 5,
  info: { publisher: '2222', pages: 430 },
  tags: [ 'software' ],
  ratings: [ { by: 'ijk', rating: 4 }, { by: 'xyz', rating: 3 } ],
  reorder: false
}
```

# Operaciones CRUD

## Operaciones de actualización

- Ejemplos

- Elimina el campo "tags"

```
db.books.updateOne ( { _id: 1 }, { $unset: { tags: 1 } } )
```

- Inserta un nuevo documento si no existe ninguno que cumpla la condición

```
db.books.replaceOne (
  { item: "ZZZ135" },
  {
    item: "ZZZ135",
    stock: 20,
    tags: [ "database" ]
  },
  { upsert: true }
)
```

```
{
  "_id" : ObjectId("542310906694ce357ad2a1a9"),
  "item" : "ZZZ135",
  "stock" : 20,
  "tags" : [ "database" ]
}
```

- Actualiza todos los documentos cuyo campo "stock" es menor o igual que 10. Si el campo "discount" no existe en algún documento, se añade con el valor "true"

```
db.books.updateMany (
  { stock: { $lte: 10 } },
  { $set: { discount: true } }
)
```

# Operaciones CRUD

## Operaciones de actualización

- Operadores de actualización de arrays
  - $pop - elimina el primer o último valor de un array
  - $pull - elimina los valores de un array que cumplan el filtro
  - $pullAll - elimina los valores específicos de un array
  - $push - añade un elemento a un array
  - $addToSet - añade elementos a un array solo si estos no existen (asegura que no se añaden elementos duplicados)
  - $each - se usa junto con **$addToSet** o **$push** para especificar que se van a añadir varios elementos al array

# Operaciones CRUD

## Operaciones de actualización

- Ejemplos

```
{ _id: 1, scores: [ 8, 9, 10 ] }
```

```
db.students.updateOne( { _id: 1 }, { $pop: { scores: -1 } } )
```

```
{ _id: 1, scores: [ 9, 10 ] }
```

```
{ _id: 2, scores: [ 0, 2, 5, 5, 1, 0 ] }
```

```
db.students.updateOne( { _id: 2 }, { $pullAll: { scores: [ 0, 5 ] } } )
```

```
{ "_id" : 2, "scores" : [ 2, 1 ] }
```

```
db.students.updateOne (
  { _id: 1 },
  { $push: { scores: 7.5 } }
)
```

```
{ "_id" : 1, "scores" : [ 9, 10, 7.5] }
```

```
db.students.updateOne (
  { _id: 2 },
  { $push: { scores: { $each: [ 9, 7.5, 8.5 ] } } }
)
```

```
{ "_id" : 2, "scores" : [ 2, 1, 9, 7.5, 8.5 ] }
```

# Operaciones CRUD

## Operaciones de actualización

- Ejemplos

```
{ "_id" : 1, "scores" : [ 9, 10, 7.5] }
{ "_id" : 2,  "scores" : [ 2, 1, 9, 7.5, 8.5 ] }
```

```
db.students.updateMany(
  { scores: { $gte: 8.5 } },
  { $set: { "scores.$[elemento]" : 10 } },
  { arrayFilters: [ { "elemento": { $gte: 8.5 } } ] }
)
```

```
{ "_id" : 1, "scores" : [ 10, 10, 7.5] }
{ "_id" : 2,  "scores" : [ 2, 1, 10, 7.5, 10 ] }
```

Esto es para buscar en arrays donde haya elementos que cumplen esta condición. Para buscar en todos, basta con poner { }

# Operaciones CRUD

## Operaciones de actualización

- Ejemplos

```
{ "_id": 1,
  "alias": [ "The American Cincinnatus", "The American Fabius" ],
  "mobile": "555-555-5555",
  "nmae": { "first" : "george", "last" : "washington" }
}
```

```
db.students.updateOne( { _id: 1 }, { $rename: { "nmae": "name" } } )
```

```
db.students.updateOne ( { _id: 1 }, { $rename: { "name.first": "name.fname" } } )
```

```
{
  "_id" : 1,
  "alias" : [ "The American Cincinnatus", "The American Fabius" ],
  "mobile" : "555-555-5555",
  "name" : { "fname" : "george", "last" : "washington" }
}
```

# Operaciones CRUD

## Operaciones de borrado

- Los métodos para eliminar documentos son db.collection.deleteMany() y db.collection.deleteOne()

- Acepta filtros para seleccionar los documentos a eliminar

```
db.users.deleteMany(
   { status: "reject" }
)
```

collection

delete filter

- Consultar toda la información referente al borrado de documentos en:

https://www.mongodb.com/docs/manual/tutorial/remove-documents/

# Operaciones CRUD

## Operaciones de borrado

- Ejemplos

- Elimina todos los documentos de la colección "productos"

```
db.productos.deleteMany ( { } )
```

- Elimina todos los documentos de la colección "productos" cuyo campo "cantidad" es mayor que 20

```
db.productos.deleteMany( { cantidad: { $gt: 20 } } )
```

# Ejercicios

- Realizar las siguientes operaciones CRUD sobre la colección "productos":

1. Añadir el color "naranja" al producto "cuaderno"

```
db.productos.updateOne( { item : "cuaderno" }, { $push : { colores: 'naranja' } } )
```

2. Añadir los colores "magenta" y "verde" al producto "bloc"

```
db.productos.updateOne( { item : "bloc" }, { $push : { colores : { $each : ['magenta', 'verde'] } } } )
```

3. Eliminar el color "blanco" al producto "cartulina"

```
db.productos.updateOne( { item : "cartulina" }, { $pull : { colores : "blanco" } } )
```

# Ejercicios

4. Aumentar en 10 las cantidades de los productos que tengan más de 50 unidades

```
db.productos.updateMany(  { cantidad : { $gt : 50 } }, { $inc : { cantidad : 10 } } )
```

5. Añadir al producto "cuaderno" un nuevo campo de tipo array que se llame "precios" y asignarle los valores 30 y 60

```
db.productos.updateOne( { item : "cuaderno" }, { $set : { precios : [30, 60] } } )
```

6. Aumentar en 3 unidades la primera dimensión del producto "tarjeta"

```
db.productos.updateOne( { item : "tarjeta" }, { $inc : { "dim_cm.0" : 3 } } )
```

# Ejercicios

- Realizar las siguientes operaciones CRUD sobre la colección "productos2":

1. Cambiar el estado del producto "tarjeta" al valor "C"

```
db.productos2.updateOne( { item : 'tarjeta' }, { $set : { estado :'C' } } )
```

2. Cambiar las dimensiones de producto "bloc" a (10, 14)

```
db.productos2.updateOne( { item : 'bloc' }, { $set : { "tam.h" : 10, "tam.w": 14 } } )
```

3. Pasar a "cm" todos los productos "in"

```
db.productos2.updateMany( { "tam.unidad" : 'in' }, { $set : { "tam.unidad" : 'cm' } } )
```

# Ejercicios

4. Añadir a todos los productos un nuevo campo "forma" dentro del campo "tam" con el valor "rectángulo"

```
db.productos2.updateMany( {  }, { $set : { "tam.forma" : "rectángulo"} } )
```

5. Eliminar los productos con estado "A"

```
db.productos2.deleteMany( { estado : 'A' } )
```

# Ejercicios

- Realizar las siguientes operaciones CRUD sobre la colección "productos3":

1. Añadir un nuevo almacén "A" al stock del producto "tarjeta" con una cantidad de 40

```
db.productos3.updateOne( { item : 'tarjeta' }, { $push : { stock : { almacen : 'A', cantidad : 40 } } } )
```

2. Aumentar en 100 unidades la cantidad del stock de todos los almacenes de tipo "C". NOTA: para resolver esta consulta hay que usar el operador *arrayFilters*

```
db.productos3.updateMany(
  { },
  { $inc : { "stock.$[elemento].cantidad" : 100 } },
  { arrayFilters: [ { "elemento.almacen": "C" } ] }
)
```

# Ejercicios

- Crear una base de datos "Agencia" y una colección "viajeros" con estos datos

```
db.viajeros.insertMany ( [
{ _id: 1, first : "Maria",
   travel : [
    { country: "Canada", visits: 3, rating: 7 },
    { country: "Poland", visits: 1, rating: 8 },
    { country: "Thailand", visits: 2, rating: 9 } ] },
{ _id: 2, first : "Chen",
   travel : [
    { country: "Thailand", visits: 3, rating: 7 },
    { country: "Canada", visits: 2, rating: 9 },
    { country: "Costa Rica", visits: 4, rating: 8 } ] },
 { _id: 3, first : "Gladys",
   travel : [
    { country: "Canada", visits: 1, rating: 8 },
    { country: "Thailand", visits: 2, rating: 9 },
    { country: "Australia", visits: 3, rating: 10 } ] }
 ] );
```

# Ejercicios

- Realizar las siguientes operaciones CRUD:
  1. Obtener el nombre de los viajeros que han visitado 2 veces Canadá y lo han valorado con un 9
  2. Obtener los viajeros que han realizado 4 visitas a un país
  3. Obtener los viajeros que han realizado más de 3 visitas a un país
  4. Obtener los viajeros que han realizado más de 2 visitas a un país y han puntuado con menos de un 8 (pueden ser distintos países)
  5. Obtener los viajeros que han realizado más de 2 visitas a un país y han puntuado con menos de un 8 pero del mismo país ($elemMatch)
  6. Igual que la anterior pero ahora la puntuación debe estar entre 6 y 9
  7. Actualizar el número de visitas de María a Canadá con 5 visitas
  8. Actualizar los viajes de Chen añadiendo una visita a España y con una puntuación de 10