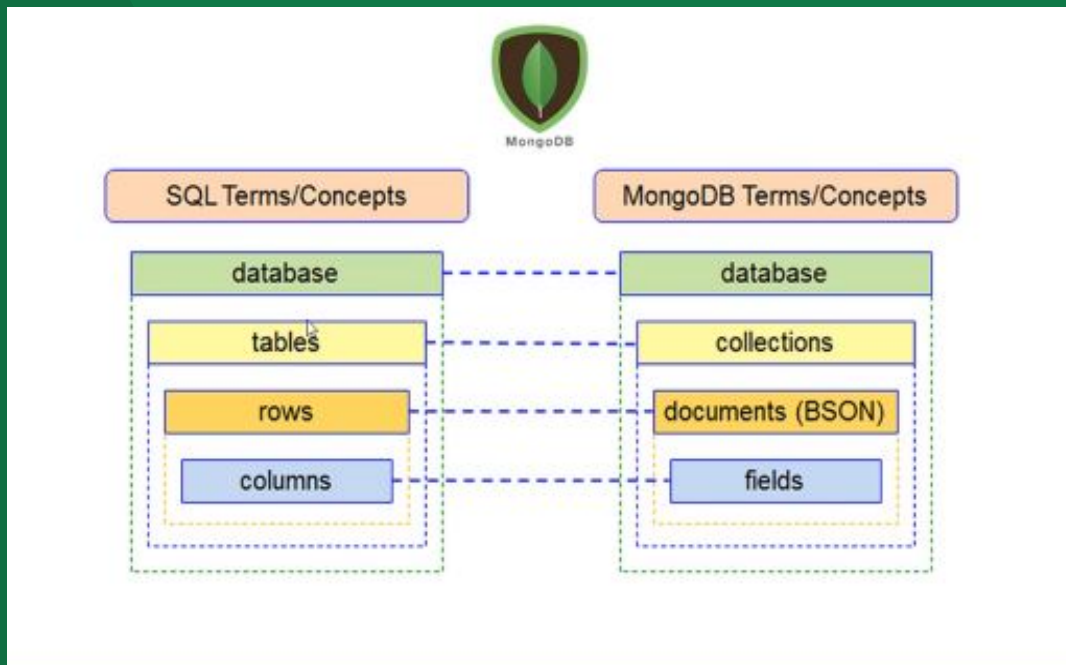


MONGO DB

MySQL → MongoDB



El concepto de *database* en ambos modelos es el mismo.

En MongoDB se le llama *collection* a la agrupación de *documents* (BSON — binary JSON) que se almacenan en una database.

Por lo tanto, las operaciones de lectura y escritura y las consultas que haremos serán sobre una database y sus collections,

SQL → NoSQL

```
> use GenteDB
```

```
> db.createCollection("usuarios")
```

o

```
> db.usuarios.insertOne(...)
```

¿¿¿¿¿
nos suena de
algo
?????

MongoDB

MongoDB es una base de datos libre de esquemas, orientada a documentos, escrita en C ++. La base de datos está basada en el almacén de documentos, lo que significa que almacena valores (denominados documentos) en forma de datos codificados.

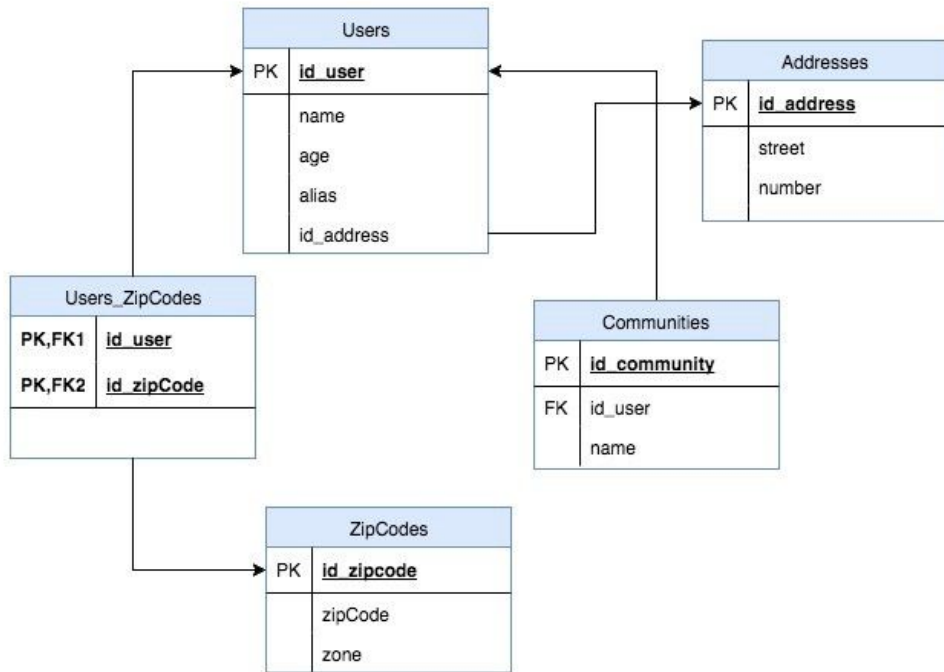
La elección del formato codificado en MongoDB es JSON. Es muy potente, porque incluso si los datos están anidados dentro de los documentos JSON, seguirá siendo consultable e indexable.

MongoDB

MongoDB tiene un sistema flexible de almacenamiento de esquemas. Lo que significa que los objetos almacenados no tienen que tener la misma estructura o los mismos campos.

MongoDB también tiene algunas características de optimización, que distribuye las colecciones de datos, mejorando el rendimiento y consiguiendo un sistema más equilibrado.

SQL → NoSQL



Users <collection>

```
{
  name : "Marcela Sena",
  age  : 29,
  alias : "MarceStarlet",
  memberOf : ["TechWo", "JavaGDL"],
  address : { street : "Main Boulevard", number : 234 },
  zipCodes : [
    { zipCode : 1234, zone: "Guadalajara" },
    { zipCode : 4321, zone: "arajaladaug" }
  ]
}
```

Estructura...

Las ventajas de tener este tipo de estructura de datos como registros es que principalmente podemos reducir las “relaciones” entre tablas que comúnmente usamos en bases de datos relacionales cuando hacemos consultas con los famosos “joins” o uniones de datos, ya que resulta muy costoso.

Otra ventaja es que de cierta forma esta estructura se corresponde con tipos de datos en varios lenguajes de programación lo cual resulta conveniente para desarrollar código de forma más ágil.

Además, también nos permite guardar grandes volúmenes de datos sin tanto coste.

MongoDB JSON

Todos los datos que se guardan en MongoDB, son almacenados en documentos JSON.

JSON es un formato estándar para datos que destaca por ser ligero y rápido (por tanto muy útil para desarrollos web).

Los datos en formato JSON pueden ser utilizados por prácticamente todos los lenguajes de programación (como Java, C#, C, C++, PHP, JavaScript, Python, etc.)

MongoDB JSON

Los archivos JSON son simples archivos de texto con extensión .json y se puede crear con cualquier editor de texto plano.

Los documentos JSON se componen de dos estructuras básicas: **arrays** y **diccionarios**.

Los arrays son un conjunto de valores que se encuentran encerrados entre corchetes [] y los diccionarios son mapas asociativos representados por clave:valor que se encuentran encerrados entre llaves { }

MongoDB JSON

Cada una de estas estructuras pueden contener a la otra o a sí misma.

El concepto para los diccionarios de clave:valor se refiere a que podemos insertar valores con algún identificador para reconocerlo más fácilmente, como por ejemplo si tuviéramos un diccionario con datos personales sería algo como esto:

```
{"nombre" : "Jason", "edad" : 20}
```

Un diccionario puede tener varios valores, éstos deben separarse por una coma, además los strings siempre deberían estar entre comillas,

MongoDB JSON

Un diccionario puede contener otros diccionarios y arrays dentro.

Cómo ingresar datos dentro de un array y de otro diccionario:

```
{  
  "nombre" : "Jason",  
  "edad" : 20,  
  "email" : {  
    "email personal" : "personal@email.com",  
    "email corporativo" : "corporativo@email.com"  
  },  
  "frutas favoritas" : [pera, manzana, ciruela]  
}
```



más fácil organizar varios datos de una persona en un mismo documento, haciendo (caso de MongoDB) que no se necesiten de otras tablas adicionales para ingresar varios datos personales a una persona, como lo son en este caso las email y frutas favoritas.

MongoDB JSON.- ejemplo de array de diccionarios

```
{  
  "marcadores": [  
    {  
      "latitude": 40.416875,  
      "longitude": -3.703308,  
      "city": "Madrid",  
      "description": "Puerta del Sol"  
    },
```

```
    {  
      "latitude": 40.417438,  
      "longitude": -3.693363,  
      "city": "Madrid",  
      "description": "Paseo del Prado"  
    },  
  ]  
}
```

MongoDB JSON.- ejemplo de conjunto de valores directo

```
{  
  "título": "Doctor",  
  "Nombre": "Pedro",  
  "Edad": 37,  
  "City": "Madrid"  
  "facebook": "https://fb.com"  
}
```

MongoDB - Características

Shards / Fragmentos

Sharding es la partición y distribución de datos a través de múltiples máquinas (nodos). Un fragmento, es una colección de nodos MongoDB.

En MongoDB es posible escalar horizontalmente agregando clusters de máquinas donde los datos se distribuyen horizontalmente a través de múltiples nodos. En el caso de que haya una aplicación que utilice un único servidor de base de datos, se puede convertir en clúster fragmentado, con muy pocos cambios en el código de la aplicación original.

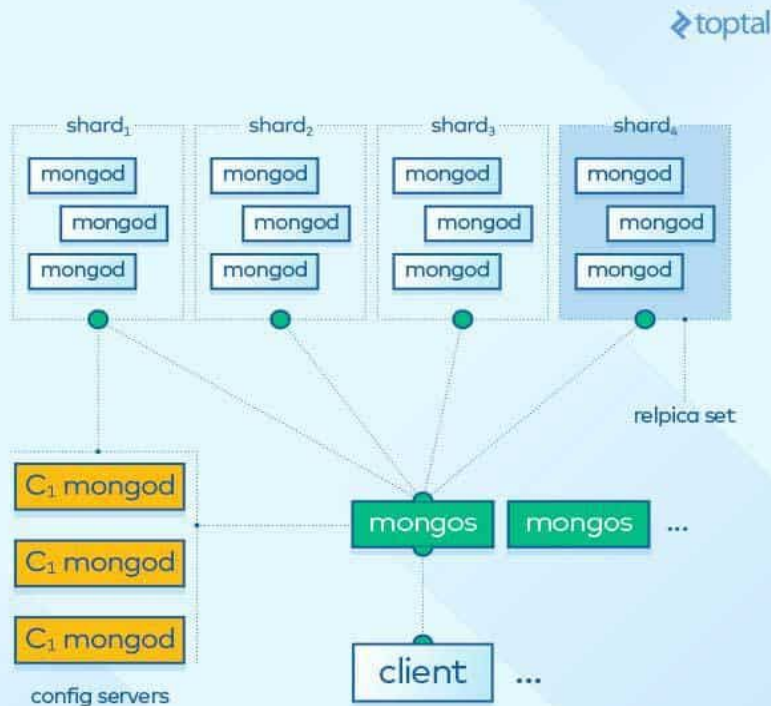
MongoDB - Características

Lenguaje de consulta Mongo

Como se mencionó anteriormente, MongoDB utiliza una API RESTful. Para recuperar ciertos documentos de una colección db, se crea un documento de consulta que contiene los campos que deben coincidir con los documentos deseados. JavaScript.

MongoDB soporta un lenguaje de consultas enriquecido para hacer operaciones de escritura y lectura (CRUD)

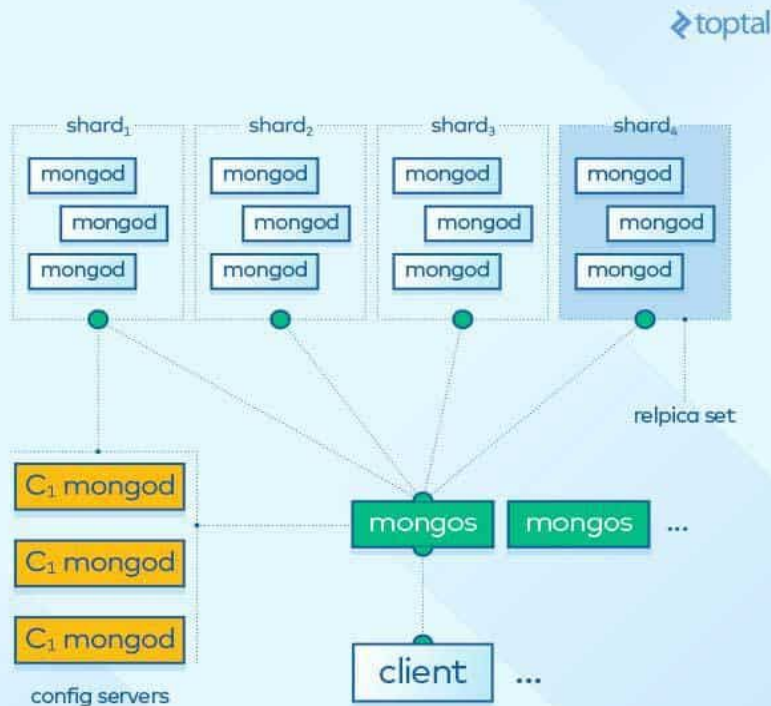
Arquitectura MongoDB



En MongoDB, hay un grupo de servidores llamados enrutadores. Cada uno actúa como un servidor para uno o más clientes. Del mismo modo, el clúster contiene un grupo de servidores denominados servidores de configuración. Cada uno contiene una copia de los metadatos que indican qué fragmento contiene qué datos. Las acciones de lectura o escritura se envían desde los clientes a uno de los servidores de enrutador del clúster y son encaminadas automáticamente por ese servidor, a los fragmentos adecuados que contienen los datos con la ayuda de los servidores de configuración.

servidores del enrutador en verde, los servidores de configuración en amarillo y los fragmentos que contienen los nodos MongoDB en azul.

Arquitectura MongoDB



Cabe señalar que el sharding (o compartir los datos entre fragmentos) en MongoDB es completamente automático, lo que reduce la tasa de fallos y hace MongoDB un sistema de gestión de base de datos altamente escalable.

Escalabilidad .- Propiedad de aumentar la capacidad de trabajo o de tamaño de un sistema sin comprometer el funcionamiento y calidad normales del mismo.

Documentos

En MongoDB un documento es un registro compuesto por pares “campo : valor”.

```
{  
  name : "Marcela Sena",  
  age  : 29,  
  alias : "MarceStarlet",  
  memberOf : ["TechWo", "JavaGDL"],  
  address : { street : "Main Boulevard", number : 234 },
```

```
  zipCodes : [  
    { zipCode : 1234, zone: "Guadalajara" },  
    { zipCode : 4321, zone: "arajaladauG" }  
  ]  
}
```

Como vemos, los valores de los campos pueden ser otros documentos, arrays y arrays de documentos.

Instalación y CRUD

INSTALACIÓN

Conectar a <https://www.mongodb.com/download-center/community>

Descargar paquete MSI

Click para iniciar el wizard de instalación

Visitar:

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

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/>

Comparativa CRUD

Podemos comenzar a manipular datos con las operaciones comunes CREATE, READ, UPDATE, DELETE; y para migrar de SQL a NoSQL con Mongo, lo haremos paso a paso con una comparativa entre sentencias en uno y otro lenguaje.

Comparativa CRUD

CREATE (base de datos) en SQL

```
CREATE DATABASE alumnos;
```

CREATE (base de datos) en
Mongo

```
use peliculas
```



Comparativa CRUD

CREATE (tablas) en SQL

```
CREATE TABLE usuarios (  
  id MEDIUMINT NOT NULL AUTO_INCREMENT,  
  nombre Varchar(30),  
  edad Number,  
  alias Varchar(20),  
  PRIMARY KEY (id)  
)
```

CREATE (colección) en Mongo

```
db.createCollection("películas")
```

o también implícitamente:

```
db.películas.insertOne(  
  {  
    titulo : "Titanic",  
    año : 1979,  
    pPal : "DiCaprio"  
  }  
)
```

Comparativa CRUD

ALTER en SQL

```
ALTER TABLE usuarios ADD nacimiento DATETIME
```



ALTER en Mongo

```
db.peliculas.updateMany(  
  {},  
  { $set: { f_revision: new Date() } }  
)
```

```
ALTER TABLE usuarios DROP COLUMN nacimiento
```



```
db.users.updateMany(  
  {},  
  { $unset: { "f_revision": "" } }  
)
```


Comparativa CRUD

Los métodos **insertOne(<document>)** e **insertMany([<doc1>,<doc2>...])** implícitamente crean una colección si no existe e insertan uno o muchos documents .

No tuvimos que especificar el “id” de nuestro documento, si no definimos un campo “_id”, Mongo lo crea automáticamente.

El método **updateMany(<filter>,<update>)** actualiza múltiples documents dentro de una collection basado en el filtro que recibe como argumento, pero además puede alterar los documentos con operaciones de **\$set** (establecer) y **\$unset** (des-establecer) para agregar o borrar campos.

La forma de alterar información en Mongo no se hace a nivel de colecciones ya que no es una modificación estructural sino de documentos (registros de nuestra colección).

Comparativa CRUD

INSERT en SQL

```
INSERT INTO usuarios(nombre, edad, alias)
VALUES ("Pepe Reina", 29, "Speaker"),
("Julián Contreras", 17, "ErJuli")
);
```

INSERT en Mongo

```
db.users.insertMany(
  {
    nombre : "Pepe Reina",
    edad   : 29,
    alias  : "Speaker"
  },
  {
    nombre : "Julián Contreras",
    edad   : 17,
    alias  : "ErJuli"
  })
```


Comparativa CRUD

UPDATE y DELETE en SQL


```
UPDATE usuarios  
SET alias = "Chavales"  
WHERE age > 25;
```

```
DELETE FROM usuarios  
WHERE alias = "erJuli" ;
```

UPDATE y DELETE en Mongo



```
db.usuarios.updateMany(  
  { age: { $gt: 25 } },  
  { $set: { alias: "Majetes" } }  
)
```



```
db.usuarios.deleteMany(  
  { alias: "erJuli" }  
)
```

Comparativa CRUD

SELECT en SQL

```
SELECT *FROM usuarios;
```

```
SELECT nombre, alias FROM usuarios
```

```
SELECT nombre, edad FROM usuarios  
WHERE alias = "Speaker"
```

SELECT en Mongo

```
db.usuarios.find()
```

```
db.usuarios.find(  
  {},  
  { nombre: 1, alias: 1, _id:0})
```


```
db.usuarios.find(  
  { alias: "Speaker" },  
  { nombre: 1, edad: 1, _id: 0 })
```

Comparativa CRUD

SELECT en SQL


```
SELECT * FROM usuarios  
WHERE alias = "erJuli"  
ORDER BY nombre DESC
```

SELECT en Mongo



```
db.usuarios.find (  
  { alias: "erJuli" }  
) .sort( { nombre: -1 } )
```

```
SELECT * FROM clientes  
WHERE nombre = "Pedrooooooo"  
ORDER BY nombre DESC
```



```
db.Clientes.find (  
  { Nombre: "Pedroooooooo" } )
```

Comparativa CRUD

DROP en SQL

```
DROP TABLE IF EXISTS usuarios;
```

DROP en Mongo

```
db.usuarios.drop ( )
```

```
DROP DATABASE IF EXISTS Juegos;
```

```
use Juegos  
db.dropDatabase ( )
```

Ejercicio CRUD alumnos

1. Crear un base de datos de personas .
2. Crear una colección de alumnos.
3. Insertar diez documentos (registros con campos comunes y no comunes)
 - a. dos con nombre Pepe y tres con nombre María.
4. Listar todos los documentos que componen la colección estudiantes.
5. Listar todos los campos de aquellos documentos que contengan el nombre Pepe.
6. Listar todos los campos de aquellos documentos que contengan el nombre María y que tengan 20 años.
7. Mostrar los apellidos de los alumnos que se llamen María.
8. Saber el número de alumnos en total.
9. Saber el número de alumnos que se llaman Pepe.
10. Obtener sólo el apellido del estudiante que tenga como mail: pp@soypepe.com.

Introducción a sentencias

CREATE

`db.createCollection(nombre)`

```
db.createCollection("Alumnos")
```

recordamos que con `db.nombColec.insertOne(..)` también se creaba de forma implícita

DROP

db.dropDatabase()

- Elimina la base de datos actual, eliminando los archivos de datos asociados.
- Bloqueará otras operaciones hasta que se complete.

```
use temp
```

```
db.dropDatabase()
```

DROP

`db.nombColección.drop()`

- Elimina la Colección nombColeccción, eliminando todos los documentos que dependan de ella.
- Bloqueará otras operaciones hasta que se complete.

```
use temp
```

```
db.archivosTemporales.drop()
```

INSERT

```
db.nombcollection.insert(  
  <document or array of documents>,  
  {  
    ordered: <boolean>  
  }  
)
```

```
db.alumnos.insert( { name: "Amanda", status: "Updated" } )
```

Si la colección no existe, se crea.

Si el documento no especifica un campo `_id`, MongoDB agregará el `_id` campo y asignará un único Object Id para el documento antes de insertarlo.

Si el documento contiene un campo `_id`, el valor debe ser único dentro de la colección para evitar un error de clave duplicado.

Los valores `Object_Id` son específicos de la máquina y la hora en que se ejecuta la operación. Como tal, sus valores pueden diferir en cada ejemplo.

INSERT

Varios documentos a la vez (sin obligar a una misma estructura en todos)

```
db.products.insert(  
  [  
    { _id: 11, item: "pencil", qty: 50, type: "no.2" },  
    { item: "pen", qty: 20 },  
    { item: "eraser", qty: 25 }  
  ]  
)
```

Los documentos en la matriz no necesitan tener los mismos campos

Por defecto, MongoDB realiza una inserción *ordenada*

Con las inserciones *ordenadas*, si se produce un error durante la inserción de uno de los documentos, MongoDB devuelve un error **sin procesar los documentos restantes en la matriz.**

INSERT

desordenada....

```
db.products.insert(  
  [  
    { _id: 20, item: "lamp", qty: 50, type: "desk" },  
    { _id: 21, item: "lamp", qty: 20, type: "floor" },  
    { _id: 22, item: "bulk", qty: 100 }  
  ],  
  { ordered: false }  
)
```

Con inserciones *desordenadas*, si ocurre un error durante una inserción de uno de los documentos, MongoDB continúa insertando los documentos restantes en la matriz.

```
db.products.find({qty:{$gt:25}},{item:1, _id:0})
```

db.collection.insertOne ()

Sin valor id

```
db.products.insertOne( { item: "card", qty: 15 } );
```

Debido a que los documentos no incluyeron _id, mongod crea y agrega el campo _id y le asigna un valor ObjectId único

Con valor id

```
db.products.insertOne( { _id: 10, item: "box", qty: 20 } );
```

La inserción de un valor duplicado para cualquier clave que forme parte de un índice único , como por ejemplo _id, lanza una excepción.

db.collection.insertMany ()

Inserta múltiples documentos en una colección.

Sin valor id

```
db.products.insertMany( [  
  { item: "card", qty: 15 },  
  { item: "envelope", qty: 20 },  
  { item: "stamps" , qty: 30 }  
] )
```

Por defecto los documentos se insertan en orden.

Con ordered a false, la operación de inserción continuará con los documentos restantes, si ocurriera un error en la inserción actual.

Con valor id

```
db.products.insertMany( [  
  { _id: 10, item: "large box", qty: 20 },  
  { _id: 11, item: "small box", qty: 55 },  
  { _id: 12, item: "medium box", qty: 30 }  
] );
```

La inserción de un valor duplicado para cualquier clave que forme parte de un índice único , como por ejemplo _id, lanza una excepción.

Borrados : `deleteOne()` , `DeleteMany()`

Borrado de documentos de una colección. Se identifican criterios/filtros para para identificar el documento a borrar

db.collection.deleteOne ()

Sólo borra el primer documento que coincide con el filtro

```
db.collection.deleteOne(  
  <filter>  
)
```

```
db.Amigos.deleteOne(  
  { _id: "CTY23a" },  
)
```

```
db.Amigos.deleteOne(  
  {  
    nombre : "Irene" ,  
    apellidos : "Juarez Trujillo"  
  },  
)
```

db.collection.deleteMany ()

Cambia TODOS los documentos en una colección, que cumplen con un criterio.

```
db.comunidades.deleteMany (  
  { casas: { $lt: 2 }, codPostal: "54021"}  
)
```

Actualizaciones

Cambia de documentos de una colección. Se identifican criterios/filtros para para identificar el documento a actualizar

db.collection.updateOne ()

Sólo actualiza el primer documento que coincide con el filtro

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

upsert: TRUE → si no existe
el documento LO CREARÁ.

```
db.Amigos.updateOne(  
  { nombre : "Paola" },  
  { $set: { telefono : "2334455667" } }  
)
```

```
db.Amigos.updateOne(  
  { nombre : "Carlos" },  
  { $set: { _id : 290, telefono: "72345678" } },  
  { upsert: true }  
)
```

db.collection.updateMany ()

Cambia TODOS los documentos en una colección, que cumplen con un criterio.

```
db.familia.updateMany(  
  { casas: { $gt: 1 } },  
  { $set: { AvisarVerano : true } }  
);
```

db.collection.replaceOne ()

Sólo actualiza el primer documento que coincide con el filtro

```
db.collection.replaceOne(  
  <filter>,  
  <replacement>,  
  {upsert: <boolean>  
)
```

```
db.Amigos.replaceOne(  
  { "nombre" : "JoseAntonio" },  
  { "nombre" : "Pepe", "deporte" : "Tenis" }  
)
```