# Contenido

[1. Contenido 1](#_Toc124355947)

[2. Introducción. 1](#_Toc124355948)

[3. Bases de datos, colecciones y documentos. 2](#_Toc124355949)

[4. Tipos de datos 2](#_Toc124355950)

[4.1. Introducción 2](#_Toc124355951)

[4.2. Tipos de datos en MongoDB 3](#_Toc124355952)

[4.3. 3](#_Toc124355953)

[4.4. Anécdotas 3](#_Toc124355954)

[5. Modelado de la información y relación entre esta (muy importante). 4](#_Toc124355955)

[5.1. Introducción 4](#_Toc124355956)

[5.2. Cuando elegir un tipo de relación u otra. 4](#_Toc124355957)

[5.3. One-to-one Embebido. 4](#_Toc124355958)

[5.4. One-to-One con referencias 4](#_Toc124355959)

[5.5. One-to-Many embebed 5](#_Toc124355960)

[5.6. One-to-many referencing. 5](#_Toc124355961)

[6. Anotaciones de operaciones CRUD 5](#_Toc124355962)

[6.1. Operaciones de lectura 5](#_Toc124355963)

[6.2. Proyecciones (Projection) 5](#_Toc124355964)

[7. Estadísticas. 6](#_Toc124355965)

[8. Enlaces directos: 6](#_Toc124355966)

[8.1. Restricciones de mongoDB 6](#_Toc124355967)

MongoDB se puede runear como un servicio de sistema operativo que siempre está “bajo las escenas ejecutándose”.

# Introducción.

Entre los lenguajes de programación y MongoDB existen drivers que actúan como puentes y permiten comunicar a lenguajes como Python, C, C++, C#, Java, Go…etc. con la base de datos mongoDB. En está web <https://www.mongodb.com/docs/drivers/> , podremos tener más info de los drivers disponibles en mongoDB.

Normalmente, entre el servicio de mongoDB y el lenguaje de programación se encuentra el driver que hace consultas necesarias para la aplicación, el servicio de mongoDB también puede recibir ordenes directas desde MongoDBSh, se suele utilizar como “playground” o para administración de la base de datos.

Existe un “Storage Engine” entre el servicio de MongoDB y la base de datos o el lugar donde se almacenan los ficheros, hay varios, cada uno con sus ventajes y desventajas y podemos elegir uno dependiendo de las necesidades de nuestro negocio.

Entre el “Storage Engine” y la base de datos, se puede leer/escribir desde archivos (más lento) o leer/escribir desde memoria (más rápido).

MongoDB almacena la información en formato BSON, Binary JSON, esto se debe a que es más eficiente, por ende, más rápido y además soporta tipos especiales como ObjectId.

# Bases de datos, colecciones y documentos.

* **Bases de datos:** Se pueden tener una o más bases de datos en un servicio o servidor de mongo.
* **Colecciones:** Una base de datos está compuesta por varias colecciones
* **Documentos:** Cada colección está compuesta por varios documentos, los cuales, contienen los datos que se guardan en la base de datos, todos los documentos tienen un tamaño máximo de 16MB (parece poco, pero se almacena texto plano)
* **Relaciones:** Máximo 100 niveles de anidamiento de un documento dentro de otro.

Por ejemplo: si hacemos un “use productos” desde la raíz de mongo shell, en caso de no existir la base de datos, se crea de manera implícita. Lo mismo sucede si una vez hacemos “use productos” dentro de esta base de datos ponemos db.electronica.insertOne({nombre: “SmartTV”, precio: 1299,39}).

Cuando damos la orden de crear un nuevo documento, para una colección y base de datos especifica, si estos elementos no existen, es decir, si la base de datos o la colección no existen se crean de manera implícita, aunque se puede forzar a que sea explicito.

# Tipos de datos

## Introducción

En MongoDB la manera en la que se estructura la información no sigue ningún esquema o plantilla que debe cumplirse siempre, por ejemplo, una persona tiene edad, nombre, email, primer apellido y segundo apellido, pero MongoDB no nos obliga a que todas las personas tengan estos campos rellenados o siquiera que existan en todas las personas e incluso se pueden tener campos nuevos y diferentes según la persona en la misma colección, esto dota de gran flexibilidad a la manera en la que se maqueta la información.

Pero ojo, esta flexibilidad sin limites hace que la responsabilidad de como se estructura la información y relaciona caiga sobre el administrador de la base de datos o desarrollador, por lo que se debe intentar encontrar el equilibro entre implementar buenos patrones arquitectónicos propios de modelos más clásicos como bases de datos SQL pero también aprovecha el potencial de la flexibilidad y mundo sin normas que es mongoDB.

Lo ideal es que todos los documentos tenga una misma estructura en común siguiendo el paradigma SQL, pero que también, siempre teniendo una estructura base común, si es necesario se puedan añadir campos específicos. Por ejemplo



Tenemos que todos los clientes siguen un patrón común de tener nombre y edad, pero que Elisa, tiene un campo adicional que define que tiene una diversidad funcional.

Normalmente los ficheros se almacenan en formato BSON, que es una especie de extensión de JSON, esto permite tener tipos de datos más específicos y necesarios para una base de datos, como por ejemplo, un tipo ObjectId.

Tenemos lo tipos simples de tipo número, string, Tiempo, Geospacial…etc.

## Tipos de datos en MongoDB

|  |  |  |
| --- | --- | --- |
| Concepto | Ejemplos | Aclaraciones |
| Texto o String | “Hola”; “Hola mundo” | Entre comillas simples |
| Booleanos | True; False |  |
| Números | 55; 1290120192; 1.234 | Los números muy grandes se deben guardar o separados en diferentes partes o como string. |
| ObjectId | ObjectId(“jkklfjkl”) | Tiene un timestamp asociado |
| ISODate | ISODate(“2018-09-09” |  |
| Timestamp | Timestamp(1212321) |  |
| Embedded Documents | { a: { b: 1, c: true}} | Documentos dentros de documentos |
| Array | { odd: [2,4,6,8,10]} |  |

## Tipo GeoSpacial data.

Este tipo dato se denomina GeoJson Objects. Este objeto esta compuesto por información geoespacial, concretamente:

Longitud: distancia angular desde el meridiano de Greenwich. Puede medirse en diferentes unidades.

Latitud: distancia angular desde el ecuador.

Se puede añadir un campo de tipo GeoSpacial como sigue:

**db.lugares.insertOne({**

**name: "Plaza de España",**

**location: {**

**type: "Point",**

**coordinates: [**

**-122.42343,**

**34.232123**

**]**

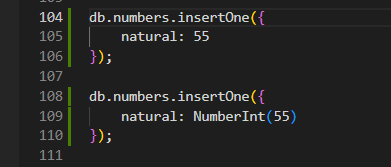
**}**

**})**

## Anécdotas

Cuando se va a crear un nuevo documento, no es necesario especificar las dobles comillas en la especificación del campo si no hay espacios.

Cuando se van a crear tipos numéricos, si se especifican tipos de datos propios de BSON para los números, los cuales, son:

* NumberInt(1231)
* NumberLong (212321321)
* NumberDecimal(12.34)

Se gestionará mejor la memoria en mongoDB y por ende todo será más eficiente. En la captural de código de la derecha se puede afirmar que el primer documento ocupara más espacio en memoria que el segundo caso.

# Modelado de la información y relación entre esta (muy importante).

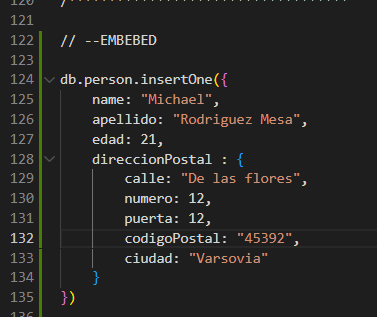
## Introducción

Podemos trabajar bien con documentos anidados o embebidos o con referencias a otros documentos.

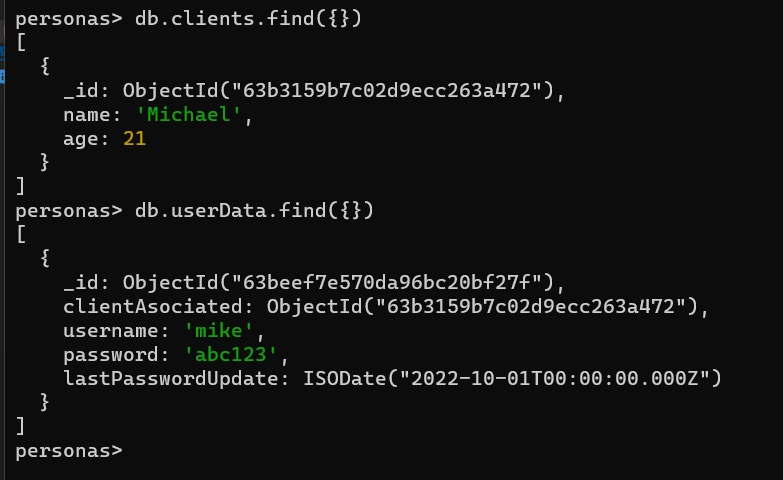
## Cuando elegir un tipo de relación u otra.

Normalmente las relaciones One-to-One van embebidas si la información que se embebe no va a crecer con el tiempo lo suficiente como para exceder los 16MB de tamaño de documento.

## One-to-one Embebido.

**Documento embebido**: Por ejemplo, una persona tiene una dirección postal, con su calle, número, puerta, código postal, país…etc. como sabemos que la información en la dirección postal, no crecerá mucho con el tiempo, no hará que el documento “persona” exceda de los 16MB en lugar de crear otra colección con la dirección postal de la persona, puede ser mejor opción anidar un documento dentro de una campo “dirección postal” y así evitar dobles consultas para obtener la dirección postal de la persona.

## One-to-One con referencias

En este caso tenemos dos colecciones una denominada clientes, y otra denominada userData, en la primera esta la información de la persona física y en la segunda esta la información del perfil de usuario virtual que tiene creado en la aplicación, como vemos en este caso, el documento con username igual a ‘mike’ hace referencia a que cliente esta asociado con el campo “clientAsociated” el cual contiene el ObjectId del cliente con nombre “Michael”, hay distintos tipos de referencias:

* Parent referencing:
* Children referencing:
* Bidirectional referencing:

## One-to-Many embebed

## One-to-many referencing.

# Anotaciones de operaciones CRUD

## Operaciones de lectura

Db.collection.find(): Devuelve un cursor, que actúa como un iterador sobre la colección completa de documentos.

Si lo que se quieren son todos los datos es mejor ejecutar db.collection.find().toArray() pero no es óptimo, en lugar de eso se puede usar el db.collection.find().forEach( ( docData) => { printjson(passengerData)} )

El forEach es mejor, porque no fuerza a obtener todo el array con todos los documentos, cargarlos en memoria y luego tener que hacer un bucle for, sino que se aprovecha el propio cursor de mongoDB para ejecutar código en cada iteración sin cargar en memoria, recordemos que las operaciones de lectura/escritura de memoria tiene tiempos altos de lectura y escritura.

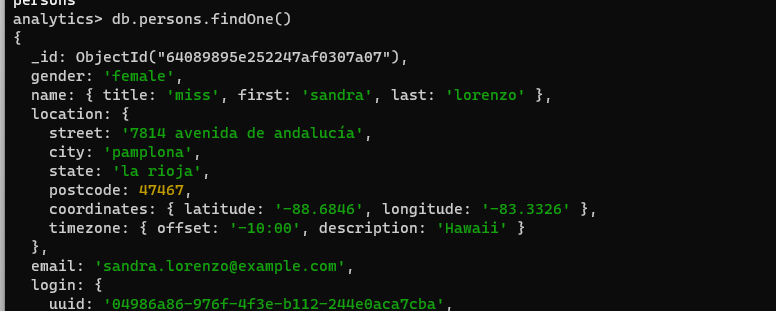
Db.collection.findOne() no nos da un cursor.

## Proyecciones (Projection)

Las proyecciones permiten que de toda la información de un determinado documento solo obtengamos la parte que corresponda, esto hace, que al viajar la información de un lado a otro, se evite enviar un documento potencialmente denso a través de la red, sino que se puede enviar el subconjunto necesario del documento para el correcto funcionamiento.

# Anotaciones de aggregation.

Se usara el siguiente modelado de datos en está sección:



Un aggregation puede entenderse como una tubería, conformada por varios filtros o fases, donde la entrada de la tubería es la colección de documentos y en cada una de los filtros o fases se van realizando operaciones (filtrado, agrupaciones, proyecciones, ordenación, etc.) sobre la salida de la anterior fase, teniendo en cuenta que la primera fase recibirá la colección, hará operaciones sobre ella y el resultado será la entrada de la siguiente fase.

**db.persons.aggregate([fase\_1, fase\_2, … ,fase\_n])**

Donde cada fase n-esima es un objeto json, donde hay diferentes operadores, por ejemplo:

En este caso la cláusula aggregate devolverá un “cursor” como en el mismo caso de find, teniendo en cuenta que la colección personas tiene personas con un género, en este caso, se hará una especie de filtro en el que solo se mostrarán aquellas personas de la colección que tenga genero femenino.

**db.persons.aggregate([  
 {**

**$match: { gender: ‘female’}**

**}**

**])**

## $project.

Es uno de los operadores más simple, proyecta solo aquellos campos que son especificados en la fase, también proyectará el “\_id” a no ser que se ponga { \_id:0} al igual que ocurre con el segundo objeto de la operación find. Se muestra una imagen que busca ilustrar el potencial de anidamiento de operadores en las proyecciones.

**db.persons.aggregate([  
 {**

**$proyect: {**

**\_id: 0 #No mostrar el \_id,**

**gender: 1, #mostrar el campo gender**

**fullName: {**

**$concat: [**

**{ $toUpper: { $substrCP: “$name.first”, 0, 1} },**

**{ $substrCP:**

**“$name.first”,**

**1,**

**{ $subtract : [{$strLenCP: ‘$name.first’}, 1]}**

**},**

**“ ”,**

**{ $toLower: “$name.last”}**

**]}**

**}**

**}**

**])**

Otro ejemplo, basado en el modelado de datos proporcionado al principio de está sección es el que se expone a continuación:

**db.persons.aggregate([**

**{**

**$proyect: {**

**\_id: 0,**

**name: 1,**

**email: 1,**

**birthdate: { $convert: { input:’$dob.date’, to: ‘date’}}**

**location: {**

**type:”Point”,**

**coordinates: [**

**{ $convert: {**

**input: “$location.coordinates.longitude”,**

**to: ‘double’,**

**onError: 0.0,**

**onNull: 0.0}**

**},**

**{ $convert: {**

**input: “$location.coordinates.longitude”,**

**to: ‘double’,**

**onError: 0.0,**

**onNull: 0.0}**

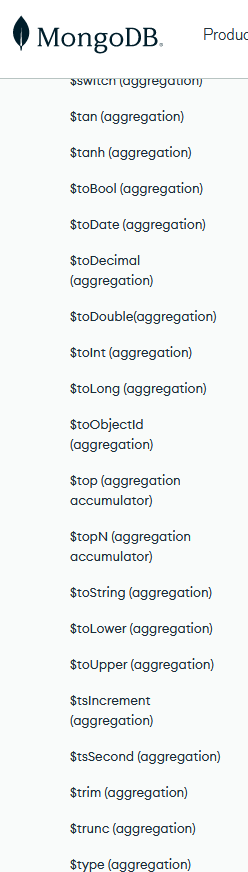
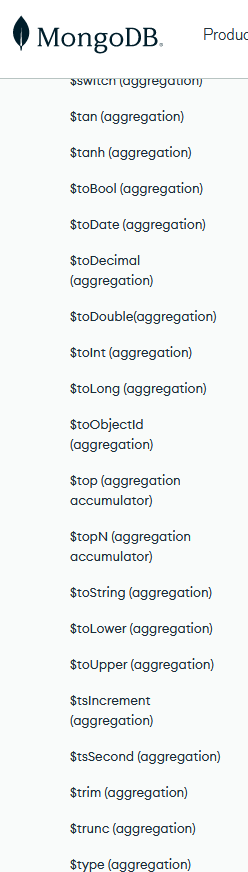
**},**

**}}**

**}**

**])**

En este caso, se puede apreciar, que se está realizando una transformación de la lontitud y la latitud que esta en formato texto, a un formato de tipo GeoJson, que tiene la estructura final que generara la proyección. Además, también se ilustro en la anterior imagen como se haría una trasnformación de tipo string a tipo date. Aunque hay unos sortcuts muy buenos que quedan de la siguiente forma:



### Anotaciones importantes en la operación $project.

Se debe tener en cuenta que si encadenamos varias fases de tipo $project entre cada una de ellas se debe volver a especificar de manera explicita que se quiere proyectar un determinado campo, incluso cuando en la anterior fase de tipo Project también se haya especificado.

## $group.

El operador $group recibirá varios parámetros, los cuales se describen en la siguiente imagen:

**db.persons.aggregate([  
 {**

**$group: {**

**\_id: {**

**state: $location.state**

**},**

**totalPersonsInThisGroup: { $sum : 1}**

**}**

**}**

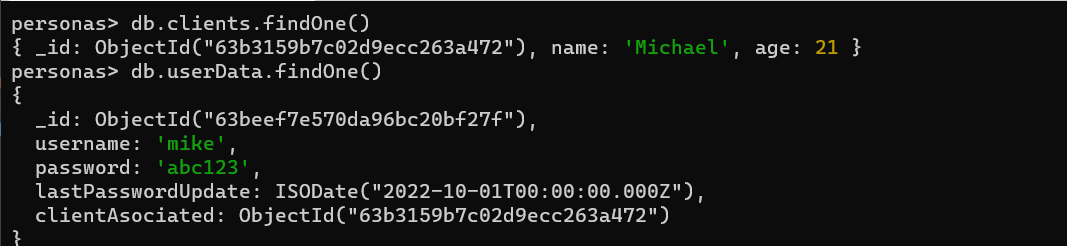
**])**

El atributo state, es como queremos llamar a ese campo por el que estamos agrupando, este nombre es a nuestra libre elección, $location.state habla de la propiedad “location.state” de los documentos que se reciben como entrada en esa fase. Además, totalPersonsInThisGroup será una variable que ira sumando 1 por cada persona que se encuentra en ese grupo, a muchos más operadores que se puede aplicar, lo cuales, pueden encontrarse en:

<https://www.mongodb.com/docs/manual/reference/operator/aggregation/group/>

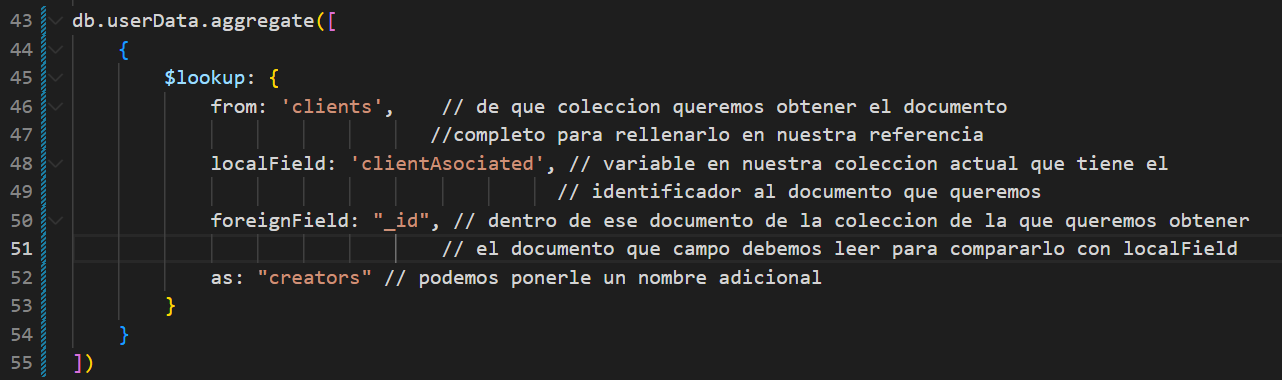
## $lookup.

Sirve para rellenar o popular las referencias a otros documentos en uno de los steps de un aggregate. Un ejemplo esclarecedor puede ser el siguiente:

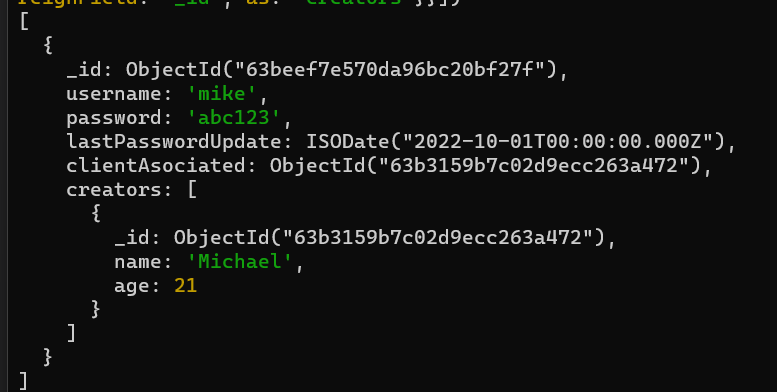


Tenemos la colección “clientes” y “user data”, como se aprecia, la colección userData tiene un usuario, cuyo campo “clientAsociated” apunta al identificador del cliente llamado Michael en la colección clientes.

El operador $lookup nos permitirá sustituir el valor de clientAsociated que es de tipo ObjectId, por el propio documento que es el clienteAsociado, es como rellenar ese campo con el objeto real. Si ejecutamos la siguiente consulta:



Teniendo como resultado:



# Estadísticas.

# Enlaces directos:

## Restricciones de mongoDB

<https://www.mongodb.com/docs/manual/reference/limits/>

## BSON Types.

https://www.mongodb.com/docs/manual/reference/bson-types/

## Query operators

https://www.mongodb.com/docs/manual/reference/operator/query/