pRÁCTICA TEMA 2 - Mongodb

USO PRÁCTICO BIG DATA

Resumen

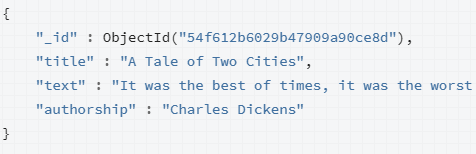
El objetivo de la siguiente práctica es que el alumno se familiarice con el manejo de una base de datos NoSQL como MongoDB, aprendiendo los conceptos y comandos básicos. Los principios que hay detrás de una base de datos documental y cómo operar con ellas de manera práctica.

### 1 Introducción

MongoDB es un sistema de base de datos multiplataforma orientado a documentos, de esquema libre. Está escrito en C++ y licenciado como GNU AGPL 3.0. Funciona en sistemas operativos Windos, linuex, OS X y Solaris.

En MongoDB, cada colección, que vendría a ser el equivalente a una tabla en el modelo relacional, contiene un conjunto de documentos que por hacer comparaciones con el modelo relacional vendrían a ser los registros. Sin embargo, las colecciones, a diferencia de la aproximación relacional puede contener documentos de diferentes formatos al no estar sometido a un esquema fijo.

Los documentos se almacenan en formato BSON o Binay JSON que permite búsquedas rápidas de datos pero en la práctica nosotros solo veremos un documento JSON tanto al almacenar como al consultar.



*Figura 7: ejemplo de documento JSON almacenado en MongoDB*

Como se puede observar en la imagen anterior, los atributos “\_id” (clave principal) pueden tener el formato que se desee, aunque MongoDB utiliza un valor parecido a un UUID en hexadecimal por defecto si no se ha especificado ninguno. Aunque parezca un valor aleatorio en realidad no lo es, utilizan como base una semilla basada en la mac de la interfaz de red de la máquina para evitar que dos máquinas diferentes puedan genera el mismo valor para la clave de un documento. Este “\_id” es el único obligatorio para un documento.

### 2. Primeros pasos

#### 2.2. Comandos.

Para abrir la consola de mongo en Windows solo tecleamos por línea de comando:

* mongo

Para abrir la consola de mongo en un terminal en Linux tendremos que iniciar el servicio.

Iniciar

* *sudo systemctl start mongod*

Saber el estado:

* *sudo systemctl status mongod*

Parar:

* *sudo systemctl stop mongod*

Reiniciar:

* *sudo systemctl restart mongod*

Para entrar a la consola de MongoDB:

* *mongo*

#### 2.2.1 Explorar la consola

Para saber la base de datos en la que estamos:

* *db*

Para saber las bases de datos disponibles

* *show dbs*

Si deseamos cambiar de base de datos

* *use myBBDD*

Si no existe no aparece hasta que se escriban datos sobre esta. (ver una vez ejecutado las bbdd disponibles con show dbs). Pero si das a db, te muestra que estás en esta nueva bbdd.

Para crear una nueva colección (similar a Tabla) y asignarle un documento (equivalente) a un registro se utiliza:

* *db.usuarios.insert({“nombre”:”Carlos”})*

Para mostrar las colecciones creadas

* *show collections*

#### 2.2.2 Importar datos

Para importar datos con Mongobd desde la consola en formato JSON o CSV se usa:

* *mongoimport --helpdb*
* *mongoimport --db myBBDD --collection profesores --jsonArray --file profesores.json*

Una vez se han importado de manera satisfactoria podemos abrir mongo:

* *use myBBDD*
* *db.profesores.count()*
* *db.profesores.find({“nombre”:”Pablo”})*

#### 2.2.3. Añadir, Modificar y Borrar datos

Ejercicios

Listar todos los profesores

|  |
| --- |
| *db.profesores.find()* |

Insertar un profesor y una asignatura en la colección

|  |
| --- |
| db.profesores.update({nombre:"Joan"},{$set:{asignatura:"Anglès"}}) |

Actualizar la asignatura del profesor que acabáis de insertar

|  |
| --- |
| db.asignaturas.insert({nombre:"Francès",curso:"4º",alumnos: 15, profesor:"Joan", ingles: false}) |

Borrar el profesor que se acaba de insertar

|  |
| --- |
| db.profesores.remove({nombre:"Joan"}) |

#### 2.2.4. Añadir campos a un documento de MongoDB

Ejercicio: Actualizar todas las edades de los profesores en una unidad

|  |
| --- |
| db.profesores.update({},{$inc:{"edad":1}},{multi:true}) |

Ejercicio: Usar upsert para incorpora un nuevo documento en caso de que no encuentre al profesor “Aitor” informando el campor “Personal”:”Fijo”

|  |
| --- |
| db.profesores.update({nombre:"Aitor"},{$set:{personal:"fixe"}}, true) |

#### 2.2.5. Filtra, Ordenar y Contar mediante consultas

Ejercicio: Listar los profesores mostrando solo los nombres:

|  |
| --- |
| db.profesores.find({},{nombre:1,"\_id":0}) |

Ejercicio: Ordenar por nombre de forma ascendente:

|  |
| --- |
| db.profesores.find({},{nombre:1,"\_id":0}).sort({nombre:1}) |

Ejercicio: Ordenar por nombre de forma descendente:

|  |
| --- |
| db.profesores.find({},{nombre:1,"\_id":0}).sort({nombre:-1}) |

Ejercicio: Limitar a 2 una búsqueda general sobre la colección profesores

|  |
| --- |
| db.profesores.find({},{nombre:1,"\_id":0}).sort({nombre:1}).limit(2) |

Ejercicio: Limitar a 2 intercalando 1 salto entre resultados

|  |
| --- |
| db.profesores.find({},{nombre:1,"\_id":0}).sort({nombre:1}).limit(2).skip(1) |

Ejercicio: Contar el número de profesores de la colección

|  |
| --- |
| db.profesores.find().count() |

Ejercicio: Contar el número de profesores que se llaman juan

|  |
| --- |
| db.profesores.find({nombre:"Juan"}).count() |

* Si queremos acceder a un campo compuesto, primero añadimos uno nuevo:
  + db.profesores.update(

{“nombre”:”Aitor”},

{$set:{“Asignaturas”:{“1A”:”Matemáticas”,”1B”:”Lenguaje”,”2C”:”Valenciano”}}}

)

* + db.profesores.find({“Asignaturas.1A”:”Matemáticas”).pretty()

##### 2.2.6: Selectores de consultas.

Ejemplos:

* Contar el número de profesores mayor de 40 años
  + db.profesores.count({edad:{$gte:40}})
* Si queremos recuperar aquellos que cuenten o no con un determinado campo
  + db.profesores.find({“edad”:{$exists:false}}).pretty()

Para hacer los siguientes ejercicios prácticos cargamos una nueva colección:

* mongoimport --db myBBDD --collection asignaturas --jsonArray --file asignaturas.json

Ejercicio: Listar las asignaturas con 23 alumnos

|  |
| --- |
| db.asignaturas.find({alumnos:23}) |

Ejercicio: Listar las asignaturas con 23 alumnos o más

|  |
| --- |
| db.asignaturas.find({alumnos:{$gte:23}}) |

Ejercicio: Listar las asignaturas que imparte el profesor Juan

|  |
| --- |
| db.asignaturas.find({profesor:"Juan"}) |

Ejercicio: Listar las asignaturas que NO imparte el profesor Juan

|  |
| --- |
| db.asignaturas.find({profesor:{$ne:"Juan"}}) |

Ejercicio: Listar las asignaturas que imparten Juan o Laura

|  |
| --- |
| db.asignaturas.find({$or:[{profesor:"Juan"},{profesor:"Laura"}]})  db.asignaturas.find({profesor:{$in:["Juan","Laura"]}}) |

Ejercicio: Listar las asignaturas que NO imparten ni Juan ni Laura

|  |
| --- |
| db.asignaturas.find({$nor:[{profesor:"Juan"},{profesor:"Laura"}]})  db.asignaturas.find({profesor:{$not:{$in:["Juan","Laura"]}}}) |

##### 2.2.7 – Operadores lógicos y de elementos de consulta

Veamos algunos ejemplos prácticos:

* Buscamos los documentos que cumplan las condiciones marcadas en el AND
  + db.asignaturas.find({$and:[{“nombre”:”Naturales”},{“profesor”:”Juan”}]})
* Cambiamos el AND por el OR para buscar aquellos que cumplan alguna de las dos condiciones
  + db.asignaturas.find({$or:[{“nombre”:”Naturales”},{“profesor”:”Laura”}]})
* Si modificamos el OR por el NOR nos devuelve el resto de documentos que no cumplen
  + db.asignaturas.find({$nor: [{“nombre”:”Naturales”},{“profesor”:”Laura”}]})
* Con la cláusula not negamos la condición de consulta
  + db.asignaturas.find({“nombre”:{$not:{$eq:”Naturales”}}}).pretty()
* Vamos a realizar consultas en función del tipo de campo que queramos buscar, para ellos es importante conocer los diferentes tipos de campos que pueden existir y cómo se nombran:

| **Type** | **Number** | **Alias** |
| --- | --- | --- |
| Double | 1 | "double" |
| String | 2 | "string" |
| Object | 3 | "object" |
| Array | 4 | "array" |
| Binary data | 5 | "binData" |
| Undefined | 6 | "undefined" |
| ObjectId | 7 | "objectId" |
| Boolean | 8 | "bool" |
| Date | 9 | "date" |
| Null | 10 | "null" |
| Regular Expression | 11 | "regex" |
| DBPointer | 12 | "dbPointer" |
| JavaScript | 13 | "javascript" |
| Symbol | 14 | "symbol" |
| JavaScript code with scope | 15 | "javascriptWithScope" |
| 32-bit integer | 16 | "int" |
| Timestamp | 17 | "timestamp" |
| 64-bit integer | 18 | "long" |
| Decimal128 | 19 | "decimal" |
| Min key | -1 | "minKey" |
| Max key | 127 | "maxKey" |

Algunos ejemplos:

* + db.asignaturas.find({“alumnos”:{$type:”number”}}).pretty()
  + db.asignaturas.find({“alumnos”:{$type:”string”}})

Ejercicio:

Unir las dos consultas anteriores en una sola sustituyendo el alias por el number (Utilizar $or):

|  |
| --- |
| db.asignatura.find({$or:[{"alumnos":{$type:1}},{alumnos:{$type:2}}]}) |

##### 2.2.8 – Operadores de evaluación de consulta

Para comenzar a ver este tipo de operadores primero vamos a cargar el resto de colecciones colecciones:

* mongoimport --db myBBDD --collection presupuesto --jsonArray --file presupuesto.json
* mongoimport --db myBBDD --collection productos --jsonArray --file productos.json
* mongoimport --db myBBDD --collection catalogo --jsonArray --file catalogo.json
* mongoimport --db myBBDD --collection articulos --jsonArray --file articulos.json

Ejercicio: Dónde estamos gastando más que nuestro presupuesto, en $expr, los campos van con ‘$’:

|  |
| --- |
| db.presupuesto.find({$expr:{$gt:["$gasto","$presupuesto"]}}) |

Con esta expresión miramos que se cumpla la condición: si la cantidad es superior a 100 entonces divide el precio entre 2, sino entre 4, y, si este resultado es menor que 5 entonces lo muestra.

* db.productos.find({

*$expr:{*

*$lt:[{*

*$cond:{*

*If: {gte:[“$cantidad”,100]},*

*then: {$divide:[“$precio”,2]},*

*else: {$divide:[“$precio”,4]}*

*}*

*},5]}*

*})*

Ahora vamos a ver algunos ejemplos con $regex

* Indicamos que busque una determinada secuencia al final de la cadena
  + db.catalogo.find({“sku”:{$regex:/9$/}})
* Con ^, acento circunflejo, para que lo busque al inicio, y lo ponemos en mayúsculas, añadiendo en las opciones la i, de insensitive recupera mayúsculas y minúsculas.
  + db.catalogo.find({“sku”:{$regex:/^ABC/i}})
* Con la opción m tiene en cuenta el salto de línea
  + db.catalogo.find({“descripcion”:{$regex:/^S/m}})

En el siguiente punto vamos a ver cómo trabajar con textos: **$text**

* Creamos un índice, ya que text solo funciona en los campos indexados como texto.
  + db.articulos.createIndex({“subject”:”text”})

Ejercicio: buscar “backing” en la colección artículos

|  |
| --- |
|  |

Ejercicio: buscar “backing” y “coffee” en un mismo comando

|  |
| --- |
|  |

Ejercicio: buscar “backing” excluyendo “coffee” en un mismo comando

|  |
| --- |
|  |

Ejercicio: buscar la frase completa “café con leche

|  |
| --- |
|  |

Ejercicio: realizar la misma consulta que la anterior pero teniendo en cuenta las tildes

|  |
| --- |
|  |

Ejercicio: realizar la misma consulta que la anterior pero teniendo en cuenta mayúsculas y minúsculas

|  |
| --- |
|  |

Con Text Search Score, la búsqueda de texto asigna una puntuación a cada documento que contiene el término de búsqueda en los campos indexados. La puntuación determina la relevancia de un documento para una consulta de búsqueda.

Para cada campo indexado en el documento, MongoDB multiplica el número de coincidencias por el peso y suma los resultados. Con esta suma, MongoDB calcula la puntuación del documento. El peso predeterminado es 1 para los campos indexados.

Un ejemplo:

* db.articulos.find({$text:{$search:”baking”}},{score:{$meta:”textScore”}})

Ejercicio

¿Y si quisiéramos mostrarlo ordenado por el textScore?

|  |
| --- |
|  |

##### 2.2.9 – Operadores con arrays

Para ver algunos ejemplos cargamos una nueva colección

* mongoimport --db myBBDD --collection inventario --jsonArray --file inventario.json

Ejercicio: Buscar todos los registros que en etiquetas tengan cuaderno y libro.

|  |
| --- |
|  |

Ejercicio: Buscar los documentos con arrays de tamaño 5

|  |
| --- |
|  |

Con el operador $slice muestra partes de un array y tiene dos formas

* clave: {$slice:n}, muestra las n primeras componentes del array. Si n es negativo muestra las n últimas.
* Clave:{$slice:[n1,n2]}, n1 es la cantidad de valores que se saltan y n2 la cantidad de valores consecutivos a mostrar.

Veamos algunos ejemplos:

* db.inventario.find({“\_id”:1},{“etiquetas”:{$slice:2},”cantidad”:0}).pretty()
* db.inventario.find({“\_id”:1},{“etiquetas”:{$slice:[1,2]},”cantidad”:0}).pretty()

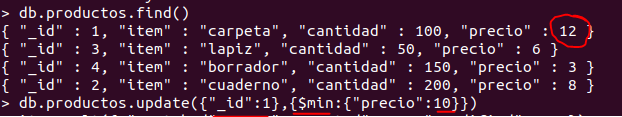
##### 2.2.10 – Comando $where

* Este operador establece una condición que deben cumplir todos los documentos.
* El formato es $where:cadena, donde cadena es una expresión JavaScript.
* El documento actual se referencia como this, y los valores asociados a sus claves como this.clave.
  + db.presupuesto.find({$where:”this.presupuesto > this.gasto”})

##### 2.2.11 – Modificadores de actualización

Los siguientes ejemplos vamos a ilustrarlos para que sea explicado mejor:

* db.productos.update({“\_id”:1},{$min:{“precio”:10}})



Si el precio que estoy proponiendo (10) es menor al precio que tiene el id1 (12) entonces actualiza, con lo que el precio se queda en 10.

Ejercicio: Asignarle al producto con id:1 un precio de 20 si el precio que tiene actualmente es menor

|  |
| --- |
|  |

Ejercicio: Multiplica por dos 2 el precio del producto \_id:1

|  |
| --- |
|  |

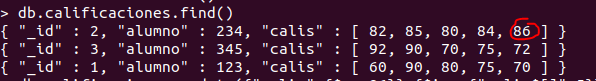
##### 2.2.12 – Modificadores de actualización de arrays

Cargamos el documento calificaciones

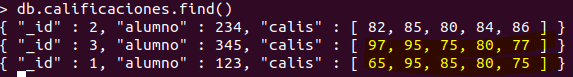
*mongoimport --db myBBDD --collection calificaciones --jsonArray --file calificaciones.json*

A continuación se muestran algunos ejemplos:

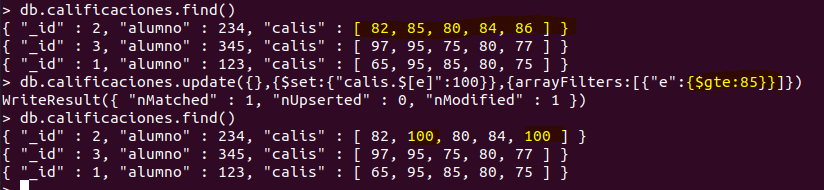
* Buscamos el documento con \_id = 2, y el elemento del array 83, actualiza a 84 para la primera coincidencia
  + db.calificaciones.update({“\_id”:2,”calis”:83},{$set:{“calis.$”:84}})
* Para modificar un array con objetos utilizamos la notación punto
  + db.inventario.update({“\_id”:3,”cantidad.tamano”:”S”},{$set:{“cantidad.$.color”:”rojo”}})
* Para modificar todos los elementos de un array, con multi:true actualizamos todos los registros.
  + db.calificaciones.update({},{$inc:{“calis.$[]”:1}},{multi:true})
* Modificar todos los documentos de un array
  + db.inventario.update({},{$inc:{“cantidad.$[].num”:-2}},{multi:true})
* Modificar los arrays especificados usando la negación del operador de búsqueda
  + db.calificaciones.update({“calis”:{$ne:86}},{$inc:{“calis.$[]:5}},{multi:true})



Lo que hace esta consulta es mirar los arrays y si encuentra un elemento con 86 no actualiza nada



* En el siguiente ejemplo vamos a utilizar el operador $[<identificación>]. Vamos a decirle que actualice solo los documentos cuyo valor del array sea igual o mayor a 85
* db.calificaciones.update({},{$set:{“calis.$[e]”:100}},{arrayFilters:[{“e”:{$gte:85}]})



* Para añadir un valor a un array usamos $addToSet, para ello vamos a añadir “lápiz” a las etiquetas del primer documento que encuentre.
  + db.inventario.update({“\_id”:2},{$addToSet:{“etiquetas”:”lápiz”}})

Ejercicio: Eliminar el primer elemento del array de la colección de calificaciones para el alumno con id=1

|  |
| --- |
|  |

Ejercicio: Eliminar el último elemento de un array

|  |
| --- |
|  |

Ejercicio: Agregar la calificación 90 al array

|  |
| --- |
|  |

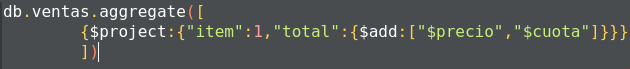
Ejercicio: Eliminar todos las “calis” coincidentes con 100 y 81 un array se usa pullAll y los valores a buscar se ponen entre corchetes.

|  |
| --- |
|  |

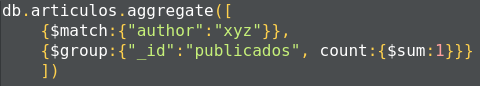
##### 2.2.13 – Agregaciones en MongoDB y Pipeline

Veamos algunos ejemplos ilustrando con Robo3t para que sea más legible

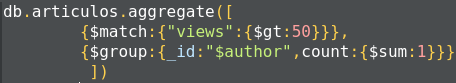
* sumamos los campos precio y cuota de del ítem 1 de ventas



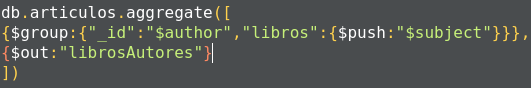
* Vamos a hacer un pipeline: Etapa 1 match y etapa 2 group, tomamos captura de robot3 para ilustrarlo mejor



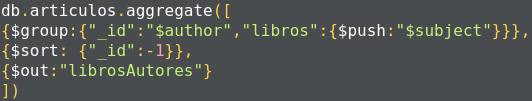
* Números de artículos con más de 50 visitas por autor



* Con el comando $out escribimos el resultado de la agregación pipeline a una colección



* Añadimos $sort

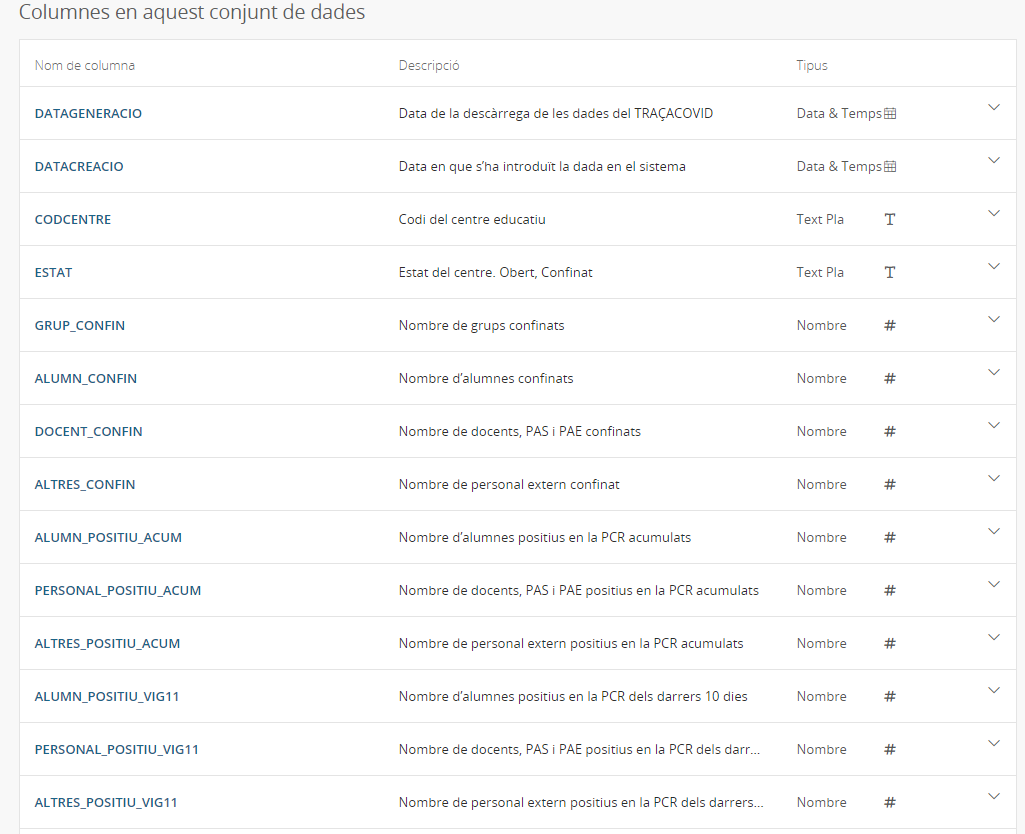


## 3. Supuesto práctico Traçacovid

Debemos acceder al siguiente enlace para acceder al dataset con el que vamos a trabajar: [enlace](https://datos.gob.es/en/catalogo/a09002970-datos-covid-19-en-los-centros-educativos).

Se trata de un dataset ofrecido por la Generalitat de Catalunya donde se recogen los datos de los casos de covid-19 que se han producido en los centros educativos a través de la aplicación Traçacovid que da soporte a los centros en el marco de protocolo de casos covid-19.

Si le echamos un primer vistazo a los datos: [enlace](https://analisi.transparenciacatalunya.cat/Educaci-/Dades-COVID-19-als-centres-educatius/fk8v-uqfv/data), vemos que se distribuyen de la siguente forma:

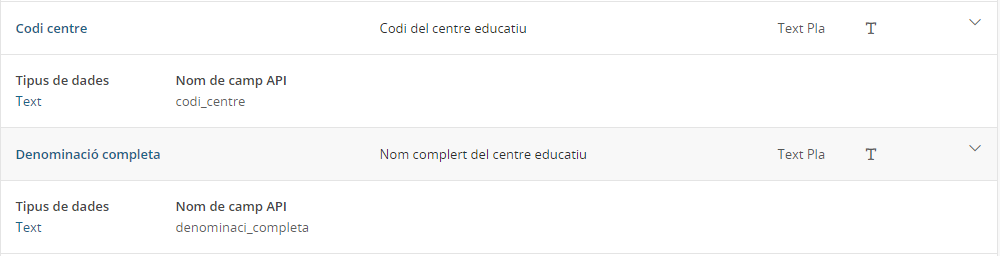


Descargamos el archivo y lo importamos en formato .csv

* Mongoimport --db myBBDD --collection covid --type=csv --headerline --file Dades\_COVID-19\_als\_centres\_educatius.cs

Adicionalmente, descargamos el directorio de centros adscritos al programa: [enlace](https://datos.gob.es/en/catalogo/a09002970-directorio-de-centros-docentes1).

La descripción de todas las columnas que conforman el dataset se explica aquí: [enlace](https://analisi.transparenciacatalunya.cat/Educaci-/Directori-de-centres-docents-anual-Base-2020/kvmv-ahh4). Aunque como resumen por ser parte del ejercicio destacamos:



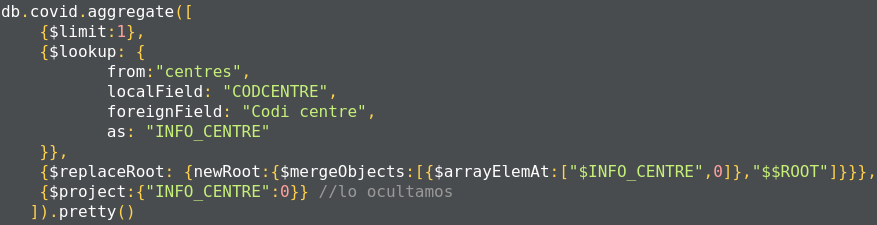
Una vez descargado lo importamos de la siguiente forma:

* mongoimport --db myBBDD --collection centres --type=csv --headerline --file Directori\_de\_centres\_docents\_anual.\_Base\_2020.csv

Para hacer este ejercicio primero es necesario conocer cómo cruzar dos colecciones en mongo, para ello usamos $lookup para unir ambas por CODCENTRE:



Esto nos añade el registro del centro correspondiente como un subdocumento del documento covid, si queremos tener un único documento:



Con lo visto anteriormente el alumno debe responder a la siguiente cuestión:

Ejercicio Final:

Crea una colección con los nombres de los 3 centros que estuvieron abierto el día 18/10/2020 con mayor número de alumnos confinados ordenados de mayor a menor.

|  |
| --- |
| db.covid.aggregate([  ... {$lookup:{from:"centres",localField:"CODCENTRE",foreignField:"Codi centre", as: "INFO\_CENTRE"}},  ... {$match:{DATAGENERACIO:"19/10/2020",ESTAT:"Obert"}},  ... {$project:{CODCENTRE:1,nom:{$arrayElemAt:["$INFO\_CENTRE.Denominació completa",0]},ESTAT:1,ALUMN\_CONFIN:1}},  ... {$sort:{ALUMN\_CONFIN:-1}},  ... {$limit:3},  ... {$out:"topConfinats"}])  Ho he fet sobre el dia 19/10/2020 perquè el dia 18 era diumenge |

Para evaluar el seguimiento del curso, enviar esta memoria a [clopez@teralco.com](mailto:clopez@teralco.com)