# EJERCICIOS

### EJERCICIO 1

Haciendo uso de mongosh, escribe los comandos necesarios para:

1. Obtener las bases de datos creadas.

show dbs

1. Sobre la base de datos sample\_training y la colección zips averigua:

use sample\_training

* 1. Cuantos documentos hay en la ciudad de SAN DIEGO.

db.zips.countDocuments({ city: "SAN DIEGO" })

* 1. Cuantos documentos tienen menos de 100 personas (campo pop).

db.zips.countDocuments({pop: {$lt: 100}})

* 1. Obtén los estados de la ciudad de SAN DIEGO (Solución: [ 'CA', 'TX' ]).

db.zips.distinct("state", { city: "SAN DIEGO" })

* 1. Cuál es el código postal de la ciudad de ALLEN que no tiene habitantes (sólo recupera el zip, no nos interesa ningún otro campo, ni el \_id).

db.zips.find({city:"ALLEN",pop:{ $eq:0} },{zip:1,\_id:0})

db.zips.findOne({ city: "ALLEN", pop: 0 }, { \_id: 0, zip: 1 })

* 1. Listado con los 5 códigos postales más poblados (muestra los documentos completos).

db.zips.find().sort({ pop: -1 }).limit(5)

* 1. Cantidad de documentos que no tienen menos de 5.000 habitantes ni más de 1.000.000 (debes utilizar el operador $nor).

db.zips.countDocuments({

$nor: [

{ pop: { $lt: 5000 } },

{ pop: { $gt: 1000000 } }

]

})

* 1. Cuantos documentos tienen más habitantes que su propio código postal (campo zip). Para hacerlo usa $expr (tendrás que investigar cómo se usa y ten en cuenta que para comparar dos campos deberían ser del mismo tipo).

db.zips.countDocuments({

$expr: { $gt: [ { $toInt: "$pop" }, { $toInt: "$zip" } ] }

})

1. Sobre la colección posts averigua:
   1. Cuantos mensajes tienen las etiquetas restaurant o moon.

db.posts.countDocuments({tags: { $in: ["restaurant", "moon"] }})

* 1. Los comentarios que ha escrito el usuario Salena Olmos.

db.posts.find({"comments.author": "Salena Olmos"},{"comments.$":1,\_id:0})

* 1. Recupera los mensajes que en body contengan la palabra earth, y devuelve el título, 3 comentarios y 5 etiquetas (tendrás que usar expresiones regulares $slice).

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

db.posts.find(

{ body: /earth/i },

{ title: 1, comments: { $slice: 3 }, tags: { $slice: 5 } }

)

db.posts.find(

{ body: {$regex: /earth/, $options:"i" }},

{ title: 1, comments: { $slice: 3 }, tags: { $slice: 5 } }

)

### EJERCICIO 2

Escribe los comandos necesarios para realizar las siguientes operaciones sobre la colección zips:

1. Crea una entrada con los siguientes datos, para Vitoria-Gasteiz:

*{*

*city: "GAS",*

*zip: '01001',*

*loc: { x: 42.859265, y: -2.681561 },*

*pop: 250500*

*}*

db.zips.insertOne({

city: "GAS",

zip: '01001',

loc: { x: 42.859265, y: -2.681561 },

pop: 250500

})

1. Crea una entrada con los datos para Bilbao y Donosti.

db.zips.insertMany([

{

city: "BIL",

zip: '48001',

loc: { x: 43.263005, y: -2.934991 },

pop: 345821

},

{

city:"DON",

zip: '20001',

loc: { x: 43.318334, y: -1.981231 },

pop: 186409

}

])

1. Modifica la población de código postal '01001' a 1.000.000.

db.zips.updateOne(

{ zip: '01001' },

{ $set: { pop: 1000000 } }

)

1. Incrementa la población de todos los documentos de Nueva York (NY) en 666 personas (puedes usar $inc)

db.zips.updateMany(

{ state: "NY" },

{ $inc: { pop: 666 } }

)

1. Añade un campo ca a los tres documentos que hemos creado cuyo valor sea Euskadi.

db.zips.updateMany(

{ city: { $in: ['GAS', 'BIL', 'DON'] } },

{ $set: { ca: 'Euskadi' } }

)

1. Modifica los documentos de Euskadi y añade un atributo tags que contenga un array vacío.

db.zips.updateMany(

{ ca: 'Euskadi' },

{ $set: { tags: [] } }

)

1. Modifica todos los documentos de Euskadi y añade al atributo tags el valor "euskera" (tendrás que usar $push).

db.zips.updateMany(

{ ca: 'Euskadi' },

{ $push: { tags: 'euskera' } }

)

1. Modifica el valor de euskera del código postal 20001 y sustitúyelo por "playa" (tendrás que usar .$).

db.zips.updateOne(

{ zip: '20001', "tags": "euskera" },

{ $set: { "tags.$": "playa" } }

)

1. Renombra en los documentos de Euskadi el atributo ca por comunidad\_autonoma (tendrás que usar $rename)

db.zips.updateMany(

{ ca: 'Euskadi' },

{ $rename: { "ca": "comunidad\_autonoma" } }

)

1. Elimina las coordenadas del código postal 35014.

db.zips.updateOne(

{ zip: '35014' },

{ $unset: { loc: '' } }

)

1. Elimina la entrada para Vitoria

db.zips.deleteOne({ city: "GAS" })

1. Elimina las entradas para Bilbao y Donosti.

db.zips.deleteMany(

{ city: { $in: ["BIL","DON"] } }

)

### EJERCICIO 3

Para las siguientes actividades, vamos a utilizar la base de datos sample\_mflix, y en concreto, la colección movies.

Haciendo uso las agregaciones en mosgosh resuelve las siguientes consultas:

1. Encuentra todas las películas que entre sus géneros (genres) se encuentre el Drama. Sólo queremos recuperar el título y la calificación (rating) de IMDB.

db.movies.aggregate([

{

$match: {

genres: "Drama"

}

},

{

$project: {

title: 1,

"imdb.rating": 1,

\_id: 0

}

}

])

1. Recupera los títulos de las tres películas románticas (Romance) con mayor calificación en IMDB que se lanzaron (released) antes del 2001.

db.movies.aggregate([

{

$match: {

genres: "Romance",

"imdb.rating": { $exists: true },

released: { $lt: ISODate("2001-01-01T00:00:00.000Z") }

}

},

{

$sort: {

"imdb.rating": -1

}

},

{

$limit: 3

},

{

$project: {

title: 1,

\_id: 0

}

}

])

1. Averigua la cantidad de películas que hay de cada categoría de calificación (rated).

db.movies.aggregate([

{

$group: {

\_id: "$rated",

count: { $sum: 1 }

}

}

])

1. Teniendo en cuenta las películas anteriores al año 2001, para cada género, recupera la media y la máxima calificación en IMDB así como el tiempo de la película más larga (runtime + 12min de trailer), ordenando los géneros por popularidad.

El resultado será similar a:

*{*

*\_id: 'Film-Noir',*

*nota\_media: 7.62,*

*mejor\_nota: 8.3,*

*tiempo\_ajustado: 123*

*}*

*{*

*\_id: 'Documentary',*

*nota\_media: 7.555313351498638,*

*mejor\_nota: 9.4,*

*tiempo\_ajustado: 1152*

*}*

*{*

*\_id: 'Short',*

*nota\_media: 7.386,*

*mejor\_nota: 8.6,*

*tiempo\_ajustado: 56*

*}*

*...*

db.movies.aggregate([

{

$match: {

released: { $lt: ISODate("2001-01-01T00:00:00.000Z") },

"imdb.rating": { $exists: true },

}

},

{$unwind: "$genres"},

{

$group: {

\_id: "$genres",

nota\_media: { $avg: "$imdb.rating" },

mejor\_nota: { $max: "$imdb.rating" },

tiempo\_ajustado: { $max: { $add: ["$runtime", 12] } }

}

},

{

$sort: {

nota\_media: -1

}

}

])

1. Queremos buscar qué película recomendar (máxima popularidad) de cada genero siempre y cuando duren un máximo de 218 minutos y tengan al menos una calificación de 7 (pista: necesitas utilizar $first). Hay que mostrar el género, el título recomendado, nota recomendado (rating), tiempo recomendado (runtime) y la popularidad del género (media del rating para el género)

El resultado será similar a:

*{*

*\_id: 'Documentary',*

*titulo\_recomendado: 'Planet Earth',*

*nota\_recomendado: 9.5,*

*tiempo\_recomendado: 60,*

*popularidad: 7.624062968515742*

*,*

*{*

*\_id: 'News',*

*titulo\_recomendado: 'Most Likely to Succeed',*

*nota\_recomendado: 8.9,*

*tiempo\_recomendado: 86,*

*popularidad: 7.622580645161291*

*},*

*...*

db.movies.aggregate([

{

$match: {

runtime: { $lte: 218 },

"imdb.rating": {$gte: 7}

}

},

{$unwind: "$genres"},

{

$sort: {

"imdb.rating": -1

}

},

{

$group: {

\_id: "$genres",

titulo\_recomendado: { $first: "$title" },

nota\_recomendado: { $first: "$imdb.rating" },

tiempo\_recomendado: { $first: "$runtime" },

popularidad: { $avg: "$imdb.rating" },

}

},

{

$sort: {

popularidad: -1

}

}

])

1. Recupera las 5 películas más comentadas (los comentarios se almacenan en la colección comments), devolviendo el título, su género y la cantidad de comentarios. Además, queremos almacenar el resultado en la colección peliculas\_mas\_comentadas.

db.movies.aggregate([

{

$lookup: {

from: "comments",

localField: "\_id",

foreignField: "movie\_id",

as: "comments"

}

},

{

$project: {

title: 1,

genres: 1,

numComments: { $size: "$comments" },

\_id: 0

}

},

{

$sort: {

numComments: -1

}

},

{

$limit: 5

},

{$out: "peliculas\_mas\_comentadas"}

])

### EJERCICIO 4

Haciendo uso de MongoDBCompass, mejora la siguiente consulta que obtiene los tres documentales más premiados, siempre y cuando hayan ganado algún premio (añade las etapas una a una mediante interfaz gráfica):

*db.movies.aggregate([*

*{ $sort: {"awards.wins": -1}}, // Ordenamos por premios ganados*

*{ $match: {"awards.wins": { $gte: 1}}},*

*{ $limit: 20}, // Obtenemos las 20 películas que han ganado más de un premio*

*{ $match: {*

*genres: {$in: ["Documentary"]}, // Nos quedamos con los documentales*

*}},*

*{ $project: { title: 1, genres: 1, awards: 1}},*

*{ $limit: 3},*

*])*

db.movies.aggregate([

{ $match: {

genres: {$in: ["Documentary"]},

"awards.wins": { $gte: 1}

}},

{ $sort: {"awards.wins": -1}},

{ $limit: 3},

{ $project: { title: 1, genres: 1, awards: 1}},

])

### EJERCICIO 5

Nuestro ayuntamiento está diseñando un plan energético para fomentar la instalación de placas solares. Aquellas viviendas que se sitúen en barrios que generan más energía de la que consumen recibirán un bono económico por el exceso de energía.

Nuestra base de datos almacena, para cada vivienda, cuanta energía produce por hora (en kW), cuanta consume, y cuanta necesita consumir de la red eléctrica.

VIVIENDA:

*{*

*"\_id": ObjectId("6317048697eb703de2add36f"),*

*"propietario": "Ainara Montoya",*

*"direccion": {*

*"calle": "Secreta",*

*"numero": "123",*

*"ciudad": "Vitoria-Gasteiz",*

*"barrio": "Zabalgana",*

*"provincia": "Araba",*

*"cp": "01006"*

*}*

*}*

ENERGÍA

*{*

*"\_id": ObjectId("6316fc1597eb703de2add36e"),*

*"propietario\_id": ObjectId("6317048697eb703de2add36f"),*

*"date": ISODate("2022-10-26T13:01:00.000Z"),*

*"kW\_hora": {*

*"consumo": 11,*

*"generado": 16,*

*}*

*}*

Consigue una consulta para recuperar cada día y para cada barrio el exceso de energía acumulado (generado – consumo).

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Introduce datos de prueba para hacer las comprobaciones adecuadas.

db.viviendas.insertOne({

\_id: 1,

nombre: "Ainara Montoya",

direccion: {

calle: "Secreta",

numero: "123",

ciudad: "Vitoria-Gasteiz",

barrio: "Zabalgana",

provincia: "Araba",

cp: "01006"

}

})

db.viviendas.insertMany([

{

\_id: 2,

nombre: "Propietario1",

direccion: {

calle: "Calle1",

numero: "1",

ciudad: "Vitoria-Gasteiz",

barrio: "Zabalgana",

provincia: "Araba",

cp: "01006"

}

},

{

\_id: 3,

nombre: "Propietario2",

direccion: {

calle: "Calle2",

numero: "2",

ciudad: "Vitoria-Gasteiz",

barrio: "El Pilar",

provincia: "Araba",

cp: "01012"

}

},

{

\_id: 4,

nombre: "Propietario3",

direccion: {

calle: "Calle3",

numero: "3",

ciudad: "Vitoria-Gasteiz",

barrio: "El Pilar",

provincia: "Araba",

cp: "01012"

}

}

])

db.energia.insertOne({

propietario\_id: 1,

date: ISODate("2022-10-26T13:01:00.000Z"),

kW\_hora: {

consumo: 11,

generado: 16,

}

})

db.energia.insertMany([

{

propietario\_id: 1,

date: ISODate("2022-10-26T14:00:00.000Z"),

kW\_hora: {

consumo: 10,

generado: 9,

}

},

{

propietario\_id: 1,

date: ISODate("2022-10-26T15:01:00.000Z"),

kW\_hora: {

consumo: 10,

generado: 26,

}

},

{

propietario\_id: 1,

date: ISODate("2022-10-26T16:01:00.000Z"),

kW\_hora: {

consumo: 10,

generado: 20,

}

},

{

propietario\_id: 3,

date: ISODate("2022-10-26T13:01:00.000Z"),

kW\_hora: {

consumo: 11,

generado: 36,

}

},

{

propietario\_id: 3,

date: ISODate("2022-10-26T13:01:00.000Z"),

kW\_hora: {

consumo: 11,

generado: 2,

}

}

])

db.viviendas.aggregate([

{

$lookup: {

from: "energia",

localField: "\_id",

foreignField: "propietario\_id",

as: "energias",

},

},

{

$unwind: "$energias",

},

{

$group: {

\_id: {

fecha: {

$dateToString: {

format: "%Y-%m-%d",

date: "$energias.date",

},

},

barrio: "$direccion.barrio",

},

consumo\_total: {

$sum: "$energias.kW\_hora.consumo",

},

generacion\_total: {

$sum: "$energias.kW\_hora.generado",

}

}

},

{

$addFields: {

excedente: {

$subtract: ["$generacion\_total","$consumo\_total",],

}

}

},

{

$project: {

\_id: 0,

fecha: "$\_id.fecha",

barrio: "$\_id.barrio",

excedente: 1,

}

}

])

### EJERCICIO 6

Define el modelo de datos necesario para modelar una aplicación de captura de datos:

* Debes tener en cuenta la gestión de los usuarios, los cuales pueden ser los usuarios finales (clientes), los técnicos de asistencia (técnicos), así como los usuarios administradores de la aplicación.
* De los clientes, además del nombre completo, sexo y la fecha de nacimiento, registraremos su enfermedad (categoría, nombre, descripción).
* Las sesiones de los clientes se van a grabar, y además de la ruta del audio (el cual almacenaremos en un sistema externo), guardaremos la fecha de la grabación y la velocidad de la grabación.
* Los textos son transcripciones de los audios que el administrador o el técnico dan de alta. Todos los textos se agruparán por categoría, y cada texto, puede tener un número de orden, así como un estado de ánimo.
* Los listados que luego necesitaremos para recuperar los datos son:
  1. Número de clientes de una determinada enfermedad, agrupando por edad y sexo.
  2. Sesiones y textos de un determinado cliente.
  3. Clientes, sesiones y textos de una determinada enfermedad.

Para ello, primero define el modelo, crea diferentes colecciones/documentos JSON con datos de muestra así como las consultas necesarias para conseguir los listados requeridos.

SOLUCIÓN 1

USUARIOS

db.usuarios.insertOne({

\_id: 1,

nombre: "Usuario 1",

rol: "cliente",

sexo: "masculino",

fecha\_nacimiento: ISODate("1990-01-15"),

enfermedad: {

categoria: "Categoría 1",

nombre: "Enfermedad 1",

descripcion: "Descripción de la enfermedad"

}

})

SESIONES

db.sesiones.insertOne({

\_id:1,

usuario\_id: 1,

fecha\_grabacion: ISODate("2022-10-26T14:01:00.000Z"),

ruta\_audio\_externo: "ruta\_del\_audio\_externo",

velocidad\_grabacion: "normal"

})

TEXTO

db.textos.insertOne({

\_id: 1,

categoria: "Saludos",

texto: "Hola, ¿cómo estás?",

numero\_orden: 1,

estado\_animo: "amigable",

sesion\_id: 1

})

LISTADOS:

**A**

db.usuarios.aggregate([

{

$match: {

"enfermedad.nombre": "Enfermedad 1"

}

},

{

$group: {

\_id: {

sexo: "$sexo",

edad: {

$subtract: [

{ $year: new Date() },

{ $year: "$fecha\_nacimiento" }

]

}

},

cantidad: { $sum: 1 }

}

}

])

**B**

db.sesiones.aggregate([

{

$match: {

usuario\_id: 1,

},

},

{

$lookup: {

from: "textos",

localField: "\_id",

foreignField: "sesion\_id",

as: "textos",

},

},

])

**C**

db.usuarios.aggregate([

{

$match: {

"enfermedad.nombre": "Enfermedad 1",

},

},

{

$lookup: {

from: "sesiones",

localField: "\_id",

foreignField: "usuario\_id",

as: "sesiones",

},

},

{

$lookup: {

from: "textos",

localField: "sesiones.\_id",

foreignField: "sesion\_id",

as: "textos",

}

}

]

SOLUCIÓN 2

USUARIOS

db.usuarios.insertOne({

\_id: 1,

nombre: "Usuario 1",

rol: "cliente",

sexo: "masculino",

fecha\_nacimiento: ISODate("1990-01-15"),

enfermedad: {

categoria: "Categoría 1",

nombre: "Enfermedad 1",

descripcion: "Descripción de la enfermedad"

}

})

SESIONES

db.sesiones.insertOne({

\_id:1,

usuario\_id: 1,

fecha\_grabacion: ISODate("2022-10-26T14:01:00.000Z"),

ruta\_audio\_externo: "ruta\_del\_audio\_externo",

velocidad\_grabacion: "normal",

textos: [

{

\_id: 1,

categoria: "Saludos",

texto: "Hola, ¿cómo estás?",

numero\_orden: 1,

estado\_animo: "amigable",

sesion\_id: 1

}

]

})

LISTADOS:

**A**

db.usuarios.aggregate([

{

$match: {

"enfermedad.nombre": "Enfermedad 1"

}

},

{

$group: {

\_id: {

sexo: "$sexo",

edad: {

$subtract: [

{ $year: new Date() },

{ $year: "$fecha\_nacimiento" }

]

}

},

cantidad: { $sum: 1 }

}

}

])

**B**

db.sesiones.aggregate([

{

$match: {

usuario\_id: 1,

},

}

])

**C**

db.usuarios.aggregate([

{

$match: {

"enfermedad.nombre": "Enfermedad 1",

},

},

{

$lookup: {

from: "sesiones",

localField: "\_id",

foreignField: "usuario\_id",

as: "sesiones",

},

}

]

### EJERCICIO 7

En una máquina virtual con Ubuntu (adaptador de red puente):

1. Instalar MongoDB con Docker (tienes los pasos a seguir al final del enunciado)
2. Crea en Ubuntu el fichero ejemplo.sh con el siguiente contenido:

use mi\_basededatos

db.createCollection("usuarios")

db.createCollection("coches")

db.usuarios.insertMany([

{

nombre: "Usuario 1",

edad: 30,

correo: "usuario1@example.com"

},

{

nombre: "Usuario 2",

edad: 25,

correo: "usuario2@example.com"

},

{

nombre: "Usuario 3",

edad: 35,

correo: "usuario3@example.com"

}

])

db.coches.insertMany([

{

marca: "Toyota",

modelo: "Corolla",

año: 2020

},

{

marca: "Ford",

modelo: "Mustang",

año: 2019

},

{

marca: "Honda",

modelo: "Civic",

año: 2021

}

])

1. Carga el fichero anterior en MongoDB (mongosh < ejemplo.sh)
2. Haz alguna consulta con mongosh (dentro de la terminal del contenedor)
3. Desde MongoDB Compass de tu equipo conéctate al servidor de MongoDB que tienes en el contenedor Docker
4. Haz alguna consulta con MongoDB Compass
5. Reiniciar la máquina y comprueba que al lanzar de nuevo el servidor MongoDB los datos siguen estando

Instalar docker:

*sudo apt update*

*sudo apt install docker.io*

Lanzar contenedor con MongoDB:

*sudo docker run -p 192.168.0.104:27017:27017 --name iabd-mongo -d mongo*

* **docker run**: comando para crear y ejecutar un contenedor Docker.
* **-p 192.168.0.104:27017:27017**: Este parámetro indica la asignación de puertos. Estás mapeando el puerto 27017 del sistema host con la dirección IP 192.168.0.104 (tendrás que usar la IP de tu equipo) al puerto 27017 dentro del contenedor. Esto permite acceder a MongoDB desde el sistema host a través de la dirección IP 192.168.0.104 y el puerto 27017.
* **--name iabd-mongo**: asigna el nombre "iabd-mongo" al contenedor.
* **-d mongo**: indica que el contenedor se basará en la imagen "mongo".

*sudo docker exec -it iabd-mongo bash*

* **docker exec**: comando para ejecutar un comando en un contenedor Docker en ejecución.
* **-it**: Estos son indicadores que se utilizan para abrir una sesión interactiva en el contenedor. El indicador **-i** permite una interacción estándar, y el indicador **-t** asigna un terminal al contenedor.
* **iabd-mongo**: Es el nombre del contenedor en el que deseas ejecutar el shell interactivo.
* **bash**: Es el comando que se ejecutará dentro del contenedor. En este caso, estás ejecutando el shell Bash, lo que te permite interactuar con el sistema dentro del contenedor.

Ya tenemos mongodb listo para ser utilizado

Otros comandos que puedes necesitar:

* **sudo docker cp prueba.js iabd-mongo:/tmp**: copia un archivo llamado "prueba.js" desde tu sistema host al contenedor Docker "iabd-mongo" en la ubicación "/tmp" dentro del contenedor.
* **mongosh < /tmp/prueba.js**: ejecuta en mongosh las instrucciones del fichero. Para hacerlo desde host o desde el virtual, instalar en mongosh o mongodb.
* **sudo docker stop/restart/start iabd-mongo**: para/reinicia/inicia el contenedor iabd-mongo