**createCollection**

**db.createCollection("nameCollection");**

Crea una colección en la base de datos, como argumento se da el nombre de la colección.

Opcionalmente esta operación recibe objetos de configuración donde se pueden agregar validaciones y restricciones a la colección.

**Ejemplo:**

**db.createCollection("users", {  
 validator: {  
 $jsonSchema: {  
 bsonType: "object",  
 required: ["name", "age", "created\_at"],  
 properties: {**

**name: {  
 bsonType: "string",  
 description: "Name of the user"  
 },**

**age: {  
 bsonType: "int",  
 minimum: 1,   
 maximum: 100,  
 description: "Age of the user"  
 },**

**create\_at: {  
 bsonType: "date",  
 description: "Date related to the user"  
 },**

**}  
 }  
 }  
});**

**Find**

Para buscar documentos en la base de datos usamos el metodo **find,** el metodo tiene diferentes formas de búsqueda filtrada, cambiando su resultado usando diferentes operadores o argumentos.

Para enlistar todos los elementos de una colección usamos find y pasamos un objeto vacio como argumento:

**db.users.find({});**

Si quiere buscar todos los usuarios cuya edad sea 25:

**db.users.find({age:25});**

**insertOne and insertMany**

Para insertar un documento a la colección se usa el método **insertOne**, como argumento se indica la estructura a guardar.

**Ejemplo:**

**db.users.insertOne({  
 name: "Julian",  
 age: 45,  
 created\_at: new Date(),  
});**

Para insertar múltiples documentos se usa **insertMany,** como argumento se da un array de estructuras a guardar.

**Ejemplo:**

**db.users.insertMany([  
 {  
 name: "Franco",  
 age: 20,  
 created\_at: new Date(),  
 },  
 {  
 name: "Juan",  
 age: 40,  
 created\_at: new Date(),  
 }  
]);**

**ObjetId**

Este campo tiene el propósito de indexación automática en búsquedas y de diferenciar de forma predeterminada cada documento. El campo tiene como nombre **“\_id”**

**{  
 "\_id" : ObjectId("65c55b7bf7c3603a6a6a1351"),  
 "name" : "Diego",  
 "age" : NumberInt(25),  
 "created\_at" : ISODate("2024-02-08T22:53:47.238+0000"),  
}**

**Operadores MongoDB**

**Operadores de comparación:**

Estos operadores nos ayudan a evaluar las condiciones específicas que queremos aplicar a nuestros datos para filtrar y encontrar los documentos deseados en nuestra base de datos.

* **$eq (Igual que):** Se utiliza para encontrar documentos con un campo que sea igual a un valor específico:

**db.users.find({ age: { $eq: 25 } })**

* **$ne (Distinto de):** Encuentra documentos donde un campo sea distinto de un valor específico:

**db.users.find({ age: { $ne: 25 } })**

* **$gt (Valores mayores que):** Encuentra documentos donde un campo sea mayor que un valor específico. **($gte es mayor o igual que)**:

**db.users.find({ age: { $gt: 25 } })**

* **$lt (Valores menores que):** Encuentra documentos donde un campo sea menor que un valor específico. **($lte es menor o igual que):**

**db.users.find({ age: { $lt: 25 } })**

* **$in (Valores que se encuentran en un array de elementos):** Encuentra documentos donde un campo coincida con cualquier valor en un array especificado:

**db.users.find({ age: { $in: [25, 30, 35] } })**

* **$nin (Valores que NO se encuentran en un array de elementos):** Encuentra documentos donde un campo no coincida con ningún valor en un array especificado:

**db.users.find({ age: { $nin: [25, 30, 35] } })**

**Operadores de elementos:**

Son útiles para verificar la existencia de campos y para comprobar sus tipos de datos. Aquí están los operadores de elementos admitidos:

* **$exists:** Este operador comprueba si un campo existe en los documentos de la colección.

**db.users.find({ age: { $exists: true } })**

* **$type:** Este operador comprueba el tipo de dato de un campo en los documentos de la colección.

**db.users.find({ name: { $type: "string" } })**

**Operadores Logicos:**

Estos operadores son fundamentales para construir consultas más complejas y precisas (OR and AND)

* **$or (O):** Filtra documento que cumpla alguna de las condiciones descrita en un array de elementos.

**db.users.find({$or: [{ name: "Diego" }, {name: "Franco"}]});**

* **$and (Y):** Filtra documentos que cumplan todas las condiciones descritas en un array de elementos.

**db.users.find({$and: [{ name: "Diego" }, { age: { $gte: 25 } }]});**

**updateOne and updateMany**

Para actualizar un documento de nuestra colección, se usa el método **updateOne**. Como argumento se le indica el filtro de búsqueda más los elementos a actualizar con el operador **$set**. El operador $set actualiza un campo existente o agrega uno nuevo en caso de que el campo no exista.

**Ejemplo:**

**db.users.updateOne(  
 {\_id:ObjectId("62c0946602d55365f6a8b737")},  
 {$set: { name: "nuevoNombre" }}  
);**

Para actualizar múltiples documentos de nuestra colección, se usa el método updateMany. Como argumento se le indica el filtro de búsqueda más los elementos a actualizar con el operador $set. Este método actualiza todas las coincidencias.

**Ejemplo:**

**db.users.updateMany(  
 {},  
 {$set: { name: "ahoraTodosSeLlamaranAsi " }}  
);**

En el caso que necesitemos eliminar una propiedad existente, lo podemos hacer mediante el operador $unset.

**Ejemplo:**

**db.users.updateOne(  
 {\_id:ObjectId("62c0946602d55365f6a8b737")},  
 {$unset: { "email": "" }}  
);**

**-----------------------------------------------------------------------------------------------------------------------------------**

**deleteOne and deleteMany**

Para eliminar un documento de nuestra colección, se usa el método deleteOne . Como argumento se le indica el filtro de búsqueda.

**Ejemplo:**

**db.users.deleteOne({\_id: ObjectId("62c0957402d55365f6a8b739")});**

Para eliminar múltiples documentos de nuestra colección, se usa el método deleteMany. Como argumento se le indica el filtro de búsqueda.

**Ejemplo:**

**db.users.deleteMany({email: "all@update.com"});**