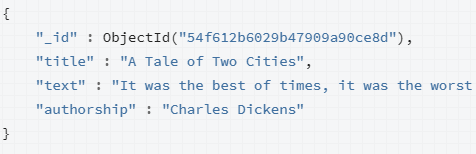


1 Introducción

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 Windows, Linux, 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 Binary JSON que permite búsquedas rápidas de datos pero en la práctica nosotros solo veremos un documento JSON tanto al almacenar como al consultar.



*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 generar el mismo valor para la clave de un documento. Este “\_id” es el único obligatorio para un documento.

2. Ejercicios

Carga los siguientes documentos y realiza los ejercicios:

[

{

"nombre":"Pablo",

"apellido":"López",

"edad":28,

"asignatura":"Matemáticas"

},

{

"nombre":"Pablo",

"apellido":"Gómez",

"edad":30,

"asignatura":"Lengua Castellana"

},

{

"nombre":"Juan",

"apellido":"Rodríguez",

"edad":40,

"asignatura":"Tecnología"

}

]

Listar todos los profesores

| ***db["profesores"].find().pretty()*** |
| --- |

Insertar un profesor y una asignatura en la colección profesores

| ***db.profesores.insertOne({nombre: "Hugo", apellido: "Peralta", edad: 21, asignatura: "Ciencia de la computacion"})*** |
| --- |

Actualizar la asignatura del profesor que acabáis de insertar

| ***db.profesores.updateOne({nombre:"Hugo"},{$set:{asignatura:"Programación"}})*** |
| --- |

Borrar el profesor que se acaba de insertar

| ***db.profesores.deleteOne({nombre: "Hugo"})*** |
| --- |

*2.2.4. Añadir campos a un documento de MongoDB*

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

| ***db.profesores.updateMany({}, {$inc:{edad: 1}})*** |
| --- |

Ejercicio: Usar **upsert** para incorpora un nuevo documento en caso de que no encuentre al profesor "Aitor" informando el campo "Personal":"Fijo"

| ***db.profesores.updateMany({nombre: "Aitor"}, {$set: {Personal: "Fijo"}}, {upsert:true})*** |
| --- |

*2.2.5. Filtrar, Ordenar y Contar mediante consultas*

Ejercicio: Listar los profesores mostrando solo los nombres:

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

Ejercicio: Ordenar por nombre de forma ascendente:

| ***db.profesores.find({}).sort({nombre: 1})*** |
| --- |

Ejercicio: Ordenar por nombre de forma descendente:

| ***db.profesores.find({}).sort({nombre: -1})*** |
| --- |

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

| ***db.profesores.find().limit(2)*** |
| --- |

Ejercicio: Limitar a 2 intercalando 1 salto entre resultados

| ***db.profesores.find().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"})

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}})

Para hacer los siguientes ejercicios prácticos cargamos una nueva colección:

* asignaturas

db.asignaturas.insertMany(

[

{

"nombre":"Matematicas",

"curso":"1º",

"alumnos":28,

"profesor":"Pedro",

"ingles": false

},

{

"nombre":"Sociales",

"curso":"2º",

"alumnos":20,

"profesor":"María",

"ingles":true

},

{

"nombre":"Naturales",

"curso":"2º",

"alumnos":25,

"profesor":"Juan",

"ingles": true

},

{

"nombre":"Educacion Fisica",

"curso":"3º",

"alumnos":23,

"profesor":"Laura",

"ingles": false

}

]

)

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”})*** |
| --- |

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"}}})

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”}]})*** |
| --- |

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

| ***db.asignaturas.find({profesor: {$nin: [“Juan”, “Laura”]}})*** |
| --- |

* 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":{$type:"int"}})
  + db.asignaturas.find({"alumnos":{$type:"string"}})

Ejercicio:

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

| ***db.asignaturas.find({$or: [{alumnos: {$type: 16}}, {alumnos: {$type: 2}}]})*** |
| --- |

2.2.8 – Operadores de evaluación de consulta

Creamos una nueva colección “productos”:

[

{

"nombre": "MERMELADA DE ALBARICOQUE",

"precioCompra": 0.9,

"pvp": 2.3,

"stock": 40

},

{

"nombre": "TURRÓN BLANDO ALMENDRA",

"precioCompra": 1.75,

"pvp": 2.45,

"stock": 150

},

{

"nombre": "PAN DE MOLDE INTEGRAL",

"precioCompra": 0.95,

"pvp": 1.8,

"stock": 70

},

{

"nombre": "CHOCOLATE VALOR 150GR",

"precioCompra": 1.9,

"pvp": 2.6,

"stock": 20

},

{

"nombre": "TOFU CON SÉSAMO",

"precioCompra": 1.6,

"pvp": 2.85,

"stock": 40

},

{

"nombre":"CROQUETAS DE SETAS GOURMET",

"precioCompra": 0.95,

"pvp": 1.5,

"stock": 2

},

{

"nombre":"PATATAS FRITAS ONDULADAS",

"precioCompra": 0.37,

"pvp": 0.67,

"stock": 90

}

]

Con esta expresión miramos que se cumpla la condición: si la cantidad es superior a 50 entonces divide el precio entre 2, sino entre 4.

db.Productos.updateMany( {},

{

$set: {

precioCompra: {

$cond: {

if: { $gte: ["$stock", 50] },

then: { $divide: ["$precioCompra", 2] },

else: { $divide: ["$precioCompra", 4] }

}

}

}

}

)

Ahora vamos a ver algunos ejemplos con **$regex:**

Supongamos que tenemos una colección llamada “catalogo”, con un campo “sku” con un código único de producto.

* 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 } })
* El modificador i hace que la búsqueda sea insensible a mayúsculas y minúsculas. Así, coincidirá con valores que empiecen con “ABC”, “abc”, “Abc”, etc.
* Con la opción m tiene en cuenta el salto de línea
  + db.catalogo.find({ "descripcion": { $regex: /^S/m } })
* Ejemplo de coincidencias:
  + "Samsung Galaxy\nDescripción detallada" -> Coincide (la primera línea comienza con “S”).
  + "Producto\nSin descripción" -> Coincide (la segunda línea comienza con “S” debido al modo multilínea).
  + "LG Smartphone" -> No coincide (no comienza con “S” y solo tiene una línea).

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

**El operador $text en MongoDB se utiliza para realizar búsquedas de texto en documentos dentro de una colección. Es especialmente útil cuando se necesitan realizar búsquedas de palabras clave o frases en campos de texto, como títulos, descripciones o cualquier campo textual de los documentos.**

Primero, creamos una colección llamada “articulos”:

[

{

"\_id": 1,

"subject": "coffee",

"author": "xyz",

"views": 50 },

{

"\_id": 2,

"subject": "Coffee Shopping",

"author": "efg",

"views": 5 },

{

"\_id": 3,

"subject": "Baking a cake",

"author": "abc",

"views": 90 },

{

"\_id": 4,

"subject": "baking",

"author": "xyz",

"views": 100 },

{

"\_id": 5,

"subject": "Café Con Leche",

"author": "abc",

"views": 200 },

{

"\_id": 6,

"subject": "Сырники",

"author": "jkl",

"views": 80 },

{

"\_id": 7,

"subject": "coffee and cream",

"author": "efg",

"views": 10 },

{

"\_id": 8,

"subject": "Cafe con Leche",

"author": "xyz",

"views": 10 }

]

* Para poder usar $text en consultas, necesitas crear un índice de texto en los campos que deseas buscar. MongoDB requiere este índice para optimizar las búsquedas de texto, y puedes crear el índice en uno o varios campos.
  + db.articulos.createIndex({subject:"text"})

Ejercicio: buscar "baking" en la colección artículos

| ***db.articulos.find({subject: /baking/i})*** |
| --- |

Ejercicio: buscar "baking" y "coffee" en un mismo comando

| ***db.articulos.find({subject: {$in: [/baking/i, /coffee/i ]}})*** |
| --- |

Ejercicio: buscar "baking" excluyendo "coffee" en un mismo comando

| ***db.articulos.find({ $text: { $search: "baking -coffee" } })*** |
| --- |

Ejercicio: buscar la frase completa "Cafe 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

|  |
| --- |

2.9 – Operadores con arrays

Para ver algunos ejemplos cargamos una nueva colección “inventario”:

[

{

"\_id":1,

"sku": "abc123",

"etiquetas": ["escolar","libro", "bolsa", "audífonos", "cuaderno"],

"cantidad": [

{"tamano":"S", "num":10, "color": "azul"},

{"tamano":"M", "num":45, "color": "azul"},

{"tamano":"L", "num":100, "color": "verde"}

]

},

{

"\_id":2,

"sku": "abc456",

"etiquetas": ["cuaderno","escolar", "libro"],

"cantidad": [

{"tamano":"6", "num":100, "color": "verde"},

{"tamano":"6", "num":50, "color": "azul"},

{"tamano":"8", "num":100, "color": "marrón"}

]

},

{

"\_id":3,

"sku": "abc567",

"etiquetas": ["escolar","libro"],

"cantidad": [

{"tamano":"S", "num":10, "color": "azul"},

{"tamano":"M", "num":100, "color": "azul"},

{"tamano":"L", "num":100, "color": "verde"}

]

},

{

"\_id":4,

"sku": "abc890",

"etiquetas": ["electronico","escolar"],

"cantidad": [

{"tamano":"M", "num":100, "color": "verde"}

]

}

]

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

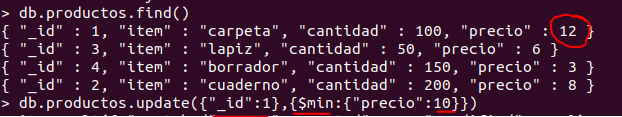
2.10 – Comando $where

* 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”})

2.11 – 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

|  |
| --- |