



MAINT[®]

TALLER DE CREACIÓN DE UNA RED SOCIAL CON REACT, REDUX, NODEJS, EXPRESS Y MONGODB

Danny Vaca

Índice

1.	Objetivo general.....	2
2.	Objetivo específico.....	2
3.	Desarrollo.....	2
3.1.	Eliminar un registro	Error! Bookmark not defined.
3.2.	Eliminar múltiples registros	Error! Bookmark not defined.
3.3.	Actualizar un registro.....	Error! Bookmark not defined.
3.4.	Actualizar múltiples registros	Error! Bookmark not defined.
3.4.1.	Agregar campos a un documento	Error! Bookmark not defined.
3.4.2.	Eliminar campos de un documento	Error! Bookmark not defined.

1. Objetivo general

Realizar consultas sobre documentos anidados en la base de datos

2. Objetivo específico

- Realizar una conexión desde NodeJS hasta mongo
- Realizar consultas sobre documentos anidados en la base de datos

3. Desarrollo

La presente práctica se desarrollará a partir de la práctica 2.

3.1. Búsqueda de coincidencias en sub documentos

Las bases de datos SQL relacionales permiten agrupar información mediante relaciones como las claves foráneas, es decir creamos una segunda tabla con información y mediante un campo común entre ambas tablas las relacionamos, esto es así debido a que las bases SQL poseen una estricta estructura, pero las bases NoSQL ofrecen la opción de anidar documentos, es decir un documento dentro de otro documento, ya que no existen rigurosas estructuras que no permitan hacer eso, de hecho esta es la mejor opción para hacer relaciones.

Es importante optimizar las consultas cuando se realizan anidaciones, es decir, si estamos desarrollando la base de datos de una red social, vamos a tener una tabla de usuarios y cada usuario va a tener posts, un usuario puede llegar a tener cientos de miles de posts, por lo que si en una consulta deseamos traer el nombre del usuario y no limitamos los campos nos traeremos todos los post, lo cual es altamente ineficiente.

En nuestra tabla de futbolistas los futbolistas tienen diferentes equipos en los que han jugado, pero queremos hacer una consulta donde extraigamos los futbolistas que han jugado en

cierto equipo, podemos hacer la búsqueda con la notación de “.” como si fuera una más de las condiciones de la consulta mediante la función **find**.

```
JS create.js JS index.js x
1  const mongodb = require('mongodb')
2  const MongoClient = mongodb.MongoClient
3  const url = 'mongodb://localhost:27017/testdatabase'
4
5  MongoClient.connect(url, { useNewUrlParser: true }, function(error, connection) {
6    if (error) throw error
7    const database = connection.db()
8    const conditions = {'teams.name': 'FC Barcelona'}
9    database.collection('players').find(conditions).toArray(function(error, player) {
10      if (error) throw error
11      console.log(player)
12      connection.close()
13    })
14  })
15  |
```

Es importante resaltar que la respuesta que me devuelve mongo, es el documento y todos los objetos del array, lo cual no es óptimo si solo se desea ver ese registro en particular y el usuario tiene cientos de miles de sub documentos.

```
[ { _id: 5bfc6b60b2d5a21444f3302c,
  name: 'Lionel Messi',
  country: 'Argentina',
  age: 31,
  teams: [ [Object] ] },
  { _id: 5bfc6b60b2d5a21444f3302e,
  name: 'Neymar Jr',
  country: 'Brazil',
  age: 26,
  teams: [ [Object], [Object], [Object] ] } ]
```

También se puede usar la función **findOne** para traer un solo registro con una búsqueda de un subdocumento, se puede realizar la búsqueda con parámetros que estén dentro del sub documento o en el documento padre, la función **find** o **findOne** devolverá los documentos que hagan match con esas condiciones.

```
JS create.js JS index.js x
1  const mongodb = require('mongodb')
2  const MongoClient = mongodb.MongoClient
3  const url = 'mongodb://localhost:27017/testdatabase'
4
5  MongoClient.connect(url, { useNewUrlParser: true }, function(error, connection) {
6    if (error) throw error
7    const database = connection.db()
8    const conditions = {
9      _id: new mongodb.ObjectId('5bfc76b6d084ab24f0a58e2a'),
10     'teams.name': 'FC Barcelona'}
11    database.collection('players').findOne(conditions, function(error, player) {
12      if (error) throw error
13      console.log(player)
14      connection.close()
15    })
16  })
17
```

De la misma forma me trae todos los sub documentos que coinciden con los criterios de búsqueda.

```
{ _id: 5bfc76b6d084ab24f0a58e2a,
  name: 'Neymar Jr',
  country: 'Brazil',
  age: 26,
  teams:
    [ { name: 'Santos', years: 3 },
      { name: 'FC Barcelona', years: 3 },
      { name: 'Paris Saint Germain', years: 2 } ] }
```

3.2. Mostrar solo subdocumento que coincida.

Cuando el documento contiene cientos de subdocumentos es muy ineficiente que la consulta nos devuelva todos los registros, ante esta problemática vamos usar las proyecciones “**projection**” para limitar que documentos se desea ver.

Se puede enviar como segundo argumento de la función **findOne** un objeto el cual puede contener un campo **projection** con el cual se puede escoger que campos se desea traer y cuales no mediante un 1 y un 0 respectivamente, pero también podemos usar el operador **\$elementMatch** para mostrar solo los campos que coincidan con cierto parámetro de búsqueda, por ejemplo, de los campos que coinciden con el **_id** igual a **5bfc76b6d084ab24f0a58e2a** se va a

Elaborado por:

Ing. Danny Vaca

Fecha de revisión:

5 noviembre del 2018

traer el campo **teams**, solamente los que tengan el nombre “**name**” exactamente igual a “**FC Barcelona**”

```
5 MongoClient.connect(url, { useNewUrlParser: true }, function(error, connection) {
6   if (error) throw error
7   const database = connection.db()
8   const conditions = { _id: new mongodb.ObjectId('5bfc76b6d084ab24f0a58e2a') },
9   const fields = {
10     projection: {
11       teams: {
12         $elemMatch: { 'name': 'FC Barcelona' }
13       }
14     }
15   }
16   database.collection('players').findOne(conditions, fields, function(error, player) {
17     if (error) throw error
18     console.log(player)
19     connection.close()
20   })
21 })
```

De esta forma se puede escoger que campos y que sub documentos se desea traer en la consulta, evitando así la carga de datos innecesarios en una consulta.

```
{ _id: 5bfc76b6d084ab24f0a58e2a,
  teams: [ { name: 'FC Barcelona', years: 3 } ] }
```

3.3. Insertar y eliminar subdocumentos

En estos array de documentos dentro de un documento es importante poder insertar y eliminar documentos, para eso tenemos que usar la función **update** o **updateOne** para actualizar el documento padre y los operadores **\$push** y **\$pull** para insertar y eliminar respectivamente sub documentos.

3.3.1. Insertar sub documentos

Par insertar un subdocumento debemos actualizar el documento padre, indicando que queremos meter un nuevo documento en su interior mediante el modificador **\$push**, el cual recibe un objeto con el nombre del campo sobre el que se desea insertar y como valor el objeto que se desea insertar.

```
MongoClient.connect(url, { useNewUrlParser: true }, function(error, connection) {
  if (error) throw error
  const database = connection.db()
  const conditions = {_id: new mongodb.ObjectId('5bfc76b6d084ab24f0a58e2f')}
  const fields = {
    $push: {
      teams: [
        {
          name: 'Ajax',
          years: '2'
        }
      ]
    }
  }
  database.collection('players').updateOne(conditions, fields, function(error, player) {
    if (error) throw error
    console.log(player)
    connection.close()
  })
})
```

Si realizamos una consulta sobre la colección players podemos observar que se ha actualizado el registro insertándole un nuevo campo.

```
> db.players.find({_id: ObjectId('5bfc76b6d084ab24f0a58e2f')}).pretty()
{
  "_id" : ObjectId("5bfc76b6d084ab24f0a58e2f"),
  "name" : "Luis Suarez",
  "country" : "Uruguay",
  "age" : 31,
  "teams" : [
    {
      "name" : "Ajax",
      "years" : "2"
    }
  ]
}
```

3.3.2. Eliminar subdocumentos

Par eliminar un subdocumento debemos actualizar el documento padre, indicando que queremos quitar un documento de su interior mediante el modificador **\$pull**, el cual recibe un objeto con el nombre del campo del que se desea eliminar el campo y como valor el objeto que debe hacer match con el campo a eliminar.

Elaborado por:

Ing. Danny Vaca

Fecha de revisión:

5 noviembre del 2018

```
3 const url = 'mongodb://localhost:27017/testdatabase'
4
5 MongoClient.connect(url, { useNewUrlParser: true }, function(error, connection) {
6   if (error) throw error
7   const database = connection.db()
8   const conditions = {_id: new mongodb.ObjectId('5bfc76b6d084ab24f0a58e2f')}
9   const fields = {
10     $pull: {
11       teams: {
12         name: 'Ajax'
13       }
14     }
15   }
16   database.collection('players').updateOne(conditions, fields, function(error, player) {
17     if (error) throw error
18     console.log(player)
19     connection.close()
20   })
21 })
```

Si realizamos una consulta sobre la colección `players` podemos observar que se ha actualizado el registro eliminado el documento que coincidía.

```
> db.players.find({_id: ObjectId('5bfc76b6d084ab24f0a58e2f')}).pretty()
{
  "_id" : ObjectId("5bfc76b6d084ab24f0a58e2f"),
  "name" : "Luis Suarez",
  "country" : "Uruguay",
  "age" : 31,
  "teams" : [ ]
}
```

Es importante destacar que no se elimina el campo **teams** cuando este queda vacío, si se lo desea eliminar se debe usar el modificador **\$unset**

```
5 MongoClient.connect(url, { useNewUrlParser: true }, function(error, connection) {
6   if (error) throw error
7   const database = connection.db()
8   const conditions = {_id: new mongodb.ObjectId('5bfc76b6d084ab24f0a58e2f')}
9   const fields = {
10     $unset: {
11       teams: 0
12     }
13   }
14   database.collection('players').updateOne(conditions, fields, function(error, player) {
15     if (error) throw error
16     console.log(player)
17     connection.close()
18   })
19 })
```

Elaborado por:

Ing. Danny Vaca

Fecha de revisión:

5 noviembre del 2018

Si realizamos una consulta sobre la colección `players` podemos observar que se ha actualizado el registro eliminado el campo **teams**.

```
db.players.find({'_id': ObjectId('5bfc76b6d084ab24f0a58e2f')}).pretty()

  "_id" : ObjectId("5bfc76b6d084ab24f0a58e2f"),
  "name" : "Luis Suarez",
  "country" : "Uruguay",
  "age" : 31
```

Elaborado por:

Ing. Danny Vaca

Fecha de revisión:

5 noviembre del 2018