Introducción a MongoDB

Luis Valencia Cabrera (lvalencia@us.es)

Research Group on Natural Computing
Departamento de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

13-12-2021, Bases de Datos

Índice

Índice

MongoDB

- SGBD multiplataforma NoSQL ***orientado a documentos***, de esquema libre (código abierto).
- Cada **registro** se denomina ***documento***.
- Los documentos se pueden agrupar en ***colecciones*** (como las tablas en una BD relacional, pero almacenando documentos muy diferentes, **sin esquema fijo**).
- Se pueden crear índices para atributos, manteniendo estructura interna eficiente para el acceso a los mismos.
- Destaca por su **velocidad** y su **rico pero sencillo sistema de consulta** de los contenidos de la base de datos.
- Buen equilibrio entre rendimiento y funcionalidad

Formato de los datos

- Los distintos **documentos** se almacenan en formato **BSON**, o ***Binary JSON***
- Versión modificada y enriquecida de JSON que permite búsquedas rápidas de datos.
- Guarda de forma explícita longitudes de campos, índices de arrays, y otra información útil para el escaneo de datos $\rightarrow$ ocupa más espacio que JSON.
- Conviene aprovechar el barato almacenamiento para ***incrementar en velocidad*** de localización de información dentro de un documento.

JSON

El formato JSON está muy ampliamente extendido para intercambio de información a través de web, con documentos que constan de pares clave-valor de distintos tipos.

```
{
  "name": "Alderaan",
  "rotation_period": 24,
  "diameter": 12500,
  "climate": "temperate",
  "terrain": "grasslands, mountains",
  "population": 2000000000,
  "residents": [
    "https://swapi.co/api/people/5/",
    "https://swapi.co/api/people/68/"
  ],
  "films": [
    "https://swapi.co/api/films/6/",
    "https://swapi.co/api/films/1/"
  ],
  "created": "2014-12-10T11:35:48.479000Z",
  "edited": "2014-12-20T20:58:18.420000Z",
  "url": "https://swapi.co/api/planets/2/"
}
```

Tipos de elementos JSON:

- Strings (cadenas de caracteres)
- Números (enteros o reales)
- Booleanos (`true` o `false`)
- Valor nulo (`null`)
- Arrays (listas ordenadas de elementos JSON)
- Objetos JSON (compuestos por sus propios conjuntos de pares clave-valor)

Tipos de valores en JSON

Lo anterior es un JSON `Object`. Pero no todo valor JSON es un objeto, veamos los posible tipos de valores[1]:

- JSON `Object`, caracterizado por sus llaves externas y las entradas para cada dato, separadas por comas. Estas entradas tienen una serie de claves cuyos valores pueden ser a su vez de cualquier tipo JSON
- JSON `Array`, conteniendo una serie de valores JSON del mismo tipo
- Cadenas (`String`)
- Valores numéricos (`Number`)
- Valores lógicos (`Bool`) → `true` o `false`
- Valor nulo (`Null`) → `null`

Puede encontrar todo tipo de ejemplos en Internet, tanto de archivos json[2] como de llamadas a API[3][4] que devuelven los datos en este formato.

[1] `https://www.json.org/`
[2] `http://json-schema.org/learn/miscellaneous-examples.html`
[3] `https://jsonapi.org/examples/`
[4] `https://github.com/endpoints/endpoints-example`

Descarga e instalación

- MongoDB es multiplataforma, luego la instalación dependerá de las características del S.O. destino. Optaremos por la versión *MongoDB community edition* 5.0.4 (*o la más reciente que haya - la 5.0.4 lo era el 04/12/2021*), de 64 bits.
- Podemos descargarla en este enlace. La instalación es distinta para cada sistema operativo:
  - Windows: seguir el asistente, quitar opción de instalar como servicio, añadir carpeta al Path y crear la carpeta `C:/data/db`.
  - Mac Os X: Instalar MongoDB en Mac, Installing MongoDB and related GUIs
  - Linux: Instalación en Ubuntu 20.04, Compass en Linux
- Existen también versiones de pago (*Enterprise*) y en la nube (*Atlas*). Haremos también alguna prueba con esta última.

Clientes para MongoDB

- Además del servidor, debemos disponer de clientes para interactuar con él
- Junto con el servidor, habremos instalado algunos clientes
  - `mongo`, programa que lanzamos desde la consola y con el que interactuamos mediante texto.
  - `MongoDBCompass`, interfaz gráfica.
- Recomendamos también instalar Robo3T, más amigable que la consola con un editor de texto en el que ir escribiendo una secuencia de instrucciones y lanzándolas contra el servidor. Ojo, descargar Robo3T, no Studio3T, que aparece también en el enlace anterior pero ese entorno es de pago, para uso profesional.

MongoDB Database Tools

- Hay programas asociados a MongoDB para tareas como importar datos, crear copias de seguridad, etc. que hasta ahora venían con MongoDB y ahora hay que instalar por separado. Son las `MongoDB Database Tools`.
- Podemos descargarlas en este enlace.
- Las instrucciones para su instalación se pueden consultar aquí.
- Además, en las versiones más recientes de MongoDB ya se recomienda emplear el nuevo shell de Mongo: `mongosh`, que se puede descargar en este enlace.

Primeros pasos con servidor y cliente

- El servidor de base de datos de MongoDB (contiene el SGDB) se lanza como un servicio, o bien mediante el programa ejecutable `mongod`.
- Por ejemplo, en Windows vamos a la línea de comandos (símbolo del sistema), navegamos hasta la carpeta `bin` dentro de la carpeta `mongodb`, y tecleamos `mongod`, lanzando así el servidor.
- El proceso para lanzar el cliente es muy similar. Abrimos otra línea de comandos, sin cerrar la del servidor, hacemos lo mismo que antes, y en el último paso lanzamos `mongo`, o si empleamos el nuevo programa de consola lanzamos en su lugar `mongosh`.
- En ese momento estamos listos para lanzar órdenes contra la base de datos.

Arrancando con la *shell*

- Para ver las bases de datos disponibles en nuestro servidor mongod, tecleamos `show dbs`.
- El comando `db` devuelve la bd actual.
- Podemos seleccionar una escribiendo: `use mydb`. Si no existe, se crea automáticamente al usarla **e insertar algo**, sin necesitar declaración explícita de creación.
- Se elimina una BD mediante `db.dropDatabase()`, y una colección individual mediante `db.micoleccion.drop()`.
- Si necesitamos acudir a la ayuda del sistema para consultar alguna funcionalidad podemos hacerlo mediante `help`, o a la ayuda de una determinada función con `db.nombrefuncion.help()`.

Consultar la referencia del lenguaje para más detalle.

Índice

Colecciones

- Una colección en MongoDB sería el *equivalente* a una tabla en modelo relacional.
- Podemos ver las existentes mediante `show collections`
- Para crear una colección, no es necesario definir explícitamente el nombre de la colección y su estructura o esquema; se generará y actualizará automáticamente conforme vayamos insertando **documentos**.
- Podemos añadir mediante `db.cosas.insert({v : 27})`.
- También podríamos hacer asignar datos a una variable (mediante Javascript), y posteriormente insertarla. Por ejemplo:

```
j = { name : "mongo" }
k = { x : 3, n : 5 }
db.cosas.insert( j )
db.cosas.insert( k )
```

Consultas

- Para realizar una consulta y poder así ver los documentos insertados, hacemos: `db.cosas.find()`, que devolverá todos los documentos de la colección cosas.
- Lo anterior devolverá los 3 registros introducidos, sin ningún criterio de filtro establecido, de forma similar a la siguiente:

```
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), "v" : 27 }
{ "_id" : ObjectId("4c2209f9f3924d31102bd84b"), "name" : "mongo" }
{ "_id" : ObjectId("4c2209fef3924d31102bd84c"), "x" : 3 , "n" : 5 }
```

- También podríamos consultar documentos por algún criterio, como por ejemplo haciendo: `db.cosas.find( { name : "mongo"} )`, que devolverá el documento cuyo name sea igual a "mongo".
- Para devolver un único documento: `db.cosas.findOne()`. Y para limitar el número: `db.cosas.find().limit(3)`.

Bucles e iteradores

- Podemos escribir bucles de tipo `for` o `while` desde el cliente de MongoDB. Por ejemplo, para añadir una serie de documentos en la colección cosas:

```
for (var i = 1; i <= 20; i++) db.cosas.insert( { x : 4 , j : i } )
```

- Ahora podemos consultar sus elementos: `db.cosas.find()`

  Esto mostrará un número máximo de documentos, por defecto 20, si bien ese límite se puede modificar; una vez mostrados los primeros podremos escribir `it` para ver más documentos.

- Podemos declarar una variable con el cursor y recorrer el iterador:

```
var c = db.cosas.find()
while ( c.hasNext() ) printjson( c.next() )
```

- También podemos acceder a una posición del array de resultados al que hace referencia la variable:

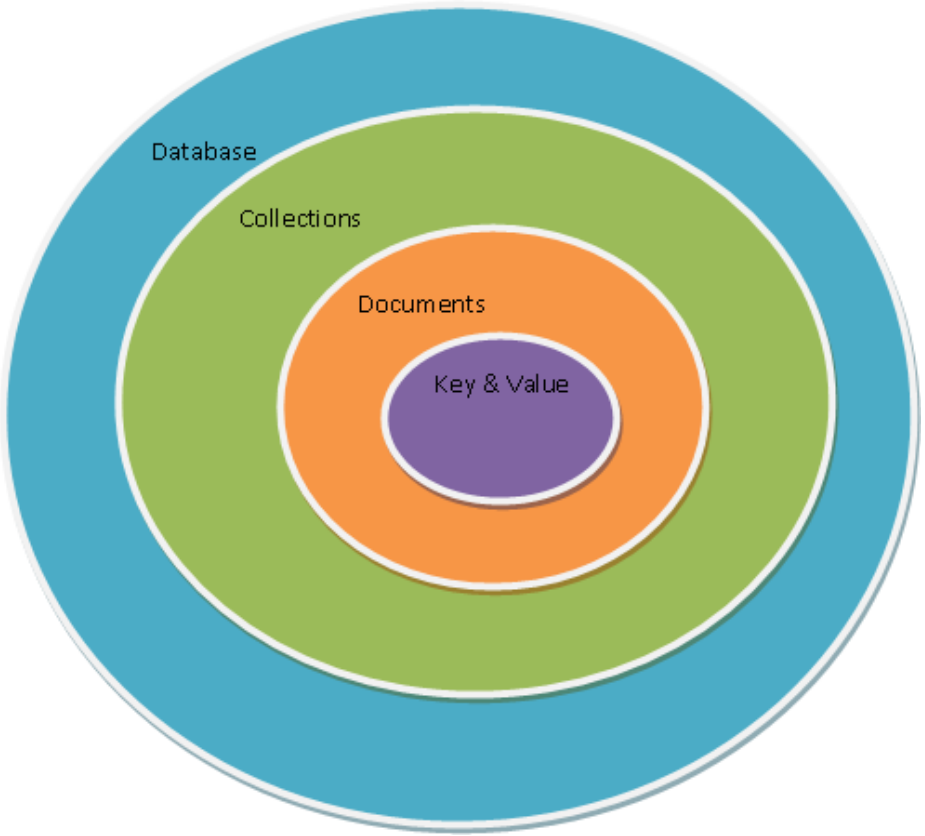
```
var c = db.cosas.find()
printjson( c [ 4 ] )
```

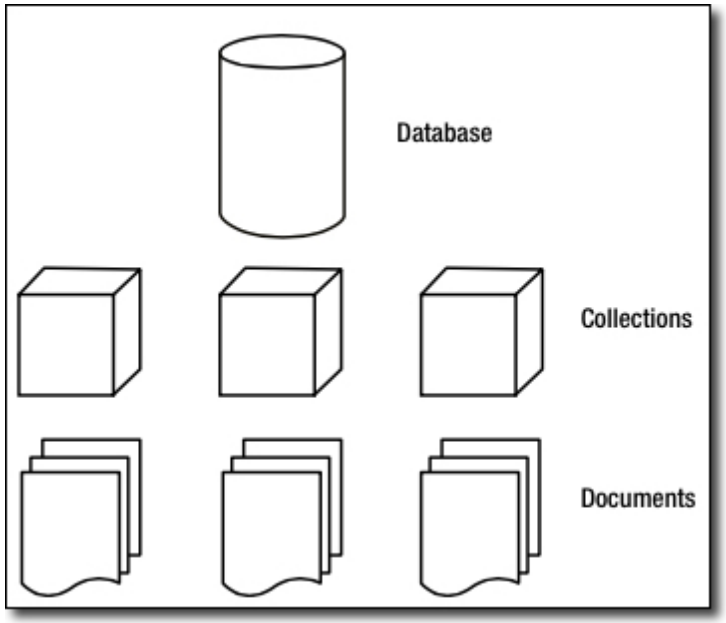
Próximos pasos

- Esta presentación proporciona únicamente una introducción a NoSQL y MongoDB.
- Encontraremos mucho más material en este curso completo, que revisaremos en las próximas diapositivas.
- Prestaremos especial atención a distintos aspectos:
  - Uso básico del shell, importación/exportación de datos
  - Inserción de datos mediante `insert`, actualización mediante `update`.
  - Consulta de documentos mediante `find`.
- Tenemos un resumen de funciones muy útil en este enlace.

NOTA (importante): *el contenido de las próximas diapositivas no es original, se toma del curso referenciado arriba; se han incluido aquí para que el alumnado tenga claro el material que entra en la asignatura*.

Estructuras en MongoDB: Vista general





[4] Imágenes tomadas de
http://expertojava.ua.es/si/nosql/nosql01.html#_mongodb

Importación de datos

Podemos importar datos de archivos JSON, CSV o TSV.
Centrándonos en la opción de JSON, podemos distinguir dos casos:

- Archivo con un objeto JSON por línea:

  ```
  mongoimport -d ejercicios -c ciudades --file mongo_cities1000.json
  ```

- Archivo con un array de objetos JSON:

  ```
  mongoimport -d ejercicios -c planetas --file planets.json --jsonArray
  ```

Si queremos que la importación elimine en su caso la colección preexistente, debemos añadir la partícula `--drop`.

Exportación de datos

Podemos exportar los datos a archivos JSON, de cualqeuira de los tipos anteriores:

- Archivo con un objeto JSON por línea:

  ```
  mongoexport -d ejercicios -c ciudades -o ciudades.json
  ```

- Archivo con un array de objetos JSON:

  ```
  mongoexport -d ejercicios -c planetas -o planetas.json --jsonArray
  ```

Copia de seguridad (backup)

Podemos hacer una copia de seguridad de la base de datos completa, y restaurarla posteriormente en el mismo u otro equipo:

- Backup: crea copia de seguridad en subcarpeta `dump`

  ```
  mongodump -d ejercicios
  ```

- Restore: restaura la base de datos con la información de la subcarpeta `dump`

  ```
  mongorestore --drop
  ```

Parámetros en consultas

Como patrón general, tenemos:
`db.micol.find(filtro,proyección)`, de modo que:

- El `filtro` debe ser un objeto JSON en el que se indiquen los criterios sobre los valores que deben tener los campos incluidos en el mismo.
- La `proyección`, en caso de proporcionarse, debe ser otro objeto JSON en el que se indique qué campos se desea incluir (poniéndoles el valor **`true`**) o excluir (asignándoles el valor **`false`**).

Por ejemplo, podemos tener
`db.pobs.find({"prov": "Sevilla"},{"nomb": true, "_id": false})`.

Opciones de filtro

Formato:`{"clave1": criterio1, ..., "claveN": criterioN}`:

- Las claves se corresponden con campos de la colección a filtrar.
- Cada criterio puede ser:
  - Un valor concreto, que compararíamos por igualdad con el valor del campo en cada documento de la colección. Por ejemplo, `"Sevilla"` como criterio para la clave `"prov"`, quedando el par como `"prov": "Sevilla"`.
  - Una condición, como por ejemplo `{"$gt": 5000}` para la clave `"pobl"` sería: `"pobl": {"$gt": 5000}`.
- El objeto de filtro quedaría como:

  `{"prov": "Sevilla", "pobl": {"$gt": 5000}}`

De forma similar a `"$gt"` (mayor que), tenemos `"$lt"` (menor que), y sus correspondientes `"$gte"` y `"$lte"` (que incluyen la igualdad).

Otros operadores

Al mismo nivel de los anteriores, disponemos de operadores como:

- $ne (no igualdad): `db.grades.find({type:{"$ne":"quiz"}})`
- $exists (existencia):
  `db.grades.find({"score":{"$exists":true}})`
- $not (negación):
  `db.grades.find({score:{"$not": {"$mod": [5,0]}}})`
- $regex (expresión regular[5]):
  `db.people.find({nombre: {"$regex":/aitor/i}})`

[5] Similar a LIKE. Consulte esta chuleta sobre expresiones regulares

Condiciones adicionales (I)

Campos anidados:

```
db.catalogo.find({"precio":{$gt:10000},
                  "reviews.calificacion":{$gte:5}})
```

Condiciones compuestas:
- Tipo OR:

```
db.grades.find({ $or:[ {"type":"exam"}, {"score":{$gte:65}} ]})
db.grades.find({ $or:[ {"score":{$lt:50}}, {"score":{$gt:90}} ]})
```

- Tipo AND:

```
db.grades.find({ $and:[ {type:"exam"}, {score:{$gte:65}} ] })
```

Nótese que esto último es equivalente a

```
db.grades.find({ type:"exam", score:{$gte:65} })
```

Condiciones adicionales (II)

- NOR ($nor, negación de OR):

```
db.grades.find({ score:{$gte:65},
                 $nor:[ {type:"quiz"}, {type:"homework"} ] })
```

- IN ($in, coincidencia con algún valor del array dado):

```
db.grades.find({ type:{$in:["quiz","exam"]} })
```

- NIN ($nin, negación de IN).

```
db.grades.find({score:{$gte:65}, type:{$nin:["quiz","homework"]} })
```

Consultas sobre arrays

- ALL ($all, devuelve los documentos que contengan todos los valores del array dado en el array indicado como clave):

```
db.people.find({ amistades: {$all: ["Juan", "David"]},
                 hobbies: {$in: ["footing", "baloncesto"]} })
```

- $elemMatch, devuelve documentos que coincidan con las condiciones impuestas a objetos dentro de arrays:

```
db.scores.find({
  results: { $elemMatch: { product: "xyz", score: { $gte: 8 } } } })
```

- $size, filtra por tamaño del array:

```
db.people.find( {hobbies : {$size : 3}} )
```

- $slice, usado en la proyección, permite restringir el número de elementos del array que mostraremos

```
db.people.find( {hijos: {$gt:1}}, {hobbies: {$slice:2}} )
```

Distinct

Devuelve los distintos valores del campo indicado, aplicando el filtro establecido. Como patrón general, tenemos: `db.micol.distinct(campo,filtro)`.

```
> db.grades.distinct('type')
---> [ "exam", "quiz", "homework" ]

> db.grades.distinct('type', { score: { $gt: 99.9 } } )
---> [ "exam" ]
```

Count

Cuenta el número de documentos que cumplen el filtro indicado.
Vemos que admite distintos tipos de uso:

```
db.grades.count({type:"exam"})
db.grades.find({type:"exam"}).count()
db.grades.count({type:"essay", score:{$gt:90}})
```

Update

Permite actualizar uno o muchos documentos. Como patrón general, tenemos:
`db.micol.update(filtro,actualización,opciones)`, siendo el último parámetro opcional

```
db.people.update({nombre:"Steve Jobs"},
                 {$set: {nombre:"Domingo Gallardo", salario: 1000000}})
db.people.update({salario: {$ge: 80000}},
                 {$set: {sueldo:"Alto"}},
                 {multi: true})
```

Como observamos, el segundo ejemplo tiene una opción para que se haga actualización múltiple. Hoy día la función `update` se considera *deprecated* (obsoleta), por lo que se recomienda usar en su lugar las funciones `updateOne` y `updateMany`. Lo mismo ocurre con la función `insert` y las correspondientes funciones `insertOne` e `insertMany`.

Upsert

Inserta un nuevo documento cuando no se encuentra ningún resultado que cumpla el criterio de filtro. Como patrón general, tenemos:
`db.micol.update(filtro,actualización,opciones)`, siendo el último parámetro opcional

```
db.people.update(
  {nombre:"Domingo Gallardo"},
  {$set: {name:"Domingo Gallardo", twitter: '@domingogallardo'}},
  {upsert: true})
```

Operadores de actualización

- $set, modifica solamente lo indicado, no sustituyendo el documento completo:

```
db.people.update({nombre:"Aitor Medrano"},{ $set:{salario: 1000000} })
```

- $inc, incrementa el valor de la variable en la cantidad indicada:

```
db.people.update({nombre:"Aitor Medrano"},{ $inc:{salario: 1000} })
```

- $unset, elimina el campo indicado:

```
db.people.update({nombre:"Aitor Medrano"},{ $unset:{twitter: ''} })
```

Otros operadores: $mul, $min, $max, $currentDate, etc.[6]

[6] Más información en este enlace

Otras actualizaciones

- `findAndModify`, encuentra, modifica y devuelve el objeto (lo indicamos con el campo **`new`** a **`true`**):

```
db.grades.findAndModify({
query: { student_id: 0, type: "exam"},
update: { $inc: { score: 1 } },
new: true
})
```

- Renombrado de campos en update (por cada par de elementos pasados al `rename`, el primer campo es el nombre antiguo y el segundo es el nuevo):

```
db.people.update( { _id: 1 },
{ $rename: { 'nickname': 'alias', 'cell': 'movil' } } )
```

Actualización sobre arrays

Añadir elementos

Push: añadir a lista

- $push simple:

```
db.people.update({nombre: "Luis"}, {$push: {aficiones: "Cine"}})
```

- $push múltiple:

```
db.people.update(
  {nombre: "Luis2"},
  {$push: {aficiones: {$each : ["Música", "Juegos de mesa"]}}})
```

AddToset: añadir a conjunto

```
db.enlaces.update({titulo:"google"}, {$addToSet: {tags:"buscador"} } )
db.enlaces.update(
  {titulo:"google"},
  {$addToSet: {tags: { $each:["drive", "traductor"]}}})
```

Actualización sobre arrays

Eliminar elementos

Pull - `$pull` elimina un elemento del array, `$pullAll` elimina varios:

```
db.enlaces.update({titulo:"google"}, {$pull: {tags:"traductor"}})
db.enlaces.update({titulo:"google"},
                  {$pullAll: {tags:["calendario","email"]}})
```

Pop: elimina elementos por el principio (-1) o por el final (1):

```
db.enlaces.update({titulo:"google"},{$pop: {tags:-1}})
```

Actualización sobre arrays

Operación posicional

Modifica el elemento de una posición del array (el referenciado por el operador $)

- Ejemplo: dada una colección que incluye el elemento `{ "_id" : 1, "grades" : [ 80, 85, 90 ] }`, y ante la necesidad de actualizar el 80 por un 82:

```
db.students.update({ _id: 1, grades: 80}, {$set: {"grades.$": 82}})
```

- Ejemplo: dada la colección siguiente

```
{"_id" : 4, "grades" :
  [{grade: 80, mean: 75, std: 8},
   {grade: 85, mean: 90, std: 5},
   {grade: 90, mean: 85, std: 3}]}
```

y la necesidad de cambiar a 6 el campo `std` de las notas con `grade` a 85:

```
db.students.update({_id: 4, "grades.grade": 85 },
                   {$set: {"grades.$.std": 6}})
```

Remove

Permite eliminar uno o muchos documentos. Como patrón general, tenemos: `db.micol.remove(filtro)`.

```
db.people.remove({nombre:"Domingo Gallardo"})
```