UNIVERSIDAD
COMPLUTENSE
MADRID

MongoDB

Gestión de la Información en la Web
Enrique Martín - emartinm@ucm.es
Grados de la Fac. Informática

Datos básicos sobre MongoDB

MongoDB

- MongoDB (de la palabra "hu**mongo**us", enorme) es una de las bases de datos orientadas a documento más famosas.
- Características:
  - Orientada a **documento** en formato JSON
  - Código abierto (C++)
  - Surge en 2007
  - Soporta replicación y *sharding*.
  - Soporta consultas avanzadas mediante el *aggregation pipeline* y MapReduce.

MongoDB

- Carece de **esquema fijo**, por lo que una colección puede contener documentos de tipos diferentes:

```
{
  “name” : ”Pepe Perez”,
  “fnac” : “21/08/1999”,
  “gustos” : [“cine”, “deporte”],
  “web” : {“url” : “www...com”, “fecha” : “Mayo2009”}
}

{
  “id” : ”1002ASR”,
  “dir” : “/home/juan”,
  “path” : [“/usr/local/bin”, “/usr/bin”]
}
```

- Como se puede ver los documentos anteriores **no coinciden** en ninguno de los campos.

Propiedades

- **Distribución**: soportan de manera sencilla *sharding* y replicación. Dependiendo de la carga de trabajo suelen presentar 2 organizaciones:
  - **Lecturas mayoritarias**: utilizan únicamente replicación con modelo maestro-esclavo. Los nuevos nodos esclavo para lectura se sincronizan automáticamente y comienzan a servir datos.
  - **Lecturas y escrituras**: utilizan *sharding* combinado con replicación. Cada *shard* se almacenará en un conjunto de nodos réplica.

Propiedades

- **Disponibilidad:**
  - El nodo maestro se elige por votación entre los nodos.
  - El *sharding* se realiza por el contenido de algún campo. Puede ser manual o *autosharding* (se reconfigura dinámicamente para balancear la carga).
- **Consistencia**: en entornos distribuidos con replicación se utiliza un valor W indicando el número de nodos que deben confirmar una escritura.

Propiedades

- **Transacciones**: las modificaciones que involucran un solo documento se realizan de manera **atómica**.
- No existe ninguna garantía de atomicidad a la hora de modificar varios documentos.

Organización de MongoDB

MongoDB

- Soporta distintas **bases de datos** (similar a MySQL) y varias **colecciones** por base de datos:
  - MongoDB
    - Base de datos 1
      - Colección 1
      - ...
      - Colección k
    - Base de datos 2
    - ...

Instalación de MongoDB

- La instalación de MongoDB es bastante sencilla, únicamente es necesario descargarse los binarios de http://www.mongodb.org
- Existen dos archivos principales:
  - **mongod**: servidor de la base de datos.
  - **mongo**: cliente JavaScript que se conecta al servidor local.
- Además tiene conectores para diversos lenguajes: C++, Java, Python, PHP, Ruby, Scala...

Instalación de MongoDB

- Antes de lanzar el servidor es necesario que exista el directorio para almacenar los datos. Por defecto los datos se almacenan en `/data/db` pero se puede utilizar otro directorio utilizando el parámetro `--dbpath`:

```
$ mongod --dbpath /home/user/mongo
```

- Una vez que se ha iniciado el servidor se puede iniciar el cliente sin problema ejecutando `mongo`

Bases de datos y colecciones

- `> use pruebas`
  `switched to db pruebas`
  Elige la base de datos “pruebas” para trabajar.
  Se crea automáticamente si no existe.
- `> show dbs`
  `local 0.078125GB`
  `pruebas (empty)`
  `test (empty)`
  Muestra las bases de datos existentes.

Bases de datos y colecciones

- `> **show collections**`
  `system.indexes`
  `users`
  `users.vip`
  Muestra las colecciones dentro de la base de datos actual.

Uso de MongoDB: inserción, actualización y eliminación

Inserción

- Para insertar documentos en la base de datos se utiliza el comando **insert**:

```
> db.users.insert({name:"Pepe", nhijos:3})
WriteResult({ "nInserted" : 1 })

> db.users.insert({name:"Juan", web:"www.juan.com"})
WriteResult({ "nInserted" : 1 })

> db.users.insert({name:"Ana", gustos:["musica","p2p"]})
WriteResult({ "nInserted" : 1 })
```

Inserta 3 usuarios en la colección **users**: Pepe, Juan y Ana; cada uno con campos diferentes.

Inserción

- **Nota importante:** los nombres de campos son siempre cadenas de texto.
- Estos siempre deberían llevar comillas, pero si no se hace el cliente **mongo** las añadirá automáticamente.
- Por tanto

  ```
  {name:"Pepe", nhijos:3}
  ```

  será transformado a

  ```
  {"name":"Pepe", "nhijos":3}
  ```

Inserción

- Todo documento insertado en MongoDB tiene campo **identificador** llamado **_id.**
- Este **_id** es **único** en cada **colección** y sirve para buscar rápidamente un documento (mediante un índice).
- Si un documento insertado no incluye el identificador _id entonces MongoDB lo añadirá **automáticamente:**

```
{ "_id" : ObjectId("5812150dd91d96e21aa4d534"),
"name" : "Pepe", "nhijos" : 3 }
{ "_id" : ObjectId("58121510d91d96e21aa4d535"),
"name" : "Juan", "web" : "www.juan.com" }
{ "_id" : ObjectId("58121513d91d96e21aa4d536"),
"name" : "Ana", "gustos" : [ "musica", "p2p" ] }
```

Inserción

- Además de **insert**, a partir de la versión 3.2 existen otras operaciones para insertar documentos:
  - insertOne()
  - insertMany()
- Podéis encontrar más información en las **Referencias** (operaciones CRUD).

Modificación

- Para reemplazar **completamente** un documento existente se utiliza la función **update**:

```
> db.users.update({name:"Ana"},{name:"oculto"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

  Reemplaza el documento con campo `name=Ana` por el documento `{name:"oculto"}`.

- A la hora de reemplazar un documento, su **identificador se conserva**.
- Por defecto, si existen varios documentos que encajen solo se **modificará uno**.

Modificación

- Para **añadir un campo nuevo** o **reemplazar** el valor de un campo existente en un documento se utiliza la función **update** con el modificador **$set**:

- ```
  > db.users.update({name:"Juan"}, {"$set":{sexo:"varon"}})
  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
  ```

  Añade al documento con `name=Juan` el campo `sexo=varon`

  ```
  { "name" : "Juan", "web" : "www.juan.com", "sexo" : "varon" }
  ```

- ```
  > db.users.update({name:"Juan"}, {"$set":{sexo:"mujer"}})
  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
  ```

  Modifica el campo `sexo` y cambia su valor a `mujer`

  ```
  { "name" : "Juan", "web" : "www.juan.com", "sexo" : "mujer" }
  WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
  ```

Modificación

- También se pueden eliminar campos utilizando la función **update** junto con el modificador **$unset**:
- `> db.users.update({name:"Juan"}, {"$unset":{sexo:1}})`
  `WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })`
  Elimina el campo sexo del documento con campo `name=Juan`.
- No importa el valor que se ponga para el campo `sexo`, en este caso es un 1 pero podía ser “”, true, etc.

Modificación

- A partir de MongoDB 3.2 existen más operaciones para modificar documentos:
  - updateOne()
  - updateMany()
  - replaceOne()
- Podéis encontrar más información en las **Referencias** (operaciones CRUD).

Eliminación

- Para eliminar documentos de una colección se usa la operación **remove**:

```
> db.users.remove({name:"Pepe"})
WriteResult({ "nRemoved" : 1 })
```

- Por defecto **remove** elimina **todos los documentos** que encajan, aunque se puede modificar con el parámetro **justOne**.

Eliminación

- A partir de MongoDB 3.2 existen más operaciones para eliminar documentos:
  - deleteOne()
  - deleteMany()
- Podéis encontrar más información en las **Referencias** (operaciones CRUD).

Uso de MongoDB: consultas

Consultas

- Para buscar documentos insertados en una colección se utiliza el comando **find**:

```
> db.users.find()
{ "_id" : ObjectId("5812150dd91d96e21aa4d534"), "name" :
"Pepe", "nhijos" : 3 }
{ "_id" : ObjectId("58121510d91d96e21aa4d535"), "name" :
"Juan", "web" : "www.juan.com" }
{ "_id" : ObjectId("58121513d91d96e21aa4d536"), "name" :
"Ana", "gustos" : [ "musica", "p2p" ] }
```

- Si no se pasa ningún parámetro, buscará todos los documentos en la colección.

Consultas

- Se puede buscar documentos concretos pasando la consulta representada como JSON a la función **find**:
- Documentos con el campo `name=”Ana”`:

```
> db.users.find({name:"Ana"})
{ "_id" : *, "name":"Ana", "gustos":["animales","aventura"]
```

- Documentos con el campo `name="Pepe"` **Y** `nhijos=3`:

```
> db.users.find({name:"Pepe", nhijos:3})
{ "_id" : *, "name":"Pepe", "nhijos" : 3 }
```

- Documentos con `name=Ana` **O** `nhijos=3`:

```
> db.users.find({$or: [{name:"Ana"},{nhijos:3}]})
{ "_id" : *, "name":"Pepe", "nhijos" : 3 }
{ "_id" : *, "name":"Ana", "gustos":["animales","aventura"] }
```

Consultas

- Observad que en MongoDB las condicones de consulta son también documentos JSON:
  - `{name:"Ana"}`
  - `{name:"Pepe", nhijos:3}`
  - `{$or: [{name:"Ana"},{nhijos:3}]}`
- Esto es algo general en MongoDB: **todo se representará como JSON**.

Consultas

- MongoDB soporta un lenguaje muy completo para representar **consultas complejas**:
  - **Comparaciones**:
    - $eq → ==
    - $gt → >
    - $gte → >=
    - $lt → <
    - $lte → <=
    - $ne → !=

Consultas

- MongoDB soporta un lenguaje muy completo para representar **consultas complejas**:
  - **Lógicas**:
    - $or
    - $and
    - $not
    - $nor

Consultas

- MongoDB soporta un lenguaje muy completo para representar **consultas complejas**:
  - **Elemento**:
    - $exists → Existencia de un campo.
    - $type → Un campo contiene un valor de un tipo determinado.
- Podéis encontrar más detalles en las **Referencias** (operadores de consulta)

Consultas

- Ejemplos con operadores de búsqueda:
- Documentos con 2 o más hijos:

```
> db.users.find({nhijos: {$gte: 2} })
{ "_id" : *, "name":"Pepe", "nhijos" : 3 }
```

- Documentos con menos de 5 hijos:

```
> db.users.find({nhijos: {$lt: 5} })
{ "_id" : *, "name":"Pepe", "nhijos" : 3 }
```

- Documentos con entre 2 y 5 hijos (incluidos):

```
> db.users.find({nhijos: {$gte: 2, $lte: 5} })
{ "_id" : *, "name":"Pepe", "nhijos" : 3 }
```

Consultas en listas

- Se puede buscar por valores en una lista. *Ej:* documentos que contienen “musica” en el campo **gustos** *(en alguna posición)*.

```
> db.users.find({gustos: "musica"})
{ "_id":*, "name":"Ana", "gustos":[ "musica", "p2p" ] }
```

- También se puede buscar aquellos documentos que tengan varios elementos en un campo lista usando el modificador **$all**.
  Ej: documentos cuyo campo **gustos** contengan “p2p” y “musica” (*no importa el orden*).

```
> db.users.find({gustos: {$all: ["p2p", "musica"]}})
{ "_id":*, "name":"Ana", "gustos":["musica", "p2p"] }
```

Consultas en campos anidados

- Para acceder a campos internos de los documentos se utiliza la notación **campo1.campo2**:

```
> db.users.insert({name:”Lola”,dir:{calle:”Mayor”,num:2} })
> db.users.find({“dir.calle”: “Mayor”}
{“_id”:*, name:”Lola”,dir:{calle:”Mayor”,num:2}
> db.users.find({“dir.num”: 2})
{“_id”:*, name:”Lola”,dir:{calle:”Mayor”,num:2}
```

- La misma notación se utiliza para acceder a los elementos de un array: **campo.N**.

```
> db.users.insert({name:"Ivan", ejemplares:[1,2,5]})
> db.users.find({"ejemplares.2":5})
{"_id":*, "name":"Ivan", "ejemplares":[1,2,5] }
```

- ***Nota**: las posiciones empiezan en 0.*

Consultas *where*

- Como hemos visto en los ejemplos, los valores usados en la consulta deben ser **constantes.** Por tanto no se pueden referir a **otro campo del documento**.
- Si buscamos los usuarios con los dos apellidos iguales, la consulta siguiente sería incorrecta:

```
> db.users.find({apellido1 : apellido2})
```

Para consultas así se utiliza la cláusula **$where.**

Consultas *where*

- La cláusula **where** nos permite incluir un predicado **JavaScript** que evalúa la condición de búsqueda:

```
> db.users.find( {"$where" :
  function () {
    if ('apellido1' in this && 'apellido2' in this)
    // Si existen los campos en el documento
      return this['apellido1'] == this['apellido2'];
    else
      return false;
  }
})
```

- Las consultas *where* requieren recorrer la colección completa → **ineficientes**, no se puede aprovechar ningún **índice**.

Proyección de resultados

- Se puede **proyectar** qué campos de los documentos se devolverán en una consulta.
- Para ello se usa el segundo parámetro de la función find():

```
> db.users.find({nhijos: {$gt:2}}, {name:1,_id:0})
```

- Pondremos a **0** los campos que queremos ocultar y a **1** los campos que queremos visualizar.
  - En el ejemplo ocultamos `_id` y mostramos únicamente `name`.

Limitar resultados

- Se puede limitar el número de resultados a mostrar mediante el modificador **limit**. *Ej: mostrar solo los 2 primeros documentos de la colección **users***:

```
> db.users.find().limit(2)
```

- También se pueden omitir los primeros documentos devueltos por la consulta mediante **skip**. *Ej: mostrar los documentos de **users** salvo los 2 primeros*:

```
> db.users.find().skip(2)
```

- **limit()** y **skip()** se pueden usar para *paginar* los resultados, aunque es mejor no usar **skip** en colecciones grandes (en esos casos mejor usar condiciones de búsqueda: `fecha >= ultima_fecha`, `precio >= ultimo_precio`, etc.)

Ordenar resultados

- Por defecto, los documentos se muestran en el **orden** en que están almacenados en **disco**.
- Para ordenar los resultados obtenidos se utiliza el modificador **sort**. Este modificador acepta una serie de campos por los que ordenar:

```
> db.users.find().sort({name:-1, nhijos:1})
```

  - `name:-1` → ordenación **descendente** por **name**
  - `nhijos:1` → ordenación **ascendente** por **nhijos**
  - Primero ordenar por **name**, y para los empates utiliza **nhijos** (orden **lexicográfico**).

Ejercicios de consultas

Ejercicios de consultas

- Consideremos una colección **usuarios** con documentos como este:

```
{
   '_id':ObjectId('5820c1e8323e95'),
   'provincia': 'Baleares',
   'nombre': 'Juan',
   'edad': 27,
   'aficiones': ['deporte', 'cine']
}
```

Ejercicios de consultas

1) Usuarios madrileños que son mayores de edad.
2) Usuarios de provincias con lengua co-oficial: Galicia, País Vasco, Navarra, Cataluña, Comunidad Valenciana, Baleares.
3) Usuarios aficionados al deporte.
4) Únicamente el nombre de los 3 primeros usuarios de Madrid.
5) Usuarios gallegos o baleares, ordenados por edad descendente.

Consultar MongoDB desde Python: pymongo

pymongo

- Para conectar, modificar y consultar bases de datos en MongoDB desde Python utilizaremos el módulo **pymongo**.
- pymongo proporciona distintas clases para realizar estas tareas, entre las que destacan:
  - MongoClient
  - Database
  - Collection
  - Cursor
- Estas clases son prácticamente las mismas que las existentes en el cliente **mongo**.

pymongo

- En la página principal de pymongo podéis encontrar instrucciones de instalación, tutoriales, ejemplos, etc.: https://api.mongodb.com/python/current/
- También contiene toda la documentación API: https://api.mongodb.com/python/current/api/index.html

MongoClient

- La clase MongoClient nos permite conectar con un servidor MongoDB:

```
from pymongo import MongoClient
mongoclient = MongoClient()
```

- Por defecto se conecta a `localhost:27017`, pero se pueden pasar parámetros:
  - Host y puerto.
  - *Timeouts* de la conexión.
  - Certificados SSL para autenticar y cifrar la conexión.

MongoClient

- Dado un objeto MongoClient, podemos acceder a una base de datos concreta mediante:
  - Atributo → `db = mongoclient.giw`
  - Corchetes → `db = mongoclient['giw']`
- Ambos métodos devuelven un objeto de la clase **Database**.
- MongoClient también se usa para obtener el **listado** de bases de datos y para **borrarlas**.

Database

- Un objeto Database nos permite acceder a sus colecciones de una manera similar a MongoClient:
  - Atributo → `c = db.users`
  - Corchetes → `c = db['users']`
- Ambos métodos devuelven un objeto de la clase **Collection**.
- Database también permite consultar las **colecciones** existentes o **eliminar** colecciones.

Collection

- **Collection** es la clase principal sobre la que se realizarán las operaciones CRUD:
  - **Create**: insert_one, insert_many
  - **Read**: find, find_one
  - **Update**: replace_one, update_one, update_many, update
  - **Delete**: delete_one, remove
- Aparte de éstos también existen otras operaciones CRUD, ver más información en las **Referencias**.

Documentos en pymongo

- Un aspecto muy cómodo de pymongo es que los **documentos JSON** necesarios en las operaciones se representan con **diccionarios**.
- Los diccionarios Python y los documentos JSON tienen la **misma sintaxis:**

```
doc = {'name':'pepe', 'edad':24}
# doc es un diccionario Python
collection.insert_one(doc)
```

Valores de retorno

- Las operaciones CRUD devuelven distintos tipos de objeto resultado:
  - InsertOneResult
  - InsertManyResult
  - UpdateResult
  - DeleteResult
  - Cursor
- Hay que consultar estos objetos para conocer el resultado de las operaciones.

Cursor

- Las consultas de colecciones (**find**) devuelven un objeto **Cursor**.
- Este objeto permite acceder a los **resultados**:
  - Recorrer → `for e in cursor`
  - Acceder a un elemento → `cursor[50]`
  - Acceder a un rango → `cursor[10:20]`

Cursor

- El objeto Cursor también permite condicionar la consulta:
  - count
  - limit
  - skip
  - sort
  - where
- Ver más detalles en las **Referencias** (Cursor).

Ejemplo con pymongo

```
from pymongo import MongoClient

mongoclient = MongoClient()
db = mongoclient['giw']
c = db['test']

ana = {'name':'ana', '_id':0, 'edad':23}
res = c.insert_one(ana)
print(res.inserted_id)

doc = c.find_one({'_id':0})
print(doc)

res = c.delete_one({'_id':0})
print(res.deleted_count)
```

Referencias y bibliografía

Referencias

- Operaciones CRUD (crear, leer, actualizar y eliminar): https://docs.mongodb.com/manual/crud/
- Operadores de consulta: https://docs.mongodb.com/manual/reference/operator/query/
- API pymongo: https://api.mongodb.com/python/current/api/index.html

Referencias

- API pymongo (Collection): https://api.mongodb.com/python/current/api/pymongo/collection.html#pymongo.collection.Collection
- API pymongo (Cursor): https://api.mongodb.com/python/current/api/pymongo/cursor.html#pymongo.cursor.Cursor

Bibliografía

- MongoDB: The Definitive Guide, 2nd Edition. Kristina Chodorow. O'Reilly, 2013.
- NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Pramod J. Sadalage, Martin Fowler. Addison-Wesley Professional, 2012.
- Seven Databases in Seven Weeks: A Guide to Modern Databases and the NoSQL Movement. Eric Redmond, Jim R. Wilson. Pragmatic Bookshelf, 2012.