Práctica 1 – Primer contacto con MongoDB

Nuestro primer objetivo es conseguir que MongoDB esté accesible y funcione.
Comenzamos arrancando en Linux→usuario local

1. Abrir un terminal (arriba a la derecha; “Actividades” y luego terminal; si ya hay un terminal y queremos otro, dar botón derecho sobre el terminal y elegir “Ventana Nueva”)
2. Tecleamos *mongo*. Si la respuesta es algo del estilo “bash: mongo: no se encontró la orden” pasamos al paso 3. Si parece que intenta entrar (aunque dé error), vamos al paso 4.
3. Tenemos que hacer que mongo esté accesible. Para ello:
    1. En el terminal tecleamos *locate mongoimport* (buscamos mongoimport y no mongo, porque suelen estar en el mismo sitio, y si ponemos mongo saldrán demasiadas carpetas con este nombre). A mí me contesta:
    `/opt/mongodb-linux-x86_64-debian71-3.0.6/bin/mongoimport`
    2. En el terminal escribir *gedit .bashrc*
    3. Debe abrirse un fichero que inicializa variables del terminal. Vamos al final y añadimos una nueva línea
    `PATH="$PATH:/opt/mongodb-linux-x86_64-debian71-3.0.6/bin/:"`
    Reemplazando la parte en azul por el nombre que nos salio en locate sin “mongoimport”
    4. Guardamos y salimos.
    5. Desde el terminal hacemos source .bashrc
    Esto solo hace falta hacerlo esta vez; a partir de ahora cada terminal que abramos podrá encontra mongo sin problemas
    6. Probar a teclear en el terminal *mongo* : nos dará un error de que no pudo conectar con el server, pero ahora ya debe encontrarlo. Si no avisar al profesor...que avisará a los técnicos...que llamarán a un amigo...que a su vez...
4. Crear un directorio donde se grabarán los datos: *mkdir datos*
5. Tecleamos mongod *--dbpath datos*
6. Ahora vamos a importar los datos que usaremos en la práctica:
    1. Bajar del campus los ficheros minitweet.json y miniuser.json
    2. Desde la consola de linux teclear

    ```
    mongoimport --db test --collection miniuser --file miniuser.json
    mongoimport --db test --collection minitweet --file minitweet.json
    ```

7. En otro terminal tecleamos *mongo* sin más...ya estamos en el shell de mongo. Allí podemos teclear `show collections` para que nos enseñe las colecciones disponibles.

A partir de ahora, cuando lleguemos al laboratorio solo habrá que repetir los pasos 4,5,6 para empezar las prácticas.

1) Escribir en la solución (solucion.txt) la representación una consulta para saber el número de usuarios con la geolocalización activada (geo_enabled). Para cada uno mostrar solo el screen_name, time_zone y location.

Solución
`db.miniuser.find({"geo_enabled":true},{"screen_name":1,"time_zone":1,"location":1,"_id":0}).count()`
`>199`

2) Dentro de la colección usuarios queremos saber cuántas zonas horarias (time_zone) distintas hay. Escribe una consulta y copiala en solucion.txt. Pista: `a.length` nos da la longitud del array `a`.
Solución
`db.miniuser.distinct("time_zone").length`

3) Escribir una consulta para saber cuántos tweets se han creado (created_at) con fecha posterior a la fecha ISODate("2016-11-11T11:11:11Z")}
Solución
`db.minitweet.find({created_at: {$gt:ISODate("2016-11-11T11:11:11Z")}}).count()`

4) Escribir una consulta que nos diga solo el screen_name de aquellos usuarios que o bien han tenido más de 100 retweets (Rtin>100) o más de 100 menciones (mentions>100). Nota: salen 8
Sol.
`db.miniuser.find({$or:[{RTIn:{$gt:100}}, {mentions:{$gt:100}}]},{screen_name:1,_id:0}).count()`

5) Aparte de find, limit, skip. Sort, existe una función map que puede ser muy útil. Map tiene como parámetro una función, que recibe documentos como entrada. Lo que hace es aplicar la función a todos los documentos del cursor, y devolver el resultado en un array,
Ejemplo:

```
> db.catalogo.find().map(function(doc){return doc.nombre})
[ "AC-26", "Piel de Manzana" ]
```

Utilizar esta función para devolver por cada usuario la proporción de tweets que ha generado que son originales (dividiendo los campos original y total del subdocumento tweets

Solución
db.miniuser.find().map(function(doc){return doc.tweets.original/doc.tweets.total})

6) Vamos a crear una colección con las siguientes instrucciones:

```
db.pru.drop()
db.pru.insert([{a:1,b:1},{a:1,b:2},{a:2,b:1},{a:2,b:2},{a:3,b:1},{a:3,b:2}])
```

Escribir una consulta que devuelva:

```
{ "a" : 3, "b" : 1 }
{ "a" : 2, "b" : 1 }
{ "a" : 1, "b" : 1 }
{ "a" : 3, "b" : 2 }
{ "a" : 2, "b" : 2 }
{ "a" : 1, "b" : 2 }
```

Solucion
db.pru.find({},{_id:0}).sort({b:1,a:-1})

7) Después de una llamada a find se puede utilizar (entre otras cosas):

- sort (doc) : ordena los documentos según indica el documento (de la forma cl:1, cl':-1….con 1 indicando ordenación ascendente y -1 ordenación descendente=
- limit(N): limita la respuesta a los N primeros elementos

¿Conmutan?
Es decir, ¿db.coleccion.find().limit(N).sort(doc) da siempre el mismo resultado que db.coleccion.find().limit(N).sort(doc)? Poner un ejemplo de consulta usando los dos documentos de 4) que muestre que la respuesta que das se verifica.
Sol.
Sí, si no estas dos darían distinto
db.miniuser.find().limit(1).sort({screen_name:-1})
db.miniuser.find().sort({screen_name:-1}).limit(1)

8) Afirmación: *Una consulta como db.coleccion.find().limit(5).sort({nombre:1}) es muy eficiente, porque solo "hay que ordenar" un máximo de 5 documentos*.
¿Cierto o Falso? Justifica la respuesta con una frase.
Sol.
Falso, primero se ordenan todos

9) Escribe la instrucción que falta en esta secuencia escrita en el terminal de Mongo:

```
x = { "a" : 1 };
y = "a";
…. // falta una instrucción aquí que utiliza x e y
 print(x.a);
2  // mongo muestra el valor 2
```

Sabiendo que:

- La instrucción debe utilizar las variables x,y
- Se ha omitido la respuesta del shell en todas las instrucciones salvo la última (2)

Sol.
x[y]++; // hay más posibilidades

10) ¿`find().limit(1)` y `findOne()` son equivalentes? Es decir, ¿muestran siempre lo mismo?
Si la respuesta es *SÍ* indica por qué (una frase). Si la respuesta es *No* escribe un contrajemplo.
Sol.
N
1Al ejecutar sobre una colección vacía find().limit(1) no devuelve nada y findOne() escribe null