Introduccion.

El objetivo del curso de MongoDB avanzado es poder abordar temas esenciales que van más allá del simple hecho de realizar querys. A lo largo del curso aprenderemos a usar operadores, aggregation framework, configuraciones, nuevos tipos de collections y los índices geoespaciales.
Para poder desarrollar el curso con mayor facilidad usaremos como herramienta el cliente RoboMongo (https://robomongo.org/) y ejemplos en Node.js (si bien MongoDB puede ser usado desde cualquier lenguaje, Node.js la tecnología ideal para los ejemplos).

Server.

Status.

Para obtener el status del servidor podemos ejecutar: "db.serverStatus()" desde la consola. El comando retornará un documento con toda la info del servidor.

Export/Import.

Export.

MongoDB nos ofrece el comando mongoexport que permite exportar una collection a un archivo en formato JSON o CSV.

mongoexport --db cursodb --collection alumnos --out alumnos.json

En el ejemplo anterior exportamos la collection "alumnos" en formato JSON a un archivo llamado alumnos.json

Argumentos.

Veamos algunos de los argumentos soportados por el comando. Si deseamos ver todos los argumentos solo basta con ejecutar: `mongoexport --help`

-d, --db	Seteamos la base de datos con la que vamos a operar.
-c, --collection	La collection que vamos a exportar.
-f, --fields	Las propiedades que vamos a exportar.
-o, --out	El archivo que vamos a generar.
-q, --query	Una query para filtrar los elementos. El resultado de la query es lo que se grabará en el archivo final.
--pretty	El JSON final es legible para humanos.
--type	El formato de salida, ["json", "csv"]
--host <hostname><:port>	Indicamos el host:port al que nos queremos conectar para realizar el export.
--user	El usuario para conectarnos a la db
--password	El password del usuario con el que nos queremos conectar.

Import.

MongoDB también nos ofrece la herramienta mongoimport, la cual permite importar un archivo con formato JSON a una collection.

Ejemplo:
mongoimport -d cursodb2 -c alumnos2 --file alumnos.json

Argumentos.

Los argumentos del comando mongoimport son los mismos que los del comando mongoexport, con la diferencia que en vez de utilizar el argumento "--out" vamos a utilizar el comando argumento "--file" para indicar el archivo.json que queremos importar.
Otro argumento interesante es "--headerline" el cual se utiliza cuando importamos archivos CSV. Con este argumento le indicamos a MongoDB que el primer registro contiene los nombres de las propiedades.

Cursores.

Cada vez que utilizamos db.collection.find éste retorna un cursor. Un cursor es un apuntador a los resultados generados por la llamada a find.
Para poder obtener los resultados debemos iterar sobre el cursor (es un iterable). En el caso de no declarar el cursor y ejecutar la consulta directamente en la consola se iterará automáticamente sobre los primeros 20 resultados.

Ejemplos:

```
use cursodb;
db.employed.insert({added: new Date(), name: 'Alejandro', age:23});
db.alumnos.insert({added: new Date(), name: 'Martin', age:22});
db.alumnos.insert({added: new Date(), name: 'Carlos', age:18});
db.alumnos.insert({added: new Date(), name: 'Juan', age:31});
db.alumnos.insert({added: new Date(), name: 'Pedro', age:28});
var results = db.alumnos.find();
```

En el ejemplo anterior creamos la collection "alumnos" dentro de la DB "cursodb".
Posteriormente asignamos a la variable "results" el cursor retornado por el método "find".

Timeout.

Cuando se crea un cursor se debe tener en cuenta que el servidor lo destruirá automáticamente después de 10 minutos (TTL).

Para ponder evitar que el cursor expire podemos usar el método "noCursorTimeout" del cursor. Una vez que terminamos de utilizar el cursor deberemos "destruirlo", por lo cual será necesario llamar al método "close" del mismo.
Otra opción para evitar el TTL de los cursores es usar el método "addOption" pasando el argumento "DBQuery.Option.noTimeout"

```
var result = db.alumnos.find().noCursorTimeout();
result.close();

var t = db.alumnos;
var cursor = t.find().addOption(DBQuery.Option.noTimeout).
                      addOption(DBQuery.Option.awaitData)
```

Como podemos ver en ambos ejemplos, el resultado es el mismo.

addOption.

El método addOption de los cursores permiten alterar el comportamiento por defecto de los cursores, pasando como argumento el comportamiento que deseamos alterar.

Flag	Description
DBQuery.Option.tailable	Hace que el cursor no cierre cuando el último registro fue alcanzado, sino que seguirá devolviendo datos a medida que se inserten nuevos datos en la collection. Para funcionar la collection debe ser "capped" y tener un size fijo.
DBQuery.Option.slaveOk	Permite la consulta en un miembro de la réplica (esclavo).

DBQuery.Option.noTimeout	Impide que el servidor ejecute el TTL del cursor.
DBQuery.Option.awaitData	Para usarse junto con DBQuery.Option.tailable; bloquea el cursor y espera que termine de llegar la información.
DBQuery.Option.exhaust	Setea que el cursor devolverá todos los resultados en forma inmediata y no por batches (BSON SIZE máximo de 16MB).
DBQuery.Option.partial	Setea al cursor que retorne data parcial desde el cluster aunque algunos nodos no respondan en vez de devolver error.

Operaciones con cursores.

Ahora que ya entendimos bastante el tema de cursores es hora de ponernos manos a la obra. Veamos un ejemplo de cómo iterar sobre un curso desde la consola de mongo.

```
var myCursor = db.alumnos.find();
while (myCursor.hasNext()) {
   print(tojson(myCursor.next()));
}
```

En el ejemplo anterior creamos el cursor myCursor, y utilizando un while para chequear si quedan elemento imprimimos el contenido de cada elemento en pantalla (para obtener el elemento usamos el método "next". Para poder imprimir correctamente el elemento en pantalla hacemos uso la función tojson que transforma el objeto en un JSON legible.