Introducción.
Que es MongoDB?
- NoSQL.
- Document-oriented database.
- Diferencias son SQL.
  - On Demand.
  - Database.
  - Collections.
  - Documents.

Primeros pasos.
- Instalación.
- Instalando en Windows.
- Instalando en MacOS.
- Instalando en Linux.

Export/Import.
- Export.
  - Argumentos.
- Import.
  - Argumentos.

Cliente.
CRUD.
- USE.
- Insert.
- Read.
  - Operadores.
- Update.
  - $set
- Remove.

Operadores.
- $gt
- $gte
- $lt
- $lte
- $ne
- $and
- $or
- $in
- $nin

Introducción.

El objetivo del curso de MongoDB avanzado es capacitar al alumno en los principios básicos, intermedios y algunos conceptos avanzados sobre la tecnología MongoDB.
A lo largo del curso aprenderemos los principios básicos de ésta base de datos, las diferencias con los sistemas SQL, a realizar operaciones CRUD, el manejo de índices simples y complejos (por ejemplo, los índices geoespaciales), etc.
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).

Que es MongoDB?

MongoDB es una base de datos NoSQL orientada a documentos, de fácil instalación, configuración (inicialmente no la requiere prácticamente), efectiva para el manejo de grandes volúmenes de datos y muy fácil de aprender a utilizar.
Está desarrollado sobre el motor V8 de Google (motor de JavaScript que utiliza por ejemplo Google Chrome o la tecnología Node.js), por lo cual para poder utilizar MongoDB es necesario contar con conocimientos de JavaScript.
Para entender un poco mejor de qué se trata, primero debemos repasar algunos conceptos.

NoSQL.

Los sistemas NoSQL son justamente, como su nombre lo indica, todo sistema en el que podemos almacenar y recibir datos, pero que no son SQL. Un ejemplo simple de sistemas NoSQL son los motores clave/valor como Memcached, o algo un poco más complejo como Redis.
No solo existe sistemas clave/valor dentro del mundo NoSQL, hay sistemas mucho más complejos, MongoDB por ejemplo, que cuentan con sofisticados mecanismos de búsquedas, almacenamiento, índices, etc. pero que carecen de las propiedades de los sistema SQL.

Document-oriented database.

Una base de datos orientada a documentos es aquella que permite realizar operaciones CRUD sobre estructuras de datos (normalmente JSON/BSON) a la que se denomina documentos. Estas estructuras suelen ser flexibles y representan objetos JSON.

Diferencias son SQL.

Entre MongoDB y los sistemas SQL encontraremos algunas diferencias bastante importantes, todas ellas propias de un sistema que no sigue las reglas estrictas de los mundos SQL.

A continuación veremos la siguiente tabla con las diferencias más importantes, las cuales explicaremos punto por punto.

SQL	MongoDB
Database	Database
Table	Collection
Row	Document
Column	Field
Join	Embedded documents, linking

On Demand.

En los sistemas SQL es responsabilidad del DBA/Desarrollador crear la base de datos y las tablas, todo en una estructura pre definida. En MongoDB no es necesario la creación de las bases de datos y las collections, ya que las mismas son creadas en el momento de su uso. Obviamente esto no quiere decir que no contemos con mecanismos para crear previamente las bases o las collections, pero es importante saber que todo será creado On Demand, ya que si por ejemplo nos equivocamos en el nombre de una collection, la misma será creada automáticamente sin que detectemos el error (técnicamente hablando, no hay error).

Database.

Al igual que los sistemas SQL, MongoDB trabaja con el concepto de bases de datos. Las bases de datos son creadas en el momento de ser usadas (en realidad cuando creamos la primer collection en su interior):

```
mongodb> use mydatabase;
```

Collections.

Las collections son la unidad en la que almacenaremos nuestros documentos. Están contenidas dentro de una base de datos

Documents.

Los documentos son una estructura de datos (JSON) almacenadas en BSON (JSON Binario). Cada documento puede tener una estructura diferente dentro de la misma collection.
Si tomamos a las collections como una carpeta para almacenar hojas, y a cada hoja como un documento, cada hoja puede ser un documento de distinto tipo y estructura (recibos, un poema, una carta notariada, etc).

Primeros pasos.

A continuación daremos los primeros pasos en MongoDB, empezando por la instalación, para finalmente poder insertar nuestro primer documento.

Instalación.

La instalación de MongoDB depende mucho del sistema operativo que estemos usando, al igual que la ejecución del Daemon (servicio). Es importante entender que una vez instalado MongoDB nos encontraremos con varios archivos ejecutables y que sin importar el sistema operativo, los archivos principales serán siempre los mismos.

Instalando en Windows.

Para instalar MongoDB en los sistemas windows debe ir al centro de descargas de MongoDB y descargar el MSI (archivo instalador en los sistemas Windows).
Al ejecutar el archivo nos aparecerá una series de ventanas típicas del proceso de instalación, sólo debemos dar click en "siguiente" (o "next" si es en ingles).
Una vez instalado podemos lanzar el servicio desde el administrador de servicios de Windows o lanzar el daemon directamente desde la consola de Windows.
Para lanzarlo desde la consola de Microsoft debemos ir a Inicio -> Ejecutar -> cmd.
Una vez dentro de la consola ejecutamos:

```
cmd:\>mkdir c:\mongo
cmd:\> cd "c:\Program Files"
c:\Program Files\> cd MongoDB\Server\3.x.x\bin
```

```
c:\Program Files\MongoDB\Server\3.x.x\bin\>mongod.exe --dbpath c:\mongo
```

En la primer línea creamos el directorio en donde MongoDB almacenará los archivos de bases de datos.

Instalando en MacOS.

Para instalar en MacOS podemos ir al centro de descargas de MongoDB y bajar el archivo de instalación. Otra opción es instalarlo con el comand brew:
~$brew update
~$brew install mongodb

Instalando en Linux.

Para instalar MongoDB en los sistemas Debian debemos agregar la key del package, agregar a los sources el path a mongodb y finalmente instalar.

```
~$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
~$echo "deb http://repo.mongodb.org/apt/debian wheezy/mongodb-org/3.2 main" | sudo tee
/etc/apt/sources.list.d/mongodb-org-3.2.list
~$ sudo apt-get update
~$ sudo apt-get install -y mongodb-org
```

Export/Import.

Los procesos de export/import nos permiten exportar e importar documentos en formato JSON. Serán de gran utilidad a lo largo de todo el curso para poder realizar las prácticas planteadas en cada capítulo, como así también hacer copias de seguridad de nuestros datos (aunque más adelante veremos comandos más eficientes para esta última tarea).

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.
--jsonArray	Con esta opción indicamos que los documentos deben ser exportados dentro de un Array y no separados por saltos de línea.

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.

Cliente.

El cliente de MongoDB (la consola) es lo que nos permitirá operá con nuestras bases, collections y documentos.
Debemos tener en cuenta que MongoDB utiliza V8, por lo cual la consola tiene cierta similitud con la consola de Google Chrome. Veamos algunos argumentos de la consola de MongoDB.

Argumento.	Descripción.
--host	Indicamos el host del servidor al que nos queremos conectar. Por defecto será "localhost".
--port	Indicamos el puerto del servidor al que nos queremos conectar. Por defecto será 27017.
-eval	Evalua la expresión JavaScript pasada como argumento (debe ir como string).
file.js	Podemos pasar como último argumento un archivo .js, el cual será ejecutado por el cliente. Una vez terminada la ejecución al cliente devolverá el prompt al sistema.
--jsonArray	Indica que el archivo json que vamos a importar es un Array de objetos y no un grupo de objetos separados por saltos de línea.

Debemos recordar que MongoDB está basado en V8, por lo que dentro de la consola debemos utilizar JavaScript.

```
> for(var x=0; x<10; x++) print(x);
```

Como vemos en ejemplo anterior, podemos imprimir los números del 0 al 9 utilizando la sentencia for.

CRUD.

A continuación veremos cómo realizar operaciones CRUD sobre MongoDB (Create, Read, Update and Delete), pero antes de comenzar debemos recordar principios básicos:

- Las bases de datos son creadas en el momento de ser usadas
- Las collections son creadas en el momento de realizar un insert (aunque podemos crearlas previamente, algo que veremos más adelante).
- Lo que almacenamos en las collections son documentos (BSON), y no es necesario que cada documento tenga el mismo esquema.

USE.

La sentencia "use" nos permite seleccionar la base de datos con la que vamos a operar.

```
> use cursodb;
```

Una vez seleccionada la base de datos "cursodb" veremos el mensaje "switched to db cursodb".
A partir de éste momento "db" será una referencia a la base de datos con la que estamos operando.

Insert.

Para poder insertar documentos dentro de una collection debemos llamar al método insert de las collections.
Después de llamar a "use", db será una referencia a la base de datos seleccionada, por lo que todas las collections (existan o no) serán una propiedad de "db".
Todas las collections cuentan con propiedades y métodos, por lo que insert es un método de las collections.

```
> db.alumnos.insert({name: "Pedro", age: 21});
```

En el ejemplo anterior insertamos un nuevo documento dentro de la collection "alumnos", la cual no existía, pero fue creada en el momento en que llamamos al método insert.

Si todo salió bien deberemos ver un mensaje como "WriteResult({ "nInserted" : 1 })".
En MongoDB cada vez que insertamos un nuevo documento automáticamente se le agregará la propiedad "_id", el cual es justamente el id de ese documento. El id en MongoDB es un hash autoincremental, del tipo ObjectId.

Read.

A diferencia de los sistemas SQL, en MongoDB el proceso de un "select" tiene una estructura totalmente diferente.
Primero que nada un select en MongoDB es un find. El método find soporta 2 argumento (opcionales):

- El primer argumento es el "where", es decir, la query a utilizar.
- El segundo argumento es "projection", es decir, que fields deseamos obtener.

Tan el "where" como "projection" son objetos (JSON), por lo cual debemos ser cuidadosos para poder realizar las consultas.

```
> db.alumnos.find();
{ "_id" : ObjectId("57a8a2922673abf7c2706449"), "name" : "Pedro", "age" : 21 }
```

En el ejemplo anterior obtenemos todos los documentos de la collection "alumnos" (recuerden que seguimos en la base de datos "cursodb").

```
> db.alumnos.find({age:21})
{ "_id" : ObjectId("57a8a2922673abf7c2706449"), "name" : "Pedro", "age" : 21 }

> db.alumnos.find({age:22})
>
```

En el ejemplo anterior hacemos 2 consultas con "where". En el primer caso le indicamos que queremos todos los documentos en los cuales la propiedad "age" sea igual a 21.
En el segundo ejemplo indicamos que queremos todos los documentos en los cuales la propiedad "age" sea igual a 22, pero como no existen documentos con esa condición, no devuelve resultados.

Operadores.

Los operadores son propiedades que podemos pasar dentro de nuestro where y que representan acciones específica en nuestro criterio de búsqueda. Siempre comienzan con el signo "$" que es un caracter válido como nombre de una propiedad de un JSON.
Más adelante en este documento veremos algunos operadores bastante útiles, y a lo largo del curso veremos operadores complejos y específicos para ciertas situaciones.

Update.

El proceso de update dentro de de MongoDB funciona un poco distinto a los sistemas SQL. En primer lugar, el update actualiza el documento literalmente, esto quiere decir que altera su estructura y no los valores de las propiedades como uno esperaría.
El método update recibe 2 argumentos, el primero es el where y el segundo la nueva estructura del documento.

```
 > db.alumnos.find({})
{ "_id" : ObjectId("57a8a9662673abf7c270644c"), "name" : "Pedro", "age" : 21 }
> db.alumnos.update({}, {firstname: "Pedro", old: 21, country: "Argentina"});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnos.find({})
{ "_id" : ObjectId("57a8a9662673abf7c270644c"), "firstname" : "Pedro", "old" : 21, "country" :
"Argentina" }
```

En en ejemplo anterior lo primero que hacemos es ver los documentos que están dentro de la collection "alumnos". Luego hacemos un update de todos los documentos (el argumento where es un object vacío), y le seteamos como nueva estructura el document:
{"firstname" : "Pedro", "old" : 21, "country" : "Argentina" }
El método update funciona como el método "remove", es decir, sólo altera el primer documento que cumpla con la condición where, al menos que le pasemos el tercer argumento con la propiedad "multi" con el valor en "true", aunque es necesario utilizar el operador $set para que funcione.

$set

El operador $set se utiliza para "setear" los cambios de valores en la estructura de un documento. Cuando utilizamos el operador $set, los documentos no serán 100% alterados, sino que sólo modificaremos las propiedades declaradas dentro de SET.

```
> db.alumnos.update({}, {$set:{ truck: true}  }, {multi: true});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.alumnos.find()
{ "_id" : ObjectId("57a8a9662673abf7c270644c"), "firstname" : "Pedro", "old" : 21, "country" :
"Argentina", "truck" : true }
>
```

Remove.

Para poder eliminar documentos de una collection debemos utilizar el método "remove" de las mismas.
El método remove espera 2 argumentos: el where y las opciones. El método remove siempre eliminará el primer documento que cumpla con el where, aunque existan más documento que cumplan con la condición. Para que elimine todos los documentos que cumplan con la condición es necesario pasar como último argumento {multi: true}.

```
> db.alumnos.remove({age:55});
WriteResult({ "nRemoved" : 0 })
```

En el ejemplo anterior indicamos que queremos eliminar todos aquellos documentos en los cuales la propiedad "age" sea igual a "55".
Como no tenemos documentos que cumplan con dicha condición, el sistema no eliminará ningun documento.

```
> db.alumnos.remove({age:21}, {multi:true});
WriteResult({ "nRemoved" : 1 })
```

En el ejemplo anterior eliminamos todos los documentos que tengan la propiedad "age" igual a "21". El segundo argumento que estamos pasando es un object con la propiedad "multi" igual a "true", por lo que sí existiera mas de un documento que cumpla con la condición, serían eliminados todos ellos. Como nosotros teníamos únicamente un solo documento, el resultado de la operación es que eliminó un solo documento.

Operadores.

Como bien hemos mencionado anteriormente, los operadores son propiedades especiales dentro del objeto query que nos permiten realizar operaciones especiales.
Para poder comprender todos los ejemplos sobre los operadores importaremos el archivo "dispatches.json" que se encuentra en el directorio "class1/json".
Para importar dicho archivo ejecutaremos la aplicación "mongoimport" de la siguiente manera.

```
$ mongoimport -d cursodb -c dispatches --jsonArray --file dispatches.json
```

Una vez ejecutado el import contaremos con la collection "dispatches" dentro de la base de datos "cursodb".

$eq

El operador $eq se utiliza para indicar que queremos todos los documentos en los cuales la propiedad N sea igual al valor de $eq.

```
> db.dispatches.find( { Piso: {$eq: 'PB'}  } , {Piso: 1, _id:0} )
{ "Piso" : "PB" }
>
```

En el ejemplo anterior indicamos que queremos la propiedad Piso de todos los documentos que cumplan con la condición:
La propiedad Piso debe ser igual a 'PB'.

$gt

La propiedad $gt indica como condición que queremos todos los documentos en los cuales la propiedad N sea mayor a Y.

```
> db.dispatches.find({Service: {$gt: 70200}} , {Service: 1, _id: 0} )
{ "Service" : 70208 }
{ "Service" : 70209 }
>
```

En el ejemplo anterior seleccionamos la propiedad Service (excluyendo la propiedad _id) de todos los documentos que cumplan con las condiciones:

- Todos los documentos en los cuales la propiedad Service sea mayor al valor 70200

$gte

El operador $gte indica como condición que la propiedad debe ser mayor o igual a N.

```
> db.dispatches.find({Service: {$gte: 70200}} , {Service: 1, _id: 0} )
{ "Service" : 70208 }
{ "Service" : 70209 }
>
```

$lt

El operador $le indica como condición que la propiedad debe ser menor a N.

```
> db.dispatches.find({Service: {$lt: 80200}} , {Service: 1, _id: 0} )
{ "Service" : 70208 }
{ "Service" : 70209 }
```

```
>
```

$lte

El operador $lte indica que la propiedad debe ser menor o igual a N.

```
> db.dispatches.find({Service: {$lte: 70208}} , {Service: 1, _id: 0} )
{ "Service" : 70208 }
>
```

$ne

El operador $ne indica que la propiedad no debe tener el valor N.

```
> db.dispatches.find({Service: {$ne: 70208}} , {Service: 1, _id: 0} )
{ "Service" : "31720" }
{ "Service" : "846" }
{ "Service" : "7610" }
{ "Service" : "60423" }
{ "Service" : "351" }
{ "Service" : "14470" }
{ "Service" : "500" }
{ "Service" : "7163" }
{ "Service" : "8526" }
...
```

$and

El operador $and se utiliza para establecer la condición "and" literalmente en una consulta, esperando como valor un Array de objetos con cada una de las condiciones "and".

```
> db.dispatches.find({ $and: [ {Activo: true}, {Inmediato: true}, {Status: 'Aceptado',
calleName: 'SEGUI'}] }, {calleName:1, ZonaAlias:1} )
{ "_id" : "52273322c51a248a370286bc", "ZonaAlias" : "9", "calleName" : "SEGUI" }
>
```

La query anterior seleccionamos las propiedades _id, ZonaAlias y calleName de todos los documentos en los cuales se dan las siguientes condiciones:

- La propiedad Activo es igual a true
- La propiedad Inmediato es igual a true
- La propiedad Status es igual a 'Aceptado'

- La propiedad CalleName es igual a 'SEGUI'

$or

El operador $or se utiliza para establecer la condición "or" dentro de las consultas. Al igual que el operador $and, el operador $or espera recibir como valor un Array de objetos.

```
> db.dispatches.find( { $or: [{Piso:'2'}, {Piso:'PB'} ] } , {Piso: 1, _id:0} )
{ "Piso" : "2" }
{ "Piso" : "PB" }
>
```

En la query anterior seleccionamos la propiedad Piso de todos los documentos (excluyendo el _id) cuando se cumplan las siguiente condiciones:

- Que la propiedad Piso sea igual a '2' o 'PB'.

$in

El operador $in se utiliza para establecer como condición posibles valores a una propiedad. Modificamos la consulta anterior para usar el operador $in en vez del operador $or.

```
> db.dispatches.find( { Piso: {$in: ['2', 'PB'] } } , {Piso: 1, _id:0} )
{ "Piso" : "2" }
{ "Piso" : "PB" }
>
```

$nin

El operador $nin indica que la propiedad no debe tener los valores N…

```
> db.dispatches.find({Service: {$nin: ["11557", "5781", "1740", "846"]}} , {Service: 1, _id:
0} )
{ "Service" : 70208 }
{ "Service" : "31720" }
{ "Service" : "7610" }
{ "Service" : "60423" }
{ "Service" : "351" }
{ "Service" : "14470" }
{ "Service" : "500" }
...
```