



Clase 18. Programación Backend

CRUD en MongoDB



OBJETIVOS DE LA CLASE

- Comprender el significado de CRUD.
- Aprender el lenguaje de consultas de MongoDB.
- Crear y configurar cuentas de usuario, para definir roles que representarán permisos de acceso y operación en la base de datos.

CRONOGRAMA DEL CURSO

Clase 17



MongoDB

Clase 18



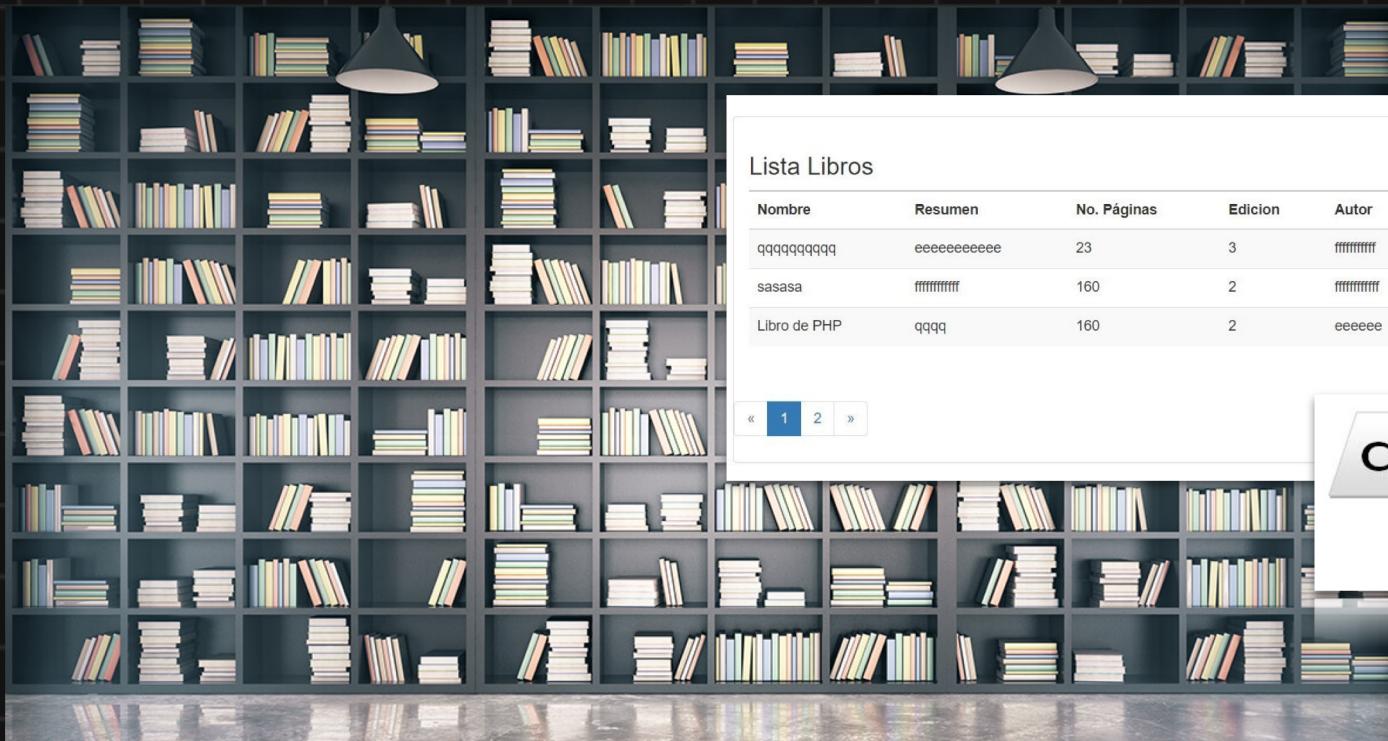
CRUD en MongoDB

Clase 19



NodeJS, MongoDB and
DBaaS

CRUD en MongoDB



Lista Libros

Nombre	Resumen	No. Páginas	Edicion	Autor	Precio	Editar	Eliminar
qqqqqqqqqqqq	eeeeeeeeeeee	23	3	fffffff	8.00		
sasasa	fffffff	160	2	fffffff	5.00		
Libro de PHP	qqqq	160	2	eeeee	5.00		

« 1 2 »

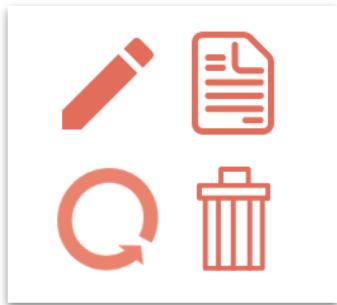
C R U D



CODER HOUSE

INTRODUCCIÓN

CODER HOUSE

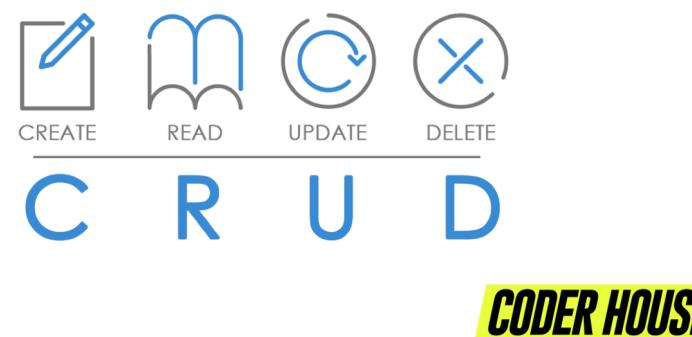


CRUD

La base de la gestión de datos

El concepto **CRUD** está estrechamente vinculado a la **gestión de datos digitales**. CRUD hace referencia a un acrónimo en el que se reúnen las primeras letras de las **cuatro operaciones fundamentales** de aplicaciones persistentes en sistemas de bases de datos

- **Create** (Crear registros)
- **Read** ó **Retrieve** (Leer registros)
- **Update** (Actualizar registros)
- **Delete** ó **Destroy** (Borrar registros)





CRUD: Conceptos



- **CRUD** resume las funciones requeridas por un usuario para **crear y gestionar datos**.
- Muchos procesos de gestión de datos están basados en **CRUD**, en los que dichas operaciones están específicamente adaptadas a los requisitos del sistema y de usuario, ya sea para la gestión de bases de datos o para el uso de aplicaciones.
- Para los expertos, las operaciones son las herramientas de acceso típicas e indispensables para comprobar, por ejemplo, los problemas de la base de datos.
- Para los usuarios, **CRUD significa crear una cuenta (create) y utilizarla (read), actualizarla (update) o borrarla (delete)** en cualquier momento.

Comandos en MongoDB

CODER HOUSE



mongoDB

MongoDB: Listado de Comandos



Conexión a la base mediante un cliente CLI

```
mongo # connects to mongodb://127.0.0.1:27017 by default
mongo --host <host> --port <port> -u <user> -p <pwd> # omit the password if you want a prompt
mongo "mongodb://192.168.1.1:27017"
mongo "mongodb+srv://cluster-name.abcde.mongodb.net/<dbname>" --username <username> # MongoDB Atlas
```

Comandos Helpers

```
show dbs
use <database_name>
db // prints the current database
show collections
load(myScript.js)
```

- **show dbs**: listado de bases no vacías
- **use**: crea y selecciona base de trabajo
- **db**: muestra la base actual
- **show collections**: listado de colecciones
- **load**: carga un script de comandos

CODER HOUSE



mongoDB

MongoDB: Listado de Comandos



Manejo de Base de datos y Colecciones

```
db.coll.drop()      // removes the collection and its index definitions
db.dropDatabase() // double check that you are *NOT* on the PROD cluster... :)

// Create collection
db.createCollection("contacts")

db.coll.stats()
db.coll.storageSize()
db.coll.totalIndexSize()
db.coll.totalSize()
db.coll.validate({full: true})
db.coll.renameCollection("new_coll", true) // 2nd parameter to drop the target collection if exist
```



MongoDB: Detalle de Comandos



- **db.coll.drop()** : borra una colección y sus índices respectivos.
- **db.dropDatabase()** : elimina la base de datos actual.
- **db.createCollection("contacts")** : crea una colección en forma explícita.
- **db.coll.stats()** : refleja estadísticas del uso de la base.
- **db.coll.storageSize()** : tamaño de almacenamiento de la colección.
- **db.coll.totalIndexSize()** : tamaño total de todos los índices de la colección.
- **db.coll.totalSize()**: tamaño total en bytes de los datos de la colección más el tamaño de cada índice de la colección.
- **db.coll.validate({full: true})** : comprueba la integridad de una colección.
- **db.coll.renameCollection("new_coll", true)** : renombra una colección, el 2do parámetro para borrar la colección destino si existe.



mongoDB

MongoDB: Comandos CRUD

CREATE y READ



Comando Create (**insert**)

```
db.coll.insertOne({name: "Max"})

db.coll.insert([{name: "Max"}, {name:"Alex"}]) // ordered bulk insert
db.coll.insert([{name: "Max"}, {name:"Alex"}], {ordered: false}) // unordered bulk insert
db.coll.insert({date: ISODate()})
db.coll.insert({name: "Max"}, {"writeConcern": {"w": "majority", "wtimeout": 5000}})
```

Comando Read (**find**)

```
db.coll.findOne() // returns a single document
db.coll.find()    // returns a cursor - show 20 results - "it" to display more
db.coll.find().pretty()
db.coll.find({name: "Max", age: 32}) // implicit logical "AND".
db.coll.find({date: ISODate("2020-09-25T13:57:17.180Z")})
```

CODER HOUSE



MongoDB: Detalle de Comandos

(2)



- **db.coll.insertOne({key:value})** : inserta un documento en la colección.
- **db.coll.insert({key:value})** : inserta un documento en la colección (en desuso).
- **db.coll.insertMany([{key:value}, {key:value}, {key:value}])** : inserta un array de documentos la colección en modo Bulk.
- **db.coll.findOne()** : busca un documento dentro de una colección.
- **db.coll.find()** : busca todos los documentos dentro de una colección.
- **db.coll.find({key:value})** : busca los documentos dentro de una colección que satisfacen el filtro de búsqueda.
- **db.coll.find().pretty()** : devuelve todos los documentos conservando el formato de objeto de salida.

FORMATO DE DOCUMENTO

CODER HOUSE



Formato de documento



Cuando insertamos un documento en **MongoDB**, el motor de base de datos crea un campo adicional llamado **ObjectId** identificado con la clave **_id**.

Este es un número compuesto por 12 bytes que asegura un identificador único para cada documento. Se considera clave primaria y contiene tres secciones:

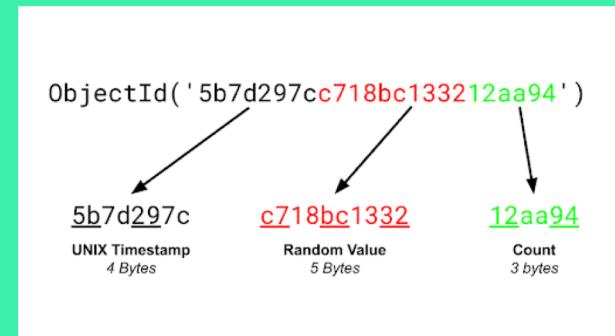
<https://docs.mongodb.com/manual/reference/method/ObjectId/>

```
> db.Employee.find()
{
    "_id" : ObjectId("563479cc8a8a4246bd27d784"),
    "Employeeid" : 1,
    "EmployeeName" : "Smith"
}

{
    "_id" : ObjectId("563479d48a8a4246bd27d785"),
    "Employeeid" : 2,
    "EmployeeName" : "Mohan"
}

{
    "_id" : ObjectId("563479df8a8a4246bd27d786"),
    "Employeeid" : 3,
    "EmployeeName" : "Joe"
}
```

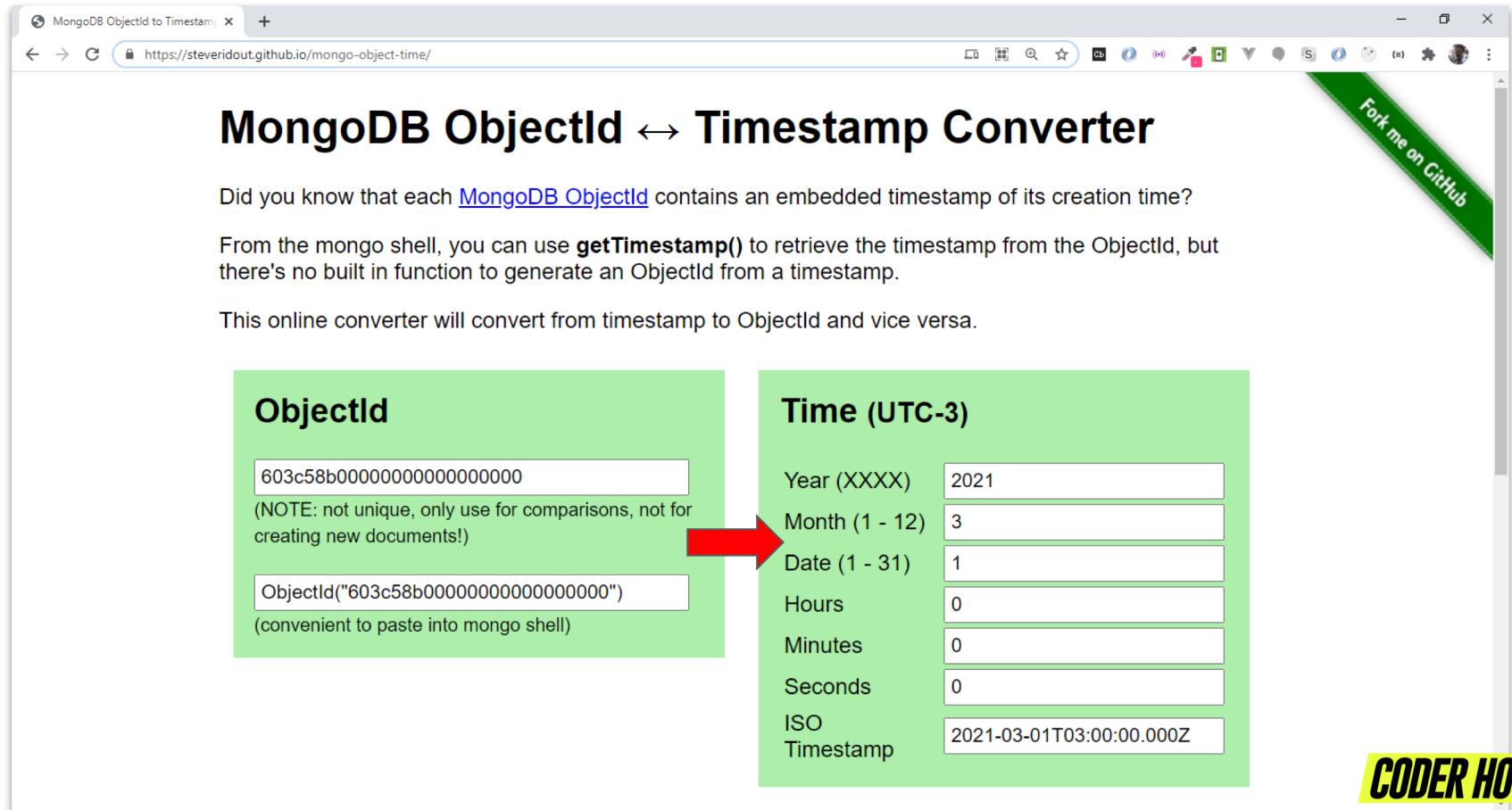
ObjectId



CODER HOUSE

ObjectId: Timestamp converter

Website: <https://steveridout.github.io/mongo-object-time/>



The screenshot shows a web browser window with the title "MongoDB ObjectId ↔ Timestamp Converter". The page content includes a note about ObjectId timestamps, instructions for using the mongo shell, and a statement about the online converter's purpose. Two main input fields are shown: "ObjectId" containing "603c58b00000000000000000000000000" and "Time (UTC-3)" with various fields filled out. A red arrow points from the ObjectId field towards the Time fields, indicating the conversion direction.

MongoDB ObjectId ↔ Timestamp Converter

Did you know that each [MongoDB ObjectId](#) contains an embedded timestamp of its creation time?

From the mongo shell, you can use `getTimestamp()` to retrieve the timestamp from the ObjectId, but there's no built in function to generate an ObjectId from a timestamp.

This online converter will convert from timestamp to ObjectId and vice versa.

ObjectId	Time (UTC-3)
603c58b00000000000000000000000000	Year (XXXX) 2021
(NOTE: not unique, only use for comparisons, not for creating new documents!)	Month (1 - 12) 3
ObjectId("603c58b00000000000000000000000000")	Date (1 - 31) 1
(convenient to paste into mongo shell)	Hours 0
	Minutes 0
	Seconds 0
	ISO Timestamp 2021-03-01T03:00:00.000Z

CODER HOUSE

CONTADORES

CODER HOUSE



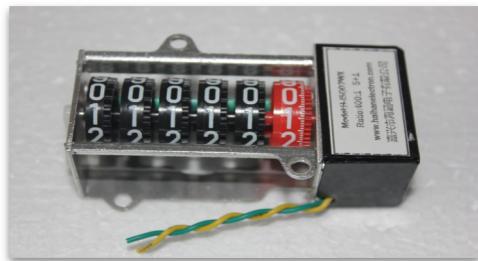
MongoDB: Contadores



Comandos Count

Son funciones que cuentan la cantidad de documentos presentes en una colección. Algunas de ellas pueden tener la opción de filtro.

```
db.coll.estimatedDocumentCount() // estimation based on collection metadata  
db.coll.countDocuments({age: 32}) // alias for an aggregation pipeline - accurate count
```



MongoDB: Contadores

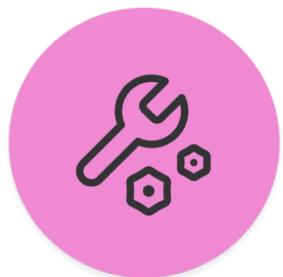


- **db.coll.estimatedDocumentCount()**

Devuelve la cantidad total de documentos encontrados en la colección.

- **db.coll.countDocuments({key: val})**

Devuelve la cantidad de documentos encontrados en la colección (con filtro de query).



CRUD

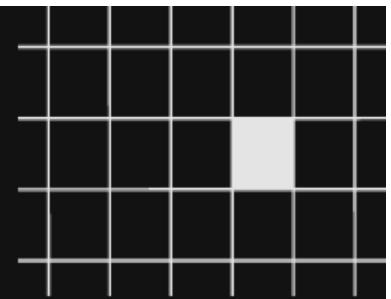
Tiempo: 10 minutos

CODER HOUSE



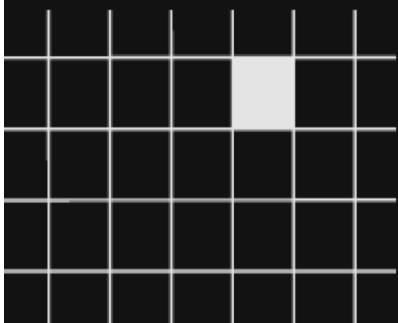
Utilizando Mongo shell, realizar las siguientes acciones:

- 1) Crear una base de datos en MongoDB llamada 'empresa'.
- 2) Crear una colección llamada 'clientes' dentro de esa base.
- 3) Insertar un documento en la colección clientes que contenga los campos 'nombre' y 'edad'.
- 4) Insertar un array de tres documentos con el mismo formato y valores distintos en la colección clientes (modo bulk).
- 5) Utilizando un sólo comando, crear una colección llamada 'articulos' e insertar dentro de ella un array de cuatro documentos con los campos 'nombre', 'precio' y 'stock'.
- 6) Mostrar las colecciones de la base 'empresa'.
- 7) Listar todos los documentos dentro de cada una de las colecciones.
- 8) Tomar el Object ID de uno de los documentos y comprobar fecha y hora de creación.
- 9) Mostrar la cantidad de documentos que tiene la colección articulos.



BREAK

¡5/10 MINUTOS Y VOLVEMOS!



FILTROS

CODER HOUSE



mongoDB

MongoDB: Comandos CRUD

READ con Filtros



Comando Read con filtros de búsqueda

db.coll.find({key: {\$operator: val}}) : devuelve los documentos según el operador de **filtro** utilizado.

```
db.coll.find({"year": {$gt: 1970}})
db.coll.find({"year": {$gte: 1970}})
db.coll.find({"year": {$lt: 1970}})
db.coll.find({"year": {$lte: 1970}})
db.coll.find({"year": {$ne: 1970}})
db.coll.find({"year": {$in: [1958, 1959]}})
db.coll.find({"year": {$nin: [1958, 1959]}})
```

CODER HOUSE



MongoDB: Operadores para Filtros de Query



- **\$and** : Realiza operación AND -> sintaxis: {\$and: [{},{}] }
- **\$or** : Realiza operación OR -> sintaxis: {\$or: [{},{}] }
- **\$lt** : Coincide con valores que son menores que un valor especificado.
- **\$lte** : Coincide con valores menores o iguales a un valor especificado.
- **\$gt** : Coincide con valores mayores a un valor especificado.
- **\$gte** : Coincide con valores mayores o iguales a un valor especificado.
- **\$ne** : Coincide con valores que no son iguales a un valor especificado.
- **\$eq** : Selecciona los documentos que son iguales a un valor especificado.



MongoDB: Operadores para Filtros de Query



- **\$exists** : Selecciona los documentos según la existencia de un campo.
- **\$in** : Selecciona los documentos especificados en un array.
sintaxis: {key:{\$in: [array of values] } }
- **\$nin** : Coincide con ninguno de los valores especificados en un array.
- **\$size** : Coincide con el número de elementos especificados.
- **\$all** : Coincide con todos los valores definidos dentro de un array.
- **\$elemMatch** : Coincide con algún valor definido dentro del query.

<https://docs.mongodb.com/manual/reference/operator/query/>

CODER HOUSE



mongoDB

MongoDB: Comandos CRUD

READ con Filtros combinados



Comando Read con filtros de búsqueda combinados

```
// Logical
db.coll.find({name:{$not: {$eq: "Max"}}})
db.coll.find({$or: [{"year" : 1958}, {"year" : 1959}]})
db.coll.find({$nor: [{price: 1.99}, {sale: true}]})
db.coll.find({
  $and: [
    {$or: [{qty: {$lt :10}}, {qty :{$gt: 50}}]}, 
    {$or: [{sale: true}, {price: {$lt: 5 }}]}
  ]
})

// Element
db.coll.find({name: {$exists: true}})
db.coll.find({"zipCode": {$type: 2 }})
db.coll.find({"zipCode": {$type: "string"}})
```

CODER HOUSE



MongoDB

Búsqueda Avanzada



- **db.coll.distinct(val)**
devuelve un array con los distintos valores que toma un determinado campo en los documentos de la colección.
- **db.coll.find({doc.subdoc:value})**
Se utiliza para filtrar subdocumentos.
- **db.coll.find({name: /[^]Max\$/i})**
filtra utilizando expresiones regulares



mongoDB

MongoDB: Comandos CRUD

READ con Filtros avanzados



Comando Read con filtros avanzados y proyección

```
// Regex
db.coll.find({name: /^Max/})    // regex: starts by letter "M"
db.coll.find({name: /Max$/i}) // regex case insensitive

// Array
db.coll.find({tags: {$all: ["Realm", "Charts"]}})
db.coll.find({field: {$size: 2}}) // impossible to index - prefer storing the size of the array
db.coll.find({results: {$elemMatch: {product: "xyz", score: {$gte: 8}}}})

// Projections
db.coll.find({"x": 1}, {"actors": 1})           // actors + _id
db.coll.find({"x": 1}, {"actors": 1, "_id": 0}) // actors
db.coll.find({"x": 1}, {"actors": 0, "summary": 0}) // all but "actors" and "summary"

// Sort, skip, limit
db.coll.find({}).sort({"year": 1, "rating": -1}).skip(10).limit(3)
```

CODER HOUSE



Proyecciones en MongoDB



- La **proyección** se utiliza para **devolver un conjunto determinado de campos** de un documento. En general devolvemos todos los campos de un documento, pero es posible que no necesitemos todos.
- Es equivalente en SQL de pasar de hacer un *SELECT ** a realizar *SELECT nombrecampo*.
- Las proyecciones deben ser incorporadas en el **segundo parámetro** del comando find. Por ej. **db.coll.find({}, {"nombre":1})** muestra sólo el campo nombre y el _id de todos documentos de la coll
- Las proyecciones se realizan indicando el nombre del campo, con valor 1 si queremos mostrarlo y 0 por el contrario.



MongoDB: sort limit skip



- **sort({ campoA: 1 ó -1 , campoB: 1 ó -1 , ... })** : Especifica el **orden** en el que la consulta devuelve documentos coincidentes. El ó los campos por los cuales ordena pueden contener los valores 1 y -1, estableciendo orden ascendente y descendente respectivamente. El orden se evalúa de izquierda a derecha en caso que los valores coincidan.
- **limit(num)**: Especifica el **número máximo** de documentos devueltos.
- **skip(offset)** : **Saltea** la cantidad de documentos especificada.

Se pueden utilizar en forma combinada:

```
db.Employee.find().skip(2).limit(3).sort({Employeeid:-1})
```

UPDATE Y DELETE

CODER HOUSE



mongoDB

MongoDB: Comandos CRUD

UPDATE



Comando Update (update)

```
db.coll.update({_id: 1}, {$set: {"year": 2016, name: "Max"}})
db.coll.update({_id: 1}, {$unset: {"year": 1}})
db.coll.update({_id: 1}, {$rename: {"year": "date"} })
db.coll.update({_id: 1}, {$inc: {"year": 5}})
db.coll.update({_id: 1}, {$mul: {price: NumberDecimal("1.25"), qty: 2}})
db.coll.update({_id: 1}, {$min: {"imdb": 5}})
db.coll.update({_id: 1}, {$max: {"imdb": 8}})
db.coll.update({_id: 1}, {$currentDate: {"lastModified": true}})
db.coll.update({_id: 1}, {$currentDate: {"lastModified": {$type: "timestamp"}}})
```



mongoDB

MongoDB: Comandos CRUD

UPDATE



Comando Update y variantes

```
// Update many
db.coll.update({"year": 1999}, {$set: {"decade": "90's"}}, {"multi":true})
db.coll.updateMany({"year": 1999}, {$set: {"decade": "90's"}})

// FindOneAndUpdate
db.coll.findOneAndUpdate({"name": "Max"}, {$inc: {"points": 5}}, {returnNewDocument: true})

// Upsert
db.coll.update({_id: 1}, {$set: {item: "apple"}, $setOnInsert: {defaultQty: 100}}, {upsert: true})

// Replace
db.coll.replaceOne({"name": "Max"}, {"firstname": "Maxime", "surname": "Beugnet"})

// Save
db.coll.save({"item": "book", "qty": 40})
```

CODER HOUSE



UPDATE: Detalle de Comando



db.collection.updateOne(query, update, options)

- **query**: especifica el filtro de documentos a ser actualizados.
- **update**: contiene los datos a ser actualizados con sus operadores respectivos: \$set, \$unset, \$inc, \$rename, \$mul, \$min, \$max, etc.
- **options**: contiene varias opciones para la actualización, entre ellas:
 - upsert (true ó false) : Es una opción para hacer un insert en caso de que el registro no exista.

db.coll.updateMany(query, update, options)

- Igual que el anterior, pero hace una actualización múltiple en caso de que el filtro de query devuelva varios resultados.



MongoDB Comandos CRUD

DELETE



Comando Delete (**delete**)

```
db.coll.deleteOne({name: "Max"})
db.coll.deleteMany({name: "Max"})
db.coll.deleteMany({}) // Deletes all the docs but not the collection itself and its index
```



DELETE

Detalle de Comando



- **db.coll.deleteOne({key: val})**: Elimina un sólo documento (el primero) que coincide con el filtro especificado.
- **db.coll.deleteMany({key: val})**: Elimina todos los documentos que coinciden con el filtro especificado.
- **db.coll.remove({key: val})**: Elimina documentos de una colección.
- **db.coll.findOneAndDelete(filter, options)**: Elimina un solo documento según el filtro y los criterios de clasificación. Algunas de las **options** pueden ser
 - **sort**: para establecer orden para el filtro
 - **projection**: para elegir campos de salida.



MongoDB Cheat Sheet Commands

Basic Mongo DB

db	Show name of current database
mongod	Start database
mongo	Connect to database
show dbs	Show databases
use db	Switch to database db
show collections	Display current database collections

Create

insert(data)	insert document(s) returns write result
insertOne (data, options)	insert one document
insertMany(data, options)	insert many documents
insertMany([{}, {}, {}])	needs square brackets

Read

db.collection.find() Display documents from collection

find(filter, options) find all matching documents

findOne(filter, options) find first matching document

Update

updateOne(filter, data, options) Change one document

updateMany(filter, data, options) Change many documents

replaceOne(filter, data, options) Replace document entirely

Delete

deleteOne(filter, options) Delete one document

deleteMany(filter, options) Delete many documents



MongoDB Cheat Sheet Commands

Filters

{ "key": "value"}	Used for filter arguments to filter collection
{key: {\$operator: value} }	Operators for querying data
{key: {\$exists: true} }	Matches all documents containing subdocument key
\$eq	Matches values that are equal to a specified value.
\$gt	Matches values that are greater than a specified value.
\$gte	Matches values that are greater than or equal to a specified value.
\$in	Matches any of the values specified in an array
syntax:	{key:{\$in: [array of values] } }
\$lt	Matches values that are less than a specified value.

Filters (cont)

\$lte	Matches values that are less than or equal to a specified value.
\$ne	Matches all values that are not equal to a specified value.
\$nin	Matches none of the values specified in an array.
\$and	Performs AND operation
syntax:	{\$and: [{}, {}] }
{key: {\$op: filter}, {filter}}	\$and operator is necessary when the same field or operator has to be specified in multiple expressions
find({doc.s- ubdoc:value})	Filter sub documents

Functions

.count()	Counts how many results
.sort(filter)	Sort ascen:1 descend: -1



OPERACIONES CON FILTROS

Tiempo: 15 minutos



Sobre la base y los datos cargados anteriormente

- 1) Insertar cinco documentos más en la colección clientes con los siguientes datos:

```
{ "nombre" : "Pablo", "edad" : 25 }  
{ "nombre" : "Juan", "edad" : 22 }  
{ "nombre" : "Lucia", "edad" : 25 }  
{ "nombre" : "Juan", "edad" : 29 }  
{ "nombre" : "Fede", "edad" : 35 }
```

- 1) Listar todos los documentos de la colección clientes ordenados por edad descendente.
- 2) Listar el cliente más joven.
- 3) Listar el segundo cliente más joven.
- 4) Listar los clientes llamados 'Juan'
- 5) Listar los clientes llamados 'Juan' que tengan 29 años.
- 6) Listar los clientes llamados 'Juan' ó 'Lucia'.



- 8) Listar los clientes que tengan más de 25 años.
- 9) Listar los clientes que tengan 25 años ó menos.
- 10) Listar los clientes que NO tengan 25 años.
- 11) Listar los clientes que estén entre los 26 y 35 años.
- 12) Actualizar la edad de Fede a 36 años, listando y verificando que no aparezca en el último listado.
- 13) Actualizar todas las edades de 25 años a 26 años, listando y verificando que aparezcan en el último listado.
- 14) Borrar los clientes que se llamen 'Juan' y listar verificando el resultado.



Usuarios y permisos



CODER HOUSE

MongoDB ***Usuarios y permisos***

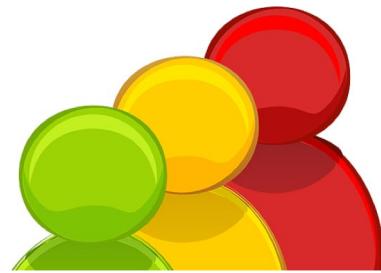
Creación de un usuario con permisos y su eliminación

```
use admin  
db.createUser({"user": "root", "pwd": "rootpwd", "roles": ["root"]})  
db.dropUser("root")
```

Permisos y Roles

<https://docs.mongodb.com/manual/tutorial/manage-users-and-roles/>

<https://docs.mongodb.com/manual/reference/built-in-roles/>



Crear usuarios y asignar roles en MongoDB

Ejemplo
en vivo



En **MongoDB** es posible **crear usuarios y asignarles acceso mediante roles**. Veremos cómo crear un usuario y asignarle un rol para que tenga ciertos accesos limitados a una base de datos.

Crearemos dos usuarios para una base de datos

- **Usuario lector:** tendrá **acceso de lectura** a la base de datos.
- **Usuario escritor:** tendrá **acceso de lectura y escritura** a la base de datos.



Creando un usuario de solo lectura

Ejemplo
en vivo



Crearemos el usuario *lector*, que solo tiene acceso de lectura.

Utilizaremos el método **createUser**. Este acepta como parámetro un objeto con las siguientes propiedades:

- **user**: nombre del usuario. Le asignaremos *lector*.
- **pwd**: contraseña para el usuario.
- **roles**: arreglo de objetos. Sirve si el usuario tendrá acceso a múltiples bases de datos, estableciendo permisos para cada acceso.



Creando un usuario de solo lectura

Ejemplo
en vivo



IMPORTANTE:

- Ejecutar el servidor con acceso root: **mongod**
- Ejecutar en el cliente **use admin** antes de **createUser(...)**

```
db.createUser(  
{  
  user: "lector",  
  pwd: "123456",  
  roles:[  
    {role: "read", db: "blog"}  
  ]  
})
```

- **MongoDB** viene con roles predefinidos. Uno de ellos es el **role read**, que permite ejecutar métodos de sólo lectura.
- La **propiedad db** es donde se indica a qué base de datos se le asignará dicho rol.

CODER HOUSE



Creando un usuario de escritura y lectura

Ejemplo
en vivo



Crearemos el usuario escritor. El proceso es similar, pero en este caso el role ya no será read sino readWrite.

```
db.createUser(  
  {  
    user: "escritor",  
    pwd: "123456",  
    roles: [  
      { role: "readWrite", db: "blog" }  
    ]  
  }  
)
```

Con el rol `readWrite` el usuario tendrá acceso a los **métodos de lectura y escritura** de la base de datos.

A continuación debemos verificar que cada usuario cuenta con los accesos correctos.

CODER HOUSE



Pruebas de acceso: usuario *read*

Ejemplo
en vivo



1. Para poder ingresar al shell de mongo con el usuario usaremos los parámetros -u y -p.

```
mongo -u lector -p 123456
```

2. Verificamos si podemos leer los posts

```
use blog;  
db.posts.find();
```

3. Obtenemos respuesta correctamente, mostrando los dos documentos existentes

```
{ "_id" : ObjectId("5e438449ea91620ed74d3cf3"), "title" : "Articulo 1" }  
{ "_id" : ObjectId("5e43844cea91620ed74d3cf4"), "title" : "Articulo 2" }
```

4. Ahora intentaremos insertar un nuevo documento

```
db.posts.insertOne({ title: "Articulo 3" });
```

5. Debería lanzarnos un error parecido al siguiente:

```
{  
  "errmsg" : "not authorized on blog to execute command" # mensaje cortado por temas de espacio  
  "codeName" : "Unauthorized"  
}
```

CODER HOUSE



Pruebas de acceso: usuario write

Ejemplo
en vivo



1. Para el usuario escritor, primero debemos salir del shell y volver a loguearnos:

```
mongo -u escritor -p 123456 --authenticationDatabase blog
```

2. Leemos los posts usando el método find:

```
db.posts.find();
# { "_id" : ObjectId("5e438449ea91620ed74d3cf3"), "title" : "Articulo 1" }
# { "_id" : ObjectId("5e43844cea91620ed74d3cf4"), "title" : "Articulo 2" }
```

3. Funciona bien el permiso de lectura, ahora intentaremos insertar el Artículo #3:

```
db.posts.insertOne({ title: "Articulo 3" });
```

4. Funciona bien el permiso de lectura, ahora intentaremos insertar el Artículo #3:

```
{
  "acknowledged" : true,
  "insertedId" :
  ObjectId("5e4385e6e2306e7554d9e60f")
}
```

Para estas pruebas, es importante ejecutar el servidor en modo autenticación: usar el comando `mongod --auth`

CODER HOUSE



USUARIO 'ENCARGADO'

Tiempo: 10 minutos

Desafío
generico



- 1) Crear un usuario que acceda con nombre 'encargado' y clave 'qwerty123' sólo a la base 'empresa' y tenga permisos de lectura/escritura.
- 2) Acceder con ese usuario y listar de la colección clientes sólo los nombres de los mismos.
- 3) Verificar que no pueda acceder si sus credenciales no corresponden.



MONGODB

CODER HOUSE

MONGODB

Formato: archivo de texto con las consultas realizadas y la carpeta de la base de datos comprimida en un zip.

Sugerencia: Si es un archivo en línea, configurar los permisos de acceso.

Desafío
entregable



>> Consigna: Utilizando Mongo Shell, crear una base de datos llamada ecommerce que contenga dos colecciones: mensajes y productos.

- 1) Agregar 10 documentos con valores distintos a las colecciones mensajes y productos. El formato de los documentos debe estar en correspondencia con el que venimos utilizando en el entregable con base de datos MariaDB.
- 2) Definir las claves de los documentos en relación a los campos de las tablas de esa base. En el caso de los productos, poner valores al campo precio entre los 100 y 5000 pesos(eligiendo valores intermedios, ej: 120, 580, 900, 1280, 1700, 2300, 2860, 3350, 4320, 4990).
- 3) Listar todos los documentos en cada colección.
- 4) Mostrar la cantidad de documentos almacenados en cada una de ellas.

MONGODB

Formato: archivo de texto con las consultas realizadas y la carpeta de la base de datos comprimida en un zip.

Sugerencia: Si es un archivo en línea, configurar los permisos de acceso.

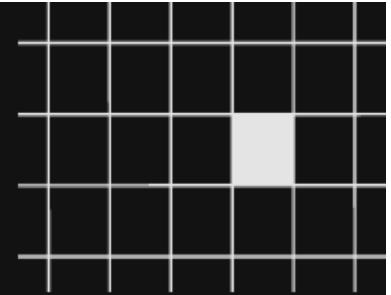
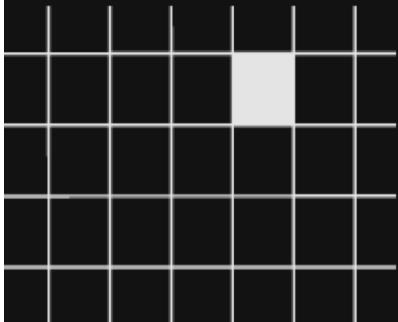
Desafío
entregable



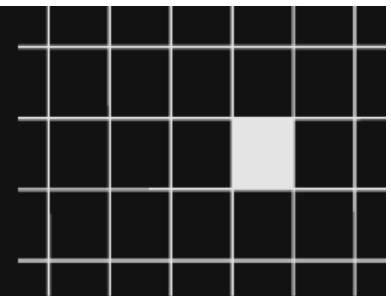
- 5) Realizar un CRUD sobre la colección de productos:
 - a) Agregar un producto más en la colección de productos
 - b) Realizar una consulta por nombre de producto específico:
 - i) Listar los productos con precio menor a 1000 pesos.
 - ii) Listar los productos con precio entre los 1000 a 3000 pesos.
 - iii) Listar los productos con precio mayor a 3000 pesos.
 - iv) Realizar una consulta que traiga sólo el nombre del tercer producto más barato.
 - c) Hacer una actualización sobre todos los productos, agregando el campo stock a todos ellos con un valor de 100.
 - d) Cambiar el stock a cero de los productos con precios mayores a 4000 pesos.
 - e) Borrar los productos con precio menor a 1000 pesos
- 6) Crear un usuario 'pepe' clave: 'asd456' que sólo pueda leer la base de datos ecommerce. Verificar que pepe no pueda cambiar la información.

CODER HOUSE

¿PREGUNTAS?

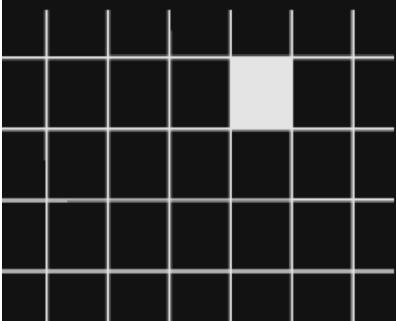


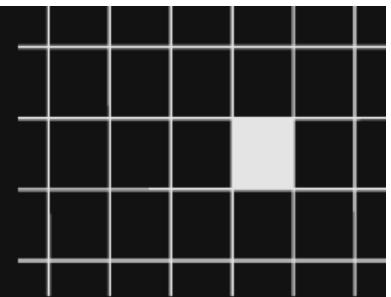
MUCHAS GRACIAS!



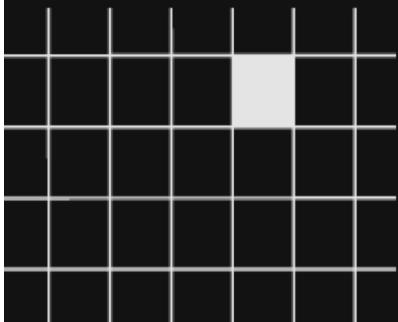
Resumen de lo visto en clase hoy:

- Operaciones CRUD en MongoDB
- Listado y detalle de comandos
- Manejo de usuarios y roles





OPINA Y VALORA ESTA CLASE



#DEMOCRATIZANDOLAEDUCACIÓN

CODER HOUSE