

Curso de MongoDB

MongoDB es orientado a documentos. En estas bases de datos se empareja cada clave con una estructura de datos compleja que se denomina '**documento**'.

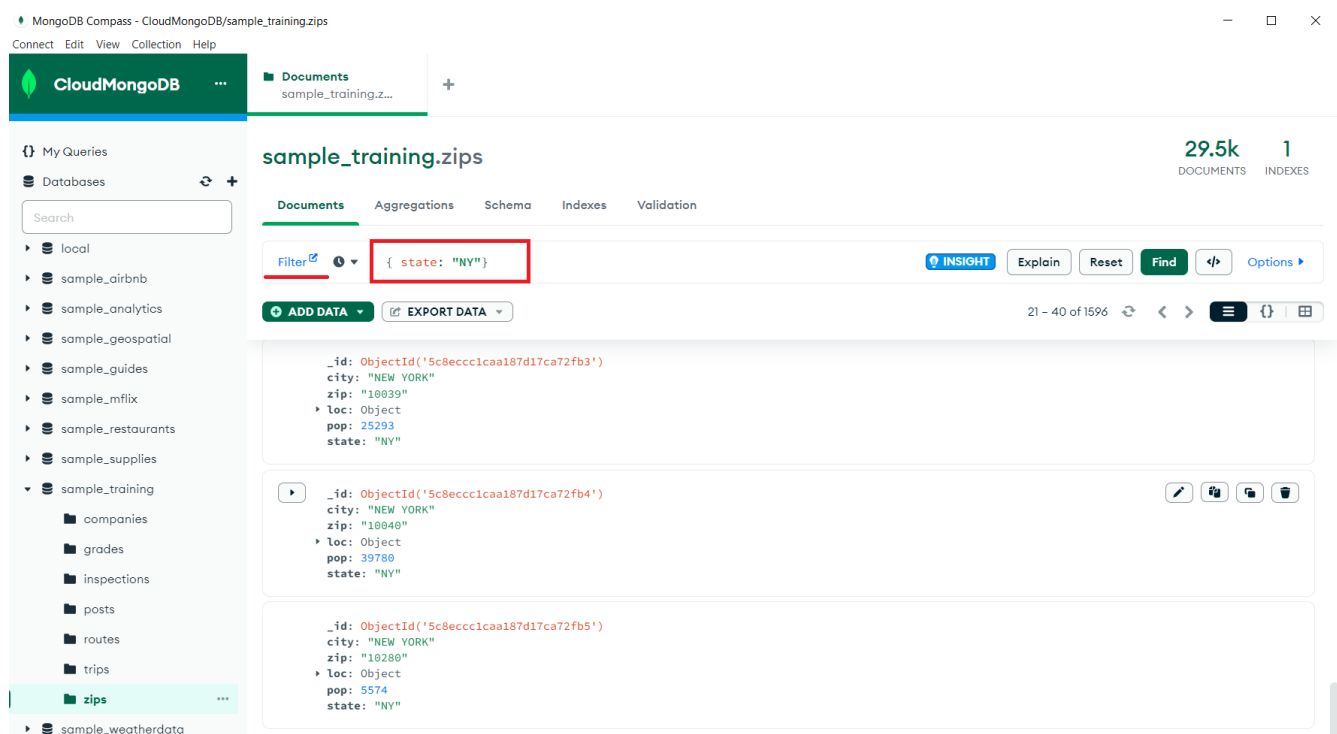
Los **grafos** se utilizan para almacenar información sobre redes de datos, como las conexiones sociales.

clave-valor: Son las bases de datos NoSQL más simples. Cada elemento de la base de datos se almacena como un nombre de atributo (o «clave»), junto con su valor.

Orientadas a columnas: Estas bases de datos, como Cassandra o HBase, permiten realizar consultas en grandes conjuntos de datos y almacenan los datos en columnas, en lugar de filas.

Document: Una forma de organizar y almacenar información con un conjunto de pares **clave-valor**.

Conectando cloudMongoDB con MongoDB compass:



2. Copy the connection string, then open MongoDB Compass

```
mongodb+srv://hmaluy:<password>@cluster0.zyrbe3x.mongodb.net/
```

Mongo en VSCode

Creamos un archivo `.gitignore` y vamos a la pagina **gitignore.io**.
Le “decimos” que queremos ignorar archivos de windows, linux y max.
Y generamos nuestro contenido para nuestro archivo `.gitignore`.

gitignore.io

Create useful .gitignore files for your project


Windows × Linux × macOS ×

Create




También creamos un archivo de `.editorconfig`.

```
.editorconfig ×
.editorconfig
1  root = true
2
3  [*]
4  indent_style = space
5  indent_size = 2
6  charset = utf-8
7  trim_trailing_whitespace = true
8  insert_final_newline = true
9  end_of_line = lf
10 # editorconfig-tools is unable to ignore longs strings or urls
11 max_line_length = off
12
13 [CHANGELOG.md]
14 indent_size = false
```


Instalamos una extensión para mongo hecha por los mismos desarrolladores de mongoDB.



MongoDB for VS Code v1.2.1

MongoDB  mongodb.com |  1,027,574 |  (34)

Connect to MongoDB and Atlas directly from your VS Code environment, navigate your dat...

Installing 

Nos conectamos a través de la extensión a cloudmongodb y así podemos acceder a las bases de datos + colecciones + documentos desde VSCode

3. Connect to your MongoDB deployment.

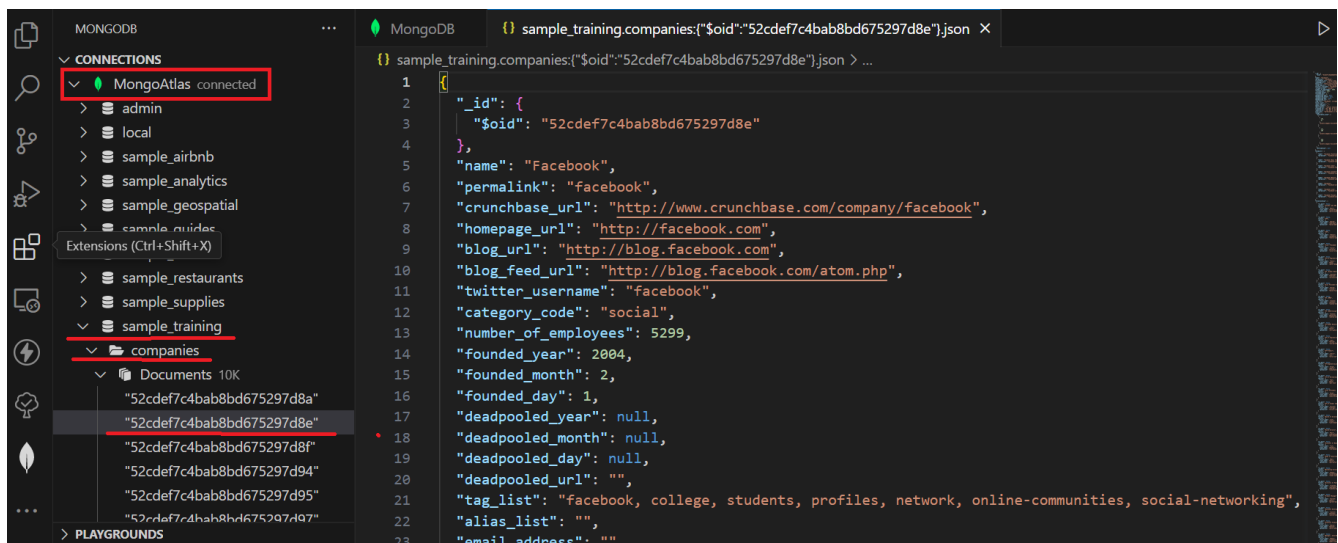
Paste your connection string into the Command Palette.

```
mongodb+srv://hmaluy:<password>@cluster0.zyrbe3x.mongodb.net/
```

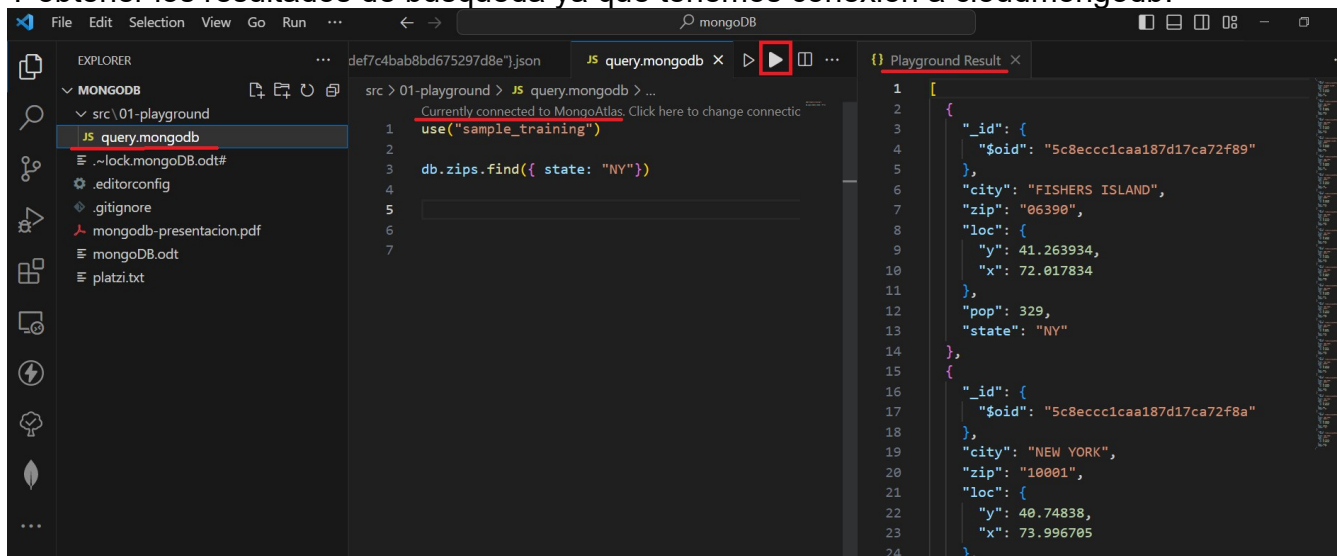
Replace **<password>** with the password for the **hmaluy** user.

When entering your password, make sure all special characters are [URL encoded](#). [↗](#)

Y finalmente tenemos la conexión activa en VSCode.



También podemos ejecutar comandos de mongo desde un archivo ejemplo (query.mongodb). Y obtener los resultados de búsqueda ya que tenemos conexión a cloudmongodb.



Recuerda que aquí estamos usando la API directa de mongo, (mongo query language). Para traernos peticiones, consultas, etc.

Después de instalar docker configuramos nuestro archivo **docker-compose.yml**

```
docker-compose.yml
1  version: '3.9'
2
3  services:
4    mongodb:
5      image: mongo:latest
6      ports:
7        - 27017:27017
8      environment:
9        - MONGO_INITDB_ROOT_USERNAME=root
10       - MONGO_INITDB_ROOT_PASSWORD=root123
11     volumes:
12       - /mongodata:/data/db
```

Y levantamos el servicio que creamos.

```
toreohm@DESKTOP-06TNP1J: /mnt/c/Users/INSPIRON 7460/OneDrive/Documentos/platzi/databases/mongoDB$ sudo docker-compose up -d mongodb
[+] Running 1/1
  Container mongodb-mongodb-1 Started
toreohm@DESKTOP-06TNP1J: /mnt/c/Users/INSPIRON 7460/OneDrive/Documentos/platzi/databases/mongoDB$ docker-compose ps -a
NAME                IMAGE             COMMAND                  SERVICE    CREATED         STATUS         PORTS
mongodb-mongodb-1   mongo:latest      "docker-entrypoint.s..."  mongodb    4 seconds ago   Up 2 seconds   0.0.0.0:27017->27017/tcp
```

Nos conectamos a través de MongoDB compass, creamos una nueva base de datos y una colección. E insertamos un documento a dicha colección:

Insert Document

To collection platzi_store.productos

VIEW  

```
1  /**
2  * Paste one or more documents here
3  */
4  {
5    "_id": {
6      "$oid": "64e7f22e62a35da454a5ddd5"
7    },
8    "name": "Product 1",
9    "price": 1200
10 }
```

Cancel

Insert

MongoDB Compass - mongodocker/platzi_store.productos

Connect Edit View Help

mongodocker ...

Documents
platzi_store.prod...

My Queries

Databases

Search

- admin
 - temproles
 - tempusers
- config
 - settings
- local
 - replset.election
 - replset.minvalid
 - startup_log
- platzi_store **Base de datos**
 - productos **Colección**

platzi_store.productos

Documents Aggregations Schema Indexes Validation

Filter ⓘ ⓘ Type a query: { field: 'value' }

ADD DATA EXPORT DATA

`_id: ObjectId('64e7f22e62a35da454a5ddd5')`
`name: "Product 1"`
`price: 1200`

Documento de la colección.

Conectándonos usando mongosh (terminal)

```
toreohm@DESKTOP-06TNPIJ:/mnt/c/Users/INSPIRON 7460/OneDrive/Documentos/platzi/databases/mongoDB$ !1995
docker-compose up -d mongodb
[+] Running 1/1
  Container mongodb-mongoddb-1 Started
toreohm@DESKTOP-06TNPIJ:/mnt/c/Users/INSPIRON 7460/OneDrive/Documentos/platzi/databases/mongoDB$ docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS                               NAMES
f051d883e106   mongo:latest   "docker-entrypoint.s..." 16 hours ago   Up 6 seconds   0.0.0.0:27017->27017/tcp        mongodb-mongoddb-1
```

Y después nos conectamos a la bash de mongo: **docker-compose exec mongodb bash**

Sacamos la url de mongo compass para conectarnos con el comando **mongosh <url>**

Compass

New connection +

Saved connections

- CloudMongoDB 24 ago 2023, 15:05
- mongodocker 24 ago 2023, 18:10**

Recents

- cluster0.zyrbe3x.mongo... 24 ago 2023, 14:59
- localhost:27017

mongodocker

Connect to a MongoDB deployment

FAVORITE

URI ⓘ Edit Connection String ⓘ

`mongodb://root:root123@localhost:27017/?authMechanism=DEFAULT&tls=false`

Advanced Connection Options

Save Connect

```

root@f051d883e106:/# mongosh "mongodb://root:root123@localhost:27017/?authMechanism=DEFAULT&tls=false"
Current Mongosh Log ID: 64e8dc5db7f1f0156a5c0dfa
Connecting to:      mongodb://<credentials>@localhost:27017/?authMechanism=DEFAULT&tls=false&directConnection=true&serverSelectionTimeoutMS=20
=mongosh+1.10.5
Using MongoDB:      7.0.0
Using Mongosh:      1.10.5

For mongosh info see: https://docs.mongodb.com/mongosh-shell/

-----
The server generated these startup warnings when booting
2023-08-25T16:01:53.387+00:00: Using the XFS filesystem is strongly recommended with the WiredTiger storage engine. See http://dochub.mongodb.org/core/notes-filesystem
2023-08-25T16:01:54.259+00:00: /sys/kernel/mm/transparent_hugepage/enabled is 'always'. We suggest setting it to 'never'
2023-08-25T16:01:54.259+00:00: vm.max_map_count is too low
-----
test> |

```

Comandos de mongo:

- **show dbs**
- **use platzi_store**
- **db.productos.find()**

Json vs. Bson

BSON significa Binary Javascript Object Notation.

Las ventajas de JSON son:

- Amigable (fácilmente podemos organizar nuestra información en un documento)
- Se puede leer
- Es un formato muy usado

Las desventajas de JSON son:

- Basado en texto
- Consume mucho espacio
- Es limitado en tipos de datos: string, boolean, number, arrays, object, null

El JSON típico sería éste:

```

{
  "_id": "5c8eccc1caa187d17ca6ed16",
  "city": "ALPINE",
  "zip": "35014",
  "loc": {
    "y": 33.331165,
    "x": 86.208934
  },
  "pop": 3062,
  "state": "AL"
}

```

Las ventajas de BSON son:

- Representación binaria de JSON
- No consume espacio
- Alto rendimiento
- Tipos de datos: +, date, raw binary, integer, long, float

Las desventajas de BSON son:

- No es estándar (No muchos utilizan este formato. mongo lo creo y lo utiliza para si mismo).
- Es un formato para la maquina.

Ejemplo: JSON

```
{  
  "hello" : "world"  
}
```

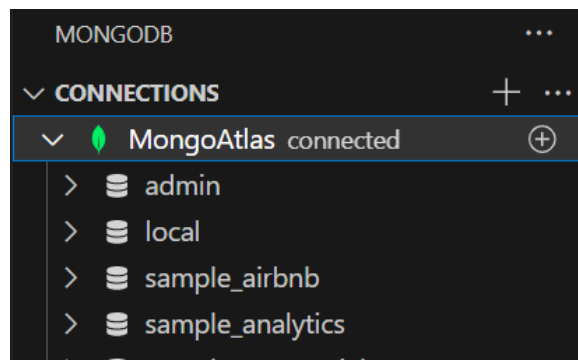
Ejemplo: BSON

```
\x16\x00\x00\x00      // total document size  
\x02                  // 0x02 = type String  
hello\x00             // field name  
\x06\x00\x00\x00world\x00 // field value  
\x00                  // 0x00 = end of object
```

Nosotros lo vamos a ver como JSON, sin embargo MongoDB internamente lo maneja como BSON.

Insertando un documento

- **docker rm -f mongodb-mongodb-1** (Por cualquier cosa quitamos nuestro docker actual)
- Nos conectamos en mongoatlas en VSCode



- **use("platzi_store")** (Con este comando mongo crea la db en caso de no existir)
- Al insertar registros, tú puedes crear tu propio ID, de lo contrario mongo lo autogenera.
- Ten en cuenta que no se puede repetir el ID, de lo contrario mongo genera un error.

```
use("platzi_store")

db.productos.insertOne({
  _id: 1,
  name: "Product 2",
  price: 1000
})

db.productos.insertOne({
  name: "Product 3",
  price: 100
})
```

← Insertar registros

Buscar registros →

```
use("platzi_store")

db.productos.find()
```

Insertando varios documentos

- **db.productos.drop()** (Con este comando borramos todos los documentos de la colección productos.)
- **db.productos.insertMany()** (Con este comando insertamos muchos documentos definidos como un objeto dentro de un array)
- Si tienes una masa de datos y varios documentos repiten el ID, mongodb te muestra un error, pero sólo van a quedar fuera los que tenga errores de colisión. Los demás registros sí se insertan. Entonces esto no es transaccional.
- Sin embargo, con respecto al punto anterior: Tomar en cuenta que después del error ya no crea más productos; deja de insertar registros (documentos).
- En la imagen de abajo, tenemos que 2 ids se repiten. Mongo marcaría el error en el tercer documento. Entonces sólo se insertan 2 documentos de un total de 4.

El consulta es la imagen de la derecha y el resultado la imagen de la izquierda:

```
use("platzi_store")

db.productos.drop()

db.productos.insertMany([
  { _id: 1, name: "Producto 1", price: 100 },
  { _id: 2, name: "Producto 2", price: 200 },
  { _id: 1, name: "Producto 3", price: 300 },
  { _id: 4, name: "Producto 4", price: 3001 }
])

db.productos.find()
```

```
[
  {
    "_id": 1,
    "name": "Producto 1",
    "price": 100
  },
  {
    "_id": 2,
    "name": "Producto 2",
    "price": 200
  }
]
```


Entonces aquí tendríamos un inconveniente, porque nos conviene tener un set de datos el cuál solo deje fuera a los documentos de los ids que se repiten. Y no que deja de insertar documentos (registros) a partir del error de colisión.

¿Cómo podemos resolver este inconveniente?

```
use("platzi_store")

db.productos.drop()

db.productos.insertMany([
  { _id: 1, name: "Producto 1", price: 100 },
  { _id: 2, name: "Producto 2", price: 200 },
  { _id: 1, name: "Producto 3", price: 300 },
  { _id: 4, name: "Producto 4", price: 3001 }
], { ordered: false })

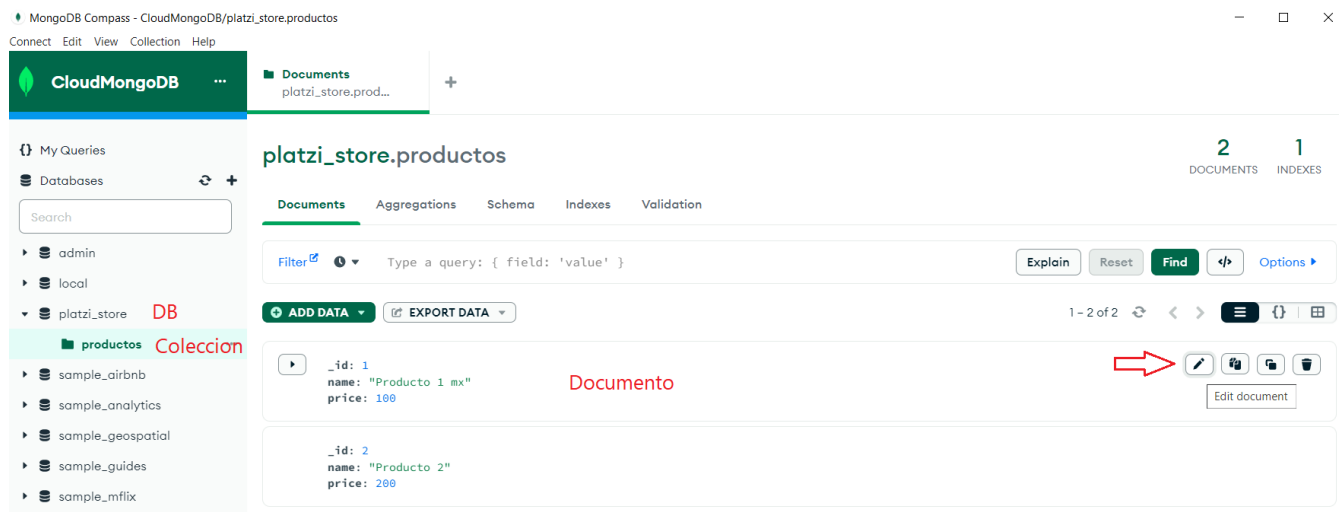
db.productos.find()
```

Resultado →

```
[
  {
    "_id": 1,
    "name": "Producto 1",
    "price": 100
  },
  {
    "_id": 2,
    "name": "Producto 2",
    "price": 200
  }
]
```

Actualizando un documento

Se puede actualizar un documento gráficamente desde MongoDB Compass de la siguiente manera.



Y con el código de la siguiente manera:

(imagenes abajo

```

use("platzi_store")

db.productos.updateOne(
  //query
  {_id: 2},
  //change => operators
  {
    $set: {
      name: "name changed!!"
    }
  })

db.productos.find()

```

resultado →

```

[
  {
    "_id": 1,
    "name": "Producto 1",
    "price": 100
  },
  {
    "_id": 2,
    "name": "name changed!!",
    "price": 200
  },
  {
    "_id": 3,
    "name": "Producto 3",
    "price": 300
  },
  {
    "_id": 4,
    "name": "Producto 4",
    "price": 3001
  }
]

```

Podemos actualizar diferentes campos a la vez.
E incluso añadir nuevos campos con el comando **updateOne()** usando el operador **\$set**.

En el siguiente ejemplo, el campo **"tags"** no existe y por lo tanto se agrega al documento con el id → 2 (imágenes abajo)

```

use("platzi_store")

db.productos.updateOne(
  //query
  {_id: 2},
  //change => operators
  {
    $set: {
      name: "name changed!!",
      price: 5000,
      tags: ['A', 'B', 'C']
    }
  })

db.productos.find()

```

resultado -->

```

[
  {
    "_id": 1,
    "name": "Producto 1",
    "price": 100
  },
  {
    "_id": 2,
    "name": "name changed!!",
    "price": 5000, Field actualizado
    "tags": [
      "A",
      "B", Nuevo field
      "C"
    ]
  },
  {
    "_id": 3,
    "name": "Producto 3",
    "price": 300
  },
  {
    "_id": 4,
    "name": "Producto 4",
    "price": 3001
  }
]

```

Con el comando **updateOne()**, le especificamos el id del documento que queremos actualizar, y con el operador **\$inc** le especificamos el field (campo, usualmente numerico) al que queremos incrementar y su cantidad para el incremento.

```
use("platzi_store")

db.productos.updateOne(
  //query
  {_id: 2},
  //change => operators
  {
    $inc: {
      //Incrementamos el precio en 100 unidades
      price: 150
    }
  }
)

db.productos.find()
```

resultado →

```
[
  {
    "_id": 1,
    "name": "Producto 1",
    "price": 100
  },
  {
    "_id": 2,
    "name": "name changed!!",
    "price": 5150,
    "tags": [
      "A",
      "B",
      "C"
    ]
  },
  {
    "_id": 3,
    "name": "Producto 3",
    "price": 300
  },
  {
    "_id": 4,
    "name": "Producto 4",
    "price": 3001
  }
]
```

ObjectId es una función para encontrar un documento con un objID. Estos son los ids que mongodb les asigna a los documentos cuando el usuario no le especifica uno al momento de su creación.

```
use("platzi_store")
//id "64e93184e4cf364935708e08"

db.productos.updateOne(
  //query
  {_id: ObjectId("64e93184e4cf364935708e08")},
  //change => operators
  {
    $inc: {
      //Incrementamos el precio en 100 unidades
      price: 150
    }
  }
)

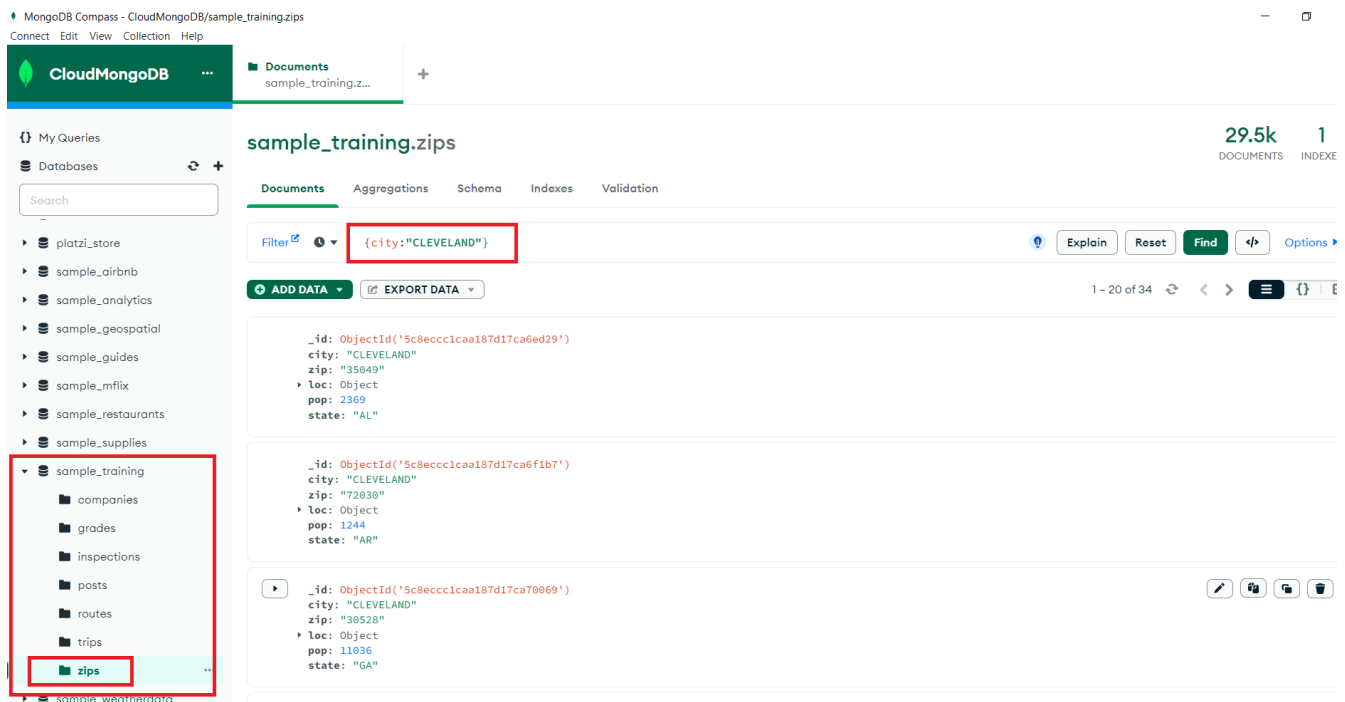
db.productos.find()
```

Operators

- **\$set**
Sets the value of a field in a document.
- **\$inc**
Increments the value of the field by the specified amount.
- **ObjectId**
Function to find a doc with objID

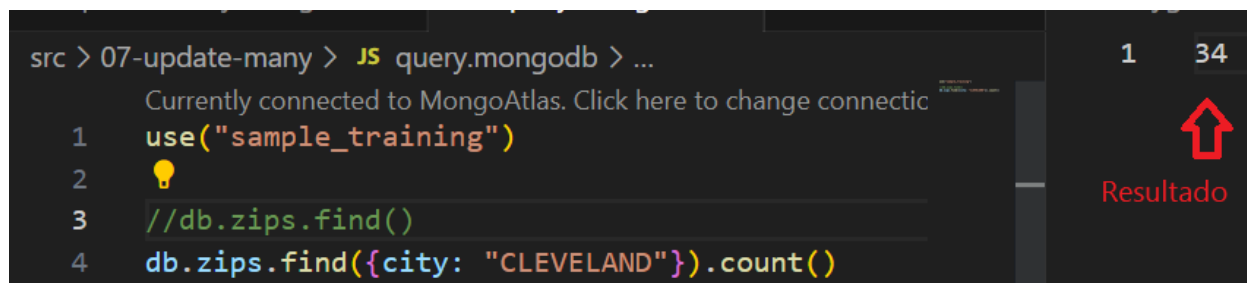
Actualizando varios documentos

Volvemos a MongoDB Compass y hacemos una consulta.



¿Cómo podemos actualizar todos los documentos que cumplan con la condición de que la ciudad sea "CLEVELAND"? Por ejemplo aumentarle en 100 o 1000 a la población (field pop).

Veamoslo en código:



Tenemos 34 registros que cumplen con la búsqueda de nuestro filtro. Y vamos a actualizar la población de cada documento de los 34 aumentándole en 1000.

(Código en la siguiente pagina)

```

use("sample_training")

db.zips.find({city: "CLEVELAND"})
db.zips.updateMany(
  //query - En este caso se esta usando como filtro.
  {city: "CLEVELAND"},
  // update - operators
  {
    //Con el operador $inc incrementamos la población en 1000 a todos los documentos que...
    //... cumplan con el filtro del query.
    $inc: {
      pop: 1000
    }
  }
)

```

Y volvemos a buscar para ver los resultados.

```

use("sample_training")

db.zips.find({city: "CLEVELAND"})

```

Recordemos que el operador **\$set** no sólo sirve para actualizar un valor sino también para agregar un nuevo valor (field-valor).

```

use("sample_training")

db.zips.find({city: "CLEVELAND"})
db.zips.updateMany(
  //query - En este caso se esta usando como filtro.
  {city: "CLEVELAND"},
  // update - operators
  {
    //Con el operador $set podemos actualizar/agregar un valor, en este caso que...
    //... cumplan con el filtro del query.
    $set: {
      saludo: "Hola!"
    }
  }
)

```

Y consultamos para ver el resultado; se agrega un nuevo campo a todos los documentos cuya ciudad sea "CLEVELAND".

```

use("sample_training")

db.zips.find({city: "CLEVELAND"})

```

Podemos renombrar los fields de los documentos de la siguiente manera con el operador **\$rename**.

```
use("sample_training")

db.zips.find({city: "CLEVELAND"})
db.zips.updateMany(
  //query - En este caso se esta usando como filtro.
  {city: "CLEVELAND"},
  // update - operators
  {
    //Con el operador $rename podemos cambiar el nombre de un field/campo.
    //En este caso cambiamos el nombre del field a los documentos que cumplan con...
    //... el filtro del query.
    $rename: {
      saludo: "saludo_mx"
    }
  }
)
```

Con **\$unset** podemos eliminar un field de un documento que ya no ocupemos.

```
use("sample_training")

db.zips.find({city: "CLEVELAND"})
db.zips.updateMany(
  //query - En este caso se esta usando como filtro.
  {city: "CLEVELAND"},
  // update - operators
  {
    //Con el operador $unset podemos eliminar un field/campo que ya no se ocupe.
    //En este caso se elimina el field de los documentos que cumplan con...
    //... el filtro del query.
    $unset: {
      saludo_mx: ""
    }
  }
)
```

Y checamos el resultado con el filtro de la ciudad "CLEVELAND":

```
use("sample_training")

db.zips.find({city: "CLEVELAND"})
```

Array Update Operators

Considerando la siguiente colección en mongoDB. Podemos hacer uso de los siguientes operadores.

```
use("platzi_store")

db.inventory.drop()

db.inventory.insertMany([
  { _id: 1, item: { name: "item ab", code: "123", description : "Single line description." }, qty: 15, tags: [ "school", "book", "bag", "headphone", "appliance" ] },
  { _id: 2, item: { name: "item cd", code: "123", description : "First line\nSecond line" }, qty: 20, tags: [ "appliance", "school", "book" ] },
  { _id: 3, item: { name: "item ij", code: "456", description : "Many spaces before line" }, qty: 25, tags: [ "school", "book" ] },
  { _id: 4, item: { name: "item xy", code: "456", description : "Multiple\nline description" }, qty: 30, tags: [ "electronics", "school" ] },
  { _id: 5, item: { name: "item mn", code: "000" }, qty: 20, tags: [ "appliance", "school" ] },
])

db.inventory.find()
```

Podemos actualizar el documento con `_id:4` en el field **tags** tomando en cuenta que su valor es un array.

Si queremos agregarle un valor a dicho array podemos hacerlo de la siguiente manera con el operador **\$push** (sólo funciona para arrays):

```
use("platzi_store")

db.inventory.updateOne({_id: 4}, {
  $push: {
    tags: "headphone"
  }
})
```

Con el operador **\$pull** podemos quitar elementos de un array. Recordemos que **\$pull** funciona solamente con los arrays. Si queremos quitar un field podemos usar **\$unset**. Podemos hacer un update masivo con `updateMany()`. Y en el caso del query (filtro) dejamos llaves vacías indicando que este update es para todos los documentos.

```
use("platzi_store")
//Para quitar un elemento de un array de un solo documento
/*db.inventory.updateOne({_id: 4}, {
  $pull: {
    tags: "headphone"
  }
})*//
//Pero tambien podemos hacer un update masivo con updateMany()

db.inventory.updateMany({}, {
  $pull: {
    tags: "book"
  }
})
```

En el ejemplo anterior, solo removimos un elemento de un array de un field para todos los documentos. Pero ¿Cómo podemos remover varios elementos de un array con mongoDB?

```
//Aquí removemos más un elemento de un array para todos los documentos

db.inventory.updateMany({}, {
  $pull: {
    tags: {
      $in: ["school", "appliance"]
    }
  }
})
```

Update or Insert

Para esta clase vamos a considerar la siguiente colección.

```
use("platzi_store")

db.iot.drop()

db.iot.insertMany([
  { _id: 1, sensor: "A001", date: "2022-01-01", readings: [1,2,3,4] },
  { _id: 2, sensor: "A001", date: "2022-01-02", readings: [1,2,3,4] },
  { _id: 3, sensor: "A002", date: "2022-01-01", readings: [1,2,3,4] },
  { _id: 4, sensor: "A002", date: "2022-01-02", readings: [1,2,3,4] },
])

db.iot.find()
```

Insertamos un registro individual.

```
use("platzi_store")

db.iot.insertOne({sensor: "A001", date: "2022-01-03", readings: [1212]}))
```

Y lo actualizamos insertando otro valor en el array.

```
db.iot.updateOne({sensor: "A001", date: "2022-01-03",},
{$push: {readings: 2323}})
```

Con el operador **\$pop** podemos eliminar un elemento del array ya sea al final o al inicio. Si le pasamos el valor -1 es al inicio, si le pasamos el valor 1 es al final.


```
//Podemos eliminar un elemento del array al inicio o al final
// Con el valor 1 es al final o con el valor -1 es al inicio
db.iot.updateOne({sensor: "A001", date: "2022-01-03"},
{$pop: {readings: 1}})
```

upsert te identifica si se debe insertar o actualizar el documento (verifica si existe o no).

```
db.iot.updateOne(
{sensor: "A001", date: "2022-01-03"},
{$push: {readings: 2323}},
{upsert: true})
```

En este caso no existe el registro que cumpla con el filtro. Así que lo crea (insert).

Si lo hacemos por segunda vez con otro valor, **upsert** detecta que ya existe el registro que cumple con el filtro (sensor A001 y la date especificada), y por lo tanto lo actualiza (update).

Eliminando documentos

Consideremos la siguiente colección en mongoDB.

```
use("platzi_store")

db.products.drop()

db.products.insertMany([
  { _id: 1, name: "Producto 1", price: 100 },
  { _id: 2, name: "Producto 2", price: 100 },
  { _id: 3, name: "Producto 3", price: 100 },
  { _id: 4, name: "Producto 4", price: 400 },
  { _id: 5, name: "Producto 5", price: 500 },
  { _id: 6, name: "Producto 6", price: 600 },
])

db.products.find()
```

Podemos eliminar un registro/documento de la siguiente manera:

```
//Eliminar un documento especificando su ID.
//Si el ID es generado con mongoDB el filtro sería {_id: ObjectId("123456677")}
db.products.deleteOne({_id: 1})
```

Podemos eliminar varios documentos con **deleteMany()** a través de un filtro. Supongamos que queremos eliminar todos los registros que tengo el precio de 100.

```
//Queremos eliminar todos los registros/documentos que tenga el precio de 100.
db.products.deleteMany({price: 100})
```

Y además...



MongoDB

<https://www.mongodb.com/docs>

db.collection.drop() — MongoDB Manual

Removes a collection or view from the **database**. The method also **removes** any indexes associated with the **dropped collection**. The method provides a wrapper ...



MongoDB

<https://www.mongodb.com/docs>

db.collection.remove()

Removes documents from a **collection**. The `db.collection.remove()` method can have one of two syntaxes. The `remove()` method can take a query **document** and an ...



Comparison Query Operators Usando \$eq(equal) y \$ne(not equal)

Ya hemos usado el operador **\$eq** implícitamente al poner un filtro (`{_id: 5}`) en la búsqueda.

```
db.inventory.find({qty: 20})
```

Si lo queremos ver el operador **\$eq** explícitamente, lo poder hacer de la siguiente manera.

```
db.inventory.find({qty: {$eq: 20}})
```

Para esta clase usaremos la siguiente colección:

```
use("platzi_store")

db.inventory.drop()

db.inventory.insertMany([
  { _id: 1, item: { name: "ab", code: "123" }, qty: 15, tags: ["A", "B", "C"] },
  { _id: 2, item: { name: "cd", code: "123" }, qty: 20, tags: ["B"] },
  { _id: 3, item: { name: "ij", code: "456" }, qty: 25, tags: ["A", "B"] },
  { _id: 4, item: { name: "xy", code: "456" }, qty: 30, tags: ["B", "A"] },
  { _id: 5, item: { name: "mn", code: "000" }, qty: 20, tags: ["A", "B", "C"] },
])

db.inventory.find()
```

Podemos buscar un field en un documento anidado de la siguiente manera:

```
//Buscando en documentos anidados
db.inventory.find({"item.name": "cd"})
```

Y también en este caso podemos usar el operador **\$eq** de manera explícita.

```
//Buscando en documentos anidados con el operador $eq
db.inventory.find({"item.name": {$eq: "cd"}})
```

En el caso del operador **\$ne** sólo se puede usar de manera explícita:

```
//El operador $ne(not equal) se usa de manera explícita
db.inventory.find({qty: {$ne: 20}})
```

Otro ejemplo con update:

```
//Se puede usar el operador $ne con update
//En este ejemplo queremos que todos los documentos cuyo qty no sea igual a 20
//se incremente en 10
db.inventory.updateMany(
  // query - operators
  {qty: {$ne: 20}},
  // update - operators
  {$inc: {
    qty: 10
  }})
```

Usando \$gt, \$gte, \$lt, \$lte (operadores de comparación)

En resumen, los operadores a ver en esta clase son:

- **\$gt** (greater than)
- **\$gte** (greater than or equal)
- **\$lt** (less than)
- **\$lte** (less than or equal)

Para esta clase trabajaremos sobre la siguiente colección de documentos:

```
use("platzi_store")

db.inventory.drop()

db.inventory.insertMany([
  { _id: 1, item: { name: "ab", code: "123" }, qty: 15, tags: ["A", "B", "C"] },
  { _id: 2, item: { name: "cd", code: "123" }, qty: 20, tags: ["B"] },
  { _id: 3, item: { name: "ij", code: "456" }, qty: 25, tags: ["A", "B"] },
  { _id: 4, item: { name: "xy", code: "456" }, qty: 30, tags: ["B", "A"] },
  { _id: 5, item: { name: "mn", code: "000" }, qty: 20, tags: ["A", "B", "C"] },
])

db.inventory.find()
```

Y tenemos ejemplos de búsqueda con dichos operadores de comparación.

```
//Búsquedas usando operadores de comparación individualmente
db.inventory.find({qty: {$gt: 20}})
db.inventory.find({qty: {$gte: 20}})
db.inventory.find({qty: {$lt: 20}})
db.inventory.find({qty: {$lte: 20}})
```

También podemos tener combinación de operadores y filtros

```
// Combinación de operadores
//Mayor o igual a 25 y menor o igual a 35
db.inventory.find({qty: {$gte: 25, $lte: 35}})

//Podemos hacer un "join", combinación de filtro con operadores de comparación
/* Aquí estamos poniendo dos condicionales: item.name (un subdocumento) con
el valor de "ab" y los operadores de comparación: $gte y $lte.
En este caso no hay ni un documento que cumpla con ambas condiciones*/
db.inventory.find({"item.name": "ab", qty: {$gte: 20, $lte: 25}})

/*Otro ejemplo usando los operadores $ne (not equal) $gte y $lte
"Buscar los documentos cuyo field code en el subdocumento no sea igual a 123 y
que la cantidad (qty) sea mayor o igual a 20 y menor o igual a 25"*/
db.inventory.find({"item.code": {$ne: "123"}, qty: {$gte: 20, $lte: 25}})
```

Más ejemplos:

Para esto usamos la siguiente colección de documentos.

```
use("platzi_store")

//data-set
db.iot.drop()

db.iot.insertMany([
  { _id: 1, sensor: "A001", date: "2022-01-01", readings: [1,2,3,4] },
  { _id: 2, sensor: "A001", date: "2022-01-02", readings: [1,2,3,4] },
  { _id: 3, sensor: "A002", date: "2022-01-01", readings: [1,2,3,4] },
  { _id: 4, sensor: "A002", date: "2022-01-02", readings: [1,2,3,4] },
])

db.iot.find()
```

Podemos combinar operadores de comparación con un update.
(imagen abajo)

```
//$pull es para quitar elementos de un array
db.iot.updateMany(
  //query
  {sensor: "A001"},
  //update
  {
    $pull: {
      readings: {$gte: 3}
    }
  }
)
```

Ejecutando consultas desde mongo compass:

The screenshot shows the MongoDB Compass interface. On the left, the 'Databases' sidebar lists various sample databases, with 'sample_training' expanded and the 'trips' collection selected. The main panel displays the 'sample_training.trips' collection. A filter is applied: `{tripduration: {$lte: 200}, usertype: "Subscriber"}`. Below the filter, two document snippets are visible, each with fields like `_id`, `tripduration`, `start station id`, `start station name`, `end station id`, `end station name`, `bikeid`, `usertype`, and `birth year`.

Usando \$regex

Para esta clase usaremos la siguiente colección de documentos:

(imagen siguiente pagina)

```

use("platzi_store")

db.inventory.drop()

db.inventory.insertMany([
  { _id: 1, item: { name: "ab", code: "123", description : "Single line description." }, qty: 15,
    tags: ["A", "B", "C"] },
  { _id: 2, item: { name: "cd", code: "123", description : "First line\nSecond line" }, qty: 20,
    tags: ["B"] },
  { _id: 3, item: { name: "ij", code: "456", description : "Many spaces before line" }, qty: 25,
    tags: ["A", "B"] },
  { _id: 4, item: { name: "xy", code: "456", description : "Multiple\nline description" }, qty: 30,
    tags: ["B", "A"] },
  { _id: 5, item: { name: "mn", code: "000" }, qty: 20, tags: ["A", "B", "C"] },
])

db.inventory.find()

```

Nota que el ultimo documento no tiene el field "description". En mongoDB existe esa flexibilidad de no tener los mismos fields en todos los documentos.

```

//Usando expresiones regulares con el operador $regex
db.inventory.find({"item.description": {$regex: /\n/}})

/*Buscamos todos los documentos que tenga la palabra line
en la descripción y que no sea case sensitive*/
db.inventory.find({"item.description": {$regex: /LINE/i}})

/*Buscamos todos los documentos que finalice con la palabra line
en la descripción y que no sea case sensitive*/
db.inventory.find({"item.description": {$regex: /LINE$/i}})

/*Buscamos todos los documentos que inicie con la palabra single
en la descripción y que no sea case sensitive*/
db.inventory.find({"item.description": {$regex: /^single/i}})

/*Buscamos todos los documentos que inicien con "s"
en la descripción tomando en cuenta los saltos de línea (m - de multiline)
y que no sea case sensitive (i)*/
db.inventory.find({"item.description": {$regex: /^s/im}})

```

Projection

En automático mongodb incluye todos los fields de los documentos que forman parte de la respuesta de la consulta. Con **project** yo puedo seleccionar los atributos/fields que van a

formar parte de la respuesta de la consulta. Es decir, limitar a sólo los campos que yo quiero que aparezcan.

sample_training.trips

sample_training.trips

1
DC

Documents

Aggregations

Schema

Indexes

Validation

Filter  {tripduration: {\$lte: 200}, usertype: "Subscriber"}



Explain

Reset

Find

Project {"start station name": 1}

Sort { field: -1 } or [['field', -1]]

MaxTimeMS 60000

Collation { locale: 'simple' }

Skip 0

Limit 0

EXPORT DATA

1 - 20 of 798

_id: ObjectId('572bb8222b288919b68abf93')
start station name: "Washington Pl & 6 Ave"

_id: ObjectId('572bb8222b288919b68abf98')
start station name: "South End Ave & Liberty St"

_id: ObjectId('572bb8222b288919b68abfcc')
start station name: "E 12 St & 3 Ave"

_id: ObjectId('572bb8222b288919b68abff1')
start station name: "N 6 St & Bedford Ave"

Podemos omitir que traiga el `_id` usando **project** de la siguiente manera:

Project {"start station name": 1, `_id`: 0}

Podemos especificar que traiga más campos de la siguiente manera: 0 significa que el campo no se muestra mientras que 1 significa que sí se muestra.

Filter  {tripduration: {\$lte: 200}, usertype: "Subscriber"}

Project {"start station name": 1, `_id`: 0, tripduration: 1, usertype: 1}

Sin embargo, con excepción del `_id`, no es necesario excluir los demás campos con el valor "0" explícitamente. Sólo basta agregar los campos que se necesitan para la consulta. En código sería de la siguiente manera:

```
use("sample_training")  
  
query projection  
db.trips.find({tripduration: {$gte: 500}}, {tripduration: 1, usertype: 1})
```

Operadores para Arrays

Para esta clase, vamos a trabajar con dos colecciones:

```
use("platzi_store")

db.inventory.drop()
db.inventory.insertMany([
  { _id: 1, item: { name: "ab", code: "123", description : "Single line description." }, qty: 15,
    tags: [ "school", "book", "bag", "headphone", "appliance" ] },
  { _id: 2, item: { name: "cd", code: "123", description : "First line\nSecond line" }, qty: 20,
    tags: [ "appliance", "school", "book" ] },
  { _id: 3, item: { name: "ij", code: "456", description : "Many spaces before      line" }, qty: 25,
    tags: [ "school", "book" ] },
  { _id: 4, item: { name: "xy", code: "456", description : "Multiple\nline description" }, qty: 30,
    tags: [ "electronics", "school" ] },
  { _id: 5, item: { name: "mn", code: "000" }, qty: 20, tags: [ "appliance", "school" ] },
])
```

y...

```
db.survey.drop();
db.survey.insertMany([
  {
    _id: 1,
    results: [
      { product: "abc", score: 10 },
      { product: "xyz", score: 5 },
    ],
  },
  {
    _id: 2,
    results: [
      { product: "abc", score: 8 },
      { product: "xyz", score: 7 },
    ],
  },
  {
    _id: 3,
    results: [
      { product: "abc", score: 7 },
      { product: "xyz", score: 8 },
    ],
  },
  {
    _id: 4,
    results: [
      { product: "abc", score: 7 },
      { product: "def", score: 8 },
    ],
  },
])
```

El operador **\$in** significa que dentro de un array pueda obtener esos elementos que se especifican en la búsqueda. Con el operador **\$in** se puede trabajar tanto con arrays como con objetos y con valores simples.

\$nin es lo contrario a **\$in**, es decir que no haya esos elementos.


```

//El operador $in significa que dentro de un array pueda obtener esos elementos
//que se especifican en la búsqueda.
//Con el operador $in se puede trabajar tanto con arrays como con objetos.
//En el siguiente query $in se usa como se fuera un or.
//Por lo que devuelve los documentos que tengan qty de 20 o 25.
db.inventory.find({qty: {$in: [20, 25]}})

//En este ejemplo buscamos valores dentro de un array
db.inventory.find({tags: {$in: ["school", "book", "electronics"]}})

//$nin es lo contrario a $in, es decir que no haya esos elementos.
//En este query se busca los documentos cuyo field qty no tienen los valores 20 y 25.
db.inventory.find({qty: {$nin: [20, 25]}})

//Aquí buscamos los documentos en cuyo field tags el cual es un array...
//... No tengan los valores "book" y "electronics"
db.inventory.find({tags: {$nin: ["book", "electronics"]}})

```

Ahora trabajaremos con operadores que sirven solo para arrays.

```

//Esto funciona a pesar de que "book" este dentro de un array
db.inventory.find({tags: "book"})

//En este caso funciona si y solo si los valores coinciden exactamente en el mismo orden.
db.inventory.find({tags: ["school", "book"]})

//Ahora trabajaremos con el operador $all.
//A diferencia del operador $in que se maneja como un or, $all es inclusivo, equivalente a un and.
//Aquí en este ejemplo se busca los documentos que tengan ambos valores,
//sin importar el orden en que los tenga.
db.inventory.find({tags: {$all: ["book", "school"]}})

//$size - Con el operador $size podemos buscar que el tamaño del array sea de dos elementos
db.inventory.find({tags: {$size: 2}})

//$elemMatch - Nos sirve cuando tenemos un array de objetos.
//Aquí buscamos los documentos que dentro del array results tengan un
//objeto con el valor "xyz"
db.survey.find({results: {$elemMatch: {product: "xyz"}}})

//Le añadimos otro filtro indicando que el score sea mayor o igual a 7
db.survey.find({results: {$elemMatch: {product: "xyz", score: {$gte: 7}}}})

```

Operadores lógicos

Trabajaremos con una colección de documentos con la siguiente estructura:

```
{
  "_id": {
    "$oid": "56d61033a378eccde8a8354f"
  },
  "id": "10021-2015-ENFO",
  "certificate_number": 9278806,
  "business_name": "ATLIXCO DELI GROCERY INC.",
  "date": "Feb 20 2015",
  "result": "No Violation Issued",
  "sector": "Cigarette Retail Dealer - 127",
  "address": {
    "city": "RIDGEWOOD",
    "zip": 11385,
    "street": "MENAHAN ST",
    "number": 1712
  }
},
```

El operador **\$and** ya lo hemos usado de forma implícita.

```
//En este ejemplo la búsqueda ya es un AND (implícito)
//ya que son dos filtros(condiciones) a cumplir
db.inspections.find({sector: {$regex: /tax/i}, result: {$regex: /unable/i}})
db.inspections.find({sector: {$regex: /tax/i}, result: {$regex: /unable/i}}).count()

//AND de forma explícita
db.inspections.find({
  $and: [
    {sector: {$regex: /tax/i}},
    {result: {$regex: /unable/i}}
  ]
}).count()
```

El operador **\$or**

```
//OR - Con que una condición se cumpla basta.
db.inspections.find({
  $or: [
    {sector: {$regex: /tax/i}},
    {result: {$regex: /unable/i}}
  ]
})
```

operador **\$nor** se refiere a todos los documentos que no incluyan los filtros especificados.

```
//operador $nor - es todos los documentos que no incluyan los filtros especificados.
db.inspections.find({
  $nor: [
    {sector: {$regex: /tax/i}},
    {result: {$regex: /^unable/i}}
  ]
},
//proyección
{result: 1, _id: 0})
```

\$not es un operador de negación.

```
//$not es un operador de negación
db.inspections.find({
  result: {$not: {$regex: /unable to/i}}
})
```

Podemos combinar operadores lógicos. Trabajaremos sobre la siguiente colección de documentos con la siguiente estructura:

```
{
  "_id": {
    "$oid": "56e9b39b732b6122f877fa31"
  },
  "airline": {
    "id": 410,
    "name": "Aerocondor",
    "alias": "2B",
    "iata": "ARD"
  },
  "src_airport": "CEK",
  "dst_airport": "KZN",
  "codeshare": "",
  "stops": 0,
  "airplane": "CR2"
},
```

Los ejercicios en la imagen de la siguiente pagina.

```

use("sample_training")

db.routes.find({airplane: "E70"})

db.routes.find({
  $or: [
    {dst_airport: "BOG"},
    {src_airport: "BOG"}
  ]
}).count()

//Ahora haremos una union entre los dos primeros queries.
//and recibe su primer filtro, y el segundo filtro es un or.
db.routes.find({
  $and: [{airplane: "E70"}, {$or: [{dst_airport: "BOG"}, {src_airport: "BOG"}]}]
}).count()

```

Expressive operator

Trabajaremos con la siguiente estructura de documento.

```

use("platzi_store")

db.monthlyBudget.drop()

db.monthlyBudget.insertMany([
  { "_id" : 1, "category" : "food", "budget": 400, "spent": 450 },
  { "_id" : 2, "category" : "drinks", "budget": 100, "spent": 150 },
  { "_id" : 3, "category" : "clothes", "budget": 100, "spent": 50 },
  { "_id" : 4, "category" : "misc", "budget": 500, "spent": 300 },
  { "_id" : 5, "category" : "travel", "budget": 200, "spent": 650 }
])

db.monthlyBudget.find()

db.monthlyBudget.find({spent: {$gte: 100}})

//Podemos hacer la consulta anterior usando el expressive operator
db.monthlyBudget.find({
  $expr: {
    /*el signo de pesos ($) en spent, se usa para seleccionar un campo
    en específico */
    $gte: ["$spent", 100]
  }
})

```

¿Si aparentemente **\$expr** hace lo mismo que el primer query, entonces por qué hacerlo más complicado? ¿Realmente cuál es su uso?

```
//Con $expr, podemos comparar dos atributos/campos con el operador $gte
//¿Cuales son los documentos en donde spent supera a budget?
db.monthlyBudget.find({
  $expr: {
    $gt: ["$spent", "$budget"]
  }
})
```

Más ejemplos con otra estructura de documento con el operador **\$expr**

```
{
  "_id": {
    "$oid": "572bb8222b288919b68abf5a"
  },
  "tripduration": 379,
  "start station id": 476,
  "start station name": "E 31 St & 3 Ave",
  "end station id": 498,
  "end station name": "Broadway & W 32 St",
  "bikeid": 17827,
  "usertype": "Subscriber",
  "birth year": 1969,
  "start station location": {
    "type": "Point",
    "coordinates": [
      -73.97966069,
      40.74394314
    ]
  },
  "end station location": {
    "type": "Point",
    "coordinates": [
      -73.98808416,
      40.74854862
    ]
  },
  "start time": {
    "$date": "2016-01-01T00:00:45Z"
  },
  "stop time": {
    "$date": "2016-01-01T00:07:04Z"
  }
},
```

Podemos hacer una búsqueda y comparar dos campos (sus valores) con el operador **\$eq** para ver cuales coinciden de la siguiente manera:

```
db.trips.find({
  $expr: {
    $eq: ["$start station id", "$end station id"]
  }
}).count()
```

Reto, el mismo ejercicio anterior(cuantos han iniciado y finalizado en el mismo punto) pero además, donde su duración fue mayor a "x" tiempo.

```
db.trips.find({
  $expr: {
    $and: [
      {$eq: ["$start station id", "$end station id"]},
      {$gt: ["$tripduration", 1200]}
    ]
  }
}).count()
```

Podemos ver que este operador sirve para poder hacer referencia a los valores que tienen los campos de un documento, en una consulta.

Query in subdocs (consultas a Arrays y SubDoc)

Si hacemos un `findOne` en vacío nos trae el primer elemento de la colección.

`db.trips.findOne()` y obtenemos el siguiente documento (la misma estructura que la imagen anterior).

Recordemos que podemos hacer consulta a subdocumentos de la siguiente manera:

```
//Consulta a subdocumentos
db.trips.findOne({"start station location.type": "Point"})
```

También podemos ir a una posición en específico de un array dentro de un documento en una consulta, de la siguiente manera:

```
//Podemos ir directamente a la posición de un array y hacer una consulta
db.companies.find({
  "relationships.0.person.last_name": "Zuckerberg"
},
//projection
{name: 1, relationships: 1})
```

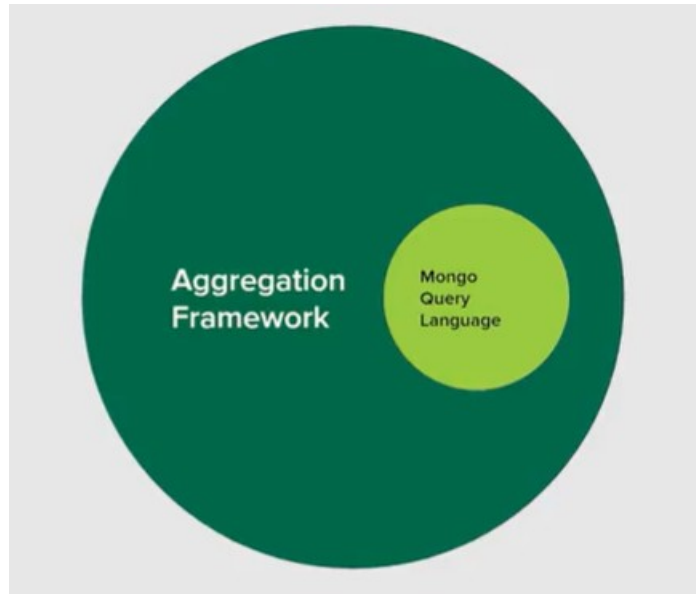
```
//Ahora preguntamos cuantos jefes de compañías se llaman Mark
db.companies.find({
  "relationships.0.person.first_name": {$regex: /mark/i}
},
//projection
{name: 1, relationships: 1}).count()
```

Si queremos consultar todas las posiciones del array podemos hacer lo siguiente:

```
//Ahora no solamente quiero buscar la posición 0 del array, sino todas las posiciones.
db.companies.find({
  relationships: {
    $elemMatch: {
      "person.first_name": "Mark"
    }
  }
},
//projection
{name: 1, relationships: 1}).count()
```

Aggregation Framework

Esta herramienta está diseñada para tener análisis de datos más profundos. Esta herramienta es más para procesar información y en comparación de Mongo Query Language (lo que hemos estado viendo) es mucho más grande. Esto se usa cuando estamos hablando de grandes volúmenes de datos.



El aggregation Framework funciona por capas, una alimenta a la anterior y he aquí su punto fuerte: La salida de un pipeline/capa es la entrada de otro.

```
db.listingsAndReviews.find({
  amenities: "Wifi"
}),
{price: 1, amenities: 1})

//Hacemos la misma consulta anterior pero ahora usando aggregation framework.
//[[]],[[]],[[]] funciona como pipelines y cada pipeline va a ser un proceso que
//va a alimentar al anterior. Es decir... la salida de un pipeline es la entrada de otro
db.listingsAndReviews.aggregate([
  {$match: {amenities: "Wifi"}}, //find
  {$project: {address: 1}}, //project
  //Podemos empezar a hacer agrupaciones
  {$group: {_id: "$address.country", count: {$sum: 1}}}
])
```

Sort, limit y skip

Obtenemos nuestra estructura de documento a usar para esta clase.

db.zips.find()

```
{
  "_id": {
    "$oid": "5c8eccc1caa187d17ca6ed16"
  },
  "city": "ALPINE",
  "zip": "35014",
  "loc": {
    "y": 33.331165,
    "x": 86.208934
  },
  "pop": 3062,
  "state": "AL"
},
```

La estructura de los documentos de la colección a la derecha, y el ejercicio con **sort()** y **limit()** en la imagen de abajo.

```
//Para hacer un sort sería de la siguiente manera.
// 1 es igual a ordenar de forma ascendente.
// -1 es igual a ordenar de forma descendente.

db.zips
  .find({pop: {$gte: 100}})
  .sort({pop: 1}) //Ascendente
  .limit(3) //limitar a tres documentos la consulta
```

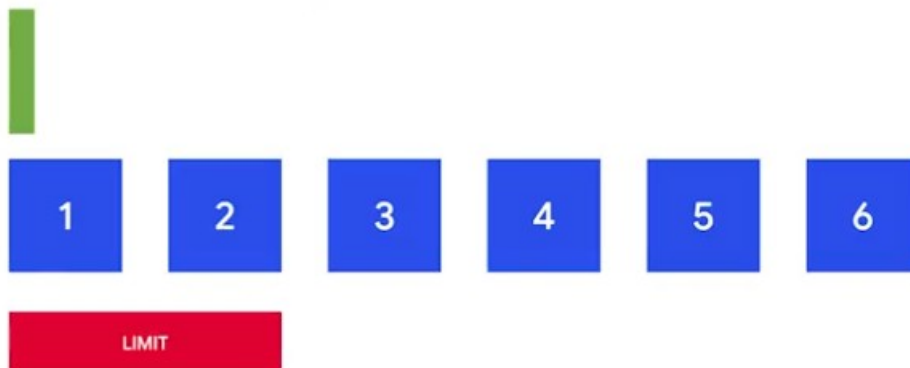
La función **skip()** puede servir para hacer técnicas de paginación.

La técnica de paginación va acompañada por un **limit()** y un **skip / offset = 0**

En este ejemplo tenemos un total de 6 documentos, en cada paginación se muestran dos, el skip viene siendo como el “salto”, si lo ponemos a dos en la siguiente paginación apuntaría al documento #3 y el limit determinaría el número de documentos a mostrar. En este caso mostraría el documento 3 y 4.

Aquí el skip esta en 0 y por lo tanto apunta al primer documento.

limit = 2
skip / offset = 0



En la imagen de la siguiente pagina vemos que el skip se pone en dos y por lo tanto apunta al documento 3, mostrando dos documentos por tener el limit en 2.

limit = 2
skip / offset = 2



Para el siguiente ejercicio de paginación tenemos la siguiente colección de documentos.

```
use("platzi_store")

db.categories.drop()

db.categories.insertMany([
  { _id: 1, name: "category 1" },
  { _id: 2, name: "category 2" },
  { _id: 3, name: "category 3" },
  { _id: 4, name: "category 4" },
  { _id: 5, name: "category 5" },
  { _id: 6, name: "category 6" },
  { _id: 7, name: "category 7" },
  { _id: 8, name: "category 8" },
  { _id: 9, name: "category 9" },
  { _id: 10, name: "category 10" },
])

db.categories.find()
```

Ejemplo 1 de paginación

```
db.categories
  .find()
  .skip(0)
  .limit(2)
```

```
[
  {
    "_id": 1,
    "name": "category 1"
  },
  {
    "_id": 2,
    "name": "category 2"
  }
]
```

Ejemplo 2 de paginación

```
db.categories
  .find()
  .skip(2)
  .limit(2)
```

```
[
  {
    "_id": 3,
    "name": "category 3"
  },
  {
    "_id": 4,
    "name": "category 4"
  }
]
```

Ejemplo 3 de paginación

```
db.categories
  .find()
  .skip(4)
  .limit(2)
```

```
[
  {
    "_id": 5,
    "name": "category 5"
  },
  {
    "_id": 6,
    "name": "category 6"
  }
]
```