Chapter 2 : The MongoDB Query Language + Atlas

CRUD

- Create
- Read
- Update
- Delete

Installing the mongo Shell

Compass no cobre toda las opciones que da MongoDB

windows

- download center
- instalar MongoDB Server versión enterprise (soporte SSL?) para acceder a Atlas Cluster
- `c:\Program Files\MongoDB\Server\3.4`
- `mongo –nodb → quit()`

osx/linux

- descargar
- modificar path

conexión

```
mongo "mongodb://cluster0-shard-00-00-jxeqq.mongodb.net:27017,cluster0-shard-00-01-jxeqq.mongodb.net:27017,cluster0-shard-00-02-jxeqq.mongodb.net:27017/test?replicaSet=Cluster0-shard-0" --authenticationDatabase admin --ssl --username m001-student --password m001-mongodb-basics
```

- cluster, le indicamos todos los servers
- test es la DB a la que le indicamos que queremos conectar, la iremos cambiando, por ejemplo: 100YWeatherSmall
- `use video`
- `show collections`
- `db.movies.find().pretty()`

Lab2

- crear cuenta en Atlas
- creación cluster en AWS (free tier)
- settings
    - project name: *M001*

- Security
  - IP Whitelist → Allow access from anywhere
  - MongoDB Users → new user
    - user: **m001-student**
    - pass: **m001-mongodb-basics**

Connecting to your sandbox cluster from mongo shell

- desde Atlas GUI, Overview → Connect
  - usar mongodb-shell connection

v3.4+

```
mongo "mongodb://sandbox-shard-00-00-cab0l.mongodb.net:27017,sandbox-shard-00-01-cab0l.mongodb.net:27017,sandbox-shard-00-02-cab0l.mongodb.net:27017/test?replicaSet=Sandbox-shard-0" --ssl --authenticationDatabase admin --username m001-student --password m001-mongodb-basics
```

Loading data into your sandbox cluster

- `show dbs`
- descargar datos de ejemplo y cargar en mi cluster
- `load(«loadMovieDetailsDataset.js»)`

Connecting to your sandbox cluster from Compass

- Desde Atlas, y en modo simplificado, identificar al servidor primario de nuestro cluster
- crear una nueva conexión, guardar como favorito

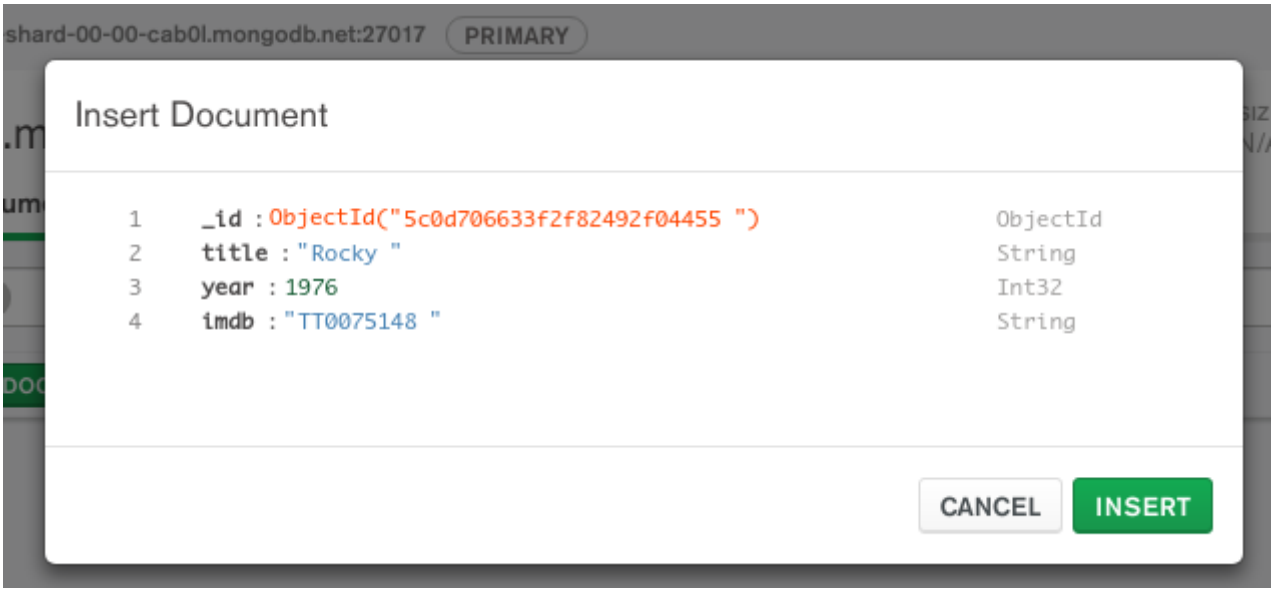
Creating Documents

- create documents = inserts

insertOne()

desde Compass:

- crear nueva colección
- insertar nuevo documento

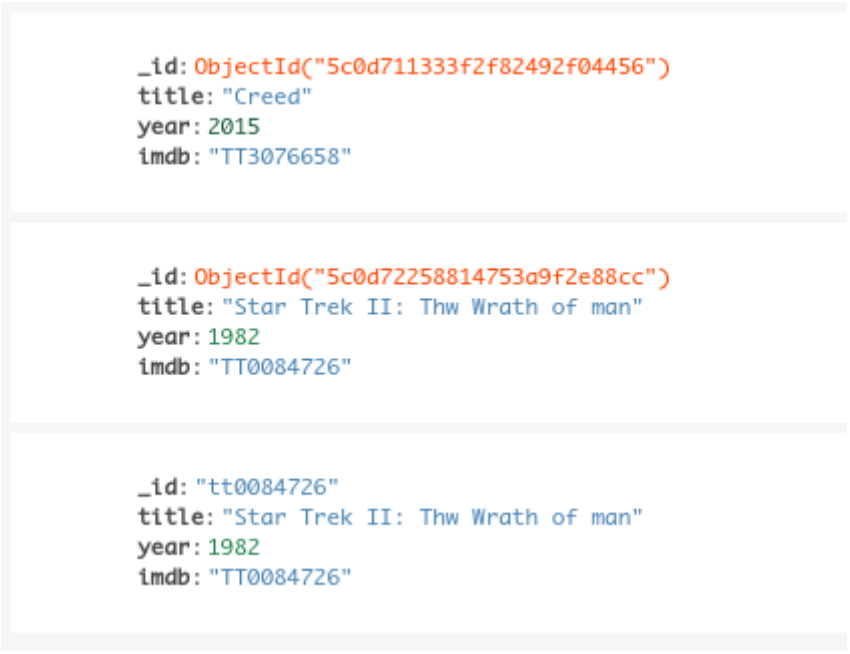


desde mongodb-shell:

- db → database en uso
- `db.moviesScratch.insertOne({title:"Star Trek II: Thw Wrath of man", year: 1982, imdb: "TT0084726"})`
  - creará la colección si esta no existiese

```
MongoDB Enterprise Sandbox-shard-0:PRIMARY> use video
switched to db video
MongoDB Enterprise Sandbox-shard-0:PRIMARY> show collections
movieDetails
moviesScratch
MongoDB Enterprise Sandbox-shard-0:PRIMARY> db
video
MongoDB Enterprise Sandbox-shard-0:PRIMARY> db.moviesScratch.insertOne({title:"Star Trek II: Thw Wrath
of man", year: 1982, imdb: "TT0084726"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5c0d72258814753a9f2e88cc")
}
MongoDB Enterprise Sandbox-shard-0:PRIMARY>
```

- todo documento de una colección ha de tener un identificador único: **_id**
- si nosotros no se lo creamos, se creará por nosotros
- `db.moviesScratch.insertOne({_id: "tt0084726", title:"Star Trek II: Thw Wrath of man", year: 1982, imdb: "TT0084726"})`
- no conviene mezclar en una colección diferentes tipos de **_id**:



insertMany()

- en lugar de pasar un objeto **{...}** le pasaremos un array **[{...},{...},...,{...}]**

```
db.moviesScratch.insertMany(
    [
        {
        "_id" : "tt0084726",
        "title" : "Star Trek II: The Wrath of Khan",
        "year" : 1982,
        "type" : "movie"
          },
          {
        "_id" : "tt0796366",
        "title" : "Star Trek",
        "year" : 2009,
        "type" : "movie"
          },
          {
        "_id" : "tt0084726",
        "title" : "Star Trek II: The Wrath of Khan",
        "year" : 1982,
        "type" : "movie"
          },
          {
        "_id" : "tt1408101",
        "title" : "Star Trek Into Darkness",
        "year" : 2013,
        "type" : "movie"
          },
          {
        "_id" : "tt0117731",
        "title" : "Star Trek: First Contact",
        "year" : 1996,
        "type" : "movie"
```

```
        }
    ]
);
```

- por defecto, **insertMany()** trabaja de manera «ordenada», podemos especificarle que no es así añadiendo un segundo parámetro al comando: **{[…],{«ordered»: false}}**
- si intentamos introducir 2 documentos con el mismo **_id** dará un error y dejará de insertar documentos.
- si le hemos indicado el **ordered: false**, no insertará aquellos que den errores, pero continuará insertando el resto

Lab 2.2

Reading documents: scalar fields

- AND: `{«campo1»: restricción1, «campo2»: restricción2}`
- OBJECTS: `{«objeto.campo»: valor1, «objeto.subobjeto.campo»: valor2}`

ejemplo/quiz

buscar en **movieDetails** cuantas películas han ganado 2 galardones y han tenido 2 nominaciones:

- mongodb shell: `db.movieDetails.find({«awards.wins»: 2, «awards.nominations»: 2}).count()`
- compass: `{awards.wins: 2, awards.nominations: 2}`

2.7 Array Fields

- `db.movies.find({cast: [«Jeff Bridges», «Tim Robbins»]}).pretty()` : muestra solo los documentos que cumplen exactamente el critero de tener 2 elementos en el array *cast* con esos actores
- `db.movies.find({cast: «Jeff Bridges»}).pretty()` : muestra los documentos donde aparece ese actor, aunque no exclusimante y sin tener importancia el orden
- `db.movies.find({«cast.0»: «Jeff Bridges»}).pretty()` : busca el actor en la primera posición del array *cast*

Cursors

- apunta al documento en curso
- en mongoshell, itera 20 resultados... si queremos más, podemos ejecutar `it`

Projections

- límita el número de campos que devuelve
- se usa (¿exclusivo?) como segundo parámetro en `find()`
  - `db.movies.find({genre: «Action,Adventure»},{title: 1})` : muestra solo el título
  - el **_id** se devuelve por defecto en todas las proyecciones, aunque se puede excluir - como otros campos- si se especifica (con un **0**):
  - `db.movies.find({genre: «Action,Adventure»},{title: 1, _id:0})` : idem anterior,

pero sin el campo **_id**
- se pueden especificar campos (con el **0**) que no se quieren mostrar

UpdateOne()

- primero se aplica un filtro al que aplicar el update
- el primero que cumpla el criterio, será el actualizado
- el segundo argumento especifica que queremos actualizar, a través de **$set**

```
db.movieDetails.updateOne({
    title: "The Martian"
  }, {
    $set: {
      poster: "http://..."
    }
  })
```

la respuesta será:`{«acknowledged»: true, «matchedCount»: 1, «modifiedCount»: 1 }`, donde:

- **acknowledged** indica que se ha ejecutado correctamente
- **matchedCount** indica cuantos registros cumplen el criterio
- **modifiedCount** si se ha modificado o no (en este caso 1 o 0, no puede haber más)

```
db.movieDetails.updateOne({
    title: "The Martian"
  }, {
    $set: {
      "awards": {
        "wins": 8,
        "nominations": 14
      }
    }
  })
```

Update Operators

For update operations, update operators specify how to modify specific fields in documents matching a filter. Fields may be added, deleted, or have their value changed in some way. Update operators define what modifications to make with respect to one or more fields in matching documents.

- $set : añade campos
- $unset : elimina campos
- $min
- $max
- $inc : incrementa el valor del campo/campos en el valor indicado
- $addToSet : añade elementos a un array (si no existen previamente)
- $pop : elimina el primer / último elemento de un array
- $pullAll : elimina los valores que coíncidan
- $pull : elimina el campo que coincida
- $push: añade un array de elementos

```
let review = [
```

```
  "aadasdsada",
  "dsdadasdsadasdsa",
  "asdasfeadfsdafasfdsfasdfsda"
].join()
db.movieDetails.updateOne({
  title: "the martian"
},{
  $push: {
    reviews: {
      rating: 4.5,
      reviewer: "Spencer H.",
      text: review
    }
  }
})
```

el documento ahora tendrá un campo llamado **reviews** del tipo array con un objeto que contendrá los campos indicados (rating,reviewer, text)

```
let review = [
  "aadasdsada",
  "dsdadasdsadasdsa",
  "asdasfeadfsdafasfdsfasdfsda"
].join()
db.movieDetails.updateOne({
  title: "the martian"
},{
  $push: {
    reviews:{
      $each: [{
        rating: 4.5,
        reviewer: "Spencer H.",
        text: review
      },{
        rating: 4.5,
        reviewer: "Spencer H.",
        text: review
      },{
        rating: 4.5,
        reviewer: "Spencer H.",
        text: review
      }]
    }
  }
})
```

añade tantos objetos en el array como hay

- ... otros en la página de MongoDB

UpdateMany()

- modifica todos los documentos que cumplen el registro

```
db.movieDetails.updateMany({
  rated: null
},{
  $unset: {
    rated: ""
  }
})
```

elimina el campo rated = null de todos los documentos

Upserts

- crear nuevos documentos

```
let detail={[...]}

db.movieDetails.updateOne({
  "imdb.id": detail.imdb.id
},{
  $set: detail
},{
  upsert: true
})
```

- **detail** contiene todos los campos del documento
- **$set** intenta actualizar el documento
- **upsert=true** inserta si no existe el documento previamente (que cumpla el criterio del primer parámetro)

2.14 replaceOne()

- para aplicaciones en las que es más fácil reemplazar todo el documento
- `detailDoc = db.movieDetails.findOne({«imdb.id»: «tt….»});`
- `detailDoc.poster;` : printea null, no está definido
- `detailDoc.poster = «http://.…»;` ; añade el campo
- `detailDoc.genres.push(«Documentary»);` este **push** es un método javascript, no el operador visto anteriormente, y añade el valor al array de **genres**
- con todos estos cambios sobre el documento, hacemos un replaceOne() para actualizarlo todo

```
db.movieDetails.replaceOne({"imdb.id": detailDoc.imdb.id}, detailDoc);
```

Update Operators