Creación de una base de datos

Bases de Datos NoSQL

Bartolomé Mestre Fons

Lunes, 4 de Noviembre de 2024

Bases de Datos SQL
2024-2025
UCM - Universidad Complutense de Madrid

Índice

1. Introducción 3
2. Ejercicio 0: Carga del dataset 3
3. Ejercicios 4
4. Ejercicios propuestos 20
5. Conclusiones 22

1. Introducción

El objetivo del presente trabajo es poner a prueba los conocimientos en el manejo de MongoDB a la hora de crear bases de datos NoSQL y manipularlas. Para ello, se realizarán una serie de ejercicios sobre una base de datos de documentos en la sección Ejercicios (previamente cargada en Ejercicio 0: Carga del Dataset). A continuación, se presentarán una serie de ejercicios planteados por el propio estudiante haciendo uso del comando *aggregate* en Ejercicios propuestos. Finalmente, en la sección Conclusiones se enunciarán las principales conclusiones a las que el estudiante ha llegado tras la realización del proyecto.

2. Ejercicio 0: Carga del dataset

Tal y como se ha adelantado en la Introducción, el dataset utilizado es la colección *movies* que aporta información sobre un conjunto de 28795 películas. Para ello, tras descargar el archivo des del enlace dispuesto por el profesor, arrastramos dicho archivo en formato *json* a la colección creada en MongoDB Compass dentro de una base de datos. Por supuesto, dicha base de datos debe conectarse a NoSQLBooster para que sobre el archivo se puedan ejecutar las *queries* deseadas. Desde MongoDB Compass, uno debería visualizar los datos siguientes tras cargar el dataset:

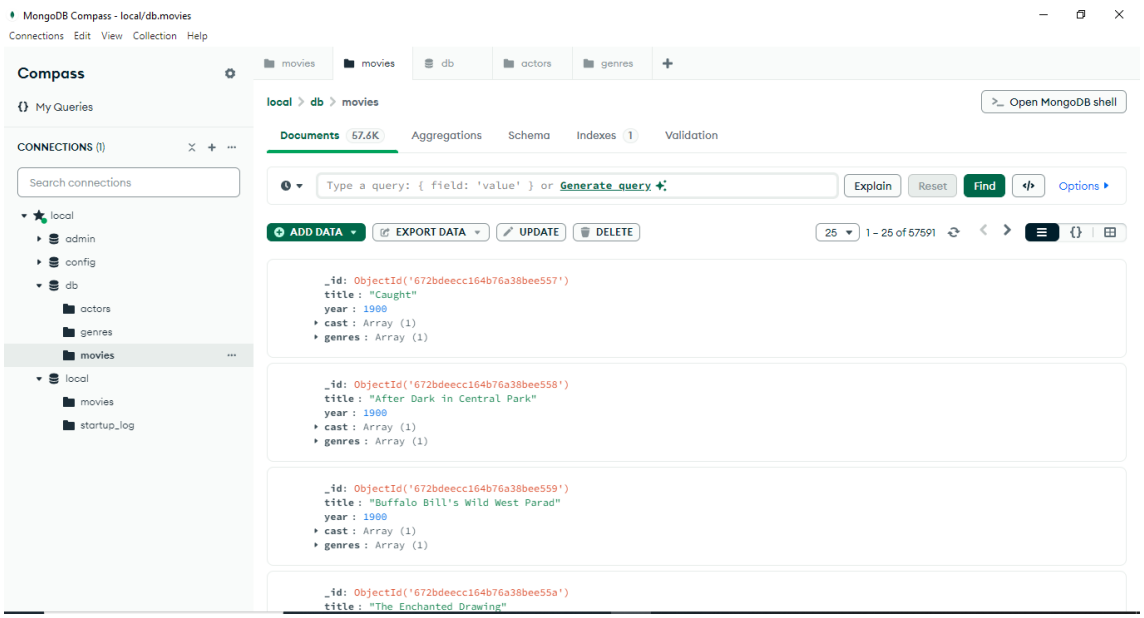


Figura 1: *Interfaz gráfica de MongoDB Compass. Se muestran los datos cargados dentro de la colección 'Movies'en la base de datos creada a propósito 'db'. Se visualizan además dos colecciones más, 'actors'y 'genres', colecciones que se crearán en ejercicios posteriores.*

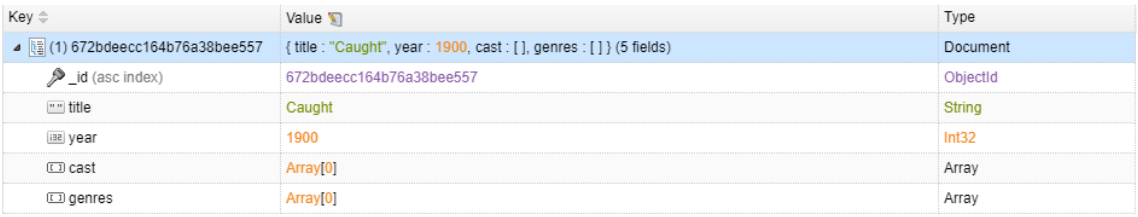
En el primer ejercicio se analizará la estructura de los documentos de la base de datos.

3. Ejercicios

1. Analizar con find la colección.

   ```
   db.movies.find({}).limit(1)
   ```

   Aquí, con find({}) se reliza una búsqueda general sobre la base de datos, y con limit(1) se limita la salida a 1 solo documento. Se obtiene lo siguiente:



Key	Value	Type
(1) 672bdeecc164b76a38bee557	{ title : "Caught", year : 1900, cast : [ ], genres : [ ] } (5 fields)	Document
_id (asc index)	672bdeecc164b76a38bee557	ObjectId
title	Caught	String
year	1900	Int32
cast	Array[0]	Array
genres	Array[0]	Array

Figura 2: *Estructura del primer documento del dataset.*

   A continuación se describe cada elemento que compone el documento:

   - **_id**:
     - **Descripción**: Identificador único del documento en MongoDB.
     - **Tipo**: ObjectId
   - **title**:
     - **Descripción**: Título del documento o de la película.
     - **Tipo**: String
   - **year**:
     - **Descripción**: Año de lanzamiento o de registro.
     - **Tipo**: Int32
   - **cast**:
     - **Descripción**: Lista del reparto o actores de la película.
     - **Tipo**: Array
   - **genres**:
     - **Descripción**: Géneros o categorías de la película.
     - **Tipo**: Array

2. Contar cuántos documentos (películas) tiene cargado.

   ```
   db.movies.find({}).count()
   ```

   Aquí, se hace uso de count( ) para contar cuántos documentos ha devuelto el comando find({}). El resultado es el siguiente:

```
1 28795
```

Figura 3: *Número de documentos en la base de datos.*

3. Insertar una película.

```
db.movies.insertOne(
  {
  '_id' : ObjectId('5f1d7e93c5b6a6bc9e8b4569' ),
  'title' : 'The Great Adventure' ,
  'year' : 1900,
  'cast' : ['John Doe' , 'Jane Smith' , 'Tom Brown' ],
  'genres' : ['Adventure' , 'Fantasy' ],
  }
  )
```

Con insertOne( ) uno puede insertar un documento. Si se quisiese insertar varios, se tendría que hacer uso de insertMany([ ]). Aquí cabe mencionar que la _id creada para esta película ha sido generada aleatoriamente, siempre que posea el mismo formato que las demás _id. Este es el resultado, que demuestra que se ha insertado el nuevo documento satisfactoriamente:

Key	Value	Type
(1)	{ acknowledged : true, insertedId : ObjectId("5f1d7e93c5b6a6bc9e8b4569") }	Object
acknowledged	true	Bool
insertedId	5f1d7e93c5b6a6bc9e8b4569	ObjectId

Figura 4: *Nuevo documento insertado en la base de datos.*

4. Borrar la película insertada en el punto anterior (en el 3).

```
db.movies.deleteOne({title:'The Great Adventure'})
```

Con deleteOne( ) se borra una película individual, que se localiza por su título 'The Great Adventure'. El resultado es el siguiente:

Key	Value	Type
(1)	{ acknowledged : true, deletedCount : 1 }	Object
acknowledged	true	Bool
deletedCount	1	Int32

Figura 5: *Eliminación del documento añadido en el ejercicio 3.*

5. Contar cuantas películas tienen actores (cast) que se llaman “and”.

   ```
   db.movies.find({cast:’and’}).count()
   ```

   Primero con find(cast: 'cast') se buscan aquellas películas donde un actor consta como 'and', y luego con count( ) se cuenta el número de aquellos documentos que cumplen dicha condición. El resultado es el siguiente:

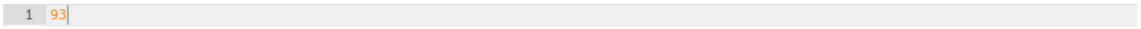


Figura 6: *Número de películas que entre sus actores consta 'and'.*

6. Actualizar los documentos cuyo actor (cast) tenga por error el valor “and” como si realmente fuera un actor. Para ello, se debe sacar únicamente ese valor del array cast. Por lo tanto, no se debe eliminar ni el documento (película) ni su array cast con el resto de actores.

   ```
   db.movies.updateMany({cast:’and’},{$pull:{cast:’and’}})
   ```

   Con updateMany( ) se actualizarán todos los documentos que cumplan una determinada condición (que cast sea 'and'para este caso), y a continuación, con $pull se eliminan aquellas entradas que cumplan dicha condición. El resultado es el siguiente:

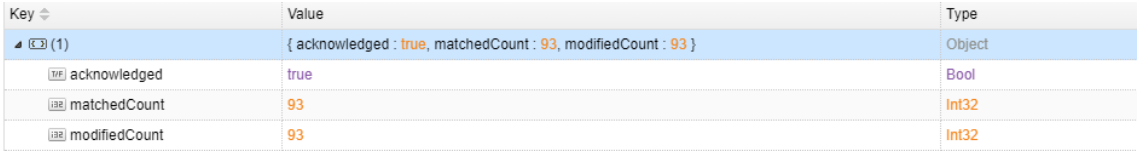


Figura 7: *Eliminación de aquellos actores que constan como 'and'.*

Efectivamente, se han eliminado las entradas 'and'en cast de aquellos 93 documentos que satisfacen dicha condición.

7. Contar cuantos documentos (películas) tienen el array 'cast'vacío.

   ```
   db.movies.find({cast: {$size: 0}}).count()
   ```

   El enunciado de este ejercicio se puede releer como contar aquellos documentos cuyo array 'cast'es de tamaño 0 (ninguna entrada). Así, con find({cast:{$size:0}}) se escogen aquellos documentos donde el array tiene 0 elementos, y con count( ) se cuenta el número de documentos que cumplen dicha condición. Se obtiene lo siguiente:

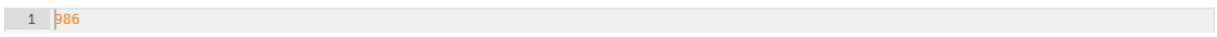
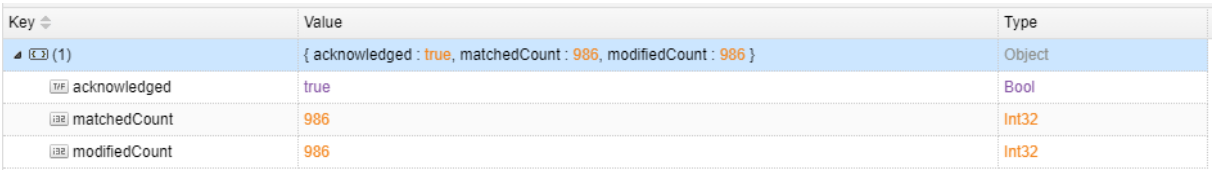


Figura 8: *Número de documentos donde el array 'cast'tiene 0 elementos.*

8. Actualizar TODOS los documentos (películas) que tengan el array cast vacío, añadiendo un nuevo elemento dentro del array con valor Undefined. Cuidado! El tipo de cast debe seguir siendo un array.

   ```
   db.movies.updateMany({cast: {$size: 0}}, {$push: {cast: 'Undefined'}})
   ```

   Primero, con {cast:{$size:0}} se actualizarán solo aquellos documentos que no tengan ningún elemento en el array 'cast', y con {$push:{cast:'Undefined'}} se añade al array un único elemento, 'Undefined'. El resultado es el siguiente:



Key	Value	Type
(1)	{ acknowledged : true, matchedCount : 986, modifiedCount : 986 }	Object
acknowledged	true	Bool
matchedCount	986	Int32
modifiedCount	986	Int32

Figura 9: *Actualización de aquellos documentos cuyo array 'cast'tiene 0 entradas a una sola, 'Undefined'.*

   En efecto, se observa como el número de documentos modificados coincide con el número obtenido en el ejercicio anterior.

9. Contar cuantos documentos (películas) tienen el array genres vacío.

   ```
   db.movies.find({genres: {$size: 0}}).count()
   ```

   Este ejercicio es análogo al ejercicio 7, donde se ejecutaba la misma *query* pero para el campo 'cast'. Se omite volver a explicar la funcionalidad de la *query* (véase el ejercicio 7 para más información). Este es el resultado:


```
1 901
```

Figura 10: *Número de documentos con 0 elementos en el campo array 'genres'.*

10. Actualizar TODOS los documentos (películas) que tengan el array genres vacío, añadiendo un nuevo elemento dentro del array con valor Undefined. Cuidado! El tipo de genres debe seguir siendo un array.

    ```
    db.movies.updateMany({genres: {$size: 0}}, {$push: {genres: 'Undefined'}})
    ```

    Este ejercicio es análogo al ejercicio 8. Se omite volver a explicar la funcionalidad de la *query*. Este es el resultado:

Key	Value	Type
(1)	{ acknowledged : true, matchedCount : 901, modifiedCount : 901 }	Object
acknowledged	true	Bool
matchedCount	901	Int32
modifiedCount	901	Int32

Figura 11: *Actualización de aquellos documentos cuyo array 'genres'tiene 0 entradas a una sola, 'Undefined'.*

11. Mostrar el año más reciente / actual que tenemos sobre todas las películas.

    ```
    db.movies.find({},{_id: 0, year: 1}).sort({year: -1}).limit(1)
    ```

    Primero, con find({}, {_id, year:1}) se permite devolver de todas las películas su año de lanzamiento (y no su _id). A continuación, con sort({year:-1}) se ordena de mayor a menor los años, y con limit(1) se devuelve tan solo el primer documento ordenado. Este es el resultado:

Key	Value	Type
(1)	{ year : 2018 }	Object
year	2018	Int32

Figura 12: *Año más reciente en el que se ha publicado una película comprendida dentro del dataset.*

12. Contar cuántas películas han salido en los últimos 20 años. Debe hacerse desde el último año que se tienen registradas películas en la colección, mostrando el resultado total de esos años.

    ```
    db.movies.find({year: {$gte: 1984}}).count( )
    ```

    Con el operador $gte elegimos aquellas películas cuyo campo 'year'sea igual o superior a 1984. El resultado es el siguiente:

```
1  8634
```

Figura 13: *Número de películas lanzadas en 1984 o después.*

13. Contar cuántas películas han salido en la década de los 60 (del 60 al 69 incluidos). Se debe hacer con el Framework de Agregación.

```
db.movies.aggregate(
    [
        {
            $match:{
                'year':{$lt: 1969, $gt: 1960}
            }
        },
        {
            $group: {_id: null, total: {$sum: 1}}
        }
    ]
)
```

En la operación con aggregate, se han realizado dos fases: en la primera se buscan aquellos documentos ($match) cuyo campo 'year'fuese mayor a 1960 y menor a 1969. A continuación, en la fase 2, se agrupan los datos y por cada uno se suma 1 al nuevo campo total, que devuelve el número total de películas lanzadas en la década de los 60. A continuación se adjunta el resultado:

Key	Value	Type
(1) null	{ total : 1115 }	Document
_id (asc index)	null	Null
total	1115 (1.1K)	Int32

Figura 14: *Número de películas lanzadas en la década de los 60.*

14. Mostrar el año u años con más películas mostrando el número de películas de ese año. Revisar si varios años pueden compartir tener el mayor número de películas.

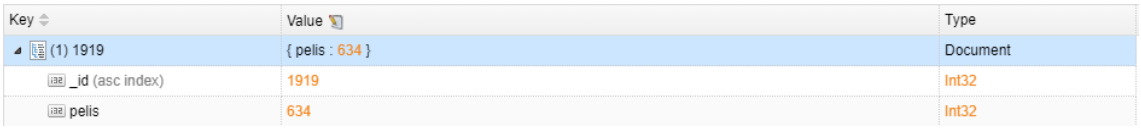
```
db.movies.aggregate([
    {
        $group: {
            _id: '$year',
            pelis: {$sum: 1}
        }
    },
    {
        $group: {
            _id: null,
            max_pelis: {$max: '$pelis'},
            years: {
                $push: {year: '$_id' , pelis: '$pelis'}
            }
        }
    },
    {
        $project: {
            years: {
                $filter: {
                    input: '$years',
                    as: 'year',
                    cond: {$eq: [ '$$year.pelis',  '$max_pelis' ]}
                }
            },
            _id: 0,
            max_pelis: 1
        }
    },
    {
        $unwind: '$years'
    },
    {
```

```
        $project: {
            _id: '$years.year ,
            pelis: '$max_pelis'
        }
    }
])
```

Antes de todo, el hecho de que se haya obtenido un código tan largo para una tarea aparentemente tan simple se debe al hecho de que puede producirse que haya dos años con el mismo número máximo de películas. Más abajo se adjunta el código correspondiente al caso en el que se supiese de antemano que un único año tuviese el número máximo de películas. Centrándonos en el primero, dentro del Framework de aggregate, primero se agrupan las películas en función de su año, tomando como _id el año de la película, y contando el número de películas por año con {$sum:1}. A continuación, generamos otro agrupamiento (como si se tratara de una iteración sucesiva) donde no agruparemos sobre nada (_id:null), sino que buscamos el número máximo de películas recogidas anteriormente (max_pelis:{$max:'$pelis'}), para que a continuación se cree un array con $push que contenga el año y el número de películas de cada registro anterior. A continuación, queremos mostrar tan solo aquellos documentos que tengan un número de películas emitidas en ese año máximo. Para ello, se utiliza $project, tal que solo se devuelve aquel array creado antes tal que el número de películas coincida con el máximo ($filter: {input: '$years', as : 'year', cond: {$eq: ['$$year.pelis', '$max_pelis']}}}). Esta tarea se ha realizado mediante la operación $filter, que parte de una input, que recibe un alias y que devuelve valores true o false en función de si una condición se cumple o no (cond). El ejercicio podría darse por terminado en este punto, si bien si se desea que el formato de la output sea como el ofrecido en la hoja de enunciados, queda tan solo separar los elementos del array como campos únicos con $unwind y finalmente realizar un $project sobre estos campos separados. A continuación se adjunta una solución mucho más simple y reducida para el caso en el que se supiese de antemano que no hay más de un año con el mismo número de películas máximo.

```
db.movies.aggregate(
    [
        {$group: {
            _id: '$year',  pelis: {$sum: 1}
            }
        },
        {
            $sort: {pelis: -1}
        },
        {
            $limit: 1
        }
    ]
)
```

El resultado, para ambos casos (ya que se da que el año donde se producen el máximo número de películas es único), es el siguiente:



Key	Value	Type
(1) 1919	{ pelis : 634 }	Document
_id (asc index)	1919	Int32
pelis	634	Int32

Figura 15: *Año con el máximo número de películas lanzadas.*

15. Mostrar el año u años con menos películas mostrando el número de películas de ese año. Revisar si varios años pueden compartir tener el menor número de películas.

```
db.movies.aggregate([
    {
        $group: {
            _id: '$year',
            pelis: {$sum: 1}
        }
    },
    {
        $group: {
            _id: null,
            min_pelis: {$min: '$pelis'},
            years: {
                $push: {year: '$_id' , pelis: '$pelis'}
            }
        }
    },
    {
        $project: {
            years: {
                $filter: {
                    input: '$years',
                    as: 'year',
                    cond: {$eq: [ '$$year.pelis',  '$min_pelis' ]}
                }
            },
            _id: 0,
            min_pelis: 1
        }
    },
    {
        $unwind: '$years'
    },
    {
        $project: {
            _id: '$years.year',
            pelis: '$min_pelis'
        }
    }
])
```

Se omite realizar algún comentario adicional ya que es la misma tarea que la del ejercicio anterior, únicamente buscando el número mínimo en vez del máximo (cabe tan solo intercambiar $max por $min). El resultado es el siguiente:

Key	Value	Type
(1) 1907	{ pelis : 7 }	Document
_id (asc index)	1907	Int32
pelis	7	Int32
(2) 1902	{ pelis : 7 }	Document
_id (asc index)	1902	Int32
pelis	7	Int32
(3) 1906	{ pelis : 7 }	Document
_id (asc index)	1906	Int32
pelis	7	Int32

Figura 16: *Años con el mínimo número de películas lanzadas.*

Se observa ahora como ya no hay una única película que satisfaga las condiciones del enunciado, sino 3.

16. Guardar en nueva colección llamada “actors” realizando la fase $unwind por actor. Después, contar cuantos documentos existen en la nueva colección.

```
db.createCollection( 'actors' );
db.movies.aggregate([
    {$unwind: '$cast' },
    {$project: {_id:0}},
    {$out: 'actors' }
    ]);
db.actors.find({}).count()
```

Primero, se crea la colección 'actors'con createCollection( ). Luego, dentro del framework de aggregate se utiliza $unwind sobre el campo 'cast'para separar los elementos dentro del array de los actores. A continuación, se utiliza $project para mostrar todos los campos menos _id, ya que tras $unwind aparecerán las llaves duplicadas, generando un error por repetición de llaves únicas. Finalmente, con $out se redirige el resultado anterior a la colección 'actors'creada anteriormente. Como resultado final, se consulta la extensión de la nueva colección poblada con la última *query*. El resultado es el siguiente:

1	83227

Figura 17: *Número de documentos en la colección 'actors'.*

17. Sobre actors (nueva colección), mostrar la lista con los 5 actores que han participado en más películas mostrando el número de películas en las que ha participado. Importante! Se necesita previamente filtrar para descartar aquellos actores llamados 'Undefined'. Aclarar que no se eliminan de la colección, sólo que filtramos para que no aparezcan.

```
db.actors.aggregate([
    {$match: {
        cast: {
            $ne: 'Undefined'
        }
    }
    },
    {$group: {
        _id: '$cast',
        pelis: {$sum: 1}
        }
    },
    {
        $sort: {pelis: -1}
    },
    {
        $limit: 5
    }
])
```

Primero, se filtran los documentos tal que tan solo nos quedamos con aquellos donde el actor ('cast') no tome el valor 'Undefined'. Una vez realizado este paso, se procede a agrupar los documentos tomando como llave el nombre del actor (_id: '$cast') y calculando el número de películas realizadas por dicho actor (pelis: {$sum: 1}). A continuación, en la siguiente fase del aggregate se ordenan los resultados en función de 'pelis'de mayor a menor ($sort: {pelis: -1}), y finalmente en la última fase con $limit: 5 se limitan los resultados a los 5 primeros. A continuación se muestra lo obtenido:

Key	Value	Type
(1) Harold Lloyd	{ pelis : 190 }	Document
_id (asc index)	Harold Lloyd	String
pelis	190	Int32
(2) Hoot Gibson	{ pelis : 142 }	Document
(3) John Wayne	{ pelis : 136 }	Document
(4) Charles Starrett	{ pelis : 116 }	Document
(5) Bebe Daniels	{ pelis : 103 }	Document

Figura 18: *5 primeros actores con mayor número de películas.*

18. Sobre actors (nueva colección), agrupar por película y año mostrando las 5 en las que más actores hayan participado, mostrando el número total de actores.

```
db.actors.aggregate([
    {$match: {
        cast:{
            $ne: 'Undefined'
        }
    }
    },
    {$group: {
        _id: {
            title: '$title',
            year: '$year',
        },
        cuenta: {$sum: 1}
        }
    },
    {
        $sort: {cuenta: -1}
    },
    {
        $limit: 5
    }
])
```

La escritura de *query* superior es muy similar a la del ejercicio anterior, la única diferencia es en que ahora se introducen dos entradas sobre _id a la hora de agrupar los documentos en función de los campos 'title'y 'year'. El resultado es el siguiente:

Key	Value	Type
(1) { title : "The Twilight Saga: Break	{ cuenta : 35 }	Document
_id (asc index)	{ title : "The Twilight Saga: Breaking Dawn - Part 2", year : 2012 }	Object
title	The Twilight Saga: Breaking Dawn - Part 2	String
year	2012	Int32
cuenta	35	Int32
(2) { title : "Anchorman 2: The Leger	{ cuenta : 33 }	Document
(3) { title : "Cars 2", year : 2011 }	{ cuenta : 32 }	Document
(4) { title : "Avengers: Infinity War", y	{ cuenta : 29 }	Document
(5) { title : "Grown Ups 2", year : 201	{ cuenta : 28 }	Document

Figura 19: *5 primeras películas con mayor número de actores que participen en ellas.*

19. Sobre actors (nueva colección), mostrar los 5 actores cuya carrera haya sido la más larga. Para ello, se debe mostrar cuándo comenzó su carrera, cuándo finalizó y cuántos años ha trabajado. Importante! Se necesita previamente filtrar para descartar aquellos actores llamados 'Undefined'. Aclarar que no se eliminan de la colección, sólo que filtramos para que no aparezcan.

```
db.actors.aggregate([
    {$match: {
        cast:{
            $ne: 'Undefined'
        }
    }
    },
    {
        $group: {
            _id: '$cast' ,
            comienza: {$min: '$year'},
            termina: {$max: '$year'}
        }
    },
    {
        $project: {
            _id: 1,
            comienza: 1,
            termina: 1,
            años: {
                $subtract: ['$termina' , '$comienza']
            }
        }
    },
    {
        $sort: {años: -1}
    },
    {
        $limit: 5
    }
])
```

En primer lugar, como se ha estado haciendo hasta ahora, se filtran aquellas películas tal que no haya ninguna con actores 'Undefined'. Luego, se agrupan en función del actor, y se escriben los años en los que empieza y termina su carrera cinematográfica (comienza: {$min: '$year'} y termina: {$max: '$year'}, respectivamente). A continuación, con $project devolvemos las cantidades anteriores más un nuevo campo creado, 'años', que corresponde a la diferencia entre el año en el que termina su carrera y el año en el que la inicia. Finalmente, con $sort y $limit se ordenan de mayor a menor los resultados y se limitan a 5. Se obtiene lo siguiente:

Key	Value	Type
(1) Harrison Ford	{ comienza : 1919, termina : 2017, "años" : 98 } (4 fields)	Document
_id (asc index)	Harrison Ford	String
comienza	1919 (1.9K)	Int32
termina	2017 (2.0K)	Int32
años	98	Int32
(2) John Doe	{ comienza : 1900, termina : 1992, "años" : 92 } (4 fields)	Document
(3) Gloria Stuart	{ comienza : 1932, termina : 2012, "años" : 80 } (4 fields)	Document
(4) Lillian Gish	{ comienza : 1912, termina : 1987, "años" : 75 } (4 fields)	Document
(5) Kenny Baker	{ comienza : 1937, termina : 2012, "años" : 75 } (4 fields)	Document

Figura 20: *5 primeros actores con la carrera cinematográfica más extensa.*

20. Sobre actors (nueva colección), Guardar en nueva colección llamada “genres” realizando la fase $unwind por genres. Después, contar cuantos documentos existen en la nueva colección.

```
db.createCollection('genres' );
db.actors.aggregate([
    {$unwind: '$genres'},
    {$project: {_id: 0} },
    {$out: 'genres'}
    ]);
db.genres.find({}).count()
```

Este ejercicio es análogo al 16, de aquí que se omita volver a repetir su explicación. El resultado es el siguiente:

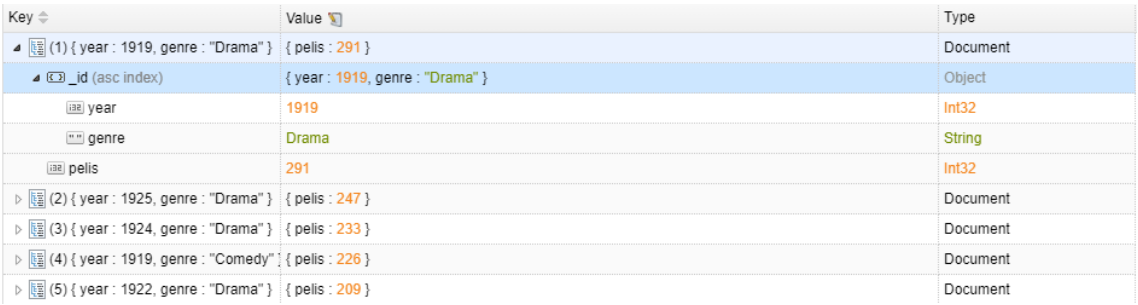
```
1  104956
```

Figura 21: *Número de documentos en la colección 'genres'.*

21. Sobre genres (nueva colección), mostrar los 5 documentos agrupados por “Año y Género” que más número de películas diferentes tienen mostrando el número total de películas.

```
db.genres.aggregate([
    {$match: {
        genres:{
            $ne: 'Undefined'
        }
    }
    },
    {$group: {
        _id: {
            year: '$year',
            genre: '$genres',
        },
        unique_titles: {$addToSet: '$title'}
        }
    },
    {
        $project: {
            year: 1,
            genre: 1,
            pelis: {$size: '$unique_titles'}
        }
    },
    {
        $sort: {pelis: -1}
    },
    {
        $limit: 5
    }
])
```

En primer lugar, se filtran aquellos documentos donde en el campo 'genres'se tenga un valor 'Undefined'. Una vez realizado este paso, se agrupan los documentos tomando como llaves los campos 'year'y 'genres'. A continuación, dentro del propio $group, se crea un set donde se vayan añadiendo las diferentes películas que se vayan agrupando en función de su año de lanzamiento y género. El set permite evitar duplicados de los títulos. En la siguiente fase, con $project se devuelven tan solo el año, el género y el tamaño del set creado antes con $size de las películas. Ya como paso final, se ordena en función de 'pelis'de mayor a menor y se limita el resultado a los 5 primeros documentos. Se obtiene lo siguiente:



Key	Value	Type
(1) { year : 1919, genre : "Drama" }	{ pelis : 291 }	Document
_id (asc index)	{ year : 1919, genre : "Drama" }	Object
year	1919	Int32
genre	Drama	String
pelis	291	Int32
(2) { year : 1925, genre : "Drama" }	{ pelis : 247 }	Document
(3) { year : 1924, genre : "Drama" }	{ pelis : 233 }	Document
(4) { year : 1919, genre : "Comedy" }	{ pelis : 226 }	Document
(5) { year : 1922, genre : "Drama" }	{ pelis : 209 }	Document

Figura 22: *5 primeros años y géneros donde se tiene más películas lanzadas, junto a dicho número.*

22. Sobre genres (nueva colección), mostrar los 5 actores y los géneros en los que han participado con más número de géneros diferentes, se debe mostrar el número de géneros diferentes que ha interpretado. Importante! Se necesita previamente filtrar para descartar aquellos actores llamados 'Undefined'. Aclarar que no se eliminan de la colección, sólo que filtramos para que no aparezcan.

```
db.genres.aggregate([
    {$match: {
        cast:{
            $ne: 'Undefined'
        }
    }
    },
    {$group: {
        _id: '$cast' ,
        generos: {$addToSet: '$genres'}
        }
    },
    {
        $project: {
            _id: 1,
            numgeneros: {$size: '$generos'},
            generos: 1
        }
    },
    {
        $sort: {numgeneros: -1}
    },
    {
        $limit: 5
    }
])
```

Este ejercicio es análogo al anterior: se filtran aquellos actores que constan como 'Undefined'. Luego, se agrupan según su nombre, y en un set se almacenan todos aquellos géneros en los que consta que han participado. Luego, mediante $project se devuelven el nombre del actor, el número de géneros mediante $size del campo 'generos', y 'generos'. Finalmente, se ordena según 'numgeneros'de mayor a menor y se

limita el resultado a los 5 primeros. Se obtiene lo siguiente:

Key	Value	Type
(1) Dennis Quaid	{ numgeneros : 20 } (3 fields)	Document
_id (asc index)	Dennis Quaid	String
generos	Array[20]	Array
numgeneros	20	Int32
(2) James Mason	{ numgeneros : 19 } (3 fields)	Document
(3) Michael Caine	{ numgeneros : 19 } (3 fields)	Document
(4) Michael Peña	{ numgeneros : 18 } (3 fields)	Document
(5) Lionel Barrymore	{ numgeneros : 18 } (3 fields)	Document

Figura 23: *5 primeros actores con más géneros en los que han sido partícipes, junto a un array que los contenga y su extensión.*

23. Sobre genres (nueva colección), mostrar las 5 películas y su año correspondiente en los que más géneros diferentes han sido catalogados, mostrando esos géneros y el número de géneros que contiene.

```
db.genres.aggregate([
    {$match: {
        genres: {
            $ne: 'Undefined'
        }
    }
    },
    {$group: {
        _id: {
            title: '$title' ,
            year: '$year'
        },
        generos: {$addToSet: '$genres' }
        }
    },
    {
        $project: {
            _id: 1,
            numgeneros: {$size: '$generos' },
            generos: 1
        }
    },
    {
        $sort: {numgeneros: -1}
    },
    {
        $limit: 5
    }
])
```

Se omite explicar cada *query* por su analogía con los ejercicios anteriores. El resultado es el siguiente:

Key	Value	Type
(1) { title : "American Made", year : 2	{ generos : ["Action", "Historical", "Crime", "Comedy", "Drama", "Biography", "Thriller"], numgeneros :	Document
_id (asc index)	{ title : "American Made", year : 2017 }	Object
title	American Made	String
year	2017	Int32
generos	Array[7]	Array
0	Action	String
1	Historical	String
2	Crime	String
3	Comedy	String
4	Drama	String
5	Biography	String
6	Thriller	String
numgeneros	7	Int32
(2) { title : "My Little Pony: The Movi	{ generos : ["Family", "Fantasy", "Musical", "Comedy", "Animated", "Adventure"], numgeneros : 6 }	Document
(3) { title : "Dunkirk", year : 2017 }	{ generos : ["Action", "Historical", "War", "Drama", "Adventure", "Thriller"], numgeneros : 6 }	Document
(4) { title : "Thor: Ragnarok", year : 2	{ generos : ["Science Fiction", "Action", "Fantasy", "Adventure", "Superhero", "Comedy"], numgenero	Document
(5) { title : "The Dark Tower", year : 2	{ generos : ["Horror", "Science Fiction", "Action", "Adventure", "Western", "Fantasy"], numgeneros : 6	Document

Figura 24: *5 primeros títulos y años que contengan el mayor número de géneros.*

4. Ejercicios propuestos

Se propone a continuación un total de tres ejercicios adicionales donde se haga uso de la pipeline de agregación.

1. Cataloga todas aquellas películas del género Erotic como '+18 'y 'No +18 'para el resto. Devuelve tan solo el título y su etiqueta +18 o no +18.

```
db.movies.aggregate([
    {
        $project: {
            title: 1,
            adult_content: {
                $cond: {
                    if: {$in: ['Erotic' , '$genres' ]},
                    then: '+18' ,
                    else: 'No +18'
                        }
            }
        }
    }
]);
```

Mediante $project se mostrarán el título de la película y el nuevo campo creado adult_content que contenga un condicional ($cond), tal que si el género 'Erotic 'se halla dentro del campo array genres, se devuelve '+18 'en el campo adult_content ($in: ['Erotic ', '$genres ']). Sino, se devuelve 'No +18 '. El resultado es el siguiente:

Key	Value	Type
(1) 672bdeecc164b76a38bee557	{ title : "Caught", adult_content : "No +18" }	Document
(2) 672bdeecc164b76a38bee558	{ title : "After Dark in Central Park", adult_content : "No +18" }	Document
(3) 672bdeecc164b76a38bee559	{ title : "Buffalo Bill's Wild West Parad", adult_content : "No +18" }	Document
(4) 672bdeecc164b76a38bee55a	{ title : "The Enchanted Drawing", adult_content : "No +18" }	Document
(5) 672bdeecc164b76a38bee55b	{ title : "Clowns Spinning Hats", adult_content : "No +18" }	Document
(6) 672bdeecc164b76a38bee55c	{ title : "Capture of Boer Battery by British", adult_content : "No +18" }	Document
(7) 672bdeecc164b76a38bee55d	{ title : "Feeding Sea Lions", adult_content : "No +18" }	Document
(8) 672bdeecc164b76a38bee55e	{ title : "How to Make a Fat Wife Out of Two Lean Ones", adult_content : "No +18" }	Document
(9) 672bdeecc164b76a38bee55f	{ title : "Searching Ruins on Broadway, Galveston, for Dead Bodies", adult_content : "No +18" }	Document
(10) 672bdeecc164b76a38bee560	{ title : "The Tribulations of an Amateur Photographer", adult_content : "No +18" }	Document
(11) 672bdeecc164b76a38bee561	{ title : "Trouble in Hogan's Alley", adult_content : "No +18" }	Document
(12) 672bdeecc164b76a38bee562	{ title : "Boarding School Girls' Pajama Parade", adult_content : "No +18" }	Document

Figura 25: *Títulos de las películas y su etiqueta en función de si son +18 o no.*

2. Calcula el número promedio de películas lanzadas por año.

```
db.movies.aggregate([
    {
        $group: {
        _id: '$year',
        pelis: {$sum: 1}
        }
    },
    {
        $group: {
            _id: null,
            promedio: {$avg: '$pelis' }
        }
    },
    {
        $project: {
            _id: 0,
            promedio: 1
        }
    }
])
```

En primer lugar, se agrupan las películas tomando como llave el año ('$year') y se guardan en pelis el número de películas lanzadas en un mismo año. Finalmente, mediante otro $group y tomando la llave como nula, se calcula el promedio del campo pelis. Finalmente, se devuelve el valor promedio mediante $project. Se obtiene lo siguiente:

Key	Value	Type
(1)	{ promedio : 241.98319327731093 }	Object

Figura 26: *Promedio de películas lanzadas en un año.*

3. Lista las 5 primeras películas con un título más extenso.

```
db.movies.aggregate([
    {
        $project: {
            title: 1,
            longitud_titulo: {$strLenCP: '$title'}
        }
    },
    {
        $sort: {longitud_titulo: -1}
    },
    {
        $limit: 5
    }
])
```

Mediante $project (dentro de aggregate), se devuelve el título de la película junto a la longitud de dicho campo, haciendo uso de $strLenCP, que devuelve la longitud de un string. Finalmente, se ordenan de mayor a menor los documentos en función de las longitudes de los textos y nos quedamos con los primeros 5 títulos. Se obtiene lo siguiente:

Key	Value	Type
(1) 672bdeecc164b76a38bee579	{3 fields}	Document
(2) 672bdeecc164b76a38bee634	{ title : "The Green Goods Man; or, Josiah and Samanthy's Experience with the Original 'American Co	Document
(3) 672bdeecc164b76a38bee58e	{ title : "Launching of the New Battleship 'Ohio' at San Francisco, Cal. When President McKinley Was	Document
(4) 672bdeecc164b76a38bee5a7	{ title : "Opening of the Pan-American Exposition Showing Vice President Roosevelt Leading the Proc	Document
(5) 672bdeecc164b76a38bee5b5	{ title : "Arrival of Prince Henry (of Prussia) and President Roosevelt at Shooter's Island (1902)" }	Document

Figura 27: *Primeras 5 películas con los títulos más extensos.*

5. Conclusiones

Se han puesto en práctica los conocimientos en MongoDB a la hora de ejecutar *queries* de complejidad variada con el fin de obtener información de interés sobre una serie de películas. Así, se han realizado satisfactoriamente ejercicios que ponen en práctica los módulos de agregación, proyección, búsqueda de documentos y actualización de campos.