Ejercicios de uso de MongoDB

Entorno de desarrollo.

Requisitos previos:

- Lectura del documento "*Uso máquina Virtual UOC: MONGO DB"*
- Instalación y arranque de la máquina virtual suministrada
- Haber descargado los ficheros de datos

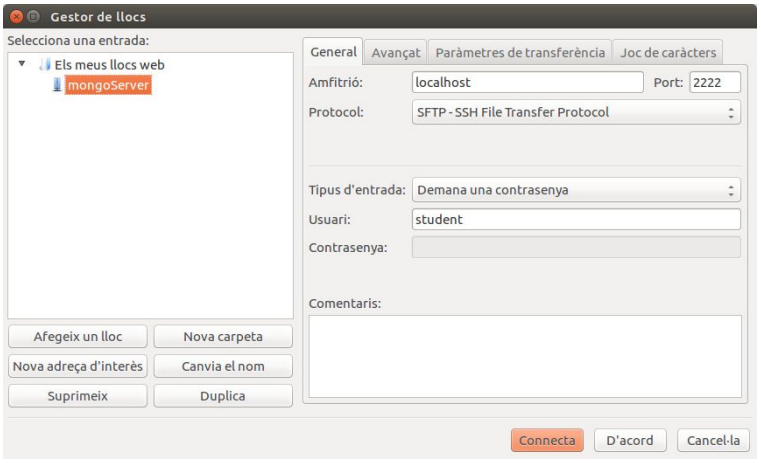
Proceso de transferencia de ficheros a la máquina virtual por SFTP

Instalamos el cliente FTP Filezilla. https://filezilla-project.org/

La documentación para su instalación la tenemos ubicada en:
https://wiki.filezilla-project.org/Documentation

Una vez instalado Filezilla creamos una conexión aprovechando la preconfiguración de la máquina virtual ***UOC NoSQL MongoDB*** que actúa de servidor. La máquina virtual viene preconfigurada con NAT de la máquina virtual de forma que el puerto 2222 del host redirige al puerto 22 del guest (máquina virtual). Utilizamos el protocolo de transferencia de ficheros SFTP. El nombre del usuario y su password es `student / student` tal y como se especifica en el documento *«Uso máquina Virtual UOC: MONGO DB»*

Nombramos a la conexión *mongoServer:*



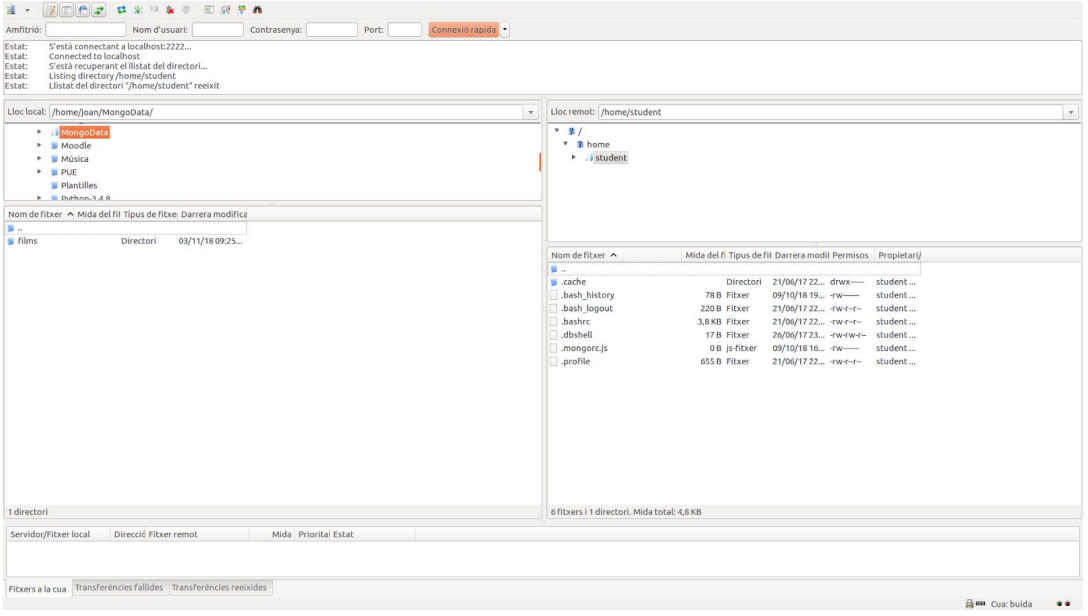
Cuando realizamos la conexión nos solicita el password:

Después de un intento de conexión exitosa, aparece una lista de archivos y directorios en el lado derecho de la ventana principal. El nombre del directorio remoto actual aparece en el campo de edición en la parte superior. Debajo veréis el árbol de directorios remotos. Debajo del directorio remoto hay una lista de los contenidos del directorio remoto actual.

Para cambiar el directorio remoto actual:

- Escribid un nombre de directorio en el campo de edición y presionad *Intro*, o
- Haced clic en un directorio en el árbol de directorios, o
- Haced doble clic en un directorio en la lista de los contenidos del directorio actual

Notaréis que un directorio llamado ".." aparece en prácticamente todos los directorios. La selección de este directorio os permite subir al directorio principal del directorio actual.



Podemos observar que en la ventana izquierda de la captura de pantalla anterior, que corresponde al sistema de directorios de la máquina host, existe el directorio `MongoData`. En su interior aparece el directorio `films` resultado de la descompresión del fichero de datos proporcionado.

Tan sólo con arrastrar con el ratón este directorio de la ventana izquierda a a la ventana derecha habremos realizado la transferencia de los ficheros que contiene.

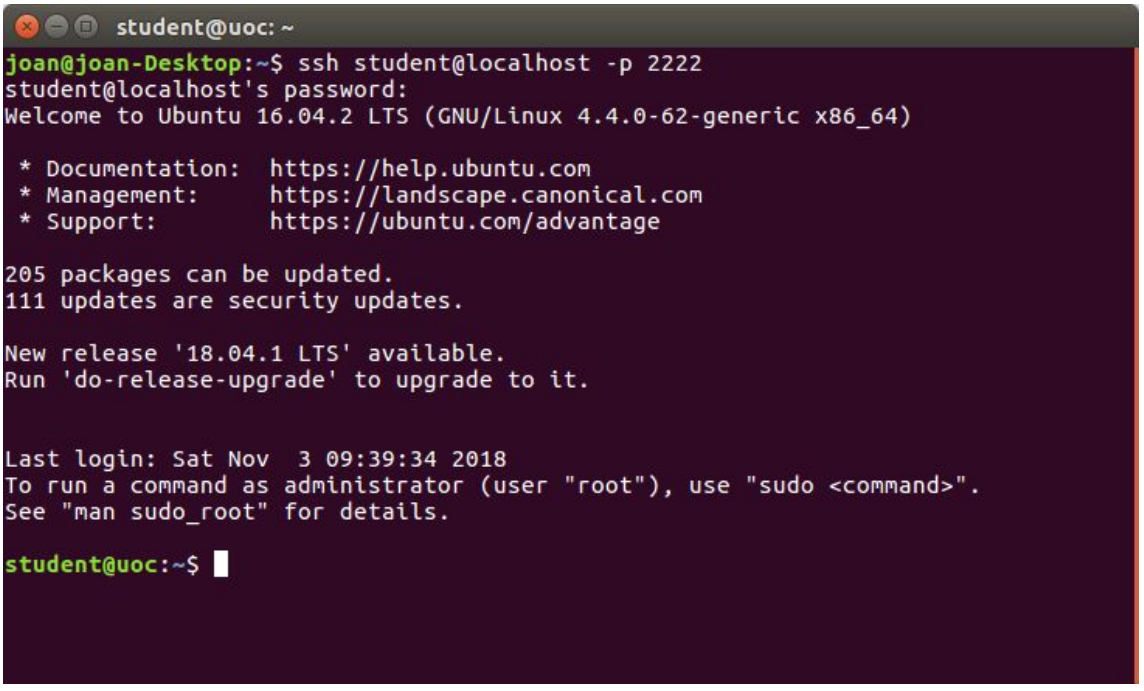
Importación de datos al servidor MongoDB.

La herramienta `mongoimport` importa el contenido de una exportación realizada anteriormente mediante `mongoexport` utilizando los formatos JSON, CSV o TSV.

https://docs.mongodb.com/manual/reference/program/mongoimport/

Nos conectamos mediante ssh desde nuestro ordenador al servidor ubicado en la máquina virtual.

```
ssh student@localhost -p 2222
```

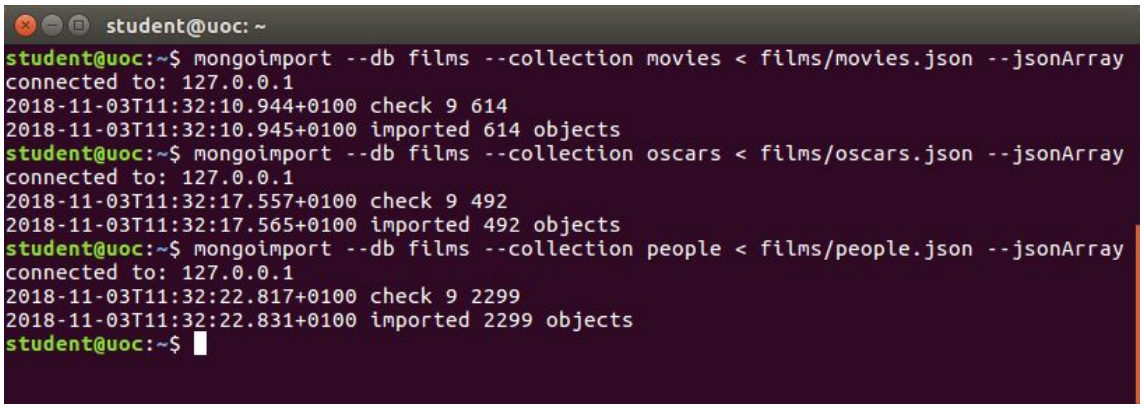


Una vez realizada la conexión importamos los tres ficheros de datos en una colección para cada uno, con el mismo nombre del fichero proporcionado, en una nueva base de datos llamada films. Notad que el contenido de cada fichero es un array en formato

JSON. Las sentencias para realizar la importación son las siguientes:

```
mongoimport --db films --collection movies < films/movies.json --jsonArray
mongoimport --db films --collection oscars < films/oscars.json --jsonArray
mongoimport --db films --collection people < films/people.json --jsonArray
```

Observamos que el comando mongoimport utiliza los modificadores *--db* para especificar la base de datos y -- collection para especificar la colección de destino. El símbolo < indica que la entrada de datos se realizará a partir del nombre del fichero especificado a continuación. Finalmente el modificador *--jsonArray* indica que el fichero de datos incluye un array en formato JSON.

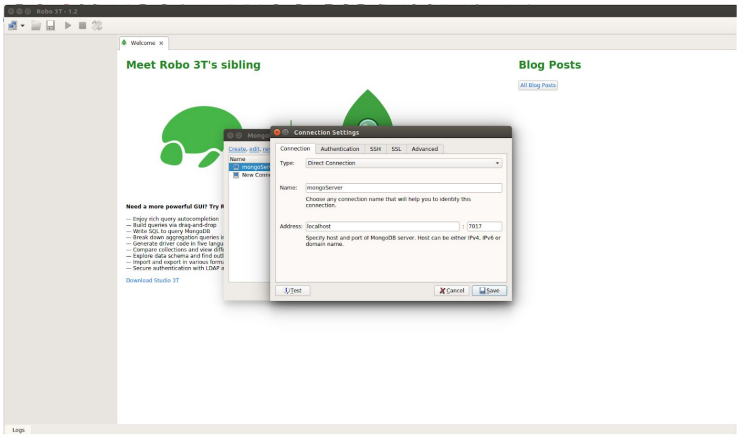


Una vez realizada la importación recomendamos usar las fichas de referencia de mongodb, https://www.mongodb.com/collateral/quick-reference-cards

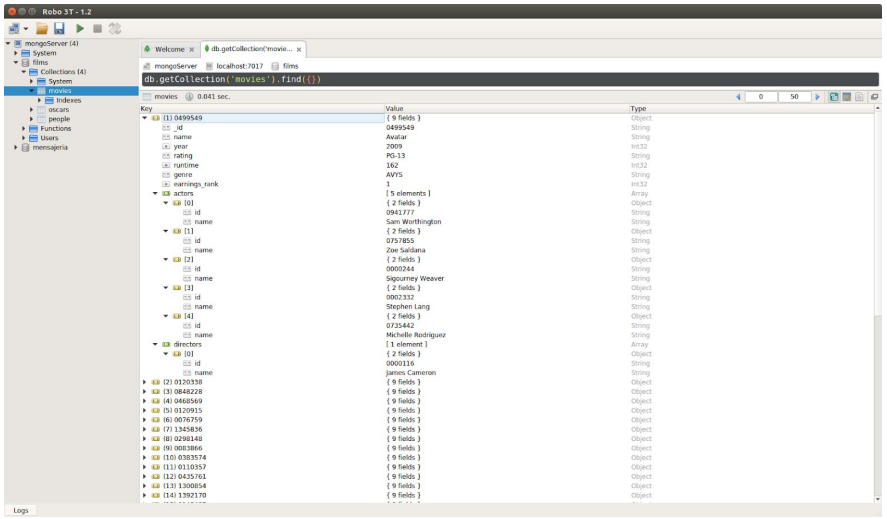
Uso de Robo 3T

Robo 3T (anteriormente Robomongo) es una GUI ligera y gratuita para los entusiastas de MongoDB. Se puede descargar desde https://robomongo.org/download

Una vez instalada ejecutamos la aplicación y creamos la conexión correspondiente:



Una vez realizada la conexión podemos visualizar el contenido de la base de datos. En el ejemplo visualizamos el contenido de la colección movies expandiendo el contenido del primer agregado.



Podéis ver que en la parte superior de Robo3T aparece una consola desde la cual podemos realizar algunas consultas.

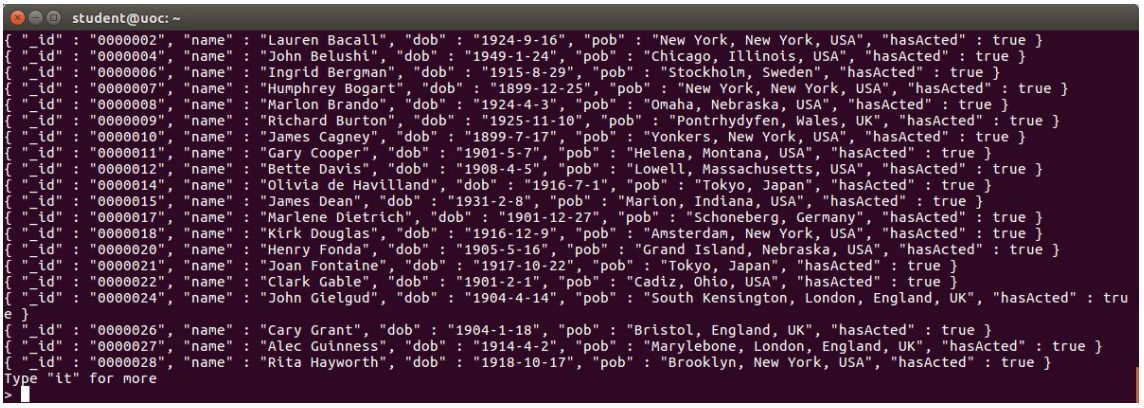
Consultas simples

https://docs.mongodb.com/manual/tutorial/query-documents/

Operaciones de selección

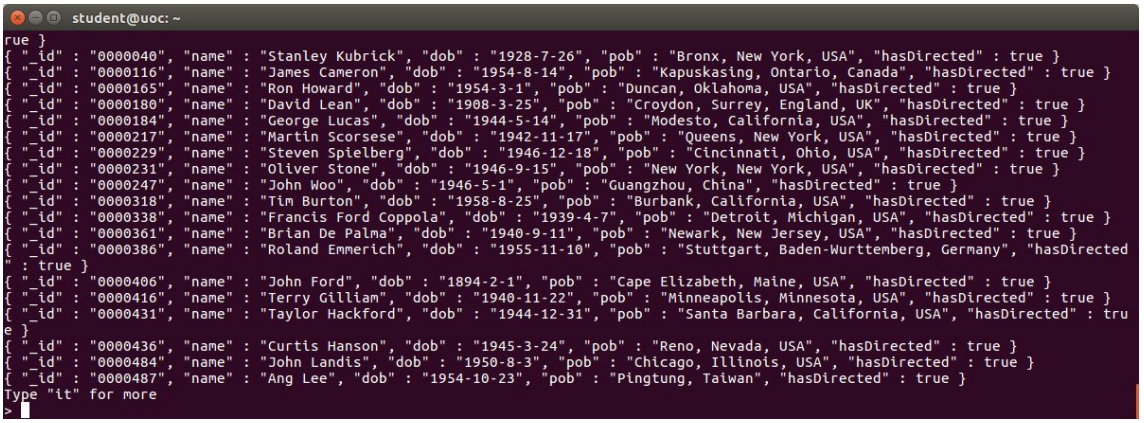
1. Buscar las personas que sólo han actuado (no dirigido) (1909 ocurrencias)

```
> db.people.find({"hasActed": {$exists: true}, "hasDirected": {$exists: false}})
```



2. Buscar las personas que sólo han dirigido (no actuado) (341 ocurrencias)

```
> db.people.find({"hasActed": {$exists: false}, "hasDirected": {$exists: true}})
```



3. Buscar las personas que han actuado y dirigido (20 ocurrencias)

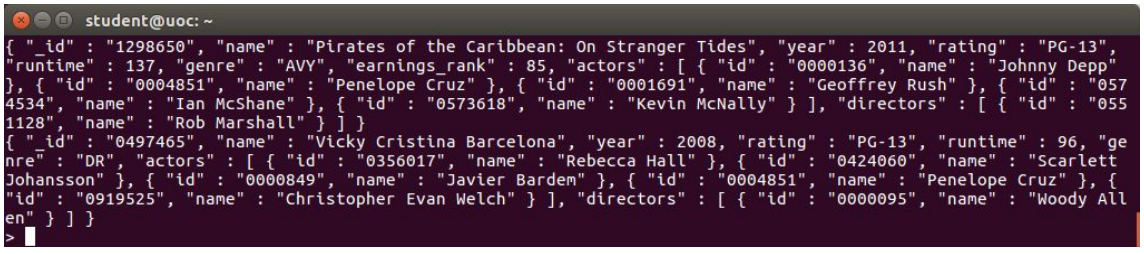
```
> db.people.find({"hasActed": {$exists: true}, "hasDirected": {$exists: true}})
```

4. Buscar las personas que ni han actuado ni dirigido (29 ocurrencias)

```
> db.people.find({"hasActed": {$exists: false}, "hasDirected": {$exists: false}})
```

5. Buscar las películas protagonizadas por Penelope Cruz (2)

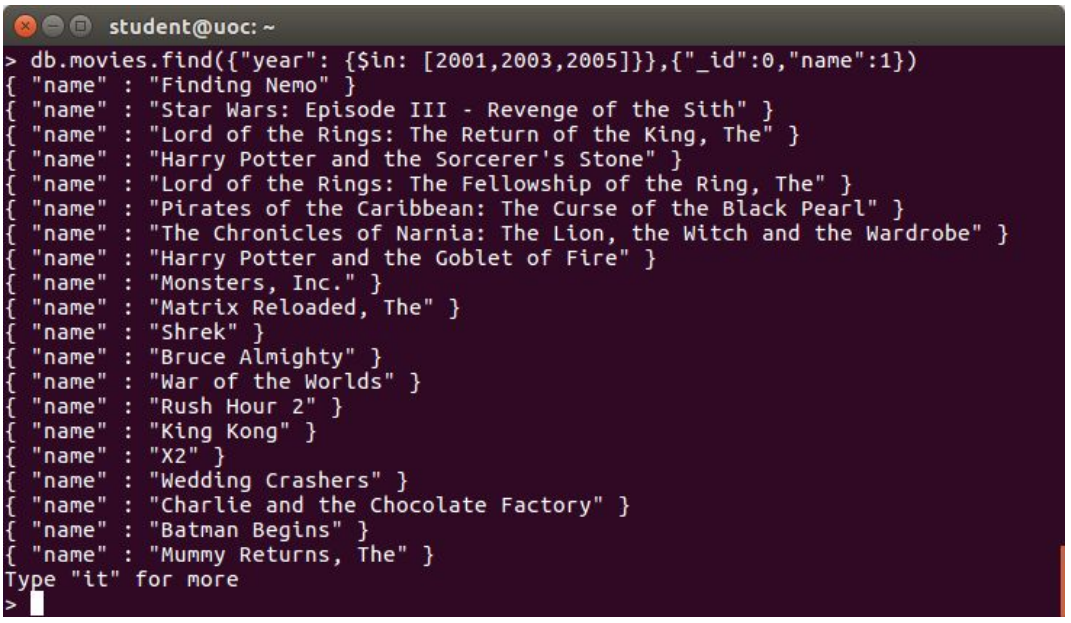
```
> db.movies.find({"actors.name": "Penelope Cruz"})
```



Operaciones de proyección

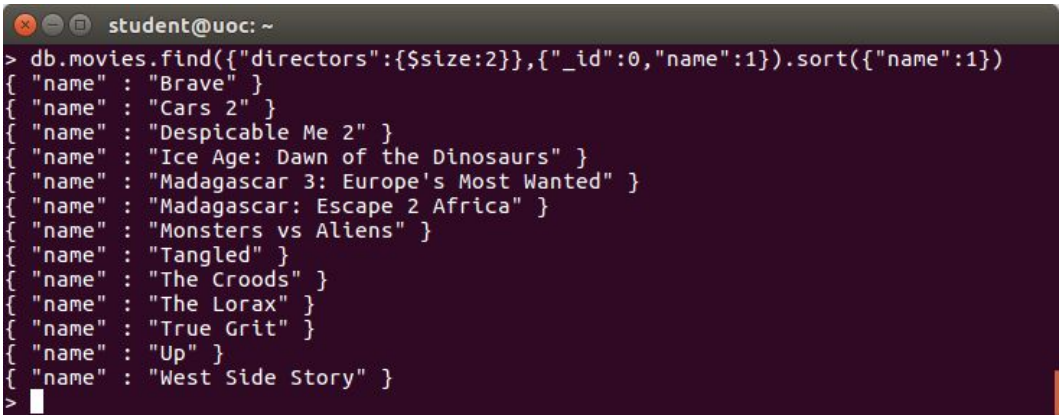
6. Muestra el título de las películas, sin mostrar el valor de _id, que se han estrenado en los años 2001, 2003 o 2005.

```
> db.movies.find({"year": {$in: [2001,2003,2005]}},
                 {"_id":0,"name":1})
```



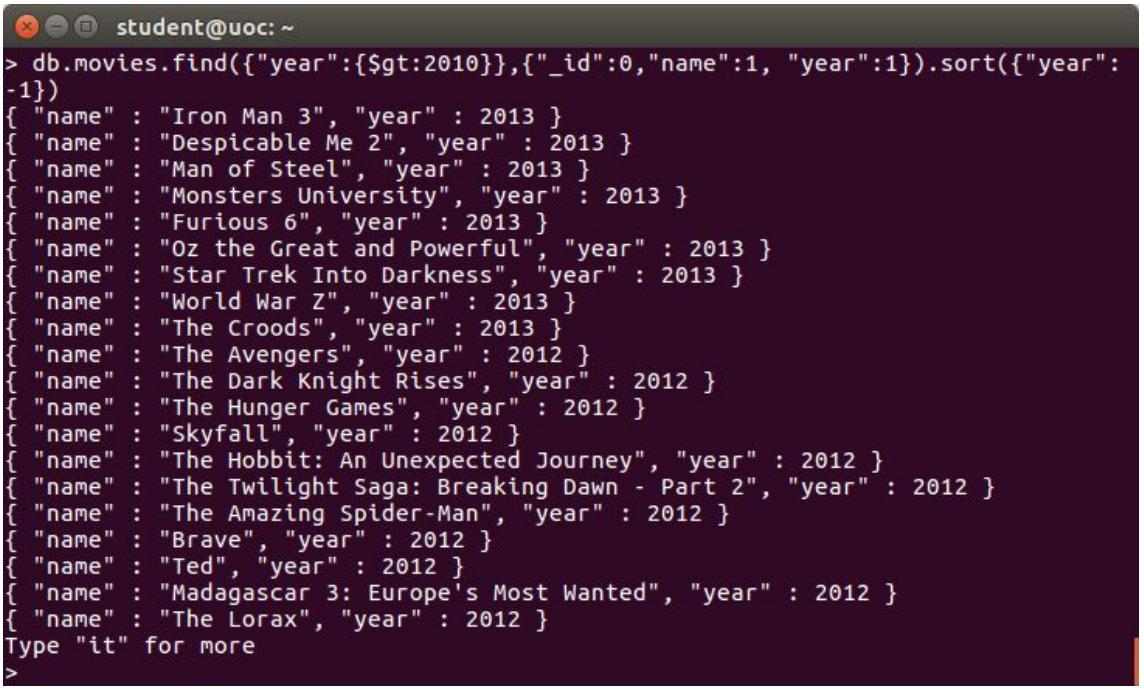
7. Muestra el título de las películas, sin mostrar el valor de _id, que han sido dirigidas por exactamente 2 directores, ordenando la salida por el título de la película en orden ascendente.

```
> db.movies.find({"directors":{$size:2}},
                 {"_id":0,"name":1}).sort({"name":1})
```



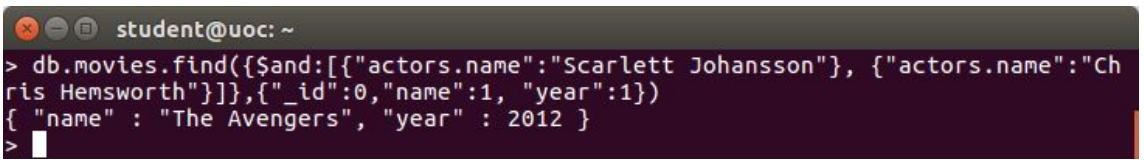
8. Muestra el título de la película y el año de estreno, sin mostrar el valor de _id, de aquellas películas que han sido estrenadas con posterioridad al año 2010, ordenando la salida por el año de estreno en orden descendente.

```
> db.movies.find({"year":{$gt:2010}},
                 {"_id":0,"name":1, "year":1})
                 .sort({"year":-1})
```



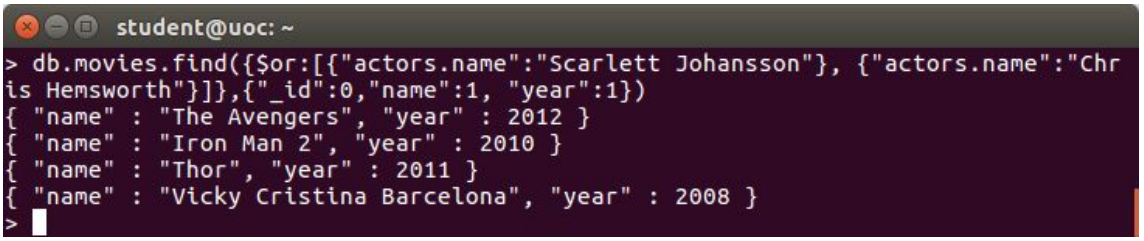
9. Muestra el título de la película y el año de estreno, sin mostrar el valor de _id, de aquellas películas que han protagonizadas por Scarlett Johansson y por Chris Hemsworth .

```
> db.movies.find({$and:[
                        {"actors.name":"Scarlett Johansson"},
                        {"actors.name":"Chris Hemsworth"}
                        ]
                },
                {"_id":0,"name":1, "year":1})
```



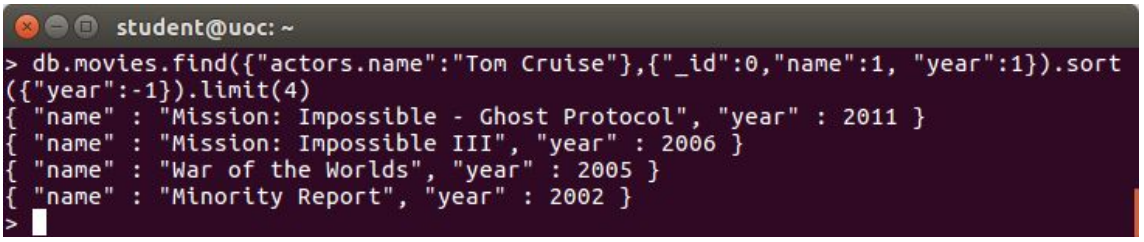
10. Muestra el título de la película y el año de estreno, sin mostrar el valor de _id, de aquellas películas que han protagonizadas por Scarlett Johansson o por Chris Hemsworth .

```
> db.movies.find({$or:[
                        {"actors.name":"Scarlett Johansson"},
                        {"actors.name":"Chris Hemsworth"}
                        ]
                  },
                  {"_id":0,"name":1, "year":1})
```



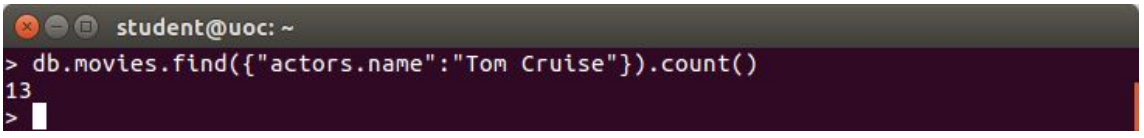
11. Muestra el título de la película y el año de estreno, sin mostrar el valor de _id, de las 4 últimas películas almacenadas en la base de datos protagonizadas por Tom Cruise.

```
> db.movies.find({"actors.name":"Tom Cruise"},
                  {"_id":0,"name":1,
            "year":1}).sort({"year":-1})
                  .limit(4)
```



12. Muestra cuántas películas hay almacenadas en la base de datos que hayan sido protagonizadas por Tom Cruise.

```
> db.movies.find({"actors.name":"Tom Cruise"}).count()
```



Almacenamiento de informaciones geoespaciales

Se recomienda la lectura del siguiente documento:

https://docs.mongodb.com/manual/geospatial-queries/

Índices

MongoDB puede almacenar cualquier tipo de datos, pero si deseamos consultar algunos atributos geoespaciales, necesitaremos usar algunas coordenadas y además crear un índice en ellas. MongoDB admite tres tipos de índices para consultas geoespaciales:

- 2d Index : Utiliza coordenadas simples (longitud, latitud). Como se indica en la documentación: El índice 2d está diseñado para pares de coordenadas heredados utilizados en MongoDB 2.2 y anteriores. Por esta razón, no detallaremos nada sobre esto en esta actividad. El índice 2d se usa para consultar datos almacenados como puntos en un plano bidimensional.
- 2d Sphere Index: Admite consultas de cualquier geometría en una esfera similar a la de la Tierra y los datos se pueden almacenar como GeoJSON.
- Geo Haystack: Se utiliza para consultar en un área muy pequeña. Hoy en día es el menos utilizado por las aplicaciones y no lo describiremos en este tutorial.

GeoJSON

Aunque podemos analizar su formato en la documentación oficial (https://tools.ietf.org/html/rfc7946) podemos empezar dando una breve explicación.

GeoJSON es un formato para codificar, en JSON, una variedad de estructuras de datos geográficos, y admite los siguientes tipos: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon y Geometry.

El formato GeoJSON es bastante sencillo, para las geometrías simples, en dos atributos: tipo y coordenadas. Tomemos un ejemplo:

La ciudad de Barcelona, tiene las siguientes coordenadas (de Wikipedia)

41°23′N 2°11′E

Esta notación es un punto, basada en la latitud y longitud utilizando el sistema WGS 84 (grados, minutos, segundos). No es muy fácil de usar por las aplicaciones, por eso también es posible representar el mismo punto usando los siguientes valores de latitud

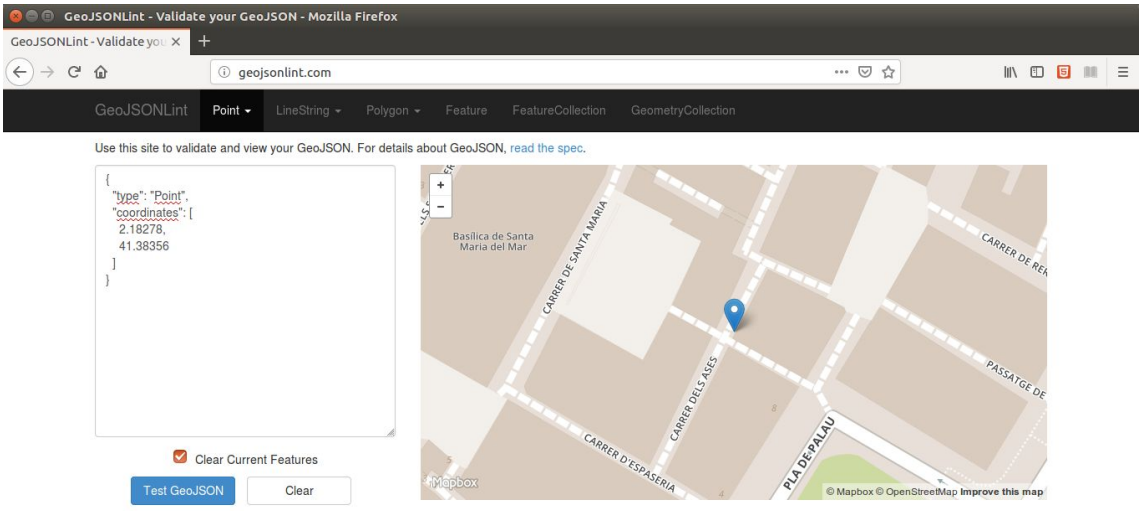
y longitud:

41.383333, 2.183333

Éste utiliza el sistema WGS 84 (grados decimales). Es el formato de coordenadas que se utiliza en la mayoría de las aplicaciones o APIs (por ejemplo: Google Maps / Earth,)

Por defecto, GeoJSON y MongoDB usan estos valores, pero las coordenadas deben almacenarse en el orden de latitud y longitud, por lo que este punto en GeoJSON se verá así:

```
{
  "type": "Point",
  "coordinates": [ 2.183333  41.383333 ]
}
```



Podéis ver el listado de los objetos GeoJSON que soporta MongoDB en en el siguiente enlace.

Prerrequisitos

Lectura de la documentación oficial de MongoDB sobre Geospatial Queries

Descomprimir el fichero suministrado con nombre *geo.zip*. Se adjuntan los ficheros de datos en formato GeoJSON siguientes:

- *shapefiles_barcelona_distrito.geojson* :
- *barris_barcelona.geojson* :
- *barcelona_edificios_protegidos.geojson* :

Nota: Estos datos han sido descargados de https://vangdata.carto.com/datasets y posteriormente depurados para poder realizar los siguientes ejemplos.

Mediante SFTP se transfieren a la máquina virtual los ficheros de datos mencionados. Para ello se recomienda utilizar *Filezilla* tal y como se ejemplifica en el apartado anterior.

Una vez los ficheros de datos han sido transferidos a la máquina virtual nos conectamos a ella vía ssh utilizando el terminal de la máquina anfitriona:

```
ssh student@localhost -p 2222
```

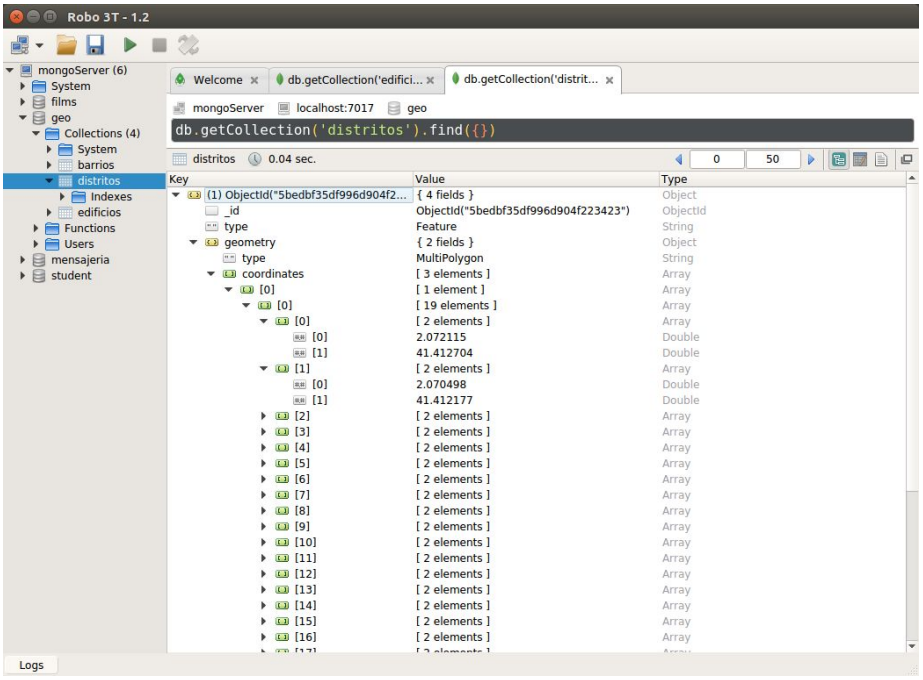
Una vez conectados a la máquina virtual ejecutamos la importación de datos al servidor MongoDB. La base de datos de destino la llamamos geo e incluye las colecciones *distritos, barrios* y *edificios* donde se almacenará suministrada por los respectivos ficheros. Las sentencias para realizar la importación son las siguientes:

```
mongoimport --db geo --collection distritos < geo/shapefiles_barcelona_distrito.geojson --jsonArray

mongoimport --db geo --collection barrios < geo/barris_barcelona.geojson

mongoimport --db geo --collection edificios < geo/barcelona_edificios_protegidos.geojson --jsonArray
```

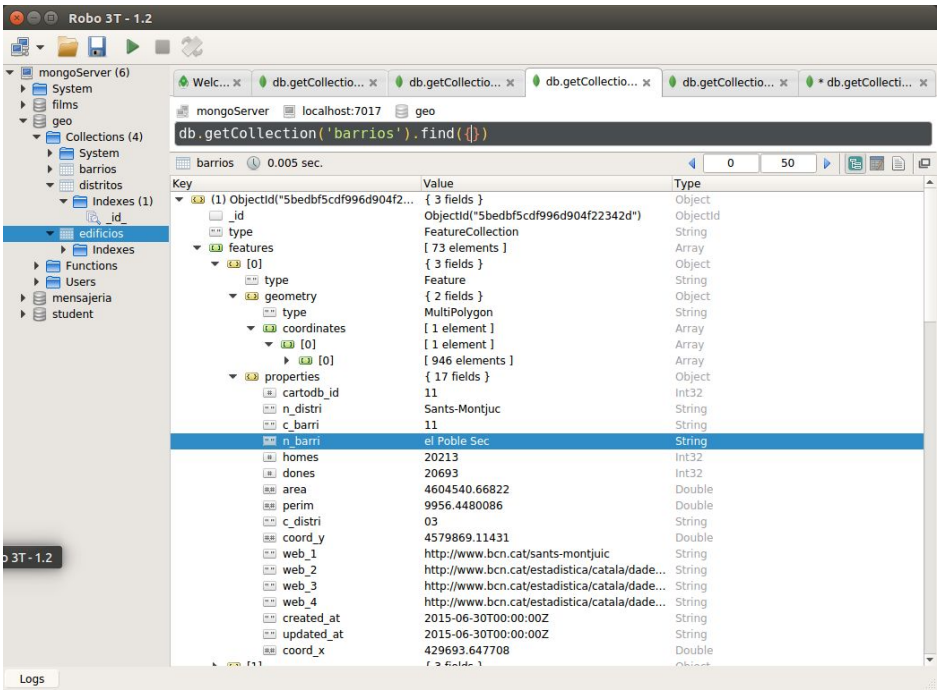
y se realiza una conexión a la base de datos desde el terminal.

```
student@uoc: ~
student@uoc:~$ mongoimport --db geo --collection distritos < geo/shapefiles_barcelona_distrito.geojson --jsonArray
connected to: 127.0.0.1
2018-11-15T19:47:17.025+0100 imported 10 objects
student@uoc:~$ mongoimport --db geo --collection barrios < geo/barris_barcelona.geojson
connected to: 127.0.0.1
2018-11-15T19:47:56.413+0100 imported 1 objects
student@uoc:~$ mongoimport --db geo --collection edificios < geo/barcelona_edificios_protegidos.geojson  --jsonArray
connected to: 127.0.0.1
2018-11-15T19:48:13.469+0100 check 9 206
2018-11-15T19:48:13.470+0100 imported 206 objects
student@uoc:~$ mongo geo
MongoDB shell version: 2.6.10
```

Mediante Robo 3T también podemos analizar la estructura de documentos y agregados.

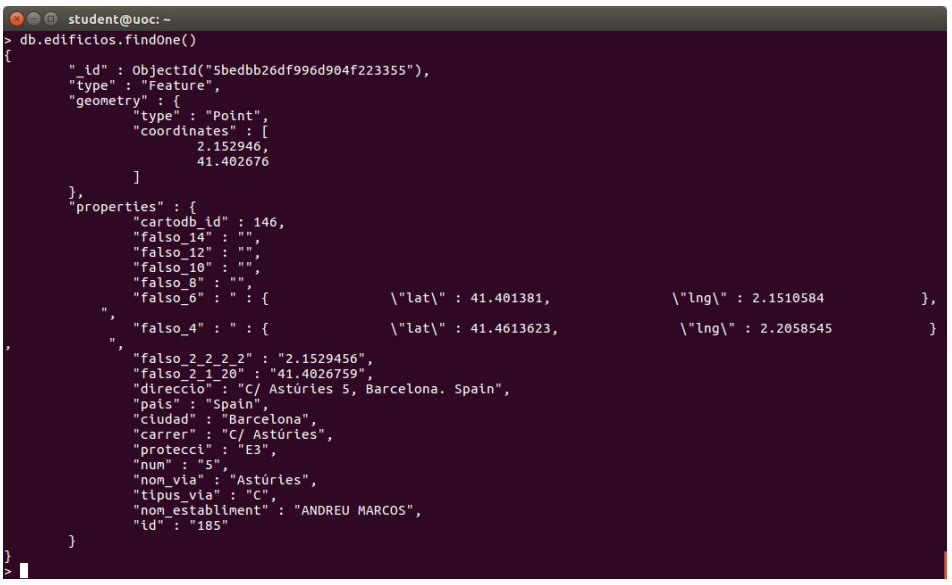


La colección *distritos* está formada por 10 documentos en los que en cada uno se almacena el agregado que proporciona información sobre un distrito. Cada uno contiene un elemento GeoJSON de tipo Multipolygon.



Observamos que en la colección *barrios* almacenamos un único documento que contiene un atributo llamado *features* que es un array de agregados en los que cada uno contiene un elemento GeoJSON de tipo Multipolygon para delimitar el área de un barrio, cuyo nombre se almacena en el atributo *properties.nbarri*

Mediante el cliente *mongo* podemos mostrar un documento de la colección *edificios*:



y mediante la sentencia

```
db.edificios.count()
```

Observamos que la colección edificios está formada 412 documentos. Tal y como se muestra en la imagen anterior cada documento contiene un elemento GeoJSON de tipo Point.

Finalmente se crean los correspondientes índices.

```
db.edificios.createIndex( { geometry : "2dsphere" } )

db.distritos.createIndex( { geometry : "2dsphere" } )

db.barrios.createIndex( { geometry : "2dsphere" } )
```

Consulta de información geoespacial.

MongoDB permite que las aplicaciones realicen los siguientes tipos de consulta entre datos geoespaciales:

- Inclusión
- Intersección
- Proximidad

Obviamente, podemos usar todos los demás operadores además de los geoespaciales. Veamos ahora algunos ejemplos concretos.

Consultas de inclusión

13. Mostar calle, número y nombre del establecimiento correspondiente a todos los edificios del distrito de “Ciutat Vella”.

Propuesta de solución:

Para ello, necesitamos obtener la ubicación de “Ciutat Vella” (MultiPolygon) que almacenamos en la variable *cal* para posteriormente usar en la consulta el comando *$geoWithin* para indicar que la localización del edificio esté dentro del área del distrito.

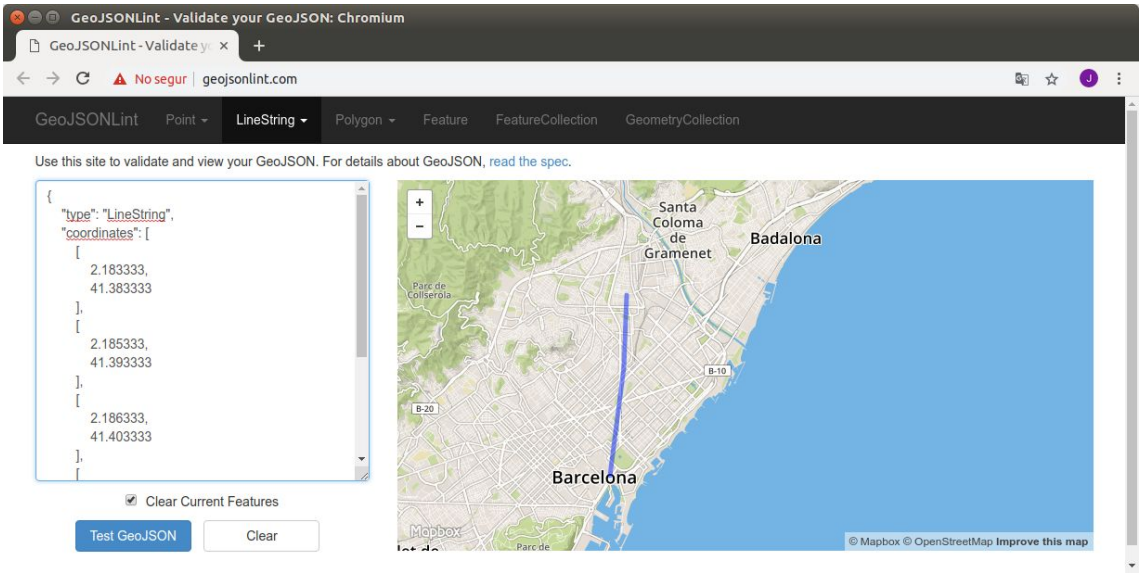
```
var cal = db.distritos.findOne( {"properties.n_distri" : "Ciutat Vella"} );


db.edificios.find( {
                  "geometry": {
                              $geoWithin : { $geometry : cal.geometry}
```

```
                    }
    },
    { "properties.nom_establiment" : 1 ,
      "properties.carrer" : 1,
      "properties.num" : 1,
      "_id": 0
    }
    );
```

```
student@uoc: ~
>
> var cal = db.distritos.findOne(  {"properties.n_distri" : "Ciutat Vella"}  );
>
> db.edificios.find(
...   {
...     "geometry": { $geoWithin : { $geometry : cal.geometry } }
...   },
...   { "properties.nom_establiment" : 1 , "properties.carrer" : 1, "properties.num" : 1, "_id": 0 }
... );
{ "properties" : { "carrer" : "C/ Ample", "num" : "28", "nom_establiment" : "ANTIGA LLAUNERIA GRASSI" } }
{ "properties" : { "carrer" : "C/ Ample", "num" : "28", "nom_establiment" : "ANTIGA LLAUNERIA GRASSI" } }
{ "properties" : { "carrer" : "C/ Escudellers", "num" : "8", "nom_establiment" : "GRILL ROOM" } }
{ "properties" : { "carrer" : "C/ Escudellers", "num" : "14", "nom_establiment" : "LOS CARACOLES" } }
{ "properties" : { "carrer" : "C/ Santa Mònica", "num" : "4", "nom_establiment" : "EL PASTIS" } }
{ "properties" : { "carrer" : "C/ La Rambla", "num" : "44", "nom_establiment" : "FARMÀCIA AGUILAR" } }
{ "properties" : { "carrer" : "C/ Vidre", "num" : "1", "nom_establiment" : "HERBORISTERIA DEL REI" } }
{ "properties" : { "carrer" : "C/ Ample", "num" : "21", "nom_establiment" : "QUEVIURES LA LIONESA" } }
{ "properties" : { "carrer" : "Pl. Reial", "num" : "3", "nom_establiment" : "BAR GLACIAR" } }
{ "properties" : { "carrer" : "C/ Escudellers", "num" : "8", "nom_establiment" : "GRILL ROOM" } }
{ "properties" : { "carrer" : "C/ Escudellers", "num" : "14", "nom_establiment" : "LOS CARACOLES" } }
{ "properties" : { "carrer" : "C/ Santa Mònica", "num" : "4", "nom_establiment" : "EL PASTIS" } }
{ "properties" : { "carrer" : "C/ La Rambla", "num" : "44", "nom_establiment" : "FARMÀCIA AGUILAR" } }
{ "properties" : { "carrer" : "C/ Vidre", "num" : "1", "nom_establiment" : "HERBORISTERIA DEL REI" } }
{ "properties" : { "carrer" : "C/ Ample", "num" : "21", "nom_establiment" : "QUEVIURES LA LIONESA" } }
{ "properties" : { "carrer" : "Pl. Reial", "num" : "3", "nom_establiment" : "BAR GLACIAR" } }
{ "properties" : { "carrer" : "C/ Nou de la Rambla", "num" : "34", "nom_establiment" : "LONDON BAR" } }
{ "properties" : { "carrer" : "C/ Sant Pau", "num" : "65", "nom_establiment" : "BAR MARSELLA" } }
{ "properties" : { "carrer" : "C/ Marqués de Barberá", "num" : "21", "nom_establiment" : "J.TORRENTE TECNO INDUSTRIA
L" } }
{ "properties" : { "carrer" : "C/ Sant Pau", "num" : "21", "nom_establiment" : "LLIBRERIA MILLÀ" } }
Type "it" for more
>
```

Consultas de intersección

14. Queremos conocer el nombre de todos los distritos por los que pasaremos si realizamos la ruta descrita en el diagrama.



Propuesta de solución:

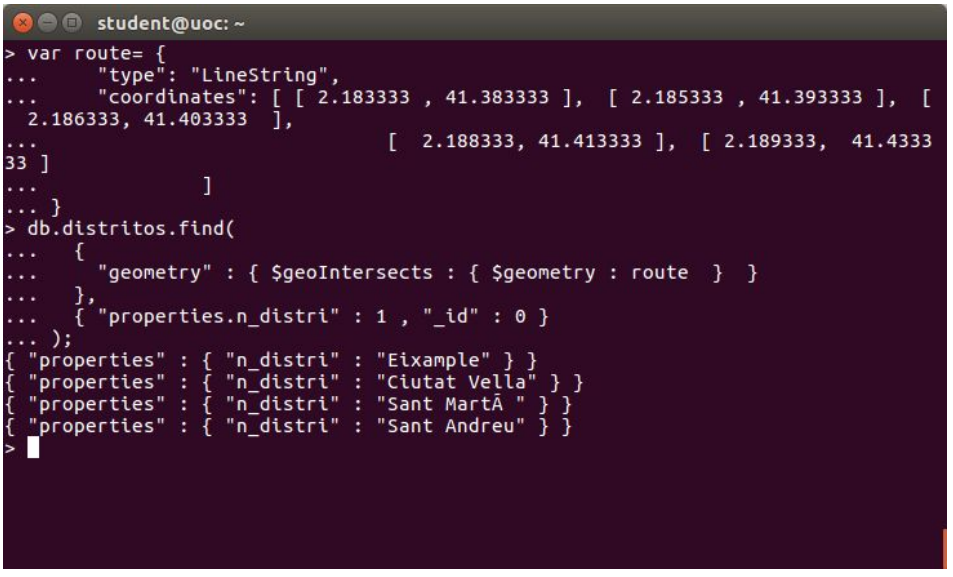
Esta ruta la podemos representar como un elemento GeoJSON de tipo *LineString* que asignamos a la variable *route*.

```
var route= {
    "type": "LineString",
    "coordinates": [ [ 2.183333 , 41.383333 ],
                     [ 2.185333 , 41.393333 ],
                     [  2.186333, 41.403333 ],
                     [  2.188333, 41.413333 ],
                     [ 2.189333,  41.433333 ]
                   ]
}
```

Posteriormente buscamos todos los distritos que tienen coordenadas que se intersectan con la ruta especificada . Esto se hace con la siguiente consulta:

```
db.distritos.find(
            {   "geometry" : { $geoIntersects : { $geometry : route  }
                             }
             },
            { "properties.n_distri" : 1 , "_id" : 0 }
);
```

Obteniendo el siguiente resultado:



Consultas de proximidad

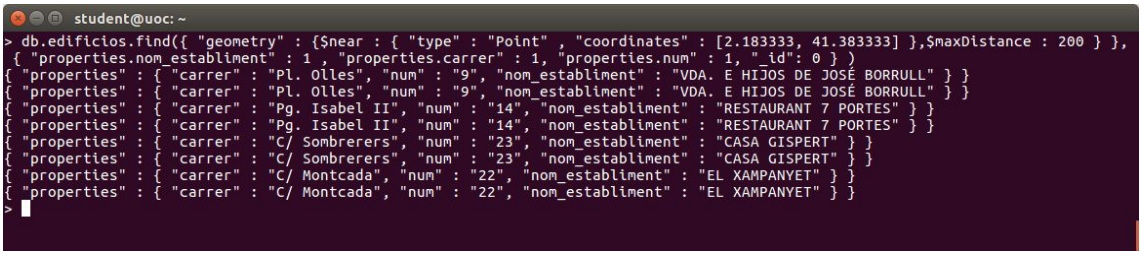
15. Mostar calle, número y nombre del establecimiento correspondiente a todos los edificios que se encuentran a menos de 200 metros del punto descrito anteriormente, en el que nos encontramos.

Propuesta de solución:

Para ello utilizaremos el operador *$near*.

```
db.edificios.find({ "geometry" : {$near : {
                                        "type" : "Point" ,
                                        "coordinates" : [2.183333,
                                                         41.383333]
                                        },
                                  $maxDistance : 200 }
                    },
                    { "properties.nom_establiment" : 1 ,
                      "properties.carrer" : 1,
                      "properties.num" : 1,
                      "_id": 0
                    }
)
```

Obteniendo el siguiente resultado:



Aggregation Pipeline

Se recomienda la lectura de los siguientes documentos:

- https://docs.mongodb.com/manual/core/aggregation-pipeline/
- https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/

16. Realiza la siguiente consulta: Muestra un agregado para cada actor. En el atributo `actor` el nombre del actor, que actuará de identificador, el atributo `movies` que informará del número de películas en las que ha intervenido y el atributo `catalog` formado por un array que incluya el título de las películas en las que ha intervenido tal y como se muestra en el siguiente ejemplo de resultado, ordenando el resultado por el valor de `movies` en orden descendente.

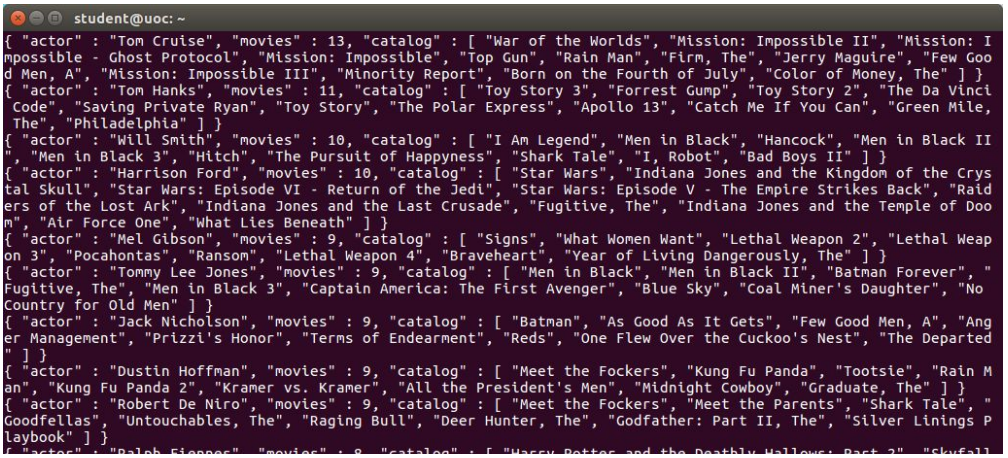
```
{       "actor" : "Jane Fonda",
        "movies" : 7,
        "catalog" : [ "Cat Ballou",
                      "They Shoot Horses, Don't They?",
                      "Klute", "Julia",
                      "California Suite",
                      "Coming Home",
                      "On Golden Pond"
        ]
}
```

Propuesta de solución:

Las etapas necesarias para realizar la operación solicitada serán las siguientes:

- Deconstruir el array *actors* del documento de entrada para generar un documento para cada elemento. Para ello se usa el operador *$unwind*
- Agrupar los documentos por el nombre del actor y dar salida a la siguiente etapa un documento para cada agrupación distinta. Los documentos de salida contendrán un campo _id que contiene la clave. Los documentos de salida contendrán los atributos computados. Se utilizará el operador $push agrega un valor específico a una matriz. Para ello se usa el operador *$group*
- Proyectar en los nuevos documentos los atributos solicitados a la siguiente etapa en la tubería. Para ello se usa el operador *$project*
- Ordenar todos los documentos de entrada según el número de películas en orden descendente.Para ello se usa el operador *$sort*

```
> db.movies.aggregate([
      {"$unwind": "$actors"},
      {"$group":
            {
             "_id": "$actors.name",
             "nmovies": {"$sum": 1},
             "lmovies": {"$push": "$name"}
            }
      },
      {"$project":
            {
             "_id": 0,
             "actor": "$_id",
             "movies": "$nmovies",
             "catalog" : "$lmovies"
            }
      },
      {"$sort": {"movies": -1}}
])
```



17. Realiza la siguiente consulta: Muestra un agregado para cada director. En el atributo `director` el nombre del director, el atributo `movies` que informará del número de películas en las que ha dirigido y el atributo `catalog` formado por un array que incluya el título de las películas que dirigido, ordenando el resultado por el valor de `movies` en orden descendente tal y como se muestra en el siguiente ejemplo de resultado.

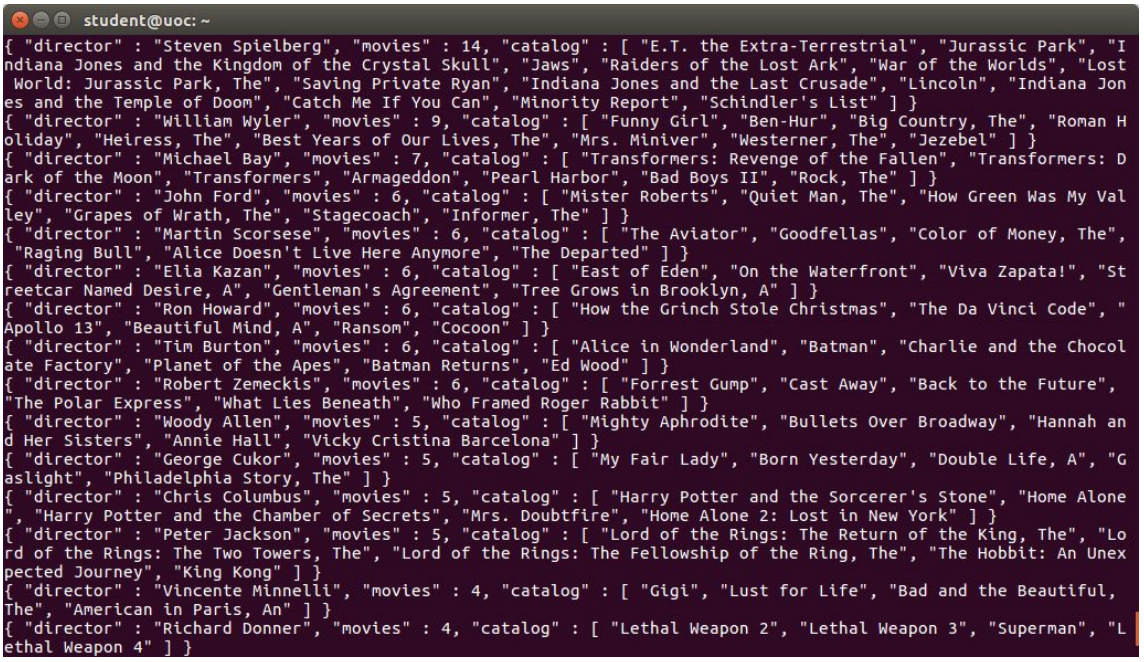
```
{       "director" : "Billy Wilder",
        "movies" : 4,
        "catalog" : [ "Lost Weekend, The",
                      "Stalag 17",
                      "Apartment, The",
                      "Fortune Cookie, The"
        ]
}
```

Propuesta de solución:

Las etapas necesarias para realizar la operación solicitada serán las siguientes:

- Deconstruir el array *directors* del documento de entrada para generar un documento para cada elemento. Para ello se usa el operador *$unwind*
- Agrupar los documentos por el nombre del director y dar salida a la siguiente etapa un documento para cada agrupación distinta. Los documentos de salida contendrán un campo _id que contiene la clave. Los documentos de salida contendrán los atributos computados. Se utilizará el operador $push agrega un valor específico a una matriz. Para ello se usa el operador *$group*
- Proyectar en los nuevos documentos los atributos solicitados a la siguiente etapa en la tubería. Para ello se usa el operador *$project*
- Ordenar todos los documentos de entrada según el número de películas en orden descendente.Para ello se usa el operador *$sort*

```
> db.movies.aggregate([
            {"$unwind": "$directors"},
            {"$group": {
                        "_id": "$directors.name",
                        "nmovies": {"$sum": 1},
                        "lmovies": {"$push": "$name"}
                       }
            },
            {"$project": {"_id": 0,
                        "director": "$_id",
                        "movies": "$nmovies",
                        "catalog" : "$lmovies"
                        }
            },
            {"$sort": {"movies": -1}}
])
```



18. Realiza la siguiente consulta: Muestra un agregado para cada actor. En el atributo `_id` el nombre del actor, el atributo `nmovies` que informará del número de películas en las que ha intervenido, el atributo `lyear` informará del año de la última película, el atributo `fyear` informará del año de la primera película, el atributo `tduration` informará de la suma de la duración de todas las películas en las que ha intervenido y el atributo `lmovies` formado por un array que incluya el título de las películas en las que ha intervenido concatenado con el año de estreno de la película, entre paréntesis, tal y como se muestra en el siguiente ejemplo de resultado, ordenando el resultado por el valor de `nmovies` en orden descendente.

```
{       "_id" : "Tom Cruise",
        "nmovies" : 13,
        "lyear" : 2011,
        "fyear" : 1986,
        "tduration" : 1690,
        "lmovies" : [ "Mission: Impossible - Ghost Protocol (2011)",
                "Mission: Impossible III (2006)",
                "War of the Worlds (2005)",
                "Minority Report (2002)",
                "Mission: Impossible II (2000)",
                "Jerry Maguire (1996)",
                "Mission: Impossible (1996)",
                "Firm, The (1993)",
                "Few Good Men, A (1992)",
                "Born on the Fourth of July (1989)",
                "Rain Man (1988)",
```

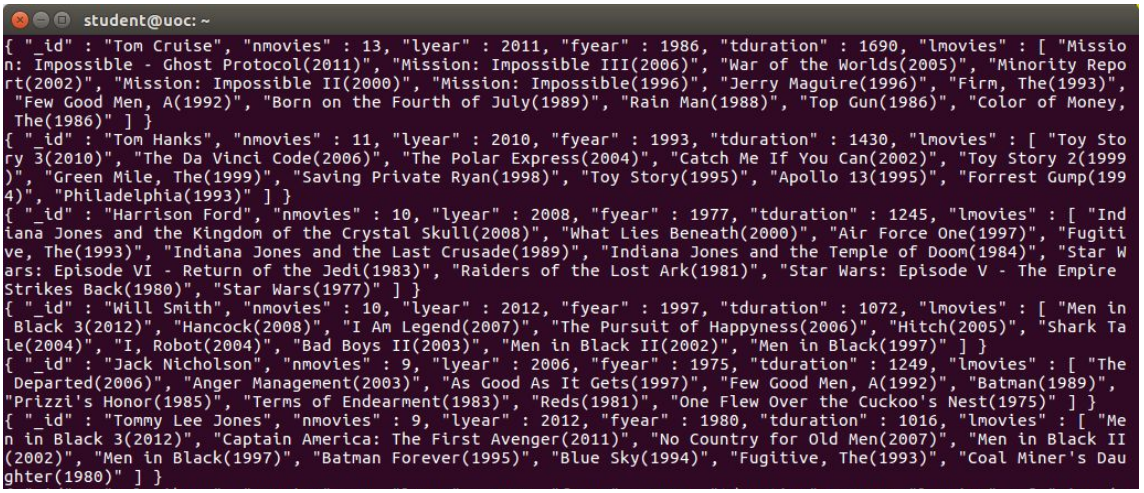
```
                "Color of Money, The (1986)",
                "Top Gun (1986)"
            ]
        }
```

Propuesta de solución:

Las etapas necesarias para realizar la operación solicitada serán las siguientes:

- Ordenar todos los documentos de entrada según el año en que se ha realizado la película en orden descendente.Para ello se usa el operador *$sort*
- Deconstruir el array *actors* del documento de entrada para generar un documento para cada elemento. Para ello se usa el operador *$unwind*
- Agrupar los documentos por el nombre del actor y dar salida a la siguiente etapa un documento para cada agrupación distinta. Los documentos de salida contendrán un campo _id que contiene la clave. Los documentos de salida contendrán los atributos computados. Se utilizará el operador $push agrega un valor específico a una matriz. Para ello se usa el operador *$group*
- Ordenar los documentos de salida por el número de películas calculado en orden descendente.

```
db.movies.aggregate([
        {"$sort": {"year": -1}},
        {"$unwind": "$actors"},
        {"$group": {    "_id": "$actors.name",
                        "nmovies": {"$sum": 1},
                        "lyear": {"$max": "$year"},
                        "fyear": {"$min": "$year"},
                        "tduration": {"$sum": "$runtime"},
                        "lmovies": { "$push": { "$concat": [
                                                    "$name",
                                                    "(",
                                                {"$substr":["$year",0,-1]},
                                                    ")"
                                                ]
                                            }
                                    }
                    }
        },
        {"$sort": { "nmovies": -1} }
])
```



19. Crear una colección con nombre `movies.report01` donde cada documento indique para un actor específico el número de películas en las que ha participado, junto a la duración de su película más larga, la duración de su película más corta, la media de duración de todas sus películas y una lista de las películas en las que ha participado (donde se debe mostrar el título, año de publicación y duración). La colección resultante debe estar ordenada en orden descendente por aquellos actores que han participado en más películas, y el listado de películas para cada actor debe estar ordenado cronológicamente, y en caso de empate, alfabéticamente por título. A continuación se muestra un ejemplo de documento de la colección deseada:

```
{
        "_id" : ObjectId("545691e469f50eba84f2fa3f"),
        "actor" : "Tom Cruise",
        "num_movies" : 13,
        "max_rutime" : 154,
        "min_runtime" : 110,
        "avg_runtime" : 130,
        "movies" : [
                {
                  "title" : "Color of Money, The",
                  "year" : 1986,
                  "runtime" : 119
                },
                {
                  "title" : "Top Gun",
                  "year" : 1986,
                  "runtime" : 110
                },
                ...
                {
```

```
                    "title" : "Mission: Impossible - Ghost Protocol",
                    "year" : 2011,
                    "runtime" : 133
                }
        ]
}
```

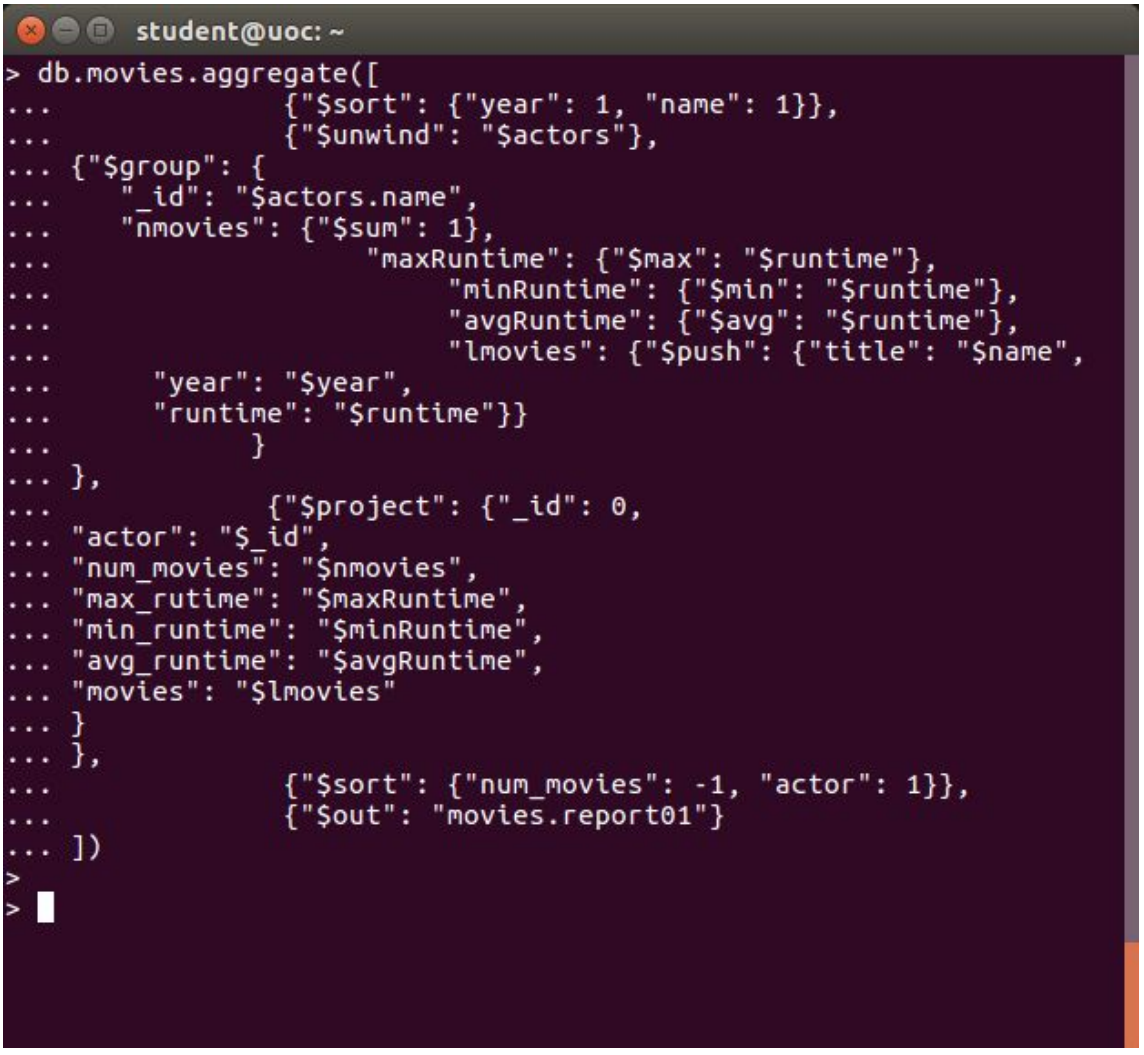
Propuesta de solución:

Las etapas necesarias para realizar la operación solicitada serán las siguientes:

- Ordenar los documentos de la colección movies según el criterio de fecha y nombre. Para ello se usa el operador *$sort*
- Deconstruir el array *actors* del documento de entrada para generar un documento para cada elemento. Para ello se usa el operador *$unwind*
- Agrupar los documentos por el nombre del actor, que actuará como identificador, en esta fase realizamos la cuenta del número de películas, el máximo, mínimo y promedio de la duración de sus películas y construimos un array, mediante el comando $push de los datos de las películas en las que ha intervenido. Para ello se usa el operador *$group* y aquellos comandos específicos para cada computación.
- Proyectar en los nuevos documentos los atributos solicitados a la siguiente etapa en la tubería. Para ello se usa el operador *$project*
- Ordenar todos los documentos de entrada según el número de películas en orden descendente.Para ello se usa el operador *$sort*
- finalmente utilizamos el operador *$out* para almacenar los documentos resultantes en la colección especificada.

```
db.movies.aggregate([
             {"$sort": {"year": 1, "name": 1}},
             {"$unwind": "$actors"},
            {"$group": {
                      "_id": "$actors.name",
                      "nmovies": {"$sum": 1},
                      "maxRuntime": {"$max": "$runtime"},
                     "minRuntime": {"$min": "$runtime"},
                     "avgRuntime": {"$avg": "$runtime"},
                     "lmovies": {"$push": {"title": "$name",
                                           "year": "$year",
                                           "runtime": "$runtime"}}
                        }
            },
            {"$project": {"_id": 0,
                         "actor": "$_id",
                         "num_movies": "$nmovies",
                         "max_rutime": "$maxRuntime",
                         "min_runtime": "$minRuntime",
```

```
                        "avg_runtime": "$avgRuntime",
                        "movies": "$lmovies"
                        }
            },
             {"$sort": {"num_movies": -1, "actor": 1}},
             {"$out": "movies.report01"}
])
```



20. Crear una colección *birth_year* donde cada documento indique para un año específico el número de personas(actores, directores,..) que han nacido ese año, junto al correspondiente listado de personas ordenadas alfabéticamente. A continuación se muestra un ejemplo de documento de la colección deseada:

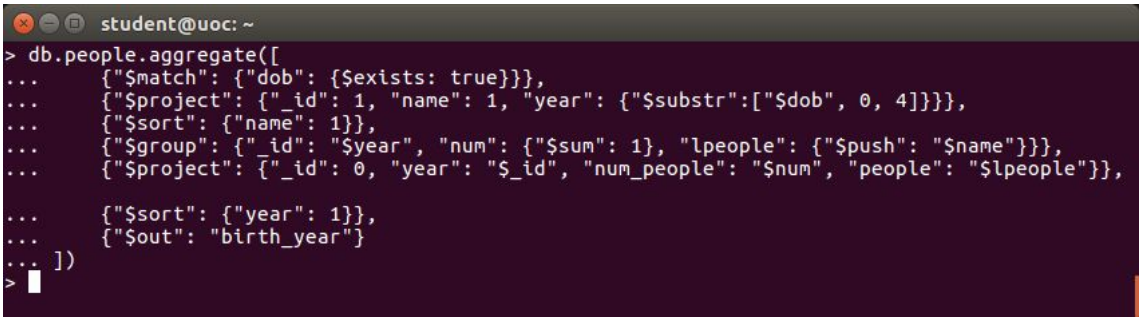
```
{
"year" : "1862",
        "num_people" : 1,
        "people" : [ "Albert Gran" ]
},
{
"year" : "1863",
"num_people" : 2,
"people" : [ "Alison Skipworth", "C. Aubrey Smith" ]
},
{
"year" : "1865",
        "num_people" : 1,
        "people" : [ "Dame May Whitty"]
},
```

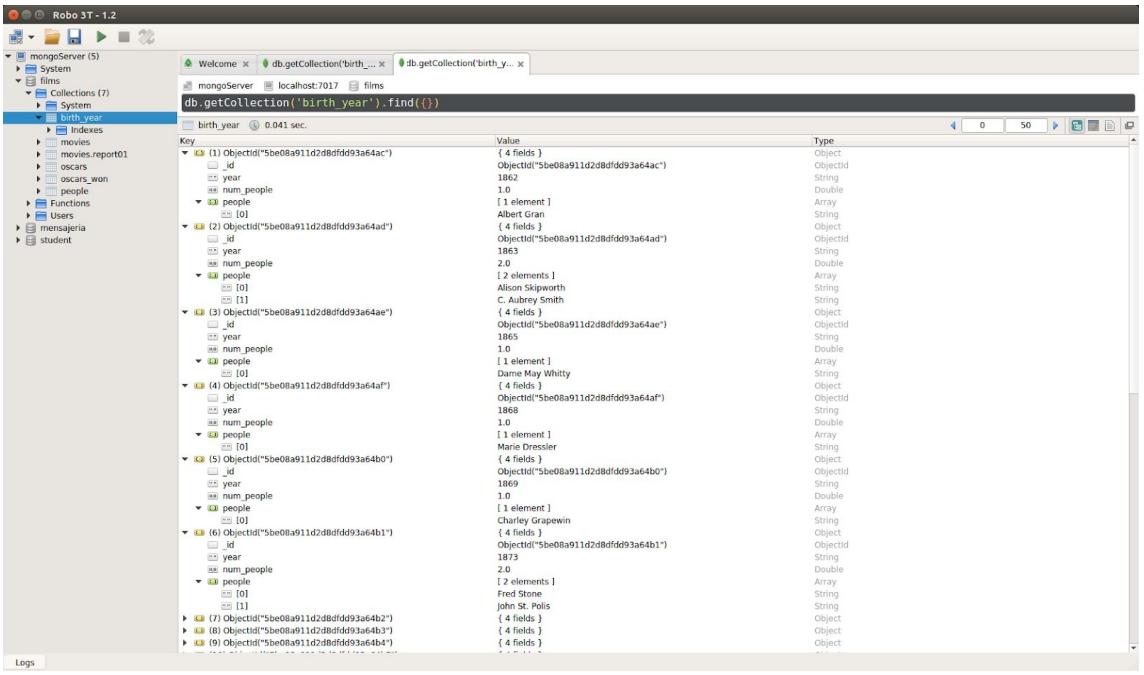
Propuesta de solución:

En este caso empezamos verificando que los documentos de entrada contengan el atributo *dob* correspondiente a la fecha de nacimiento. Ello es una selección con una condición de existencia.

```
db.people.aggregate([
      {"$match": {"dob": {$exists: true}}},
      {"$project": {
                        "_id": 1,
                        "name": 1,
                        "year": {"$substr":["$dob", 0, 4]}
                  }
      },
      {"$sort": {"name": 1}},
      {"$group": {
                  "_id": "$year",
                  "num": {"$sum": 1},
                  "lpeople": {"$push": "$name"}
                  }
      },
      {"$project": {
                        "_id": 0,
                        "year": "$_id",
                        "num_people": "$num",
                        "people": "$lpeople"
                    }
```

```
        },
        {"$sort": {"year": 1}},
        {"$out": "birth_year"}
])
```





21. Generar una colección `winners` que muestre para cada persona el número de oscars conseguidos, y el listado cronológico de estos oscars (indicando año, premio recibido y la correspondiente película) A continuación se muestra un ejemplo de documento de la colección deseada:

```
{       "name" : "Katharine Hepburn",
        "num_oscars" : 4,
        "oscars" : [
        { "type" : "BEST-ACTRESS",
                "year" : 1934,
                "movie" : "Morning Glory" },
        { "type" : "BEST-ACTRESS",
                "year" : 1968,
                "movie" : "Guess Who's Coming to Dinner"   },
        { "type" : "BEST-ACTRESS",
                "year" : 1969,
                "movie" : "The Lion in Winter" },
        { "type" : "BEST-ACTRESS",
                "year" : 1982,
                "movie" : "On Golden Pond"          }
        ]
        },
{       "name" : "John Ford",
        "num_oscars" : 4,
        "oscars" : [
        { "type" : "BEST-DIRECTOR",
                "year" : 1936,
                "movie" : "Informer, The" },
        { "type" : "BEST-DIRECTOR",
                "year" : 1941,
                "movie" : "Grapes of Wrath, The" },
        { "type" : "BEST-DIRECTOR",
                "year" : 1942,
                "movie" : "How Green Was My Valley" },
        { "type" : "BEST-DIRECTOR",
                "year" : 1953,
                "movie" : "Quiet Man, The" }
        ]
  },
```

Propuesta de solución:

En este caso empezamos verificando que los documentos de entrada contengan el atributo *person*. Como en la consulta anterior utilizamos una condición de existencia.

```
db.oscars.aggregate([
        {$match: {"person": {$exists: true}}},
        {$sort: {
                        "year": 1,
```

```
                    "movie.name": 1
                }
        },
        {$group: {
                "_id": "$person.name",
                "noscars": {$sum: 1},
                "loscars":{$push:{
                                "type":"$type",
                                "year": "$year",
                                "movie": "$movie.name"
                                }
                        }
                }
        },
        {$project: {
                    "_id": 0,
                    "name": "$_id",
                    "num_oscars": "$noscars",
                    "oscars": "$loscars"
                    }
        },
         {$sort: {"num_oscars": -1}},
         {"$out": "winners"}
])
```