

Uso de bases de datos NoSQL

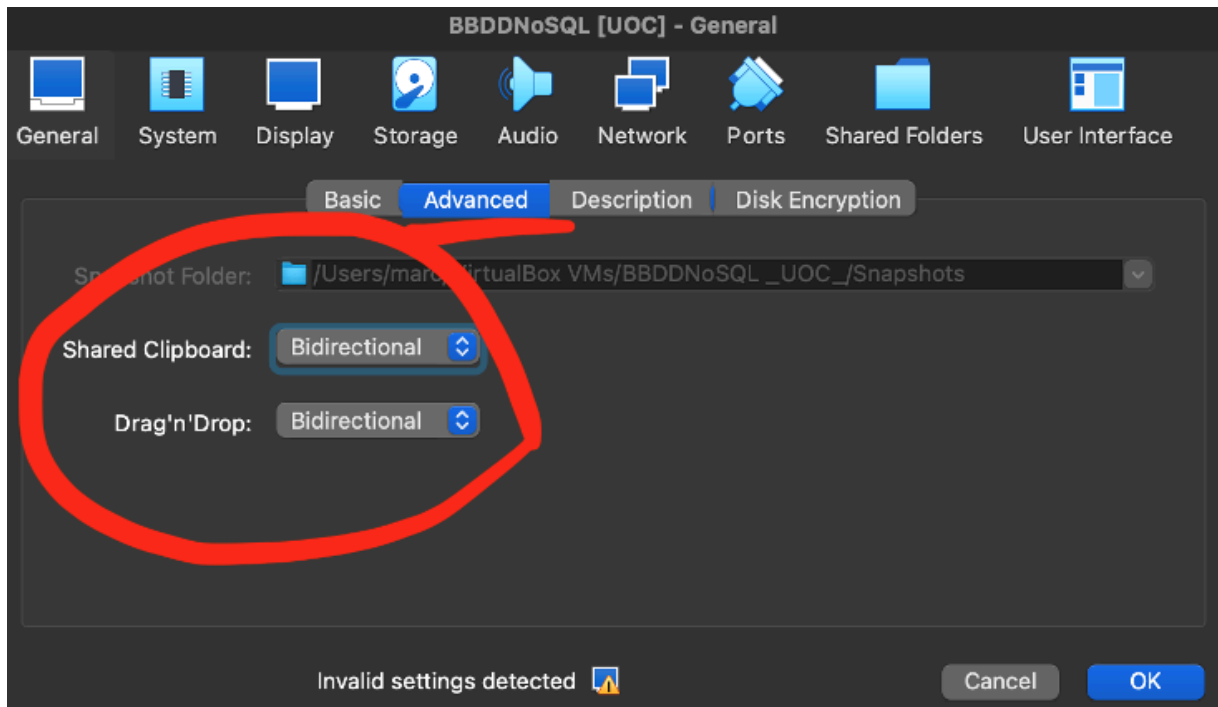
PRA1

Propuesta de solución

Instrucciones

Para poder realizar la práctica, el alumno debe usar la máquina virtual proporcionada que se encuentra en el área de recursos y consultar el manual de usuario proporcionado:

- Máquina virtual Linux Mint suministrada.
- Documento “Máquina virtual Linux Mint (Manual)”
- Importante: Para trabajar de forma óptima, se recomienda activar la opción de compartir el portapapeles (para poder copiar y pegar) y arrastrar archivos desde la máquina local a la máquina virtual. Para ello, una vez iniciada la MV, debéis ir al menú **Machine** → **Settings**, y activar las opciones de *drag and drop* y de *compartir el portapapeles*, tal como se muestra:



Ejercicio 1: Neo4j (30%)

Considera el documento que se encuentra en los materiales del curso “*Diseño de una base de datos para analizar la actividad de usuarios en Twitter*” que describe la implementación de una base de datos en Neo4j. También se necesitará la máquina virtual de LinuxMint (que encontraréis en los recursos del aula) que contiene una instalación de Neo4j que tiene cargada la base de datos descrita anteriormente. El manual para utilizar e instalar la máquina virtual también está en los recursos del aula.

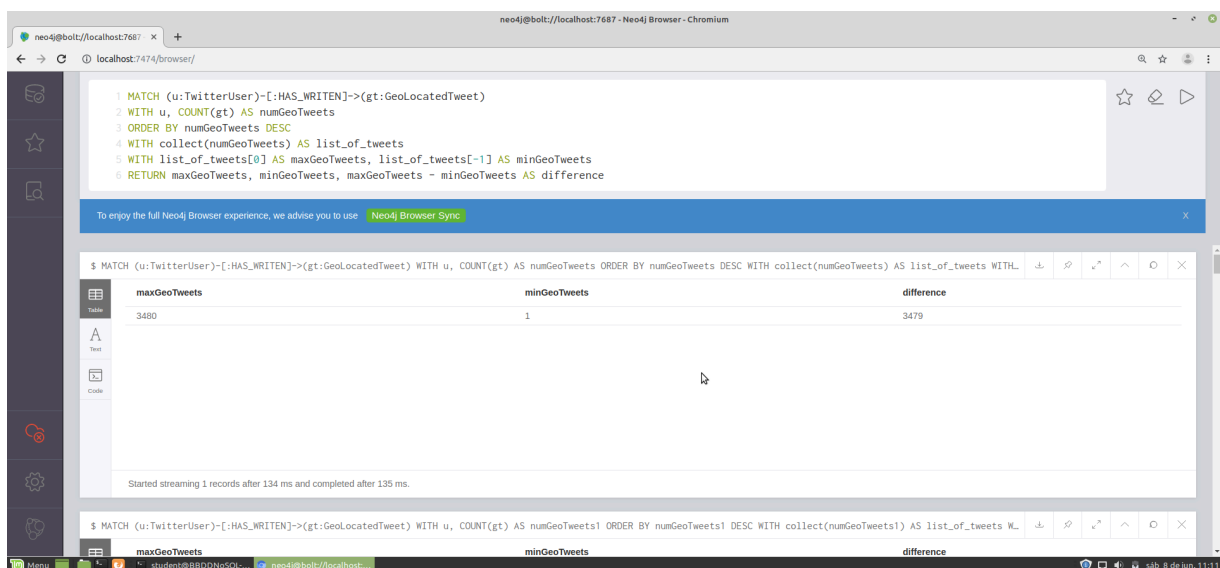
Se pide proporcionar las siguientes consultas en Cypher y una captura de pantalla con los resultados que se obtienen para las siguientes operaciones:

Consulta 1 (20%)

Considera el usuario que más tweets geolocalizados ha enviado y el que menos tweets geolocalizados ha enviado. Muestra la diferencia de tweets enviados.

El código es el siguiente:

```
MATCH (u:TwitterUser)-[:HAS_WRITEN]->(gt:GeoLocatedTweet)
WITH u, COUNT(gt) AS numGeoTweets
ORDER BY numGeoTweets DESC
WITH collect(numGeoTweets) AS list_of_tweets
WITH list_of_tweets[0] AS maxGeoTweets, list_of_tweets[-1] AS minGeoTweets
RETURN maxGeoTweets, minGeoTweets, maxGeoTweets - minGeoTweets AS difference
```



The screenshot shows the Neo4j Browser interface with the following Cypher query and results:

```
$ MATCH (u:TwitterUser)-[:HAS_WRITEN]->(gt:GeoLocatedTweet) WITH u, COUNT(gt) AS numGeoTweets ORDER BY numGeoTweets DESC WITH collect(numGeoTweets) AS list_of_tweets WITH list_of_tweets[0] AS maxGeoTweets, list_of_tweets[-1] AS minGeoTweets RETURN maxGeoTweets, minGeoTweets, maxGeoTweets - minGeoTweets AS difference
```

maxGeoTweets	minGeoTweets	difference
3480	1	3479

Started streaming 1 records after 134 ms and completed after 135 ms.

Consulta 2 (20%)

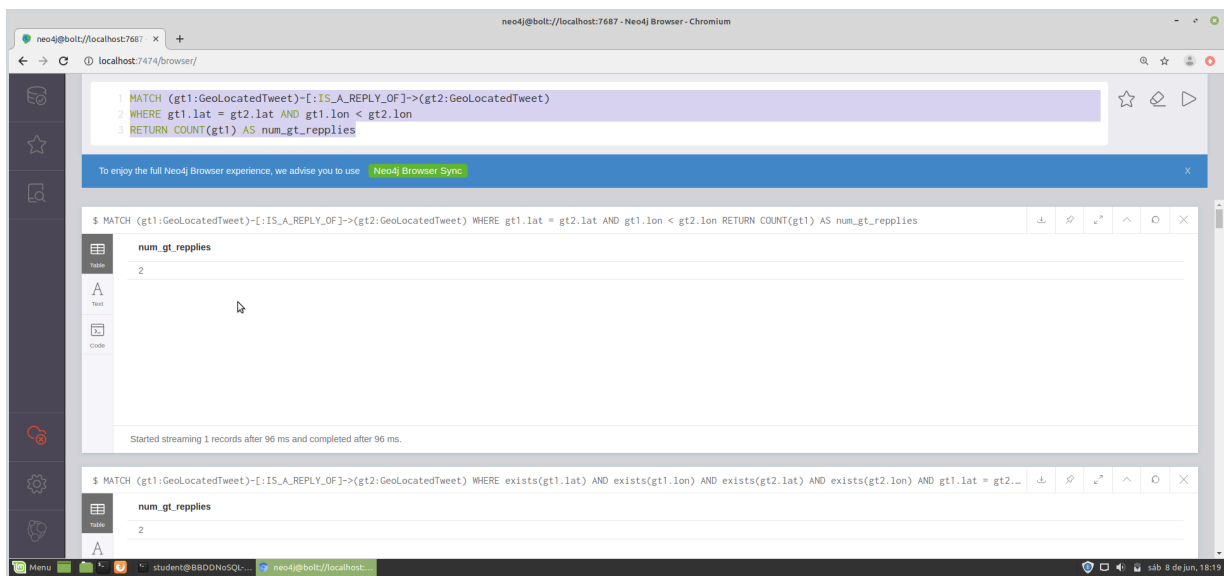
Encontrar cuántos tweets geolocalizados que son réplicas a otros tweets geolocalizados y tal que los tweets que son replicas cumplen que el atributo latitud del mismo coincide con la latitud del tweet al que replica y su longitud es menor que la longitud del tweet al que replica.

Consulta complementaria para saber los nombres de los atributos de GeoLocatedTweet:

```
MATCH (gt:GeoLocatedTweet)
RETURN keys(gt) AS attributes
```

Los atributos de longitud y latitud se llaman “lon” y “lat” respectivamente. Por lo tanto el código es el siguiente:

```
MATCH (gt1:GeoLocatedTweet)-[:IS_A_REPLY_OF]->(gt2:GeoLocatedTweet)
WHERE gt1.lat = gt2.lat AND gt1.lon < gt2.lon
RETURN COUNT(gt1) AS num_gt_replies
```



Consulta 3 (20%)

Considera el usuario relevante “UOC Estudiante”, y obtén a cuántos usuarios relevantes sigue este usuario y tal que esos usuarios le siguen también a él.

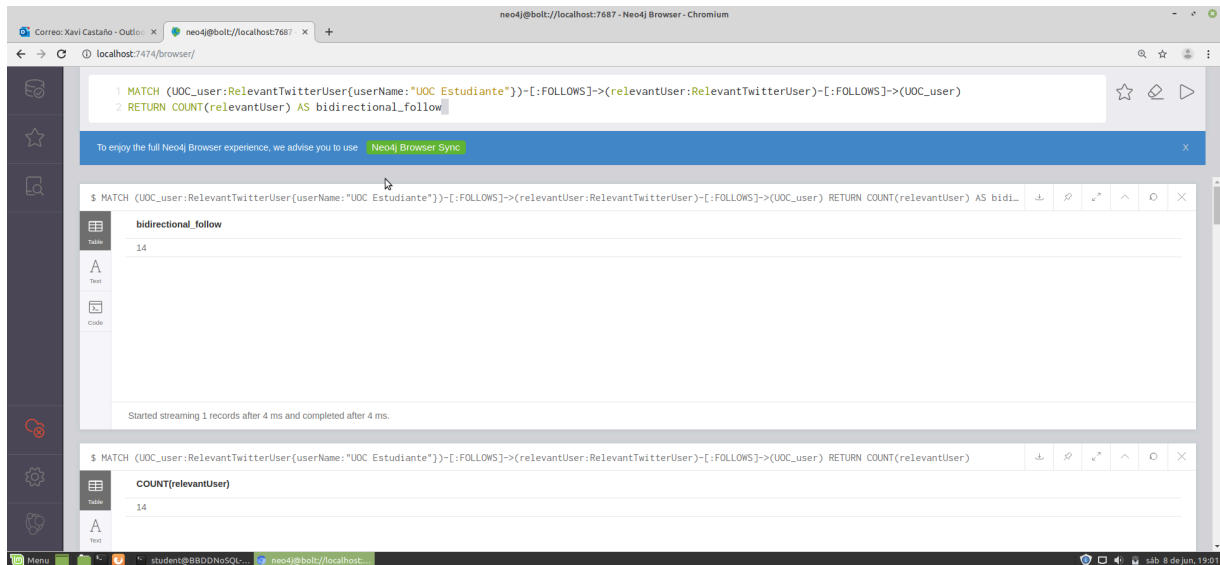
Consulta de apoyo para saber nombre del atributo que buscamos:

```
MATCH (user:RelevantTwitterUser)
RETURN keys(user)
```

Podemos ver que el atributo que el nombre figura como “userName”:

```
MATCH (UOC_user: RelevantTwitterUser{userName: "UOC Estudiante"}) -
[:FOLLOWS] -> (relevantUser: RelevantTwitterUser) - [:FOLLOWS] ->
(UOC_user)
```

```
RETURN COUNT (relevantUser) AS bidirectional_follow
```

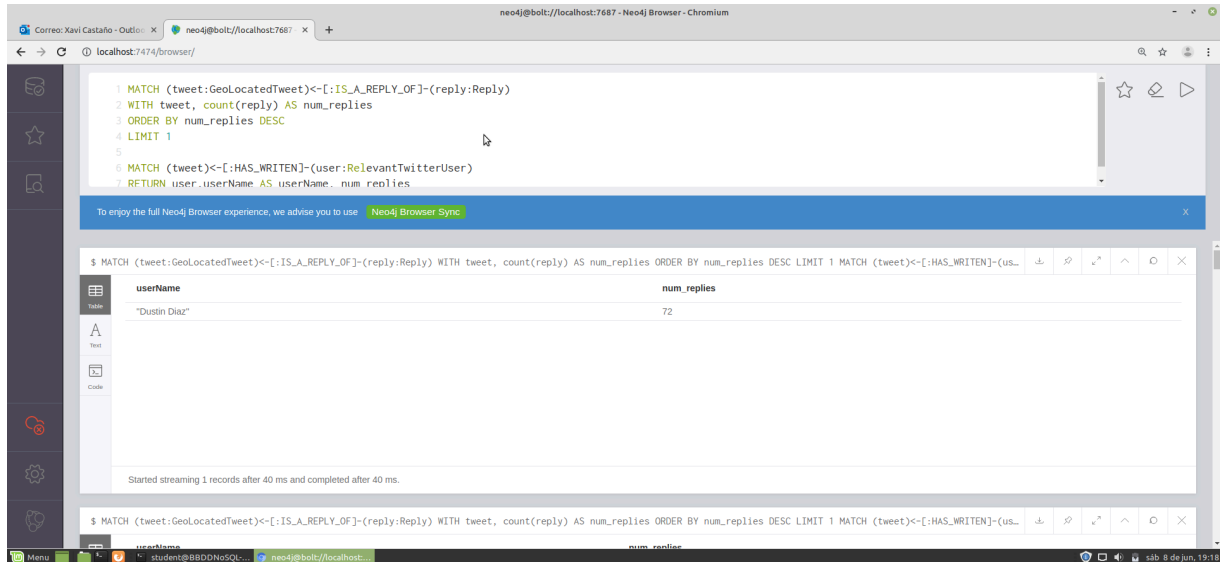


Consulta 4 (20%)

Considera el usuario relevante que ha escrito el tweet geolocalizado que más réplicas ha obtenido. Muestra el nombre del usuario y el número de réplicas recibidas.

```
MATCH (tweet: GeoLocatedTweet) <-[:IS_A_REPLY_OF]-(reply: Reply)
WITH tweet, COUNT(reply) AS num_replies
ORDER BY num_replies DESC
LIMIT 1
```

```
MATCH (tweet) <-[:HAS_WRITTEN]-(user: RelevantTwitterUser)
RETURN user.userName AS userName, num_replies
```

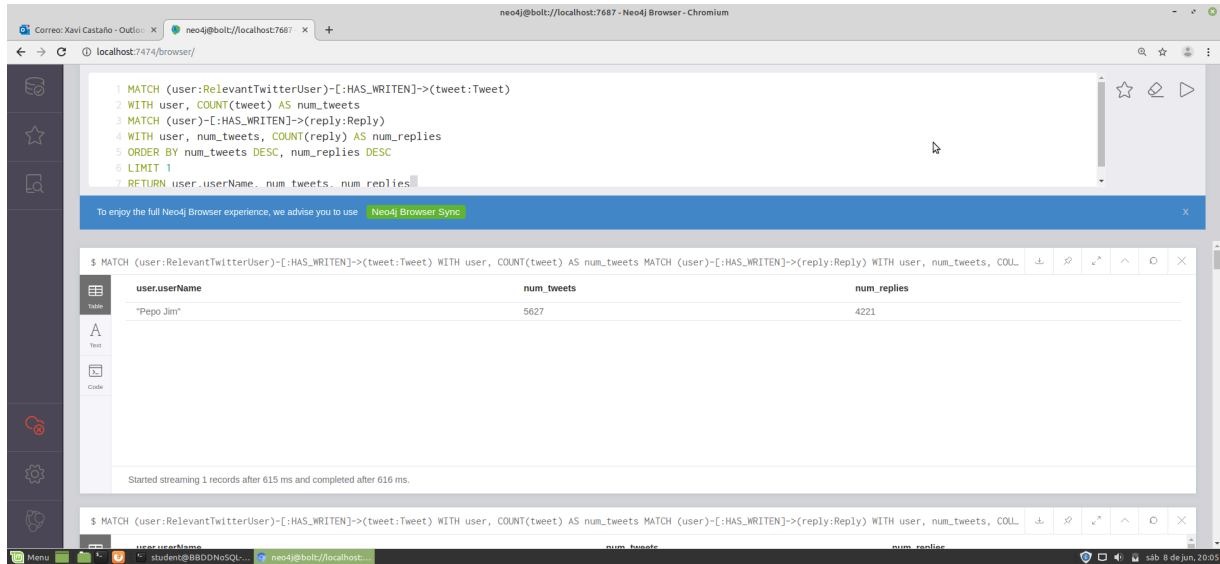


Consulta 5 (20%)

Encuentra el usuario relevante que más tweets ha enviado y ha replicado. Muestra el nombre del usuario, el número de tweets que ha enviado y el número de tweets que ha replicado.

```
MATCH (user:RelevantTwitterUser)-[:HAS_WRITTEN]->(tweet:Tweet)
WITH user, COUNT(tweet) AS num_tweets
MATCH (user)-[:HAS_WRITTEN]->(reply:Reply)
WITH user, num_tweets, COUNT(reply) AS num_replies
ORDER BY num_tweets DESC, num_replies DESC
LIMIT 1
RETURN user.userName, num_tweets, num_replies
```

*Esto solo lo podemos encontrar porque efectivamente existe un usuario que tiene el mayor número de tweets a la vez que el mayor número de replies, de hecho se puede comprobar en la imagen cómo ocurre esto por azar. Pero podría pasar que el usuario relevante con mayor número de tweets no fuese aquel que tiene el mayor número de replies. De todas maneras, si efectivamente existe, podremos encontrarlo en esta consulta Cypher.



The screenshot shows the Neo4j Browser interface in a web browser. The top bar indicates the user is logged in as 'neo4j@bolt://localhost:7687'. The main area displays a Cypher query and its results.

Cypher Query:

```
1 MATCH (user:RelevantTwitterUser)-[:HAS_WRITEN]->(tweet:Tweet)
2 WITH user, COUNT(tweet) AS num_tweets
3 MATCH (user)-[:HAS_WRITEN]->(reply:Reply)
4 WITH user, num_tweets, COUNT(reply) AS num_replies
5 ORDER BY num_tweets DESC, num_replies DESC
6 LIMIT 1
7 RETURN user.userName, num_tweets, num_replies
```

Results Table:

user.userName	num_tweets	num_replies
"Pepo Jim"	5627	4221

Below the table, a status message reads: "Started streaming 1 records after 615 ms and completed after 616 ms."

The bottom of the interface shows the same Cypher query repeated, along with a status bar at the very bottom indicating the date and time: "sáb 8 de jun, 2024".

Ejercicio 2: Neo4j (35%)

Contexto: Los datos sobre los que se realizará el ejercicio corresponden a la historia literaria desarrollada en la saga **Juego de Tronos**. Para su desarrollo no es necesario conocer dicha saga, ya que el objetivo es únicamente el de estudiar un conjunto de datos con múltiples relaciones y ser capaces de extraer cierta información a partir de ellos.

Fuentes de datos y agradecimientos: se debe agradecer y reconocer a los creadores de los siguientes repositorios su disposición a compartir su excelente trabajo:

<https://github.com/neo4j-examples/game-of-thrones>

<https://github.com/joakimskoog/AnApiOfIceAndFire>

Para realizar este ejercicio se debería utilizar **la versión de Neo4j ya instalada** en la máquina virtual. Por favor, ignore la base de datos Twitter Neo4j y recuerde que el usuario / contraseña para acceder a Neo4j son: **neo4j / uoc**

La propuesta de solución a todos los apartados debe proporcionar las consultas Cypher en formato texto y una captura de pantalla de las consultas Cypher con los resultados obtenidos.

2.1 Carga de datos (no puntúa)

A continuación se presentan las instrucciones para añadir el grafo de interés en Neo4j.

La versión de Neo4j utilizada (*community edition*) permite sólo una base de datos activa. Para no tener problemas con el ejercicio 1, se recomienda no eliminar la base de datos existente (de Twitter).

A continuación cargaremos los nodos y relaciones de la nueva base de datos. Para ello, deberéis descargar el documento *goT_data.cypher*, que contiene las sentencias de Cypher para crear los datos de *Games of Thrones*.

Una vez descargado dicho fichero en vuestra máquina virtual, se recomienda abrir una ventana de terminal y ejecutar el siguiente comando en el mismo directorio donde está el fichero descargado:

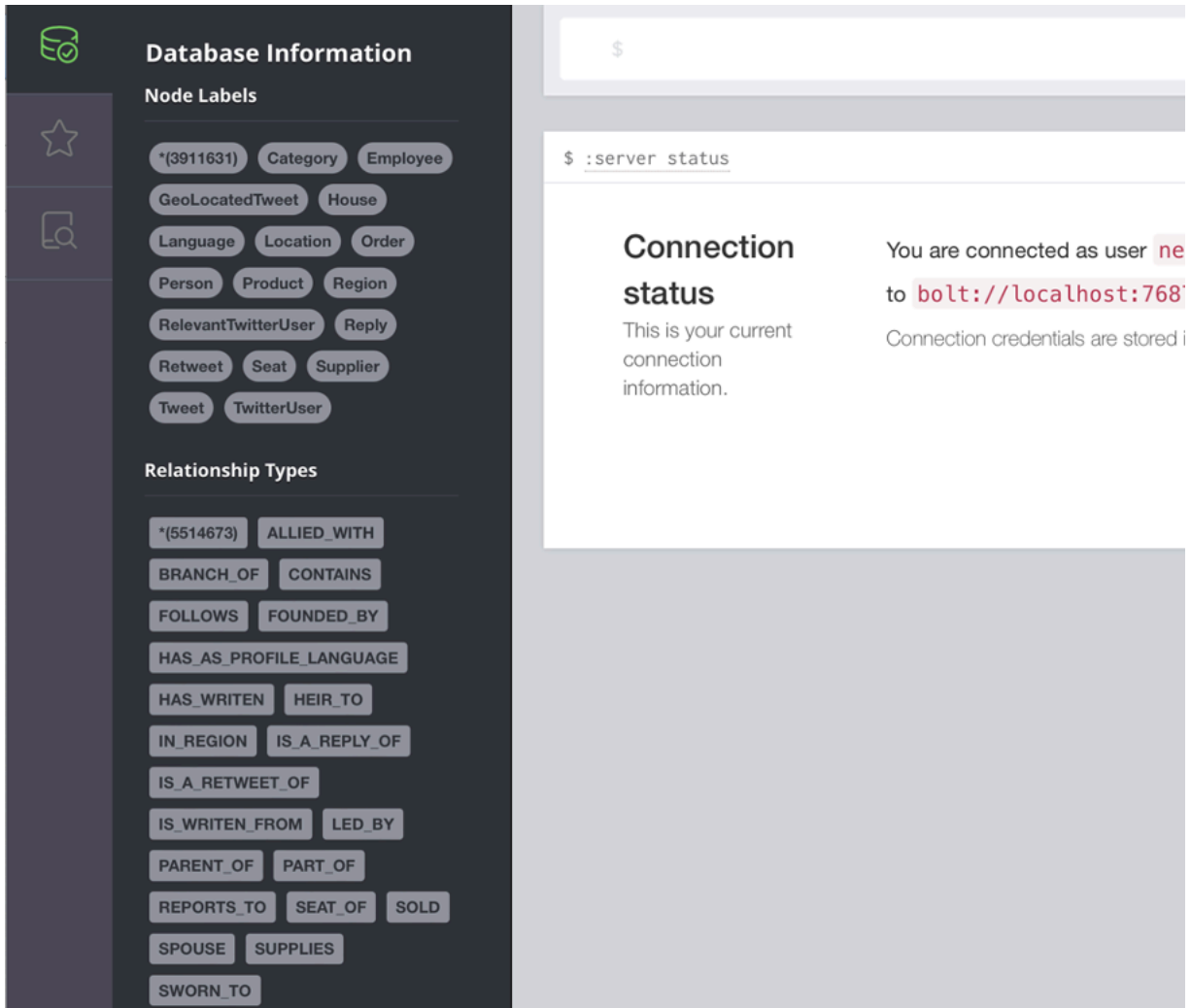
```
cat goT_data.cypher | cypher-shell -u neo4j -p uoc
```

Este comando enviará todos los comandos en Cypher del fichero al shell de cypher para que estos puedan crearse en la base de datos. Nota: para que funcione correctamente, deberéis tener la base de datos corriendo.

```
student@BBDDNoSQL-VM:~/Descargas$ cat goT_data.cypher |cypher-shell -u neo4j -p
uoc
student@BBDDNoSQL-VM:~/Descargas$
```

Una vez cargados los datos, os podéis conectar a la base de datos de Neo4j indicando la siguiente URL en el navegador web, después de haber arrancado la base de datos:

localhost:7474



The screenshot shows the Neo4j Browser interface. On the left, there is a sidebar with icons for Database Information, Query, and Relationships. The main panel is titled 'Database Information' and contains two sections: 'Node Labels' and 'Relationship Types'. The 'Node Labels' section lists various labels such as '(3911631)', 'Category', 'Employee', 'GeoLocatedTweet', 'House', 'Language', 'Location', 'Order', 'Person', 'Product', 'Region', 'RelevantTwitterUser', 'Reply', 'Retweet', 'Seat', 'Supplier', 'Tweet', and 'TwitterUser'. The 'Relationship Types' section lists various types such as '(5514673)', 'ALLIED_WITH', 'BRANCH_OF', 'CONTAINS', 'FOLLOWS', 'FOUNDED_BY', 'HAS_AS_PROFILE_LANGUAGE', 'HAS_WRITEN', 'HEIR_TO', 'IN_REGION', 'IS_A_REPLY_OF', 'IS_A_RETWEET_OF', 'IS_WRITEN_FROM', 'LED_BY', 'PARENT_OF', 'PART_OF', 'REPORTS_TO', 'SEAT_OF', 'SOLD', 'SPOUSE', 'SUPPLIES', and 'SWORN_TO'. On the right, there is a 'Connection status' section that indicates the user is connected as 'neo4j' to 'bolt://localhost:7687'.

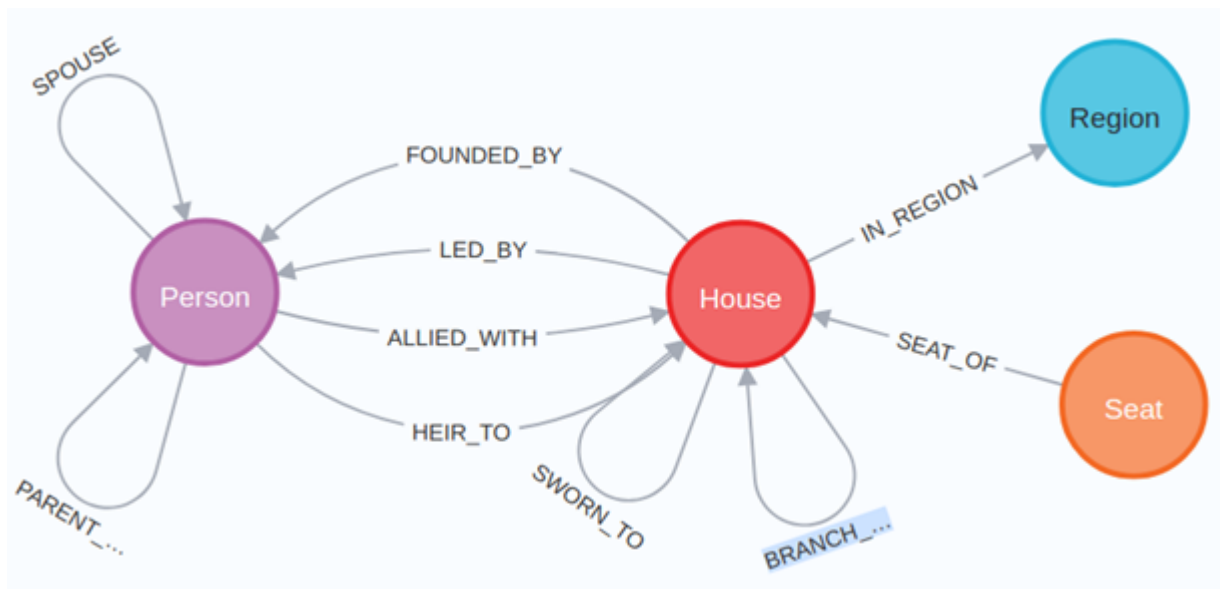
Podréis comprobar que los nodos y relaciones de Game of Thrones se han insertado correctamente.

2.2 Contexto de los datos (no puntúa)

Contexto: los datos pertenecen a un conjunto de casas aristocráticas, incluyendo a sus más destacados miembros así como otros datos de interés. El contexto, dejando tintes fantásticos de lado, evoca al existente en la Europa Medieval, donde el poder se repartía entre las casas reales de cada región, y los señores feudales de diferente rango juraban lealtad a una determinada casa real.

Estos datos representan un conjunto de información histórica, donde algunos de los más destacados personajes o casas ya han desaparecido. Podemos localizar a las Personas más importantes, ya sea por su título, por haber fundado una casa gobernante, por ser herederos del gobierno de la Región o de una zona de su Región. Podremos conocer las alianzas entre Casas, desde qué fortaleza (Seat) se gobiernan, matrimonios y lealtades entre esta aristocracia...

Estudiar el modelo de datos en grafo obtenido: (`CALL db.schema()`) desde la interfaz web disponible en la dirección <http://localhost:7474/>



Breve descripción de los nodos y relaciones actuales:

- House BRANCH_OF House - -> casa que es una rama o escisión de otra casa
- House SWORN_TO House - -> casa que ha jurado lealtad a otra casa

- House IN_REGION Region - -> región, zona geográfica de influencia de la casa
- Seat SEAT_OF House - -> fortaleza donde se ha establecido la casa
- House FOUNDED_BY Person - -> personaje que ha fundado la casa
- House LED_BY Person - -> gobernante de la casa
- Person ALIED_WITH House - -> personaje perteneciente o aliado a una casa
- Person HEIR_TO House - -> personaje que hereda el gobierno de la casa
- Person PARENT_TO Person - -> padres de un personaje que ha gobernado una casa
- Person SPOUSE Person - -> pareja consorte de la persona heredera

2.3 Consultas

Se propone responder a las siguientes consultas una vez realizada la carga de datos, utilizando Cypher y su interfaz web.

Para que una propuesta de solución sea considerada válida debe proporcionar:

- Las consultas Cypher planteadas y los resultados obtenidos, presentados ambos **en formato texto (no serán válidas capturas de pantalla)**.
- Las consultas deben estar justificadas, exponiendo los aspectos claves del diseño que permiten satisfacer los requisitos de la pregunta.

Consulta 1 (10%)

Contar y presentar el número de personajes que aparecen en el libro 7 de esta historia de ficción.

En la siguiente consulta recuperamos la información de todos los nodos de tipo "Person" filtrados mediante la condición de que la "key" o propiedad con el nombre "books" sea igual a 7 y, finalmente, queremos que la información filtrada se nos presente agrupada de tal manera que se cuente el número total de nodos tipo "Person" en una columna llamada numCharacters y todos los nombres de estos personajes agrupados en un array que contiene las cadenas de caracteres que forman los nombres de los personajes, dentro de una columna llamada characterNames.

Consulta cypher:

```
MATCH (p:Person)
WHERE 7 IN p.books
RETURN COUNT(p) AS numCharacters, COLLECT(p.name) AS characterNames
```

Output:

numCharacters	characterNames
71	["Addam Frey", "Tybolt Lannister", "Aegon Blackfyre", "Uthor Underleaf", "Valarr Targaryen", "Aegon I", "Aegon IV", "Aegon V", "Aegor Rivers", "Viserys Plumm", "Aemon Blackfyre", "Aemon Targaryen", "Aerion Targaryen", "Aerys I", "Walder Frey", "Alyn Cockshaw", "Willem Wylde", "Addam", "Ambrose Butterwell", "Arian of Pennytree", "Armond Caswell", "Argrave the Defiant", "Baelor Targaryen", "Beron Stark", "Brynden Rivers", "Buford Bulwer", "Clarence Charlton", "Cosgrove", "Daeron II", "Daeron Targaryen", "Dagon Greyjoy", "Daemon Blackfyre", "Daemon II Blackfyre", "Damon Lannister", "Danelle Lothston", "Ferret", "Eden Risley", "Franklyn Frey", "Galtry the Green", "Haegon Blackfyre", "Glendon Flowers", "Gormon Peake", "Jenny", "Harbert Paege", "Kiera of Tyrosh", "Kirby Pimm", "Kyle", "Lothar", "Lucas Nayland", "Joffrey Caswell", "Mallor", "Melaquin", "Leo Tyrell", "Mortimer Boggs", "Lucas Inchfield", "Ned", "Maekar I", "Matarys Targaryen", "Maynard Plumm", "Pudding", "Rafe", "Otho Bracken", "Roger of Pennytree", "Quentyn Ball", "Rhaegel Targaryen", "Steely Pate", "Tanselle", "Roland Crakehall", "Tommard Heddle", "Unknown", "Will"]

Consulta 2 (10%)

Presentar el nombre de la casa que ostenta un mayor número de casas leales en su historia y el número total de casas que le son o han sido leales.

La siguiente consulta devuelve un registro de todas las ocurrencias de nodos tipo “House” que tienen una relación “SWORN_TO” con otros nodos tipo “House”, es decir, de las casas que son vasallas de otras casas. Acto seguido se realiza un conteo por cada casa a la que se le jura lealtad de las casas que les son leales. Luego se ordena todo por orden descendente de número de casas contadas por casa a la que cualquier casa es leal y se extrae solo el primer ejemplo, es decir, la casa a la que más casas le son leales. Finalmente se pide que se presente este resultado en una tabla con una columna que contenga el nombre de la casa con más casas vasallas, y el conteo de las casas que le son leales llamado “servant_houses” y solo se muestra el primer resultado de la tabla, es decir, la casa con más casas vasallas.

Consulta:

```
MATCH (h:House)-[:SWORN_TO]->(H:House)
WITH H, COUNT(h) AS servant_houses
ORDER BY servant_houses DESC
LIMIT 1
RETURN H.name, servant_houses
```

Output:

H.name	servant_houses
"House Tyrell of Highgarden"	59

Consulta 3 (15%)

Presentar el nombre y las características del escudo de armas de aquellas casas de menor rango (aquellas a las que ninguna otra casa les ha jurado lealtad), que incorporen en la descripción de dicho escudo un león (lion).

Se recuperan todos los nodos del tipo “House” donde la “key” llamada “coatOfArms” contenga dentro de su cadena de texto la cadena de texto “lion”, comparando una versión de “coatOfArms” que contiene tan solo texto en minúsculas por si, por ejemplo, “lion” es la primera palabra de por lo tanto la primera letra es mayúscula. Y luego también se filtra por la condición de que dichas casas no tengan casas vasallas. Finalmente, se pide que, de esas casas, se devuelva el nombre y el “coatOfArms” en cuestión, que es una descripción del escudo.

Consulta:

```
MATCH (h:House)
WHERE toLower(h.coatOfArms) CONTAINS "lion" AND NOT
(h) <- [:SWORN_TO] - (:House)
RETURN h.name, h.coatOfArms
```

Output:

h.name	h.coatOfArms
"House Bracken of Stone Hedge"	"A red stallion upon a gold shield, on a brown field(Tenné, on an escutcheon or a stallion rampant gules)"
"House Lannister of Darry"	"Quarterly, a gold lion on a red field; a black plowman on a brown field(Quarterly, first and fourth gules a lion rampant or (for Lannister), second and third tenné a plowman sable (for Darry))"
"House Jast"	"An inverted yellow pall between three yellow lions' heads, on a black field(Sable, a pall reversed between three lions' heads erased or)"
"House Reyne of Castamere "	"A red lion rampant regardant with a forked tail, with gold teeth and claws, on a silver field(Argent, a lion rampant regardant queue-fourché gules, armed and langued or)"
"House Grandison of Grandview"	"A black sleeping lion, on a yellow field(Or, a lion dormant sable)"

"House Musgood"	"Quarterly: A golden pavilion on a blue field, a green laurel crown on a white field(Quarterly, first and fourth, azure, a pavilion or, second and third, argent, a garland of laurel vert)"
"House Vikary"	"Quarterly: a red boar's head on a white field; beneath a gold bend sinister, a silver lion rampant regardant with a forked tail, with gold teeth and claws, on a red field."
"House Manning"	"A red sea lion between two black pallets on white"
"House Parren"	"Per saltire: burgundy and white stripes; a black lion's head on a gold field(Per saltire, the first paly gules and argent, the second or, a lion's head erased sable)"
"House Wydman"	"Five splintered lances, 3, 2, striped blue and white with blue pennons, on a yellow field, beneath a white chief bearing a red castle, a green viper, a black broken wheel, a purple unicorn and a yellow lion."

Consulta 4 (15%)

Necesitamos obtener datos fundacionales sobre las casas. Para ello queremos presentar agrupados, por año de creación, los nombres de las casas fundadas en dicho año. Se pide presentar los 2 años con mayor número de casas fundadas no extintas (sín año de extinción), el número de casas fundadas en dicho año que aún perduran y una lista donde se agrupen los nombres de dichas casas.

Se pide recuperar todos los nodos tipo "House" filtrados por la key "diedOut" cuando no contiene valores (la casa no está extinta) y cuando la key "founded" también contiene valores (ya que, de algunas casas muy antiguas, no se sabe el año en que se fundaron).

A continuación se reestructuran los datos de tal manera que, para cada año que se guarda en "founded_year", se realice un conteo del número de casas que se fundaron con el nombre "numHouses" y un array (o lista) de cadenas de texto con los nombres de las casas que se fundaron ese año que se almacena en "housesNames". Adicionalmente se ordenan en sentido descendente por numHouses y finalmente se pide que se muestre la información ordenada en una tabla con una columna que contenga founded_year, otra que contenga numHouses, y una última para housesNames. Esta tabla tan solo contendrá la dos primeras filas de esta tabla, es decir, los dos primeros años con el mayor número de casas fundadas, sus nombres, y la cantidad.

Consulta:

```
MATCH (H:House)
WHERE NOT H.diedOut IS NULL AND H.founded IS NOT NULL
WITH H.founded AS founded_year, COUNT(H.name) AS numHouses,
COLLECT(H.name) AS housesNames
ORDER BY numHouses DESC
LIMIT 2
RETURN founded_year, numHouses, housesNames
```

Output:

founded_year	numHouses	housesNames
"Age of Heroes"	3	["House Gardener of Highgarden", "House Durrandon", "House Darklyn of Duskendale"]
"184 AC"	1	["House Blackfyre of King's Landing"]

Consulta 5 (15%)

Presentar las casas del reino que han sufrido más de 3 escisiones. Es indiferente que las casas escindidas se hayan extinguido o no. La consulta debe retornar como resultado el nombre de la casa principal, el número de casas escindidas y, en un solo campo agrupado, los nombres de las casas escindidas, ordenando los resultados descendientemente por número de casas escindidas.

Se pide a la base de datos recuperar todas las instancias de nodos tipo "House" almacenados en "newHouse" que tengan la relación de ser escisiones "Branch_OF" o ramas de otros nodos tipo "House" que llamamos "origHouse". De cada nodo "origHouse" se realiza un conteo de las nuevas casas creadas a partir de la vieja (conteo denominado "newHouses") y una lista de cadenas de texto que contienen cada una el nombre de estas casas nuevas llamada "housesNames" (llamado houseNames, porque contiene solo la key "name" de cada nodo "origHouse"). Esta información se ordena por orden descendente de número del conteo de nuevas casas surgidas de cada casa. Y se devuelve en una tabla en la que la primera columna es "houseNames", la segunda es "newHouses" y la tercera es "housesNames".

Consulta:

```
MATCH (newHouse:House)-[:BRANCH_OF]->(origHouse:House)
WITH origHouse.name AS houseName, COUNT(newHouse) AS newHouses,
COLLECT(newHouse.name) AS housesNames
```

```
ORDER BY newHouses DESC
RETURN houseName, newHouses, housesNames
```

Output:

houseName	newHouses	housesNames
"House Lannister of Casterly Rock"	5	["House Lanny", "House Lantell", "House Lannett", "House Lannister of Lannisport", "House Lannister of Darry"]
"House Harlaw of Harlaw"	4	["House Harlaw of Grey Garden", "House Harlaw of Harlaw Hall", "House Harlaw of the Tower of Glimmering", "House Harlaw of Harridan Hill"]
"House Baratheon of Storm's End"	4	["House Baratheon of King's Landing", "House Wensington", "House Bolling", "House Baratheon of Dragonstone"]
"House Durrandon"	3	["House Wensington", "House Baratheon of Storm's End", "House Bolling"]
"House Goodbrother of Hammerhorn"	3	["House Goodbrother of Corpse Lake", "House Goodbrother of Dowlndelving", "House Goodbrother of Crow Spike Keep"]
"House Stark of Winterfell"	2	["House Karstark of Karhold", "House Greystark of Wolf's Den"]
"House Rowan of Goldengrove"	2	["House Webber of Coldmoat", "House Osgrey of Standfast"]
"House Flint of Widow's Watch"	2	["House Flint of the mountains", "House Flint of Flint's Finger"]

"House Flint of the mountains"	2	["House Flint of Widow's Watch", "House Flint of Flint's Finger"]
"House Tyrell of Highgarden"	1	["House Tyrell of Brightwater Keep"]
"House Frey of the Crossing"	1	["House Frey of Riverrun"]
"House Osgrey of Standfast"	1	["House Osgrey of Leafy Lake"]
"House Targaryen of King's Landing"	1	["House Blackfyre of King's Landing"]
"House Arryn of the Eyrie"	1	["House Arryn of Gulltown"]
"House Dayne of Starfall"	1	["House Dayne of High Hermitage"]
"House Royce of Runestone"	1	["House Royce of the Gates of the Moon"]
"House Fossoway of Cider Hall"	1	["House Fossoway of New Barrel"]
"House Darklyn of Duskendale"	1	["House Darke"]
"House Reyne of Castamere"	1	["House Vikary"]
"House Kenning of Harlaw"	1	["House Kenning of Kayce"]

"House Baelish of the Fingers"	1	["House Baelish of Harrenhal"]
"House Foote"	1	["House Foote of Nightsong"]
"House Casterly of Casterly Rock"	1	["House Lannister of Casterly Rock"]
"House Brune of the Dyre Den"	1	["House Brune of Brownhollow"]
"House Farwynd of Sealskin Point"	1	["House Farwynd of the Lonely Light"]

Consulta 6 (15%)

Presentar a los 6 personajes con mayor número de descendientes que llegaron a ser gobernantes de su casa, incluyendo el dato de cuántos hijos totales tuvieron. La consulta debe incluir el nombre del personaje, presentar su relación de parentesco ("mother"/"father") con sus descendientes, los nombres de estos descendientes (agrupados en un array de strings), el número de descendientes incluidos en dicho array (campo por el que se ordenará el resultado) y el número total de hijos del personaje (no todos tienen por qué haber sido gobernantes, ni siquiera tener derecho a gobernar si son "ilegítimos"). El resultado se presentará en orden descendente por el número de descendientes gobernantes y, en caso de igualdad, en orden descendente por la cantidad de hijos totales.

La siguiente consulta empieza por recuperar de la base de datos el conjunto de nodos tipo "Person" que van a denominar "parent" que tienen una relación con otros nodos del tipo person, que se van a referenciar en la consulta como "child" y, además se pide que de esos nodos se tenga en cuenta también si los nodos recuperados "child" tienen a su vez la relación "HEIR_TO" con nodos tipo "House", estos últimos van a ser referenciados como "h". (Optional match podemos entenderlo como una alternativa al LEFT JOIN en Cypher ya que los nodos iniciales child que no tienen una relación "HEIR_TO" con nodos tipo "House" siguen siendo tenidos en cuenta dentro de la consulta, y no desaparecen). A continuación la agregación se realiza por "parent" y se extrae, por cada uno de ellos, "numChildren", que es un conteo del número de total de hijos, "mother", que es la key que revela si "parent" cumple la condición de ser mujer o no, "childrenNames", que es una lista de cadenas de texto que contienen los nombres de los hijos contados en "numChildren" y, por último

“numHeirChildren”, que es un conteo de todos aquellos children cuyos que efectivamente tienen una relación de “HEIR_TO” con algún nodo “h”, en caso contrario, no son tenidos en cuenta para el conteo de “child” por cada “parent”. Finalmente, la información es devuelta en una tabla cuyas columnas son “parentName”, “numChildren” y “sex” (que será “mother” si la key “isFemale” se da y “father” de lo contrario). Finalmente las filas resultantes de esta tabla se ordenan por “numHeirChildren” en orden descendente (hay que tener en cuenta que, como existe un valor por cada “parent”, existe un valor por cada fila que se puede utilizar para ordenarlas aunque no aparezca tal información en forma de fila) y luego por orden descendente de “numChildren” en caso de empates en “numHeirChildren”. Finalmente, de esta selección solo vamos a coger las primeras 6 filas.

Consulta:

```
MATCH (parent:Person)-[:PARENT_OF]->(child:Person)
OPTIONAL MATCH (child)-[:HEIR_TO]->(h:House)
WITH
parent.name AS parentName,
COUNT(child) AS numChildren,
parent.isFemale AS mother,
COLLECT(child.name) AS childrenNames,
SIZE(COLLECT(CASE WHEN h IS NOT NULL THEN child.name END)) AS
numHeirChildren
RETURN
parentName,
numChildren,
CASE mother WHEN true THEN 'mother' ELSE 'father' END AS sex
ORDER BY numHeirChildren DESC, numChildren DESC
LIMIT 6
```

Output:

parentName	numChildren	sex
"Daenaera Velaryon"	3	"mother"
"Aegon III"	3	"father"
"Alyssa Velaryon"	2	"mother"
"Aegon I"	2	"father"
"Aenys I"	2	"father"
"Cersei Lannister"	2	"mother"

Consulta 7 (20%)

De los resultados de la consulta anterior nos ha llamado la atención el caso de "Robert I Baratheon". Se desea obtener información sobre sus hijos, por lo que necesitamos una consulta que nos presente, por cada uno de sus hijos: el nombre del hijo/hija, su fecha de nacimiento y defunción (si existen), sus títulos y sus alias. Para los hijos/hijas que no tengan títulos, alias o no hayan fallecido, se deberá mostrar en la columnas respectivas los siguientes contenidos: "sin títulos" (para los que no tengan títulos), "sin alias" (para los que no tengan alias y "aún vivo" (para los que aún no han fallecido).

Se recuperan todos los nodos de tipo "Person" que tienen como key "name" la cadena de texto "Robert I Baratheon", a los que se asigna el nombre "Rob", que están relacionados con otros nodos también de tipo "Person" a los que se va a llamar "child" mediante la relación "PARENT_OF" de los "Rob" hacia los "child". A partir de aquí de esa información se devuelve una tabla en la cuál las columnas son keys de "child". La primera es "name", de la key "name", "birth", de la key "born", "death_time" que adopta el valor de la key "died" siempre que este exista, sino adopta el valor "aún vivo", "title", que adopta el valor de la key "titles" siempre que este exista, sino adopta el valor "sin títulos" y, por último, "aliases" que adopta el valor de la key "aliases" siempre que este exista, sino adopta el valor "sin alias".

Consulta:

```
MATCH (Rob:Person{name:"Robert I Baratheon"})-[:PARENT_OF]->(child:Person)
RETURN
child.name AS name,
child.born AS birth,
CASE WHEN child.died IS NOT NULL THEN child.died ELSE "aún vivo"
END AS death_time,
CASE WHEN child.titles IS NOT NULL THEN child.titles ELSE "sin
títulos" END AS title,
CASE WHEN child.aliases IS NOT NULL THEN child.aliases ELSE "sin
alias" END AS aliases
```

Output:

name	birth	death_time	title	aliases
"Tommen Baratheon "	"291 AC"	"aún vivo"	["King of the Andals, the Rhoynar and the First Men", "Lord of	["The Boy King"]

			the Seven Kingdoms"]	
"Joffrey Baratheon "	"286 AC, at King's Landin g"	"300 AC, at Red Keep, King's Landing"	["King of the Andals, the Rhoynar and the First Men", "Lord of the Seven Kingdoms", "Protector of the Realm"]	["Joffrey the Illborn", "The Young Usurper", "Aerys the Third", "Joffrey-called-Barathe on"]

Por último, agradecer a Joakim Skoog y a Michael Hunger su aportación sobre la base de datos que hemos utilizado en este ejercicio.

Ejercicio 3: MongoDB (35%)

Para crear la base de datos necesaria para realizar todos los ejercicios de MongoDB, deberá seguir las instrucciones detalladas en el apartado MongoDB disponible en el manual de la máquina virtual.

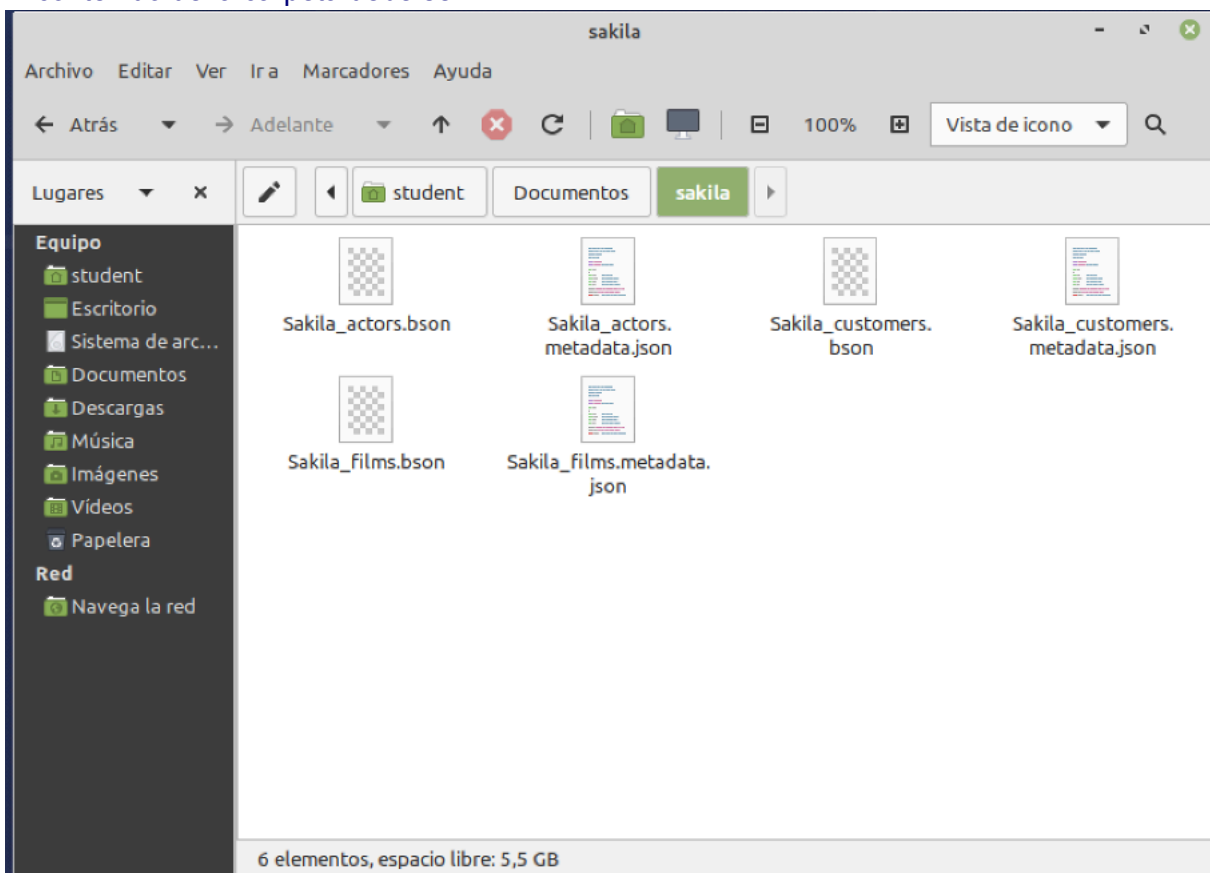
Carga de datos

En primer lugar, debéis descargar la carpeta sakila proporcionada junto con este enunciado.

Para cargar los datos, moved la carpeta sakila (en minúsculas) dentro de la carpeta Documentos. Dentro de la carpeta creada (sakila) copiar y pegar los archivos sin comprimir proporcionados junto con el enunciado:

- Sakila_actors.bson
- Sakila_actors.metadata.json
- Sakila_customers.bson
- Sakila_customers.metadata.json
- Sakila_films.bson
- Sakila_films.metadata.json

El contenido de la carpeta debe ser:



Después de haber iniciado el servicio de MongoDB (haciendo doble clic en el icono *Start MongoDB* , tal y como se indica en el manual), abra un terminal de pedidos desde la barra de tareas del sistema y ejecute el siguiente comando:

Ya podemos conectarnos a mongoDB y comprobar que los datos se han cargado correctamente. Desde un terminal de pedidos podemos ejecutar:

mongo

El command prompt cambiará a:

```
student@BBDDNoSQL-VM:~$ mongo
MongoDB shell version v3.6.3
connecting to: mongoddb://127.0.0.1:27017
MongoDB server version: 3.6.3
Server has startup warnings:
2023-05-26T17:16:22.560+0200 I STORAGE [initandlisten]
2023-05-26T17:16:22.560+0200 I STORAGE [initandlisten] ** WARNING: Using the XFS
filesystem is strongly recommended with the WiredTiger storage engine
2023-05-26T17:16:22.560+0200 I STORAGE [initandlisten] ** See http://c
ochub.mongodb.org/core/prodnotes-filesystem
2023-05-26T17:16:23.447+0200 I CONTROL [initandlisten]
2023-05-26T17:16:23.447+0200 I CONTROL [initandlisten] ** WARNING: Access control
ol is not enabled for the database.
2023-05-26T17:16:23.447+0200 I CONTROL [initandlisten] ** Read and write
te access to data and configuration is unrestricted.
2023-05-26T17:16:23.447+0200 I CONTROL [initandlisten]
> use sakila
switched to db sakila
> show collections
Sakila_actors
Sakila_customers
Sakila_films
>
```

Y una vez en la interfaz de comandos de mongoDB podemos ejecutar:

use sakila

Que nos situará en la base de datos de Sakila. Para comprobar que todo ha ido bien podemos ejecutar

show collections

Que nos debe retornar:

```
> show collections
Sakila_actors
Sakila_customers
Sakila_films
>
```

Ya estamos en la base de datos acabada de cargar y podemos hacer las consultas.

Recordad que si apagáis la máquina virtual y la volvéis a encender, deberéis repetir los pasos indicados en el manual para iniciar el servicio MongoDB y los pasos para conectarse a la base de datos (incluyendo la instrucción `use sakila`).

Para saber si está en la base de datos correcta, escribiendo el comando `db` sabrá a qué base de datos está trabajando actualmente. Por ejemplo:

```
> db
```

Retorna:

```
Sakila
>
```

Por tanto estamos en la base de datos correcta. Si el pedido devuelve otra base de datos como:

```
> db
test
>
```

Indica que está en la base de datos de test, por lo que no encontrará las colecciones cargadas en la base de datos sakila. Tenga en cuenta que MongoDB es *case sensitive* (sensible a mayúsculas y minúsculas). Por tanto, la base de datos `Sakila` (incorrecta) no es la misma que `sakila` (correcta). Si teclea por error `use Sakila` en lugar de `use sakila` se habrá situado en una base de datos vacía llamada `Sakila`.

Por último, **recuerde que después de reiniciar no es necesario repetir los pasos para cargar la base de datos para que los datos ya se guardan en la máquina virtual. Recuerde también que al volver a realizar la carga perderá los cambios realizados.** En otras palabras, si vuelve a ejecutar el comando `mongorestore` la base de datos se volverá a restaurar y perderá todo el trabajo realizado.

Consultas

Los datos que hay cargados son una base de datos ficticia de un videoclub como habréis deducido por los nombres de las colecciones. Como sabéis, MongoDB es *schemaless* por lo que la estructura de los documentos que hay en las colecciones puede ser diferente. Para ver cuál es la estructura de los documentos de cada colección tendréis que ir seleccionando

los documentos de las mismas. También podéis acceder al siguiente enlace para conocer en profundidad su estructura: <https://dev.mysql.com/doc/sakila/en/sakila-structure.html>

Consulta de ejemplo

Retorna un actor de la colección Sakila_actors en un formato JSON que facilite la legibilidad del mismo.

SOLUCIÓN DE EJEMPLO:

Ejecutamos:

```
db.Sakila_actors.find().limit(1).pretty();
```

Que retorna:

```
> db.Sakila_actors.find().limit(1).pretty();
{
  "_id" : 1,
  "FirstName" : "PENELOPE",
  "LastName" : "GUINNESS",
  "phone" : "XXX-XXXX-XXX",
  "address" : "XXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}
```

Por favor, adjuntar las respuestas en modo texto como se hace en este ejemplo, no capturas de pantalla. Si no se indica lo contrario, las consultas de este ejercicio retornan una cantidad de datos que cabe perfectamente en la práctica como texto y debe adjuntarse entera. Si los resultados no caben en pantalla, se indicará en el enunciado y podréis segmentar la salida en vuestra respuesta.

3.1 Consulta (15%)

Como no hay una colección con todos los posibles valores de las calificaciones de las películas (Rating) ni una restricción de integridad relacional con los valores posibles, queremos obtener los valores únicos de las calificaciones que hay en la colección Sakila_films. La consulta debe retornar solamente los valores únicos (no toda la lista completa de todas las clasificaciones de todas las películas) y por orden alfabético **descendente**. No debe retornar el _id del documento.

```
> var ratings = db.Sakila_films.distinct("Rating");
> ratings.sort()
[ "G", "NC-17", "PG", "PG-13", "R" ]
```

3.2 Consulta (10%)

Ahora que sabemos los códigos de calificación, queremos obtener el recuento de las películas que no son aptas para menores de 17 años (valor de calificación "NC-17"). La consulta debe retornar solamente la cifra.

```
> db.Sakila_films.count({ Rating: { $ne: "NC-17" } });
790
```

3.3 Consulta (15%)

Queremos saber los 3 actores que han participado en más películas. Se debe mostrar el recuento de los actores con su identificador y después un listado de esos tres actores mostrando el nombre y el apellido. Es posible que este ejercicio no se pueda resolver con una sola consulta, en la primera tendréis que obtener el recuento y en la segunda obtener el nombre y apellido.

Sugerencia: Para agrupar correctamente los actores en la colección de películas se debe usar el campo actorId.

Nota: El listado de los tres actores no es necesario que esté ordenado.

```
> var topActors = db.Sakila_films.aggregate([
...   { $unwind: "$Actors" },
...   { $group: { _id: "$Actors.actorId", count: { $sum: 1 } } },
...   { $sort: { count: -1 } },
...   { $limit: 3 }
... ]).toArray();
> printjson(topActors);
[
  {
    "_id" : 107,
    "count" : 42
  },
  {
    "_id" : 102,
    "count" : 41
  },
  {
    "_id" : 198,
    "count" : 40
  }
]
> var actorIds = topActors.map(actor => actor._id);
> db.Sakila_actors.find(
...   { _id: { $in: actorIds } },
...   { _id: 0, "FirstName": 1, "LastName": 1 }
... ).toArray();
```

```
[
  {
    "FirstName" : "WALTER",
    "LastName" : "TORN"
  },
  {
    "FirstName" : "GINA",
    "LastName" : "DEGENERES"
  },
  {
    "FirstName" : "MARY",
    "LastName" : "KEITEL"
  }
]
```

3.4 Consulta (15%)

Queremos saber las películas en las que han trabajado más actores. La consulta debe retornar un único registro o documento con el nombre de la película y el recuento de actores.

```
> db.Sakila_films.aggregate([
...   { $unwind: "$Actors" },
...   { $group: { _id: "$Title", countActors: { $sum: 1 } } },
...   { $sort: { countActors: -1 } },
...   { $limit: 1 },
...   { $project: { _id: 0, Title: "$_id", countActors: 1 } }
... ]).toArray();
[ { "countActors" : 15, "Title" : "LAMBS CINCINATTI" } ]
```

3.5 Consulta (20%)

Para todos los alquileres de los clientes queremos añadir una valoración numérica de las películas. Los valores irán del 0 al 5 y se utilizará el valor -1 para aquellos clientes que no han hecho ninguna valoración. El nuevo campo a añadir se llamará "Rate" y como es lógico se inicializará a -1.

Se debe adjuntar:

- El comando de actualización junto a su resultado.


```
> db.Sakila_customers.updateMany(
...   {},
...   { $set: { "Rentals.$[].Rate": -1 } }
... );
{ "acknowledged" : true, "matchedCount" : 599, "modifiedCount" : 599 }
```

- Una consulta que muestre la lista de los alquileres con el campo añadido **del décimo cliente de la colección ordenada por apellido ascendentemente**. La consulta de comprobación solamente debe retornar el apellido (campo "Last Name") y la lista de alquileres del cliente en cuestión.

```
> db.Sakila_customers.find().sort({ "Last Name": 1
}).skip(9).limit(1).pretty();

{
  "_id" : 183,
  "Address" : "1839 Szkesfehrvr Parkway",
  "City" : "Luzinia",
  "Country" : "Brazil",
  "District" : "Gois",
  "First Name" : "IDA",
  "Last Name" : "ANDREWS",
  "Phone" : "947468818183",
  "Rentals" : [
    {
      "rentalId" : 1279,
      "staffId" : 2,
      "Film Title" : "CITIZEN SHREK",
      "Payments" : [
        {
          "Payment Id" : 4975,
          "Amount" : 0.9900000095367432,
          "Payment Date" : "2005-06-15
08:13:57.0"
        }
      ]
    }
  ]
}
```

```

    ],
    "Rental Date" : "2005-06-15 08:13:57.0",
    "Return Date" : "2005-06-18 09:36:57.0",
    "filmId" : 153,
    "Rate" : -1
  },
  {
    "Return Date" : "2005-06-29 18:11:59.0",
    "filmId" : 302,
    "rentalId" : 3381,
    "staffId" : 2,
    "Film Title" : "FANTASIA PARK",
    "Payments" : [
      {
        "Payment Date" : "2005-06-21
14:02:59.0",
        "Payment Id" : 4978,
        "Amount" : 5.989999771118164
      }
    ],
    "Rental Date" : "2005-06-21 14:02:59.0",
    "Rate" : -1
  },
  {
    "rentalId" : 3869,
    "staffId" : 1,

```

```

        "Film Title" : "MANCHURIAN CURTAIN",

        "Payments" : [

            {

                "Payment Id" : 4979,

                "Amount" : 2.9900000009536743,

                "Payment Date" : "2005-07-06
17:56:46.0"

            }

        ],

        "Rental Date" : "2005-07-06 17:56:46.0",

        "Return Date" : "2005-07-10 20:44:46.0",

        "filmId" : 557,

        "Rate" : -1

    },

    {

        "Return Date" : "2005-07-09 10:42:24.0",

        "filmId" : 641,

        "rentalId" : 4134,

        "staffId" : 1,

        "Film Title" : "ORANGE GRAPES",

        "Payments" : [

            {

                "Payment Date" : "2005-07-07
08:14:24.0",

                "Payment Id" : 4980,

                "Amount" : 0.99000000095367432

```

```

    }

],

    "Rental Date" : "2005-07-07 08:14:24.0",

    "Rate" : -1

},

{

    "Rental Date" : "2005-07-07 09:04:26.0",

    "Return Date" : "2005-07-08 09:55:26.0",

    "filmId" : 69,

    "rentalId" : 4157,

    "staffId" : 1,

    "Film Title" : "BEVERLY OUTLAW",

    "Payments" : [

        {

            "Amount" : 2.990000009536743,

            "Payment Date" : "2005-07-07

09:04:26.0",

            "Payment Id" : 4981

        }

    ],

    "Rate" : -1

},

{

    "staffId" : 2,

    "Film Title" : "WEDDING APOLLO",

    "Payments" : [

```

```

        {
            "Amount" : 1.9900000095367432,
            "Payment    Date"      :      "2005-07-09
04:56:30.0",

            "Payment Id" : 4982
        }
    ],
    "Rental Date" : "2005-07-09 04:56:30.0",
    "Return Date" : "2005-07-13 23:53:30.0",
    "filmId" : 966,
    "rentalId" : 5069,
    "Rate" : -1
},
{
    "Film Title" : "ANNIE IDENTITY",
    "Payments" : [
        {
            "Amount" : 0.9900000095367432,
            "Payment    Date"      :      "2005-07-10
12:39:28.0",

            "Payment Id" : 4983
        }
    ],
    "Rental Date" : "2005-07-10 12:39:28.0",
    "Return Date" : "2005-07-11 14:08:28.0",
    "filmId" : 26,

```



```

        "rentalId" : 5756,

        "staffId" : 2,

        "Rate" : -1

    },

    {

        "Return Date" : "2005-07-20 06:23:40.0",

        "filmId" : 166,

        "rentalId" : 6569,

        "staffId" : 2,

        "Film Title" : "COLOR PHILADELPHIA",

        "Payments" : [

            {

                "Payment Date" : "2005-07-12

05:47:40.0",

                "Payment Id" : 4985,

                "Amount" : 4.989999771118164

            }

        ],

        "Rental Date" : "2005-07-12 05:47:40.0",

        "Rate" : -1

    },

    {

        "Rental Date" : "2005-07-27 14:51:04.0",

        "Return Date" : "2005-07-31 16:03:04.0",

        "filmId" : 39,

        "rentalId" : 7359,

```

```

        "staffId" : 2,

        "Film Title" : "ARMAGEDDON LOST",

        "Payments" : [

            {

                "Amount" : 0.9900000095367432,

                "Payment    Date"      :      "2005-07-27
14:51:04.0",

                "Payment Id" : 4986

            }

        ],

        "Rate" : -1

    },

    {

        "Film Title" : "ELEPHANT TROJAN",

        "Payments" : [

            {

                "Amount" : 4.989999771118164,

                "Payment    Date"      :      "2005-07-31
11:34:32.0",

                "Payment Id" : 4988

            }

        ],

        "Rental Date" : "2005-07-31 11:34:32.0",

        "Return Date" : "2005-08-04 08:20:32.0",

        "filmId" : 277,

        "rentalId" : 9818,

```

```

        "staffId" : 1,

        "Rate" : -1

    },

    {

        "Film Title" : "CAMPUS REMEMBER",

        "Payments" : [

            {

                "Amount" : 2.990000009536743,

                "Payment Date" : "2005-08-02
18:24:03.0",

                "Payment Id" : 4991

            }

        ],

        "Rental Date" : "2005-08-02 18:24:03.0",

        "Return Date" : "2005-08-06 21:22:03.0",

        "filmId" : 115,

        "rentalId" : 11386,

        "staffId" : 1,

        "Rate" : -1

    },

    {

        "Return Date" : "2005-08-20 08:20:42.0",

        "filmId" : 610,

        "rentalId" : 12451,

        "staffId" : 1,

        "Film Title" : "MUSIC BOONDOCK",

```

```

        "Payments" : [
            {
                "Amount" : 0.9900000095367432,
                "Payment Date" : "2005-08-18
11:04:42.0",
                "Payment Id" : 4992
            }
        ],
        "Rental Date" : "2005-08-18 11:04:42.0",
        "Rate" : -1
    },
    {
        "Return Date" : "2005-08-22 20:23:15.0",
        "filmId" : 6,
        "rentalId" : 12764,
        "staffId" : 1,
        "Film Title" : "AGENT TRUMAN",
        "Payments" : [
            {
                "Amount" : 3.990000009536743,
                "Payment Date" : "2005-08-18
23:14:15.0",
                "Payment Id" : 4993
            }
        ],
        "Rental Date" : "2005-08-18 23:14:15.0",
    }

```

```

        "Rate" : -1
    },
    {
        "filmId" : 356,
        "rentalId" : 13482,
        "staffId" : 1,
        "Film Title" : "GIANT TROOPERS",
        "Payments" : [
            {
                "Amount" : 2.9900000009536743,
                "Payment Date" : "2005-08-20
01:14:30.0",
                "Payment Id" : 4995
            }
        ],
        "Rental Date" : "2005-08-20 01:14:30.0",
        "Return Date" : "2005-08-24 04:57:30.0",
        "Rate" : -1
    },
    {
        "Payments" : [
            {
                "Amount" : 0.99000000095367432,
                "Payment Date" : "2005-05-27
10:12:00.0",
                "Payment Id" : 4974
            }
        ]
    }

```

```

    }

  ],

  "Rental Date" : "2005-05-27 10:12:00.0",

  "Return Date" : "2005-05-31 15:03:00.0",

  "filmId" : 949,

  "rentalId" : 382,

  "staffId" : 1,

  "Film Title" : "VOLCANO TEXAS",

  "Rate" : -1

},

{

  "Rental Date" : "2005-06-18 01:19:04.0",

  "Return Date" : "2005-06-25 03:59:04.0",

  "filmId" : 155,

  "rentalId" : 2188,

  "staffId" : 2,

  "Film Title" : "CLEOPATRA DEVIL",

  "Payments" : [

    {

      "Amount" : 1.99000000095367432,

      "Payment Date" : "2005-06-18

01:19:04.0",

      "Payment Id" : 4976

    }

  ],

  "Rate" : -1

```

```

    },
    {
        "Film Title" : "ROSES TREASURE",
        "Payments" : [
            {
                "Payment Date" : "2005-06-18
20:31:00.0",
                "Payment Id" : 4977,
                "Amount" : 5.989999771118164
            }
        ],
        "Rental Date" : "2005-06-18 20:31:00.0",
        "Return Date" : "2005-06-24 18:01:00.0",
        "filmId" : 745,
        "rentalId" : 2471,
        "staffId" : 2,
        "Rate" : -1
    },
    {
        "rentalId" : 6472,
        "staffId" : 1,
        "Film Title" : "SPICE SORORITY",
        "Payments" : [
            {
                "Amount" : 4.989999771118164,

```

```

01:33:25.0",
    "Payment Date" : "2005-07-12",
    "Payment Id" : 4984
  }
],
"Rental Date" : "2005-07-12 01:33:25.0",
"Return Date" : "2005-07-15 20:26:25.0",
"filmId" : 827,
"Rate" : -1
},
{
  "rentalId" : 9672,
  "staffId" : 2,
  "Film Title" : "WARS PLUTO",
  "Payments" : [
    {
      "Amount" : 5.9899999771118164,
      "Payment Date" : "2005-07-31",
      "Payment Id" : 4987
    }
  ],
  "Rental Date" : "2005-07-31 06:34:06.0",
  "Return Date" : "2005-08-08 10:29:06.0",
  "filmId" : 960,
  "Rate" : -1
}

```



```

    },
    {
        "Rental Date" : "2005-07-31 15:18:19.0",
        "Return Date" : "2005-08-04 14:23:19.0",
        "filmId" : 757,
        "rentalId" : 9931,
        "staffId" : 2,
        "Film Title" : "SAGEBRUSH CLUELESS",
        "Payments" : [
            {
                "Amount" : 2.9900000009536743,
                "Payment Date" : "2005-07-31
15:18:19.0",
                "Payment Id" : 4989
            }
        ],
        "Rate" : -1
    },
    {
        "Payments" : [
            {
                "Payment Id" : 4990,
                "Amount" : 5.989999771118164,
                "Payment Date" : "2005-08-01
15:09:17.0"
            }
        ]
    }

```

```

],
  "Rental Date" : "2005-08-01 15:09:17.0",
  "Return Date" : "2005-08-09 13:58:17.0",
  "filmId" : 673,
  "rentalId" : 10620,
  "staffId" : 1,
  "Film Title" : "PERSONAL LADYBUGS",
  "Rate" : -1
},
{
  "Return Date" : "2005-08-28 05:22:43.0",
  "filmId" : 416,
  "rentalId" : 12831,
  "staffId" : 1,
  "Film Title" : "HIGHBALL POTTER",
  "Payments" : [
    {
      "Payment Date" : "2005-08-19
01:40:43.0",
      "Payment Id" : 4994,
      "Amount" : 3.990000009536743
    }
  ],
  "Rental Date" : "2005-08-19 01:40:43.0",
  "Rate" : -1
},

```

```

    {
      "rentalId" : 13536,
      "staffId" : 2,
      "Film Title" : "CHISUM BEHAVIOR",
      "Payments" : [
        {
          "Amount" : 4.989999771118164,
          "Payment Date" : "2005-08-20
03:35:16.0",
          "Payment Id" : 4996
        }
      ],
      "Rental Date" : "2005-08-20 03:35:16.0",
      "Return Date" : "2005-08-25 04:06:16.0",
      "filmId" : 145,
      "Rate" : -1
    }
  ]
}
```

Advertencia: la instrucción skip es en base 0 y debe estar después de la ordenación.

Sugerencia: [La documentación](#) acerca de modificación de arrays de MongoDB os puede ser útil.

3.6 Consulta (25%)

Ahora mismo tenemos la puntuación de cada alquiler correctamente registrada. Sin embargo, cada vez que queramos obtener la puntuación media de una película tendremos que hacer una consulta a la colección de Sakila_customers y calcular la valoración media de todos los alquileres de esa película. Este proceso es muy costoso, teniendo en cuenta que habrá muchas más consultas que modificaciones, por lo que se nos solicita que diseñemos una solución más eficiente.

La aplicación que modifica los datos cada vez que se produce un alquiler, es decir, la aplicación que ejecuta la consulta para actualizar la colección de Sakila_customers, está capacitada para ejecutar más consultas o comandos en el momento de la actualización. Por lo tanto, podemos mantener actualizada la puntuación media de una película cada vez que se introduce una valoración nueva (excluyendo en el cálculo las puntuaciones con valor -1 que significa "no puntuación"). En caso de inconsistencia temporal (por ejemplo debido a una modificación realizada por otra aplicación) se podrían volver a recalcular todos los valores en un proceso de recálculo de todas las medias de todas las películas (este caso no se aborda en este ejercicio).

Se nos solicita que realicemos esta implementación y diseñemos las consultas a ejecutar para mantener la media actualizada en la colección que hayáis elegido. Con este objetivo en mente, resuelve las siguientes dos preguntas:

Pregunta 1: ¿En qué colección guardarías el atributo para mantener el valor calculado?

La mejor colección para albergar las medias de cada película es Sakila_films, ya que dentro del documento están las diferentes películas dentro de las cuales podemos añadir un atributo adicional que consista en esa media para cada película.

Pregunta 2: Se recibe un alquiler de la película con título "AFFAIR PREJUDICE" y "filmId" 4 con un nuevo alquiler en el array Rentals del cliente con identificador 1. En este caso el programa deberá calcular la nueva media de esta película y guardar el valor donde hayáis decidido en el punto anterior. Para realizar el ejercicio, deberéis insertar el alquiler con la valoración usando la siguiente instrucción:

```
db.Sakila_customers.updateOne(
  { "_id": 1 },
  { $push:
    {
      "Rentals":
      {
        "filmId": 4,
        "rentalId": 10000,
        "staffId": 2,
        "Film Title": "AFFAIR PREJUDICE",
        "Payments": [
          {
            "Amount": 10.7,
            "Payment Date": "2024-01-01 00:00:12.0",
            "Payment Id": 3
```

```

    }
  ],
  "Rental Date": "2023-12-29 00:00:12.0",
  "Return Date": "2024-01-01 00:00:12.0",
  "Rate": 3
}
}
}
);

```

Se piden dos consultas:

- La consulta para obtener la calificación media de esta película teniendo en cuenta el registro insertado en la instrucción anterior.

```

> var rates = db.Sakila_customers.aggregate([
  { $unwind: "$Rentals" },
  { $match: { "Rentals.filmId": filmId, "Rentals.Rate": { $ne:
-1 } } },

  { $project: { _id: 0, "Rentals.Rate": 1 } }
]).toArray();

> printjson(rates);

[{"Rentals" : {"Rate": 3}}]

```

- La consulta para actualizar el valor en la colección que habéis elegido en la pregunta anterior. Como MongoDB es flexible, no hace falta que estén todos los documentos de la colección con ese atributo y podemos asumir que ese atributo se irá añadiendo a todos los documentos a medida que se vayan produciendo actualizaciones en las calificaciones de las películas.

```

var filmId = 4;
var rates = db.Sakila_customers.aggregate([
  { $unwind: "$Rentals" },
  { $match: { "Rentals.filmId": filmId, "Rentals.Rate": { $ne:
-1 } } },
  { $group: { _id: "$Rentals.filmId", averageRate: { $avg:
"$Rentals.Rate" } } }
]).toArray();

if (rates.length > 0) {
  var averageRate = rates[0].averageRate;
  db.Sakila_films.updateOne(
    { "filmId": filmId },
    { $set: { "averageRate": averageRate } }
  );
}

```

Criterios de valoración

En cada ejercicio se valorará la validez de la solución y la claridad de la argumentación. Cada ejercicio tiene indicado en el enunciado su peso en la valoración final.

Formato y fecha de entrega

Tenéis que enviar la PRA1 en “Contenidos > Entrega de la actividad PRA1” disponible en el aula. El formato del archivo que contiene vuestra solución puede ser .pdf, .odt, .doc y .docx. Para otras opciones, por favor, contactar previamente con vuestro profesor colaborador. El nombre del fichero debe contener el código de la asignatura, vuestro apellido y vuestro nombre, así como el número de actividad (PRA1). Por ejemplo nombreakellido1_nosql_pra1.docx.

La fecha límite para entregar la PRA1 es el **23/06/24**.

Propiedad intelectual

Al presentar una práctica o PEC que haga uso de recursos ajenos, se tiene que presentar junto con ella un documento en que se detallen todos ellos, especificando el nombre de cada recurso, su autor, el lugar donde se obtuvo y su estatus legal: si la obra está protegida por el copyright o se acoge a alguna otra licencia de uso (Creative Commons, licencia GNU, GPL etc.). El estudiante tendrá que asegurarse que la licencia que sea no impide específicamente su uso en el marco de la práctica o PEC. En caso de no encontrar la información correspondiente tendrá que asumir que la obra está protegida por el copyright.

Será necesario, además, adjuntar los ficheros originales cuando las obras utilizadas sean digitales, y su código fuente, si así corresponde.