

Uso de bases de datos NoSQL

PRA1

Propuesta de solución

Instrucciones

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

También tenéis a vuestra disposición dos vídeos, uno de uso general de la máquina virtual y otro con sugerencias para trabajar con la base de datos de Cassandra. Aunque estéis familiarizados con el uso de máquinas virtuales o estas bases de datos, os recomendamos que los veáis, ya que podéis encontrar alguna sugerencia o idea nueva que os facilite el desarrollo de las actividades:

- Consejos sobre el uso de la máquina virtual: <https://vimeo.com/539103098>
- Consejos sobre el uso de Cassandra: <https://vimeo.com/539103072>

Para esta práctica no se usará ninguna de las bases de datos pre-instaladas en la máquina virtual. En cada ejercicio encontraréis las instrucciones detalladas para cargar la base de datos necesaria para resolverlo.

Ejercicio 1: Cassandra (30%)

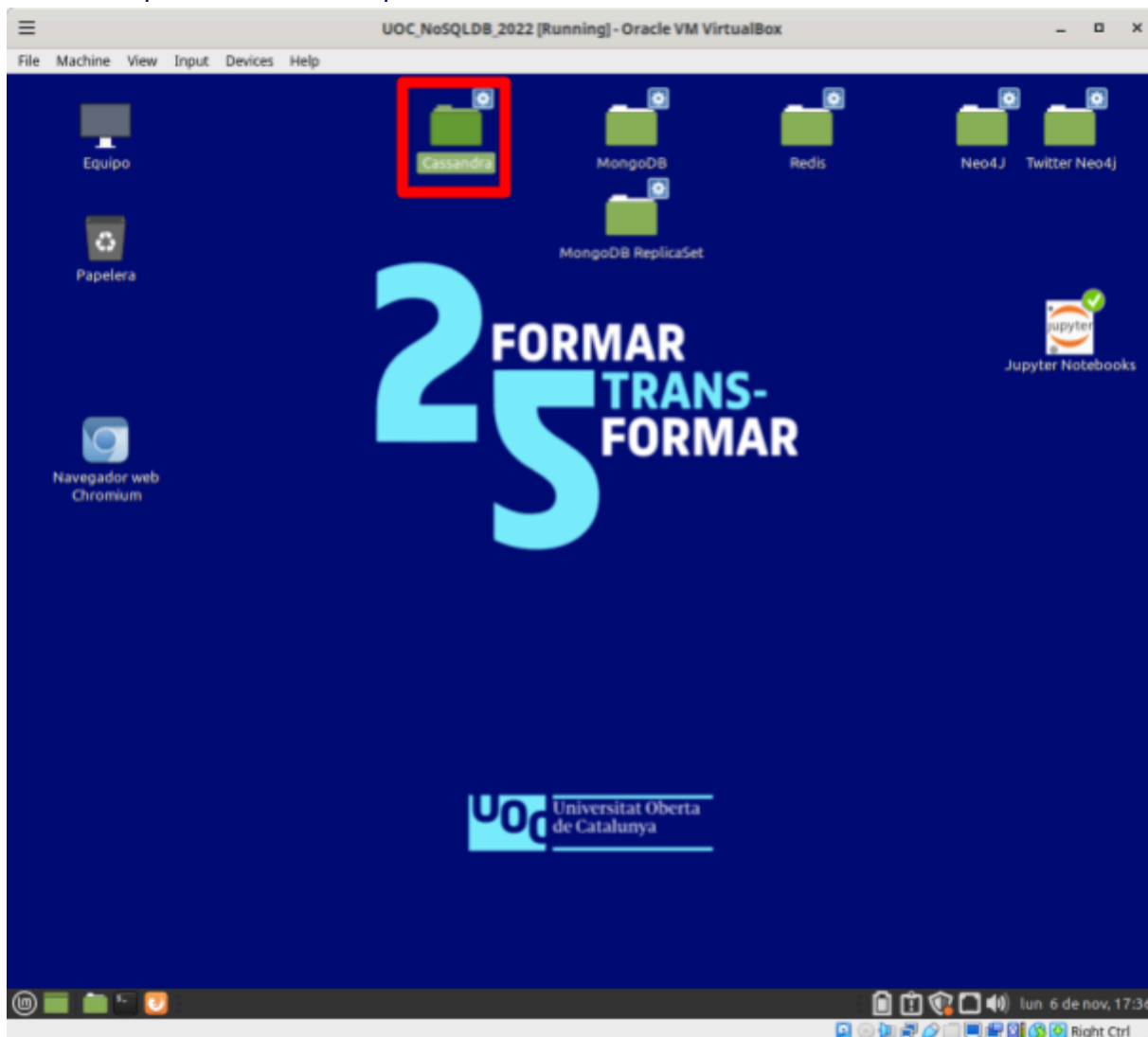
Descarga y carga de datos en la base de datos

Para la resolución de la práctica se facilitan unos archivos csv y cada archivo es un agregado. Tened en cuenta que puede que no sea necesario cargar todos los archivos en la base de datos para resolver todas las consultas. **Se valorará la resolución de las consultas usando el mínimo número de agregados (archivos csv).**

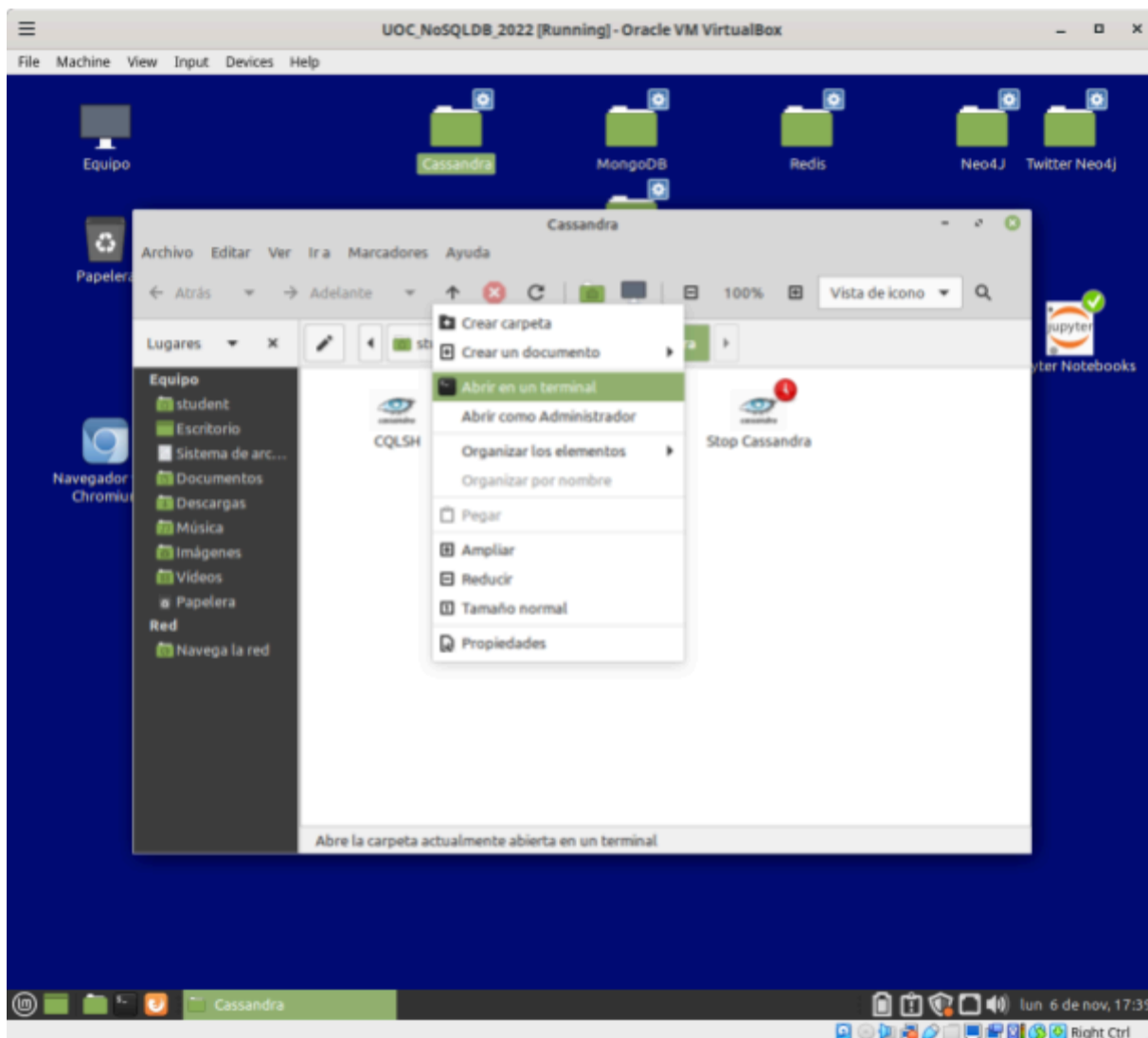
Descarga de los archivos

Una vez tengáis la máquina virtual en ejecución, seguid los siguientes pasos:

Abrid la carpeta Cassandra disponible en el escritorio.



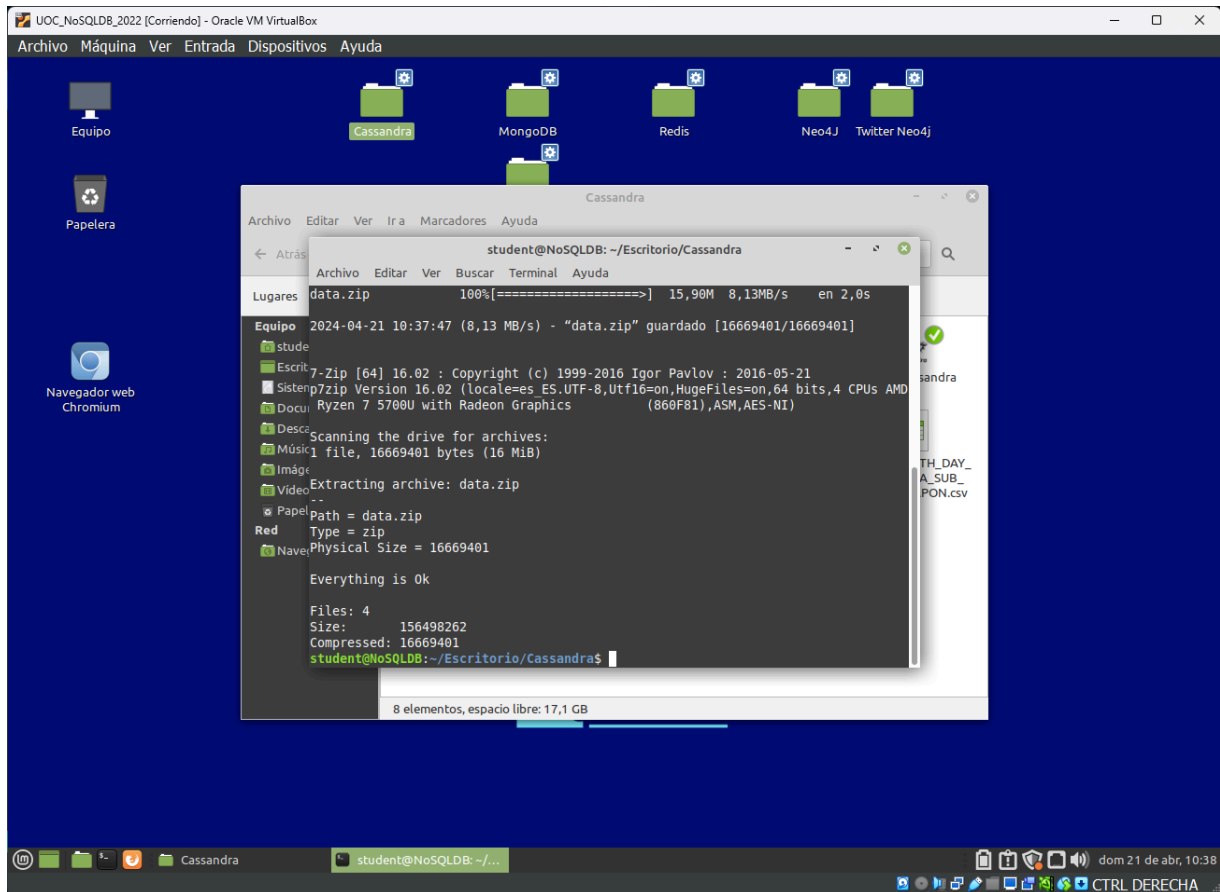
Haced clic con el botón derecho del ratón en un espacio vacío de la carpeta y luego en “Abrir en un terminal”.



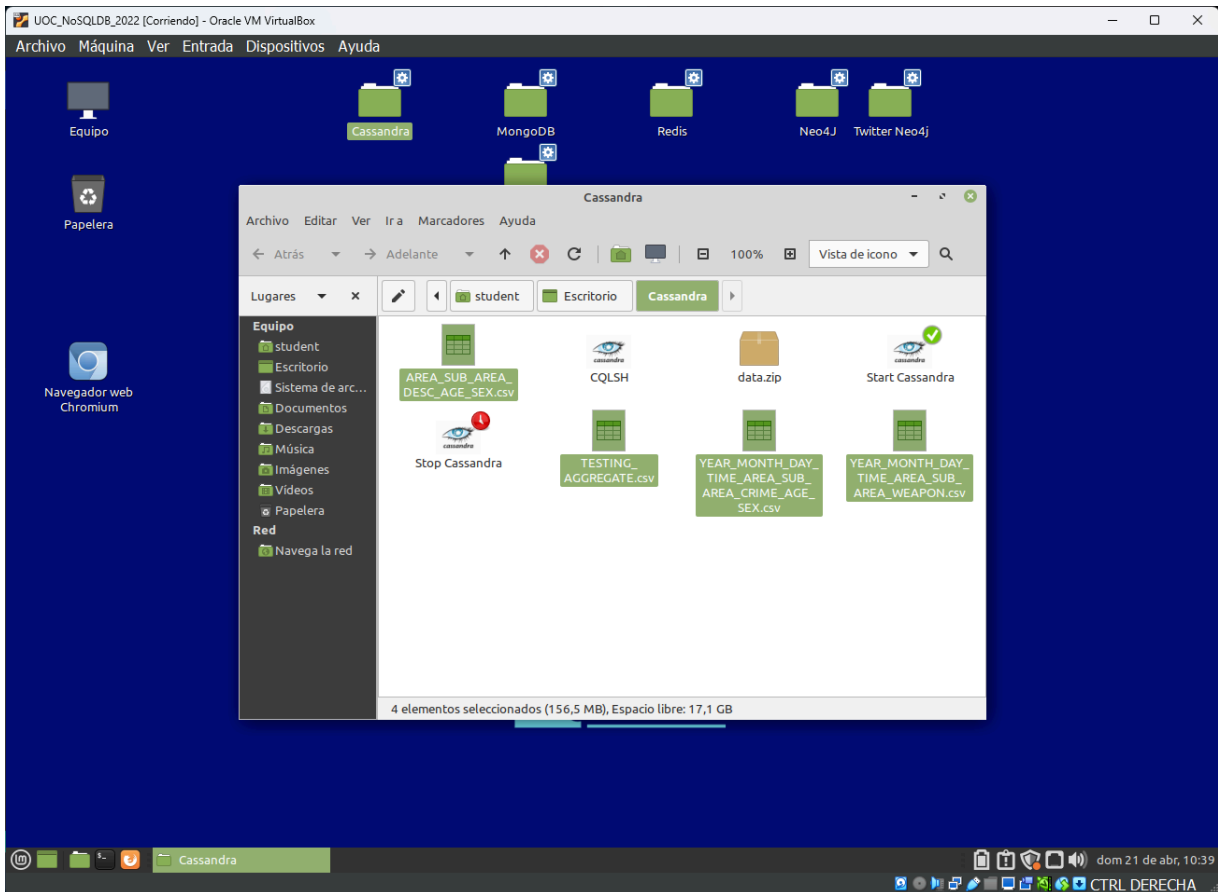
Copiad (en el documento CTRL + C) y pegad en la ventana del terminal (CTRL + MAYÚSCULAS + V) todas las líneas a la vez del comando siguiente:

```
wget https://github.com/uoc-bbdd-no-tradicionales/\  
docker-compose-replicaset/\  
raw/pra-202302/data.zip && 7z x data.zip
```

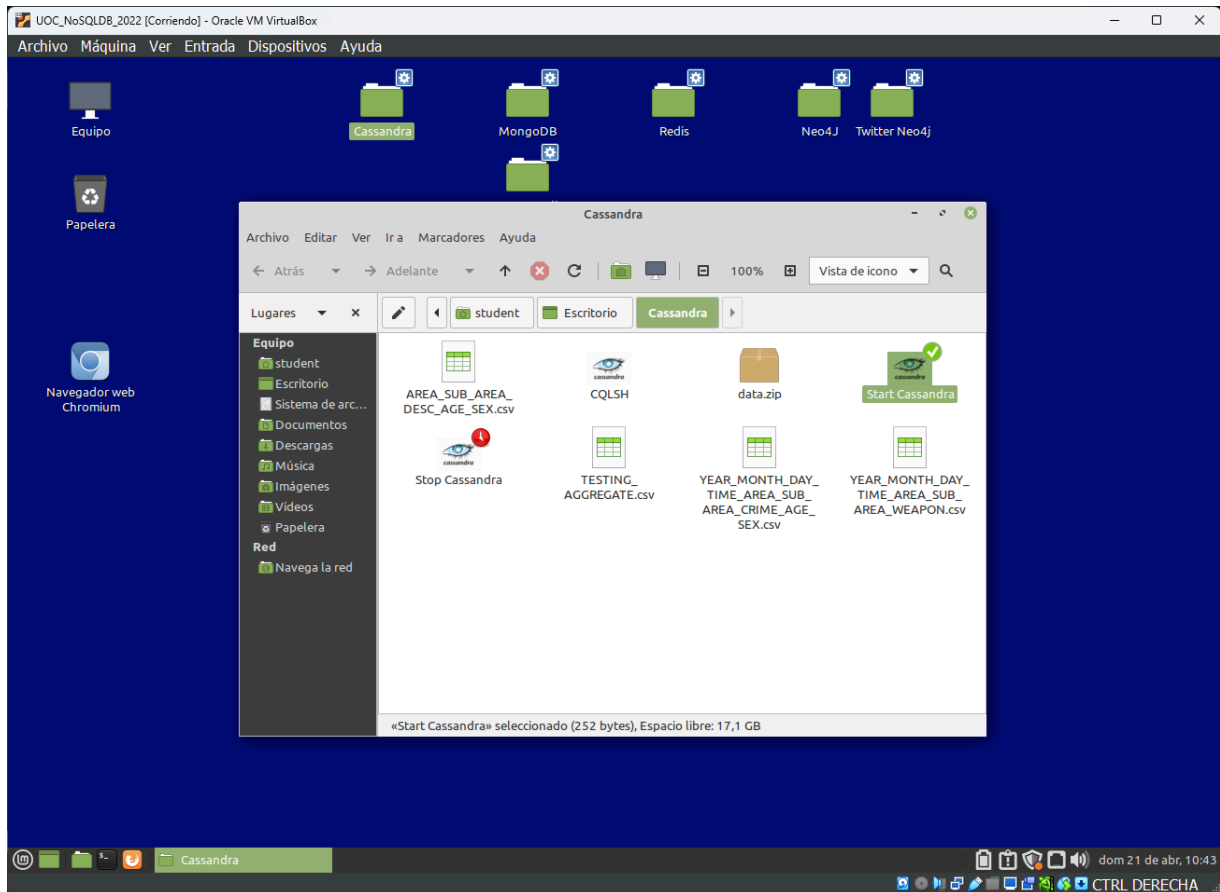
Pulsad intro y esperad a que termine la ejecución. Deberíais ver este resultado:



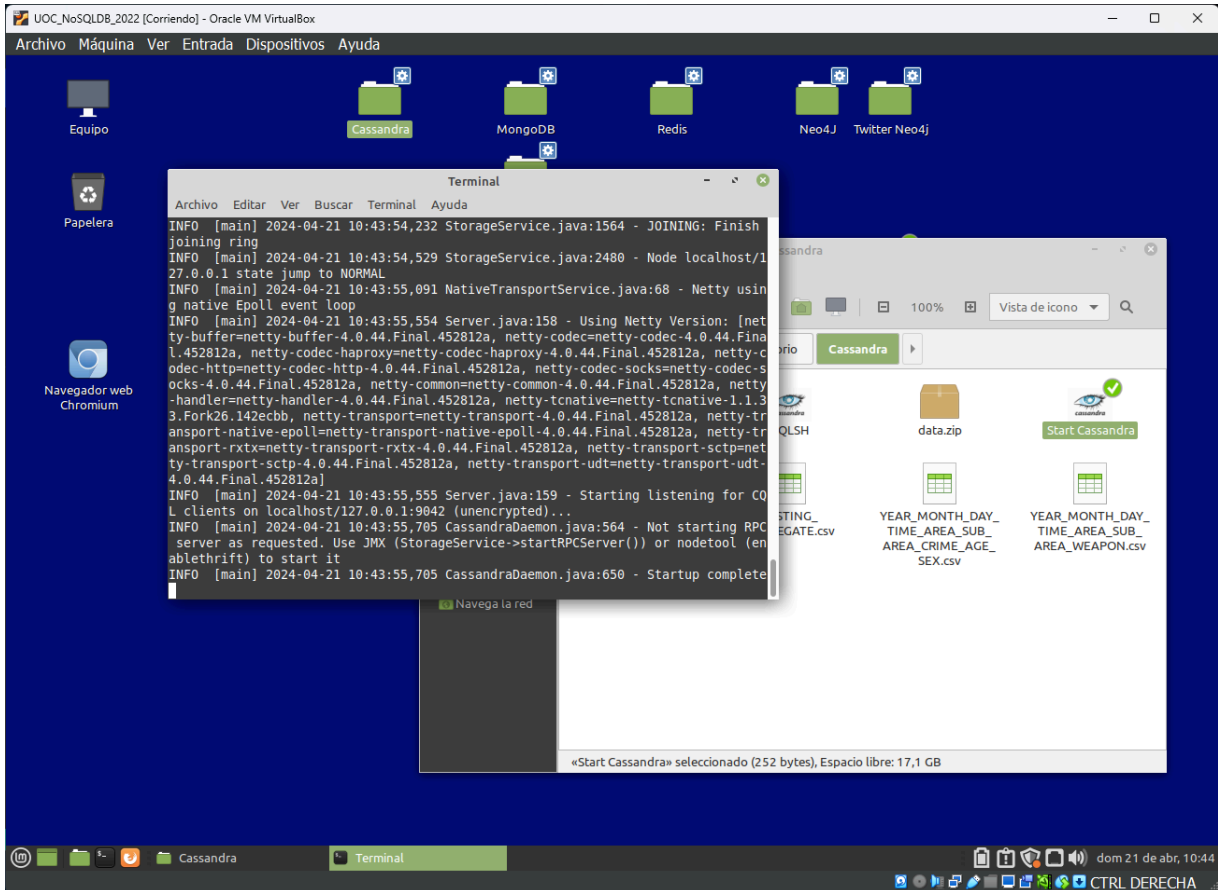
Ya podéis cerrar la ventana del terminal y ya tenéis los datos en el directorio. Deberíais ver los archivos como se muestra a continuación (los archivos descargados se han seleccionado):



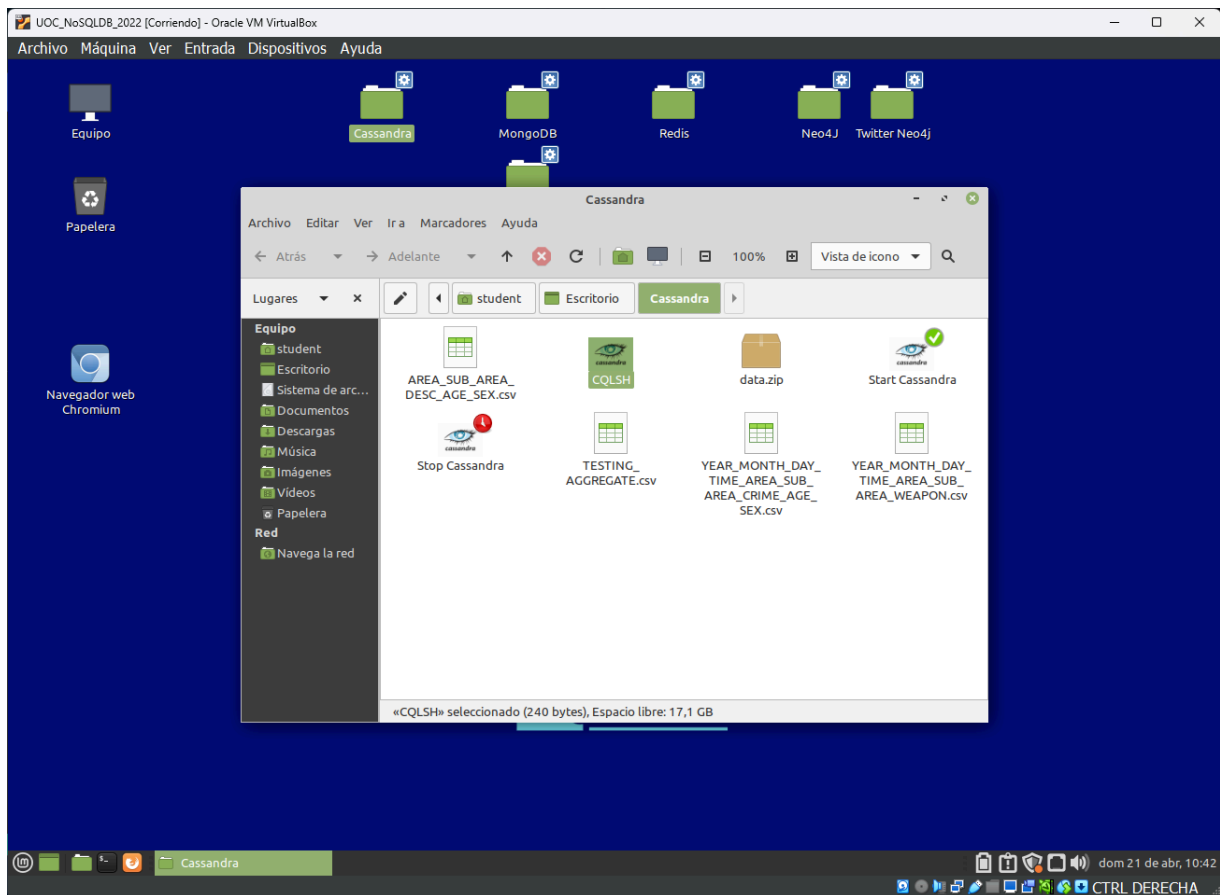
A continuación, haced doble clic en el icono “Start Cassandra”. Se abrirá una ventana de terminal. **No la cerréis, ya que pararíais el proceso de Cassandra.** Podéis minimizarla.



Esta es la ventana del terminal. Al principio de la ejecución puede dar un error de que no se tienen permisos para escribir en el archivo de log. El error se puede ignorar, la base de datos se ejecutará normalmente.



Luego haced doble clic en el icono CQLSH para situaros en el terminal de Cassandra. Si el terminal se abre y pasados unos segundos se cierra, suele ser porque el servicio de Cassandra todavía está arrancando. Solamente tendréis que intentarlo pasados unos segundos (puede que algún minuto) y os aparecerá la línea de comandos en la ventana que se abre.



Ahora ya estáis situados en la interfaz de comandos de Cassandra.

Carga de datos

Primero creamos un keyspace llamado CRIMES con el siguiente comando:

```
CREATE KEYSPACE CRIMES WITH REPLICATION = { 'class':
'SimpleStrategy', 'replication_factor': 1};
```

Este es el resultado (el comando no retorna ningún mensaje):

```
cqlsh> CREATE KEYSPACE CRIMES WITH REPLICATION = { 'class':
'SimpleStrategy', 'replication_factor': 1};
cqlsh>
```

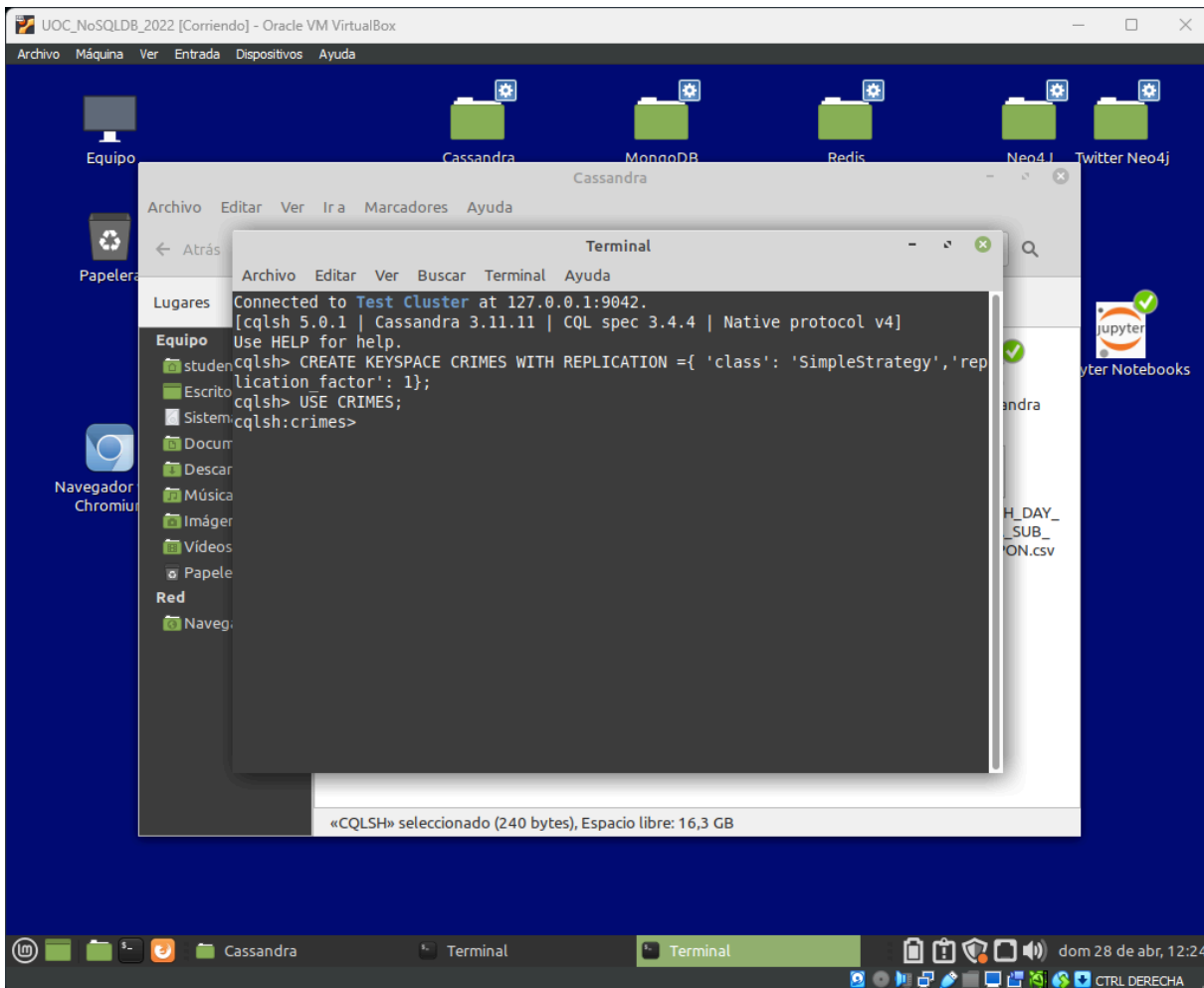
Indicamos que queremos utilizar o situarnos en el keyspace creado ejecutando:

```
USE CRIMES;
```

Que nos cambiará el *command prompt* a:


```
cqlsh:crimes>
```

Este debería ser el resultado:



Una vez estamos en el keyspace ya podemos cargar los datos. **Recordad que cada vez que volváis a iniciar el servicio de Cassandra y conectaros a la base de datos, deberéis ubicaros en el keyspace CRIMES nuevamente.**

Primero, a modo de ejemplo y para resolver el primer ejercicio, debemos crear una familia de columnas (tabla en Cassandra) y lo haremos para el agregado TESTING_AGGREGATE.csv. Los comandos son los siguientes:

Paso opcional. Por si existiera la tabla (o si queréis recrearla para empezar el ejercicio de nuevo) podéis ejecutar el siguiente comando para eliminarla. Si creáis la base de datos por primera vez o sabéis que la tabla no existe, no es necesario.

```
DROP TABLE IF EXISTS TESTING_AGGREGATE;
```

Creamos la tabla con el comando:

```
CREATE TABLE TESTING_AGGREGATE
(
    COLUMNA1 TEXT,
    COLUMNA2 TEXT,
    COLUMNA3 TEXT,
    COLUMNA4 TEXT,
    PRIMARY KEY (COLUMNA1, COLUMNA2, COLUMNA3)
);
```

Una vez tenemos la tabla creada, cargamos los datos con el comando:

```
COPY CRIMES.TESTING_AGGREGATE (COLUMNA1, COLUMNA2, COLUMNA3,
COLUMNA4) FROM
'/home/student/Escritorio/Cassandra/TESTING_AGGREGATE.csv'
WITH DELIMITER=',' AND HEADER=TRUE;
```

Comprobamos que los datos se han cargado correctamente ejecutando la consulta:

```
SELECT *
FROM TESTING_AGGREGATE;
```

La consulta retorna en modo texto (no captura de pantalla):

```
cqlsh:crimes> SELECT * FROM TESTING_AGGREGATE;
```

| columna1 | columna2 | columna3 | columna4 |
|----------|----------|----------|----------|
| clave1_1 | clave2_1 | clave3_1 | valor1 |
| clave1_2 | clave2_2 | clave3_2 | valor2 |

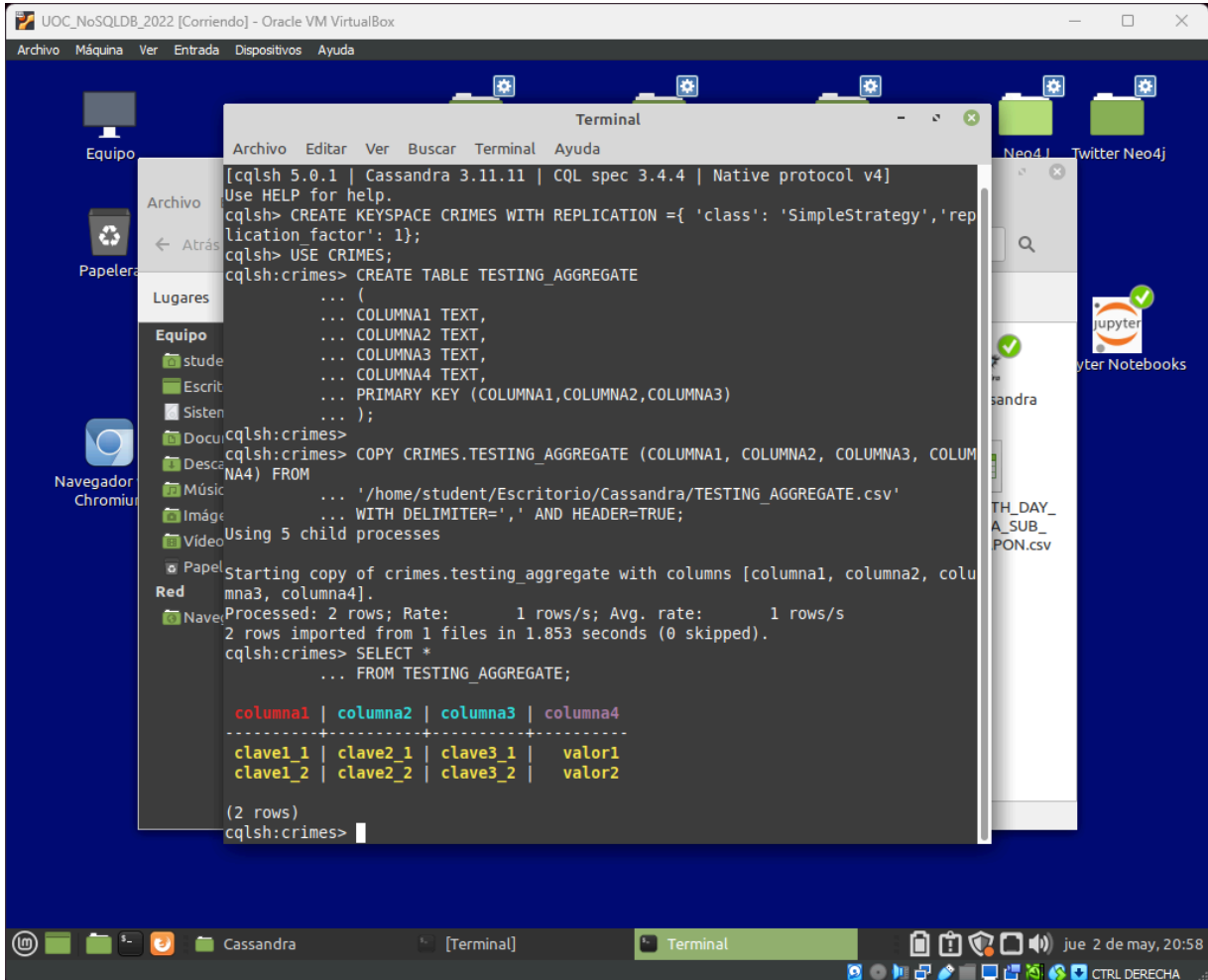
```
(2 rows)
```

```
cqlsh:crimes>
```

Recordad que el uso del asterisco en la cláusula select (SELECT *) no es una buena práctica en entornos de producción.

El formato texto (tal y como lo veis arriba) es el recomendado para adjuntar los resultados en vuestras respuestas a los ejercicios de Cassandra. La fuente Courier alinea mejor los resultados ya que todas las letras y símbolos tienen el mismo ancho. Si la consulta no cabe en el ancho del documento, podéis reducir el tamaño de la fuente para que quepa. Por favor, **para este ejercicio adjuntad los resultados de este apartado en modo texto, no capturas de pantalla.**

Y en la captura de pantalla, que se adjunta a modo de tutorial, podéis ver toda la secuencia de comandos ejecutados en el terminal.



```
[cqlsh 5.0.1 | Cassandra 3.11.11 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> CREATE KEYSPACE CRIMES WITH REPLICATION ={'class': 'SimpleStrategy','replication factor': 1};
cqlsh> USE CRIMES;
cqlsh:crimes> CREATE TABLE TESTING_AGGREGATE
... (
... COLUMN1 TEXT,
... COLUMN2 TEXT,
... COLUMN3 TEXT,
... COLUMN4 TEXT,
... PRIMARY KEY (COLUMN1,COLUMN2,COLUMN3)
... );
cqlsh:crimes> COPY CRIMES.TESTING_AGGREGATE (COLUMN1, COLUMN2, COLUMN3, COLUMN4) FROM
... '/home/student/Escritorio/Cassandra/TESTING_AGGREGATE.csv'
... WITH DELIMITER=';' AND HEADER=TRUE;
Using 5 child processes
Starting copy of crimes.testing_aggregate with columns [column1, column2, column3, column4].
Processed: 2 rows; Rate: 1 rows/s; Avg. rate: 1 rows/s
2 rows imported from 1 files in 1.853 seconds (0 skipped).
cqlsh:crimes> SELECT *
... FROM TESTING_AGGREGATE;

column1 | column2 | column3 | column4
-----+-----+-----+-----
clave1_1 | clave2_1 | clave3_1 | valor1
clave1_2 | clave2_2 | clave3_2 | valor2
(2 rows)
cqlsh:crimes>
```

1.1 Ejercicio, modificación de datos (10%)

Ejecutar el siguiente comando:

```
INSERT INTO TESTING_AGGREGATE (COLUMN1, COLUMN2, COLUMN3,
COLUMN4) VALUES ('clave1_1', 'clave2_1', 'clave3_1', 'valor3');
```

Después de ejecutar el comando anterior, volver a ejecutar la consulta para ver todos los datos de la tabla y pegar a continuación los resultados igual que se ha hecho en el ejemplo.

SOLUCIÓN:

Después de ejecutar el comando insert detallado en el enunciado, ejecutamos la consulta:

```
SELECT *
FROM TESTING_AGGREGATE;
```

Que retorna:

```
cqlsh:crimes> SELECT * FROM TESTING_AGGREGATE;
```

| columna1 | columna2 | columna3 | columna4 |
|----------|----------|----------|----------|
| clave1_1 | clave2_1 | clave3_1 | valor3 |
| clave1_2 | clave2_2 | clave3_2 | valor2 |

(2 rows)

```
cqlsh:crimes>
```

Después, ejecutar el siguiente comando:

```
UPDATE TESTING_AGGREGATE
SET COLUMNA4 = 'valor4'
WHERE COLUMNA1 = 'clave1_3' AND COLUMNA2 = 'clave2_3' AND COLUMNA3
= 'clave3_3';
```

Después de ejecutar el comando anterior, volver a ejecutar la consulta para ver todos los datos de la tabla y pegar a continuación los resultados igual que se ha hecho en el ejemplo.

SOLUCIÓN:

Después de ejecutar el comando update detallado en el enunciado, ejecutamos la consulta:

```
SELECT *
FROM TESTING_AGGREGATE;
```

Que retorna:

```
cqlsh:crimes> SELECT * FROM TESTING_AGGREGATE;
```

| columna1 | columna2 | columna3 | columna4 |
|----------|----------|----------|----------|
| clave1_1 | clave2_1 | clave3_1 | valor3 |
| clave1_2 | clave2_2 | clave3_2 | valor2 |
| clave1_3 | clave2_3 | clave3_3 | valor4 |

(3 rows)

```
cqlsh:crimes>
```

Finalmente, explica qué ha pasado en cada comando de modificación de datos (insert update) que se ha ejecutado en los pasos anteriores. Contesta las siguientes preguntas:

¿Por qué ha pasado ésto? ¿Qué diferencias hay entre este comportamiento y el de una base de datos relacional o tradicional? Este es el único apartado que puntúa de la pregunta, los anteriores son guiados y no puntúan.

SOLUCIÓN:

Podemos ver cómo el comando de INSERT, en lugar de insertar, ha actualizado el valor de la primera fila como si se tratara de un UPDATE. Después, podemos ver cómo el comando UPDATE ha insertado una fila en lugar de realizar una actualización, como si se tratara de un INSERT. De hecho, en la consulta de actualización los valores en la cláusula WHERE no existen, por lo que en un sistema relacional no se hubiera realizado ninguna actualización al no haber ninguna fila que cumpliera con el criterio definido en el WHERE. Sin embargo, por eficiencia, Cassandra no realiza ninguna comprobación. Si la fila no existe la crea y si existe la actualiza, pero siempre realiza la misma operación.

En el caso de la consulta INSERT, un sistema relacional hubiera retornado una excepción indicando que ya hay un registro con esa clave primaria sin hacer ningún cambio en la base de datos. Sin embargo, Cassandra tampoco realiza ninguna comprobación, por lo que si la fila existe la actualiza directamente y si la fila no existe la crea.

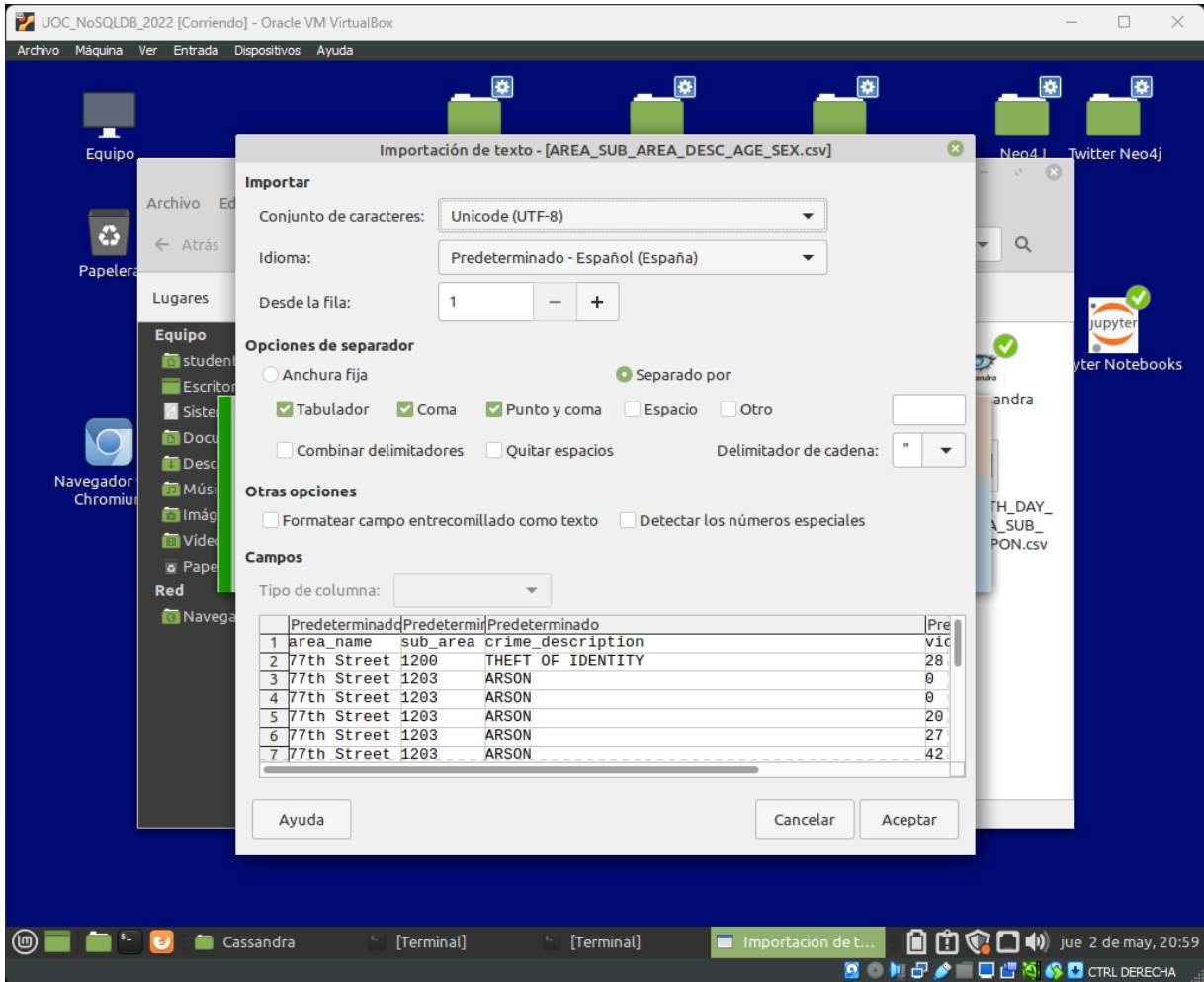
Si se busca documentación adicional acerca del comportamiento de Cassandra con las consultas de modificación de datos (update e insert) encontraréis que podemos afirmar que **casi siempre** un insert = update = upsert.

Aquí tenéis información sobre el comportamiento de las [instrucciones insert y update en Cassandra](#). Tened en cuenta que en [otra entrada del mismo hilo](#) se indica que la afirmación no es 100% precisa, de ahí el “casi siempre”.

Como hemos indicado anteriormente, los sistemas relacionales no realizan ningún cambio en la base de datos cuando se ejecuta un update con unas condiciones que no coinciden con ninguna fila de la tabla. Sin embargo, Cassandra realiza un insert porque es más eficiente ahorrarse comprobaciones.

Base de datos para consultas

Para las demás consultas del ejercicio deberéis cargar los datos de agregados que se han descargado anteriormente. Todos los agregados contienen algunas dimensiones y un recuento de reclamaciones por cada clave o fila agregada. Para ver su contenido gráficamente podéis usar el LibreOffice que está instalado en la misma máquina virtual. Solamente tendréis que hacer doble clic en el archivo de datos de la misma carpeta y podréis verlo como se muestra a continuación.



Descripción de la base de datos

Con el objetivo de analizar el comportamiento criminal y delictivo, el Ayuntamiento de la ciudad de Los Ángeles ha decidido crear una base de datos con todos los delitos y crímenes que tienen lugar en su municipio. Esta es una base de datos real de hechos delictivos ocurridos entre el 01/01/2020 y el 31/12/2023.

Para realizar la práctica se proporcionan los datos ya agregados. Cada agregado se corresponde con un archivo de los descargados anteriormente, excepto el agregado TESTING_AGGREGATE.csv que contiene los datos para el primer ejercicio y sirve para mostrar cómo importar los datos en Cassandra.

Descripción de las columnas

En los agregados os podréis encontrar los siguientes nombres de columnas. Aquí tenéis una descripción del significado de todas las columnas que se pueden encontrar. Si no se indica lo contrario, son en formato texto:

- AREA_NAME. Nombre del área de la ciudad donde se ha cometido el delito.
- SUB_AREA. Código alfanumérico que indica el área dentro del área donde se ha cometido el delito.
- CRIME_DESCRIPTION. Descripción del crimen.
- WEAPON_DESC. Descripción del arma usada.
- VICT_SEX. Género de la víctima.
- VICT_AGE edad de la víctima en formato entero (INT).
- YEAR_OCC. Año del delito.
- MONTH_OCC. Mes del delito.
- DAY_OCC. Día del delito.
- TIME_OCC. Hora del delito.
- TOTAL_COUNT Recuento de los delitos cometidos en formato entero (INT).

Error conocido en creación de índices

Durante la resolución de las consultas puede que tengáis que crear un índice. Es posible que si ejecutáis la consulta inmediatamente después de la creación del índice la base de datos retorne el error:

```
ReadFailure: Error from server: code=1300 [Replica(s) failed to
execute read] message="Operation failed - received 0 responses and 1
failures" info={'failures': 1, 'received_responses': 0,
'required_responses': 1, 'consistency': 'ONE'}
```

Esto es debido a que en Cassandra los índices se crean de forma asíncrona, por lo que puede que todavía se esté creando el índice al hacer la consulta. Solamente deberéis esperar un rato (suele ser menos de uno o dos minutos), volver a ejecutar la consulta y se ejecutará correctamente sin errores. Si el error persiste, puede que tengáis que asignar más recursos a la máquina virtual (más CPU y memoria). Tener en cuenta que la máquina viene configurada por defecto con muy pocos recursos.

Consultas en Cassandra

Con los datos que habéis descargado en el apartado anterior, se requiere realizar las consultas detalladas y adjuntar su resultado en modo texto (**no capturas de pantalla**).

Puede que algunas consultas retornen error y no se puedan ejecutar directamente. De cara a elegir los datos y resolver las consultas prestad atención a:

- Crear las claves de partición adecuadas.
- Crear las claves primarias adecuadas.
- Puede que se tengan que crear índices. Es mejor crear un índice que cargar otro agregado entero.
- Se pueden resolver varias consultas con los mismos agregados y puede que no todos los agregados sean necesarios.

Sugerencia: antes de realizar ninguna carga de datos adicional para resolver cualquier consulta, se recomienda mirar cuáles son los requisitos de las consultas siguientes para minimizar el número de agregados a cargar en la base de datos. **Cuantos menos agregados se carguen para resolver todas las consultas del ejercicio mejor.**

Se valorará todo el procedimiento realizado para ejecutar las consultas y obtener la información que se pide en el enunciado. En cualquier caso, **siempre debéis adjuntar las consultas que os den error y todas las sentencias CQL, comandos o acciones que utilizéis para poder realizar la consulta (incluyendo las sentencias utilizadas para cargar datos)**. Si hubiera más de una opción, también se tendrá en cuenta que la opción elegida sea la que tenga el mejor rendimiento.

1.2 Consulta 1 (20%)

Total de crímenes ocurridos durante los meses de enero, febrero, marzo, abril, mayo y junio del año 2020. La consulta debe retornar el año, el mes y el recuento total de crímenes de ese periodo.

Sugerencia: el operador OR no se puede utilizar para esta consulta, deberás utilizar el operador IN.

SOLUCIÓN:

Esta consulta se puede resolver con los agregados:

- YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_WEAPON
- YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX

Si observamos las demás consultas el agregado que más nos conviene utilizar es el YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX ya que se podrá reutilizar en más consultas que tendremos que resolver más adelante. Lo cargamos con las siguientes instrucciones:

```
DROP TABLE IF EXISTS
YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX;

CREATE TABLE YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
(
    YEAR_OCC TEXT,
    MONTH_OCC TEXT,
```



```

DAY_OCC TEXT,
TIME_OCC TEXT,
AREA_NAME TEXT,
SUB_AREA TEXT,
CRIME_DESCRIPTION TEXT,
VICT_AGE INT,
VICT_SEX TEXT,
TOTAL_COUNT INT,
PRIMARY KEY ((YEAR_OCC, MONTH_OCC), DAY_OCC, TIME_OCC,
AREA_NAME, SUB_AREA, CRIME_DESCRIPTION, VICT_AGE, VICT_SEX)
);

COPY CRIMES.YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
(YEAR_OCC, MONTH_OCC, DAY_OCC, TIME_OCC, AREA_NAME, SUB_AREA,
CRIME_DESCRIPTION, VICT_AGE, VICT_SEX, TOTAL_COUNT) FROM
'/home/student/Escritorio/Cassandra/YEAR_MONTH_DAY_TIME_AREA_SUB_ARE
A_CRIME_AGE_SEX.csv'
WITH DELIMITER=',' AND HEADER=TRUE;

```

Y podemos ejecutar la consulta:

```

SELECT YEAR_OCC, MONTH_OCC, SUM(TOTAL_COUNT)
FROM YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
WHERE YEAR_OCC = '2020' AND MONTH_OCC IN ('1', '2', '3', '4', '5',
'6')
GROUP BY YEAR_OCC, MONTH_OCC;

```

Que retorna:

| year_occ | month_occ | system.sum(total_count) |
|----------|-----------|-------------------------|
| 2020 | 1 | 18524 |
| 2020 | 2 | 17261 |
| 2020 | 3 | 16175 |
| 2020 | 4 | 15689 |
| 2020 | 5 | 17215 |
| 2020 | 6 | 17042 |

(6 rows)

Warnings :

Aggregation query used on multiple partition keys (IN restriction)

cqlsh:crimes>

1.3 Consulta 2 (20%)

Total de homicidios. Los homicidios se corresponden con el total de delitos con `CRIME_DESCRIPTION = 'CRIMINAL HOMICIDE'`.

Restricción: No se puede utilizar la cláusula **ALLOW FILTERING** debido al impacto negativo en el rendimiento del clúster y porque será retirada en las próximas versiones de Cassandra.

SOLUCIÓN:

Esta consulta se puede resolver con los agregados:

- `AREA_SUB_AREA_DESC_AGE_SEX`
- `YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX`

Como ya hemos cargado uno de estos agregados en el ejercicio anterior, intentamos reutilizarlo.

Ejecutamos la consulta:

```
SELECT SUM(TOTAL_COUNT)
FROM YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
WHERE CRIME_DESCRIPTION = 'CRIMINAL HOMICIDE';
```

Pero retorna un error:

```
cqlsh:crimes> SELECT SUM(TOTAL_COUNT)
... FROM YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
... WHERE CRIME_DESCRIPTION = 'CRIMINAL HOMICIDE';
InvalidRequest: Error from server: code=2200 [Invalid query]
message="PRIMARY KEY column "crime_description" cannot be restricted
as preceding column "month_occ" is not restricted"
cqlsh:crimes>
```

Teniendo en cuenta que un agregado más implica otro índice, gestión de más particiones, gestión del espacio asignado y son más datos que se tienen que replicar (y por lo tanto controlar la consistencia de los mismos por nuestra parte) resulta más eficiente crear un índice sobre un agregado existente.

Crearemos un índice:

```
CREATE INDEX CRIME_DESCRIPTION ON
YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX(CRIME_DESCRIPTION);
```

La respuesta puede que sea inmediata, pero la creación de un índice lleva tiempo. Pasados unos segundos tendremos el índice disponible y podremos volver a ejecutar la consulta:

```
SELECT SUM(TOTAL_COUNT)
FROM YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
WHERE CRIME_DESCRIPTION = 'CRIMINAL HOMICIDE';
```

Que retorna:

```
cqlsh:crimes> SELECT SUM(TOTAL_COUNT)
... FROM YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
... WHERE CRIME_DESCRIPTION = 'CRIMINAL HOMICIDE';
```

```
system.sum(total_count)
-----
1509
```

(1 rows)

Warnings :

Aggregation query used without partition key

```
cqlsh:crimes>
```

1.4 Consulta 3 (15%)

Total de delitos con CRIME_DESCRIPTION = 'PICKPOCKET' (carteristas).

Restricción: No se puede utilizar la cláusula **ALLOW FILTERING** debido al impacto negativo en el rendimiento del clúster y porque será retirada en las próximas versiones de Cassandra.

SOLUCIÓN:

Esta consulta es directa con el agregado cargado y el índice creado anteriormente.

```
SELECT SUM(TOTAL_COUNT)
FROM YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
WHERE CRIME_DESCRIPTION = 'PICKPOCKET';
```

Que retorna directamente y sin errores:

```
system.sum(total_count)
-----
2590
```

(1 rows)

Warnings :

Aggregation query used without partition key

```
cqlsh:crimes>
```

A partir de aquí se puede optar por dejar el índice en la base de datos o eliminarlo, dependiendo de la frecuencia con la que se consultarán y modificarán los datos. Este paso es opcional y no es necesario para obtener el 100% de la valoración. El comando para eliminar el índice sería:

```
DROP INDEX CRIME_DESCRIPTION;
```

Eliminar un índice cuando no se necesita es una buena práctica, puesto que los índices tienen un impacto muy positivo en las consultas, pero requieren de un mantenimiento en cada modificación de datos. El criterio general para decidir qué hacer con los índices es el siguiente: Si hay muchas consultas que necesitan filtrar por las columnas que se han indexado (como por ejemplo las que se han escrito en este ejercicio) y estas consultas se realizan con cierta frecuencia, puede que sea conveniente dejar los índices creados para no tener que crearlos cada vez que se ejecuta una consulta. Si las consultas que necesitan filtrar los datos por estas columnas no se ejecutan con cierta frecuencia, suele ser mejor eliminar el índice.

Un índice es una estructura que ocupa un espacio en disco y tiene una forma determinada dependiendo de los datos indexados. Normalmente son estructuras en forma de árbol que se tienen que reestructurar cada vez que se modifica un dato (ya sea inserción, modificación o eliminación). El árbol permite acceder a los datos rápidamente, pero requiere de un mantenimiento costoso. [En este vídeo de 15 segundos](#) podéis ver una animación de las operaciones que se tienen que realizar cuando se inserta un valor en un árbol de tipo AVL equilibrado. Este ejemplo es solamente ilustrativo. Es posible que las estructuras internas de los índices utilizados en Cassandra no sean iguales que el ejemplo del vídeo, pero este es el tipo de operaciones que se tienen que realizar para mantener un índice cada vez que se modifica un dato.

Esto añade otro criterio para tomar la decisión de eliminar o no el índice: la frecuencia con la que se modifican los datos. Si los datos indexados se modifican con mucha frecuencia se causará un estrés a la base de datos. El estrés será todavía mayor si los datos se modifican de manera concurrente. Por lo tanto, si las consultas son frecuentes y el rendimiento es crítico, merece la pena mantenerlo. Sin embargo, suele ser mejor eliminar el índice si hay muchas modificaciones de datos, pocas consultas sobre los mismos, el índice no es imprescindible para realizar las consultas, o el rendimiento de las consultas es aceptable sin índices.

1.5 Consulta 4 (20%)

Total de crímenes agrupado por sexo y filtrado por el nombre de área 'Hollywood' y sub_area con código '0643'

SOLUCIÓN:

La consulta puede resolverse con los siguientes agregados:

- AREA_SUB_AREA_DESC_AGE_SEX
- YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX

Como ya tenemos creado el agregado

YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX, intentaremos resolver la consulta con el mismo.

Si ejecutamos la consulta:

```
SELECT AREA_NAME, SUB_AREA, VICT_SEX, SUM(TOTAL_COUNT) as total
FROM YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
WHERE AREA_NAME = 'Hollywood' and sub_area = '0643'
GROUP BY AREA_NAME, SUB_AREA, VICT_SEX
ORDER BY VICT_SEX;
```

Nos retorna el error:

```
cqlsh:crimes> SELECT AREA_NAME, SUB_AREA, VICT_SEX, SUM(TOTAL_COUNT)
as total
... FROM YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX
... WHERE AREA_NAME = 'Hollywood' and sub_area = '0643'
... GROUP BY AREA_NAME, SUB_AREA, VICT_SEX
... ORDER BY VICT_SEX;
InvalidRequest: Error from server: code=2200 [Invalid query]
message="PRIMARY KEY column "area_name" cannot be restricted as
preceding column "day_occ" is not restricted"
cqlsh:crimes>
```

Aquí tenemos un problema con la cláusula GROUP BY. Para que la consulta pueda ejecutarse, todas las columnas especificadas en el GROUP BY tienen que formar parte de la clave primaria. Además, para ejecutar esta consulta tenemos disponibles dos agregados, por lo que tenemos dos opciones:

- Crear otra tabla con el mismo agregado pero diferente clave primaria.
- Considerar un agregado alternativo.

Sabemos que el tamaño del segundo es más pequeño, por lo que nos resultará más eficiente porque se podrá trabajar con un espacio asignado menor. Menos cantidad de datos, menos costes de almacenamiento y administración.

Cargamos el agregado AREA_SUB_AREA_DESC_AGE_SEX con los comandos:

```
DROP TABLE IF EXISTS AREA_SUB_AREA_DESC_AGE_SEX;
```

```
CREATE TABLE AREA_SUB_AREA_DESC_AGE_SEX
(
    AREA_NAME TEXT,
    SUB_AREA TEXT,
    CRIME_DESCRIPTION TEXT,
    VICT_AGE INT,
    VICT_SEX TEXT,
    TOTAL_COUNT INT,
    PRIMARY KEY ((AREA_NAME, SUB_AREA), VICT_SEX, VICT_AGE,
    CRIME_DESCRIPTION)
);

COPY CRIMES.AREA_SUB_AREA_DESC_AGE_SEX (AREA_NAME, SUB_AREA,
CRIME_DESCRIPTION, VICT_AGE, VICT_SEX, TOTAL_COUNT) FROM
'/home/student/Escritorio/Cassandra/AREA_SUB_AREA_DESC_AGE_SEX.csv'
WITH DELIMITER=',' AND HEADER=TRUE;
```

Ahora podemos ejecutar la consulta con el otro agregado:

```
SELECT AREA_NAME, SUB_AREA, VICT_SEX, SUM(TOTAL_COUNT) as total
FROM AREA_SUB_AREA_DESC_AGE_SEX
WHERE AREA_NAME = 'Hollywood' and sub_area = '0643'
GROUP BY AREA_NAME, SUB_AREA, VICT_SEX
ORDER BY VICT_SEX;
```

Que retorna:

| area_name | sub_area | vict_sex | total |
|-----------|----------|----------|-------|
| Hollywood | 0643 | F | 750 |
| Hollywood | 0643 | M | 975 |
| Hollywood | 0643 | NULL | 187 |
| Hollywood | 0643 | X | 218 |

(4 rows)
cqlsh:crimes>

Tal cual está creada la clave primaria, no se necesita ningún índice adicional.

1.6 Warning en consultas (5%)

Algunas de las consultas ejecutadas anteriormente retornan la advertencia *warning: Aggregation query used without partition key*. ¿Qué significa?

SOLUCIÓN:

La consulta devuelve un warning, que indica que la agrupación se ha realizado sin tener en cuenta ninguna clave de partición. Eso puede ser problemático en un entorno real, ya que puede implicar consultar todas las filas del cluster al tener que procesar la consulta en todos sus nodos, afectando muy negativamente al rendimiento de esta y otras consultas.

1.7 Detalle de tablas usadas (10%)

Por cada tabla o agregado utilizado para resolver el ejercicio, indica:

- Nombre de tabla, agregado utilizado en la tabla y consultas en las que se ha usado.
- Columnas que componen la clave primaria, clave de partición elegida
- **El porqué de la clave primaria y la partición (punto imprescindible para puntuar en la pregunta).**

Nota: contestar solamente con un listado de tablas y columnas de clave primaria y particiones no puntúa. Lo que puntúa es la justificación de la elección de las columnas de clave primaria y de partición.

SOLUCIÓN:

Tabla YEAR_MONTH_DAY_TIME_AREA_SUB_AREA_CRIME_AGE_SEX usada en las consultas 1, 2 y 3.

La clave primaria está compuesta por las columnas YEAR_OCC, MONTH_OCC, DAY_OCC, TIME_OCC, AREA_NAME, SUB_AREA, CRIME_DESCRIPTION, VICT_AGE, VICT_SEX.. Éstas son todas las columnas que componen las dimensiones del agregado. Si no fuera así, los datos no se cargarían correctamente ya que muchos inserts no insertarían todas las filas, sino que sobrescribirían valores ya existentes.

La tabla se puede crear definiendo una clave de partición compuesta como en el ejemplo (notar que las columnas YEAR_OCC y MONTH_OCC van entre paréntesis en la definición de la clave primaria) o se puede dejar por defecto de manera que se utilizará la primera columna (YEAR_OCC). En el segundo caso, habrá menos diversidad de valores para una buena distribución o dispersión de los datos en las particiones dado que el número de años puede ser una columna con menor diversidad de valores. Por lo tanto, se ha elegido utilizar la combinación de las columnas YEAR_OCC y MONTH_OCC para definir la clave de partición y así tener más valores para una mejor distribución o dispersión.

Tabla AREA_SUB_AREA_DESC_AGE_SEX usada en la consulta 4.

La clave primaria está compuesta por las columnas AREA_NAME, SUB_AREA, VICT_SEX, VICT_AGE, y CRIME_DESCRIPTION. La clave de partición también es compuesta y la componen las columnas AREA_NAME y SUB_AREA. El AREA_NAME también puede ser una columna con una diversidad de valores reducida, por lo que para garantizar una buena dispersión de los datos se utiliza la combinación de dos columnas. El motivo de la elección

de la clave primaria es el mismo que el detallado en la tabla anterior. Si las columnas de agregación no forman parte de la clave primaria (todas ellas) los datos no se cargarían correctamente. Tal y como se ha visto en el ejercicio 1.1, las instrucciones de update e insert, insertan o sobrescriben los datos (respectivamente) tanto si existen como si no.

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 la realización de este ejercicio se deberá utilizar la **versión 4.3 de Neo4j** ya instalada en la máquina virtual (detalles en las páginas 20 y 21 del manual “Uso de máquina virtual_Bases de datos no convencionales.pdf”). Por favor, ignorad la base de datos Twitter Neo4j.

2.1 Carga de datos (no puntúa)

A continuación se presentan las instrucciones para crear un nuevo grafo en Neo4j. Previamente será eliminada toda la información del grafo cargado por defecto. El apartado no puntúa, pero es fundamental para una correcta ejecución del resto del ejercicio.

Introducción: una de las formas más habituales de obtener datos es importarlos desde otros sistemas, que los proporcionarán en un formato determinado como: CSV, JSON o XML.

En este apartado se muestra cómo importar datos en formato JSON a partir de las funciones de la librería **APOC** (*Awesome Procedures on Cypher*). Los datos están ubicados en un repositorio público como un array de objetos JSON. Al importarlos se realizará un proceso de extracción y acondicionamiento de los mismos para adaptarlos a un diseño de grafo concreto.

TAREA PREVIA: Eliminación de todos los nodos, relaciones, índices y restricciones de la base de datos por defecto.

Se recomienda realizar las sentencias mediante la versión gráfica de Neo4j. Se puede acceder indicando la siguiente URL en el navegador web, después de haber arrancado la base de datos: `localhost:7474`. El usuario y contraseña es: **neo4j / uoc**.

La versión de Neo4j utilizada (*community edition*) permite sólo una base de datos activa. Por ello, se recomienda utilizar la base de datos “neo4j” para realizar la actividad. Para situarnos en esa base de datos podemos ejecutar el siguiente comando (notar que no hay punto y coma final):

```
:USE neo4j
```

Una vez en la base de datos, será conveniente borrar los datos que puedan haber.

- El siguiente comando borra todos los nodos y sus relaciones:

```
MATCH (n) DETACH DELETE n;
```

- El siguiente comando presenta todos los índices y restricciones que aún permanecen en la base de datos, **y que deben ser eliminados**:

```
SHOW indexes;
```

NOTA: En caso de que alguien tenga una versión anterior a Neo4j (o la versión *community*), las sentencias `SHOW indexes` y `SHOW constraints` pueden no funcionar. En dichos casos deberá utilizarse `CALL db.indexes()` y `CALL db.constraints()`

- Los siguientes comandos presentan ejemplos de eliminación particular de cada uno de los índices y restricciones. Adaptarlos para eliminar todos los índices y restricciones de la salida del comando anterior:

```
DROP INDEX index_f7700477;
```

```
DROP INDEX index_343aff4e;
```

```
DROP CONSTRAINT constraint_59c89b17;
```

CREACIÓN DE LOS NUEVOS NODOS Y SUS RELACIONES

Nodos de los Personajes principales y sus relaciones (ejecutad todas las líneas siguientes hasta el final incluyendo el `return p.id, p.name`; y cuidad de que no queden “rotas” las URL’s).

```
CALL
apoc.load.jsonArray('https://gist.githubusercontent.com/JosepMeseguer/f7a7df6f3b10d77ff57e82e1e9d075/raw/6479a608e41fd385a21aeeca57f28f2fc6fdf3d8/characters.json') yield value

with apoc.convert.toMap(value) as data

with apoc.map.clean(data, [],['',''],[],null)) as data

with apoc.map.fromPairs([k in keys(data) |
[toLowerCase(substring(k,0,1))+substring(k,1,size(k)), data[k]]]) as
data

MERGE (p:Person {id:data.id})

SET

p += apoc.map.clean(data,
['allegiances','father','spouse','mother'],['',''],[],null)),

p.name = coalesce(p.name,head(p.aliases))

FOREACH (id in data.allegiances | MERGE (h:House {id:id}) MERGE
(p)-[:ALLIED_WITH]->(h))

FOREACH (id in case data.father when null then [] else
[data.father] end | MERGE (o:Person {id:id}) MERGE (o)-[:PARENT_OF
{type:'father'}]->(p))

FOREACH (id in case data.mother when null then [] else
[data.mother] end | MERGE (o:Person {id:id}) MERGE (o)-[:PARENT_OF
{type:'mother'}]->(p))

FOREACH (id in case data.spouse when null then [] else
[data.spouse] end | MERGE (o:Person {id:id}) MERGE
(o)-[:SPOUSE]->(p))

return p.id, p.name;
```

Después de unos segundos de ejecución retornará algo parecido a lo siguiente (al ingerir datos que están disponibles en Internet es posible que el número de registros pueda cambiar en el futuro, aunque las consultas planteadas continúan siendo válidas):

"Added 2421 labels, created 2421 nodes, set 17125 properties, created 1824 relationships, started streaming 2124 records"

Nodos de las Casas, las relaciones entre ellas y con los Personajes principales (copiad y pegad todas las líneas hasta el final incluyendo return h.id, h.name;).

```
CALL
apoc.load.jsonArray('https://gist.githubusercontent.com/JosepMeseguer/ee0622915fde2d6d6f510098d7c9a1d5/raw/e79f8ae83de3f17296dfef9f673fa7b8761d77c8/houses.json') yield value

with apoc.convert.toMap(value) as data

with apoc.map.clean(data, [],['',''],[],null]) as data

with apoc.map.fromPairs([k in keys(data) |
[toLowerCase(substring(k,0,1))+substring(k,1,size(k)), data[k]]]) as
data

MERGE (h:House {id:data.id})

SET

h += apoc.map.clean(data,
['overlord','swornMembers','currentLord','heir','founder','cadetBranches','region','seats'],['',''],[],null])

FOREACH (id in data.swornMembers | MERGE (o:Person {id:id}) MERGE
(o)-[:ALLIED_WITH]->(h))

FOREACH (s in data.seats | MERGE (seat:Seat {name:s}) MERGE
(seat)-[:SEAT_OF]->(h))

FOREACH (id in data.cadetBranches | MERGE (b:House {id:id}) MERGE
(b)-[:BRANCH_OF]->(h))

FOREACH (id in case data.overlord when null then [] else
[data.overlord] end | MERGE (o:House {id:id}) MERGE
(h)-[:SWORN_TO]->(o))

FOREACH (id in case data.currentLord when null then [] else
[data.currentLord] end | MERGE (o:Person {id:id}) MERGE
(h)-[:LED_BY]->(o))
```

```
FOREACH (id in case data.founder when null then [] else
[data.founder] end | MERGE (o:Person {id:id}) MERGE
(h)-[:FOUNDED_BY]->(o))

FOREACH (id in case data.heir when null then [] else [data.heir]
end | MERGE (o:Person {id:id}) MERGE (o)-[:HEIR_TO]->(h))

FOREACH (r in case data.region when null then [] else [data.region]
end | MERGE (o:Region {name:r}) MERGE (h)-[:IN_REGION]->(o))

return h.id, h.name;
```

Que después de unos segundos de ejecución retornará algo parecido a lo siguiente:

```
"Added 383 labels, created 383 nodes, set 1929 properties, created
1406 relationships, started streaming 444 records"
```

CREACIÓN DE RESTRICCIONES E ÍNDICES

Creación de restricciones que garanticen identificadores únicos para Personajes y Casas

```
create constraint on (p:Person) assert p.id is unique;
create constraint on (h:House) assert h.id is unique;
create index for (p:Person) on (p.name);
create index for (h:House) on (h.name);
create index for (s:Seat) on (s.name);
create index for (r:Region) on (r.name);
```

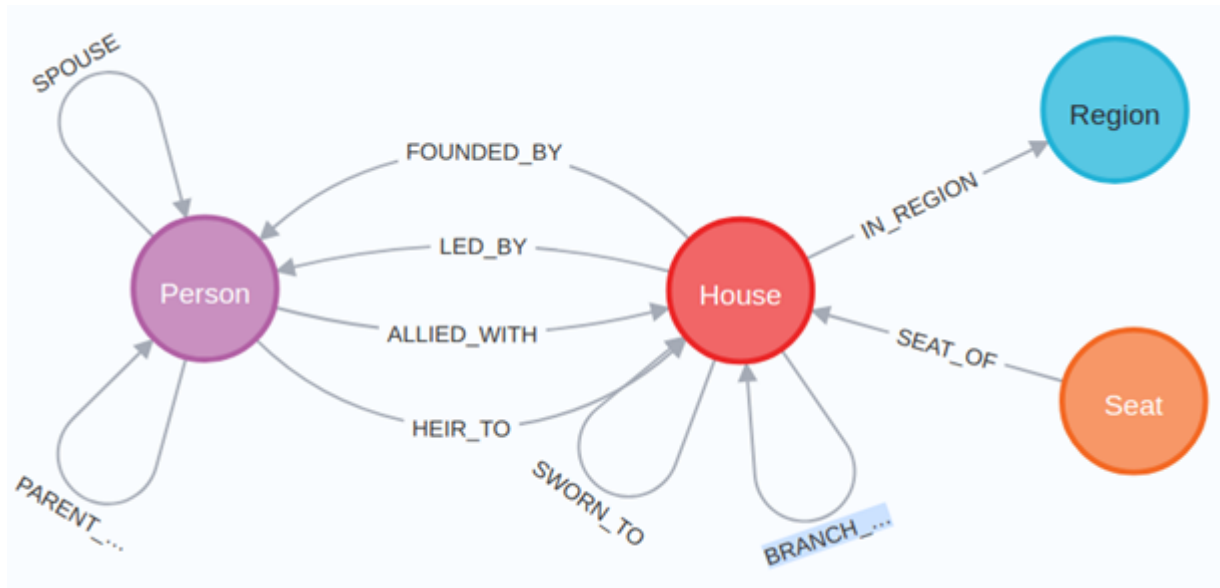
2.2 Contexto de los datos (no puntúa)

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 herederas 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.visualization()) 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 ALLIED_WITH House - -> personaje perteneciente o aliado a una casa
- Person HEIR_TO House - -> personaje que hereda el gobierno de la casa

- Person PARENT_OF 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.

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.

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).

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 (sin 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.

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.

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.

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).

SOLUCIÓN:

Consulta 1:

```
MATCH (p:Person)
WHERE 7 IN p.books
RETURN count(p)
```

| count(p) |
|----------|
| 71 |

La cláusula IN nos permite buscar dentro del array de números de libros al libro 7, simplemente contaremos cuántos personajes aparecen en y retornaremos el resultado

Consulta 2:

```
MATCH (sworn:House)-[:SWORN_TO]->(h:House)
RETURN h.name as BigHouse, count(sworn) as SwornHouses
ORDER BY SwornHouses DESC
LIMIT 1
```

| BigHouse | SwornHouses |
|------------------------------|-------------|
| "House Tyrell of Highgarden" | 59 |

Necesitamos relacionar los nodos House a través de la relación SWORN_TO para conocer las relaciones de lealtad y así poder contar las casas reales para poderlas presentar.

Consulta 3:

```
MATCH (h:House)
WHERE not exists ((:House)-[:SWORN_TO]->(h)) and
toLower(h.coatOfArms) contains "lion"
```

RETURN h.name, h.coatOfArms

| "h.name" | "h.coatOfArms" |
|--------------------------------|---|
| "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 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." |

Los requisitos de esta consulta precisan filtrar aquellos nodos que pertenecen a casas importantes a las que otras de menor poder han jurado lealtad. Para ello utilizamos **not exists** sobre la relación SWORN_TO entre nodos etiquetados como House para descartar a las grandes casas reales y buscar en la descripción del escudo de armas del resto la palabra “lion”.

Por precaución se convierte el texto de la descripción a minúsculas ya que si esta palabra aparece al comienzo de una frase aparecería como Lion y no sería considerada. Además, se añade un espacio en blanco al principio para evitar coincidencias incorrectas como por ejemplo “stallion”.

Aún así, la condición no es absolutamente correcta para cualquier caso, pues podrían producirse falsas coincidencias con palabras que tuvieran “lion” como prefijo (por ejemplo “lionel”), aspecto que no se da en ningún resultado. Pero sí cubre los casos en se finaliza con signos de puntuación al final de la frase (por ejemplo “lion.”, “lion,”, “lion’s”, etc). También podría darse el caso de que la palabra “lion” estuviera precedida de otros signos como (lion) o “lion”, o en textos en castellano podríamos tener casos como ¡león! (estos casos no se dan en los resultados de esta práctica) esta condición tampoco funcionará. Además, en inglés la

expresión “lions' heads” es gramaticalmente correcta y sería un resultado que deberíamos incluir en la consulta.

Debido a la casuística real del texto con el que trabajamos se acepta como correcto poner “lion”.

La solución más óptima (y no necesaria para tener un 10) sería el uso de una expresión regular compleja con el operador \approx . Pero no es el objetivo de esta práctica entrar en el uso avanzado de las expresiones regulares de Neo4j, que debería de tener en cuenta un completo conjunto de formas de redacción como la excepción gramatical detallada anteriormente. Por tanto una expresión regular que cubra los casos reales que podemos encontrar se considerará como válida.

Estos son los problemas que os podéis encontrar buscando coincidencias de texto tanto en esta base de datos como en cualquier otra.

Consulta 4:

```
MATCH (h:House)
WHERE h.diedOut is null and h.founded is not null
RETURN h.founded as Founded, count(h) as NumHouses, collect(h.name)
as CurrentHouses
ORDER BY NumHouses DESC
LIMIT 2
```

| Founded | NumHouses | CurrentHouses |
|-----------------|-----------|---|
| "Age of Heroes" | 9 | ["House Stark of Winterfell", "House Lannister of Casterly Rock", "House Bracken of Stone Hedge", "House Greyjoy of Pyke", "House Blackwood of Raventree Hall", "House Royce of Runestone", "House Bolton of the Dreadfort", "House Massey of Stonedance", "House Casterly of Casterly Rock"] |
| "299 AC" | 5 | ["House Lannister of Darry", "House Frey of Riverrun", "House Tyrell of Brightwater Keep", "House Baelish of Harrenhal", "House Foote of Nightsong"] |

En esta consulta necesitamos validar que el nodo House posee el atributo con información sobre su fecha de creación y a la vez no contiene el atributo con la fecha de su extinción. Una vez filtrados los nodos se presenta la fecha de creación, el número de casas que se fundaron y aún existen, y una agrupación de los nombres de estas casas. Limitando el resultado a los dos años con mayor número de casas aún existentes.

Consulta 5:

```

MATCH (branch:House)-[:BRANCH_OF]->(root:House)
WITH root.name as leadHouse, collect(branch.name) as branchHouses, count(branch) as
numBranchs
WHERE numBranchs > 3
RETURN leadHouse, branchHouses, numBranchs ORDER BY numBranchs DESC

```

| leadHouse | branchHouses | numBranchs |
|------------------------------------|--|------------|
| "House Lannister of Casterly Rock" | ["House Lannister of Darry", "House Lannister of Lannisport", "House Lannett", "House Lanny", "House Lantell"] | 5 |
| "House Harlaw of Harlaw" | ["House Harlaw of Harridan Hill", "House Harlaw of Grey Garden", "House Harlaw of the Tower of Glimmering", "House Harlaw of Harlaw Hall"] | 4 |
| "House Baratheon of Storm's End" | ["House Baratheon of Dragonstone", "House Baratheon of King's Landing", "House Bolling", "House Wensington"] | 4 |

En esta consulta la relación a utilizar entre casas es BRANCH_OF, se contarán el número de casas escindidas a la vez que se agrupan con collect sus nombres en un array de strings, a continuación filtramos los registros que no superen las 3 escisiones y se presentan los resultados ordenados según el criterio de la consulta.

Consulta 6:

```

MATCH (p:Person)-[pa:PARENT_OF]->(d:Person)
RETURN p.name as ParentName, pa.type as Parent, collect(d.name) as
DescendantNames, count(d) as DescCount, CASE size(p.children) WHEN NULL
THEN 0 ELSE size(p.children) END as SonsCount
ORDER BY DescCount DESC, SonsCount DESC
LIMIT 6

```

| ParentName | Parent | DescendantNames | DescCount | SonsCount |
|----------------------|----------|---|-----------|-----------|
| "Aegon III" | "father" | ["Daeron I", "Daena Targaryen", "Baelor I"] | 3 | 5 |
| "Daenaera Velaryon" | "mother" | ["Daeron I", "Baelor I", "Daena Targaryen"] | 3 | 0 |
| "Robert I Baratheon" | "father" | ["Joffrey Baratheon", "Tommen Baratheon"] | 2 | 8 |
| "Aenys I" | "father" | ["Jaehaerys I", "Alysanne Targaryen"] | 2 | 4 |
| "Daeron II" | "father" | ["Aerys I", "Maekar I"] | 2 | 4 |

| | | | | |
|-----------|----------|-------------------------|---|---|
| "Aegon I" | "father" | ["Aenys I", "Maegor I"] | 2 | 2 |
|-----------|----------|-------------------------|---|---|

En este caso necesitamos relacionar los nodos etiquetados como Person a través de la relación PARENT_OF para conocer las relaciones de parentesco con los gobernantes de la casa. En la propia relación encontramos la propiedad que sirve para conocer si se trata del padre o madre del gobernante. Si buscamos esta información por el atributo isFemale (pero siempre sobre los nodos "p" y no "d") la consulta podría ser incorrecta y puntuará menos, ya que al ser schemaless no podemos garantizar la existencia de dicho atributo en Person. Para satisfacer los requisitos de la consulta deben utilizarse las funciones collect (para guardar los nombres de los gobernantes, count (para contar las ocurrencias de la relación) y size para contar el número de posiciones de un array (utilizando una sentencia CASE para presentar un valor 0 en lugar de null) y facilitar el orden por el número total de descendientes. Se ordena descendientemente por descendientes coronados y, en caso de igual número de descendientes, se ordenará también descendientemente por número de hijos totales, limitando el resultado a los 6 primeros registros.

Consulta 7:

MATCH (p:Person{name:"Robert I Baratheon"})

UNWIND p.children **as** children

MATCH (son:Person{id:children})

RETURN son.name **as** name, son.born **as** born,

CASE son.died **WHEN** NULL **THEN** "aún vivo" **ELSE** son.died **END** **as** died,

CASE son.titles **WHEN** NULL **THEN** "sin títulos" **ELSE** son.titles **END** **as** titles,

CASE son.alias **WHEN** NULL **THEN** "sin alias" **ELSE** son.alias **END** **as** alias

| name | born | died | titles | alias |
|----------------------|--------------------------------|---------------------------------------|---|---|
| "Joffrey Baratheon" | "286 AC, at King's Landing" | "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-Baratheon"] |
| "Myrcella Baratheon" | "In 290 AC, at King's Landing" | "aún vivo" | ["Princess"] | "sin alias" |
| "Tommen Baratheon" | "291 AC" | "aún vivo" | ["King of the Andals, the Rhoynar and | ["The Boy King"] |

| | | | | |
|---------------|---|--------------------------------|---|---|
| | | | the First Men", "Lord of the Seven Kingdoms"] | |
| "Mya Stone" | "In 279 AC or 280 AC, at The Vale of Arryn" | "aún vivo" | "sin títulos" | "sin alias" |
| "Bella" | "In 283 AC, at Stony Sept" | "aún vivo" | "sin títulos" | "sin alias" |
| "Gendry" | "In 284 AC, at King's Landing" | "aún vivo" | ["Ser"] | ["The Bull", "Ser Gendry of the hollow hill"] |
| "Edric Storm" | "In 287 AC" | "aún vivo" | "sin títulos" | "sin alias" |
| "Barra" | "In 298 AC, at King's Landing" | "In 299 AC, at King's Landing" | "sin títulos" | "sin alias" |

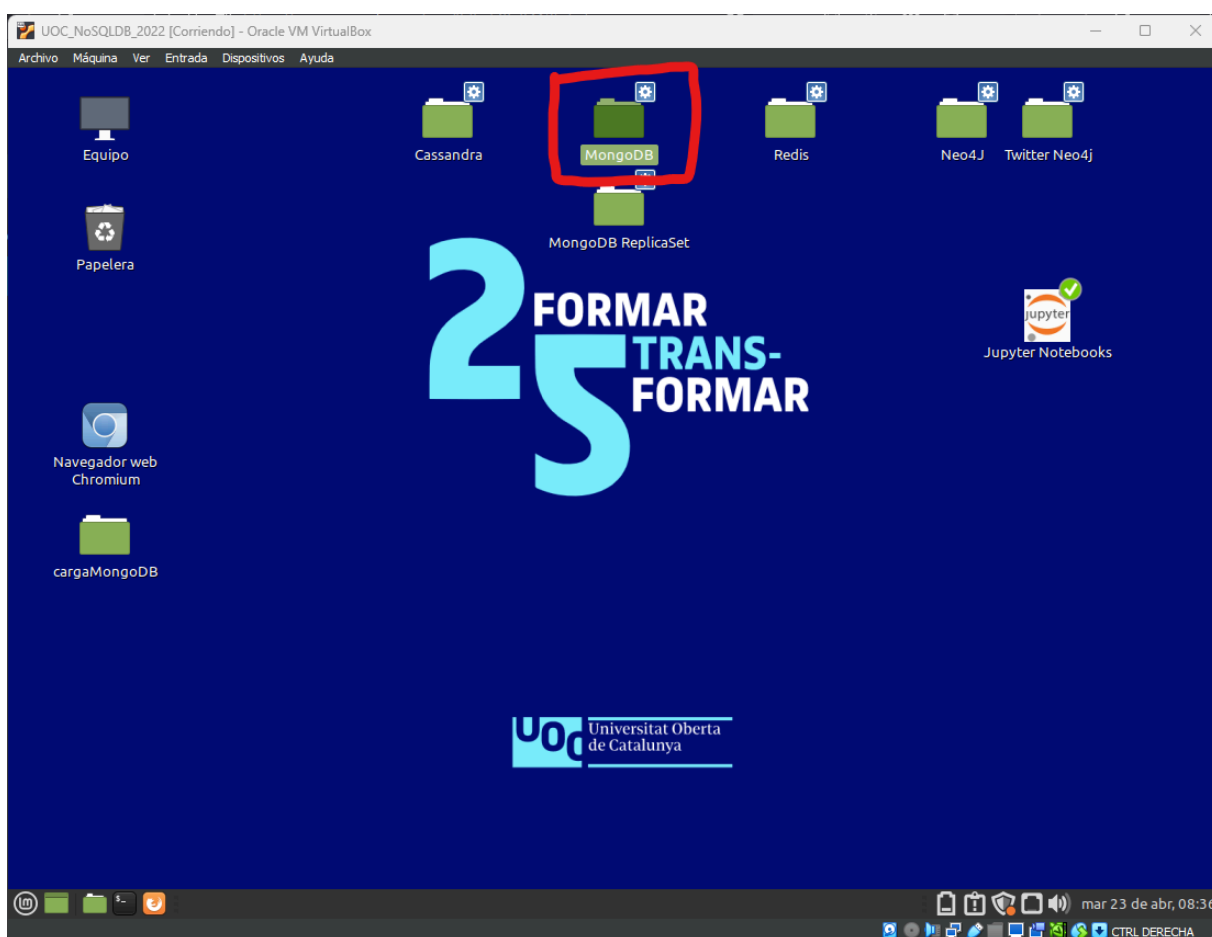
En esta consulta comenzamos obteniendo el nodo con el nombre del personaje a estudiar. Accedemos al array que contiene los identificadores de sus hijos y a través de UNWIND descomponemos el array en atributos individuales que utilizaremos en la consulta MATCH donde uno por uno obtenemos los nodos particulares de cada hijo a través de su ID para finalizar presentando los datos correspondientes a cada hijo y requeridos por la consulta. Para mejorar la presentación de los resultados utilizamos las sentencias CASE para cambiar los valores NULL por unos textos más adecuados.

Ejercicio 3: MongoDB (35%)

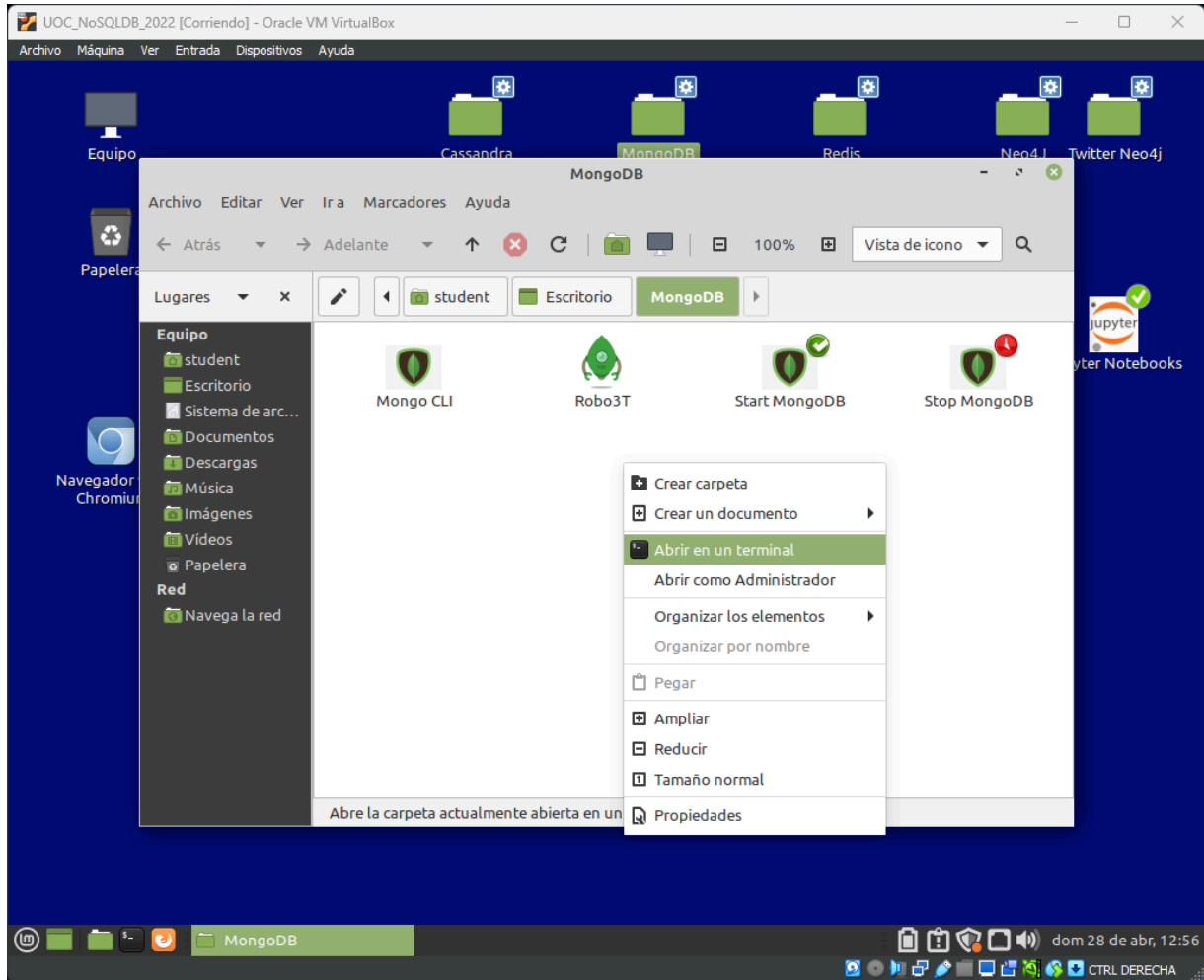
Para crear la base de datos necesaria para realizar todos los ejercicios de MongoDB, deberéis seguir las instrucciones detalladas en el apartado MongoDB disponible en el manual “Uso de máquina virtual”. No confundir con el apartado MongoDB replicaset, este apartado no será necesario para esta práctica.

Carga de datos

Nos situamos en la carpeta de MongoDB haciendo doble clic sobre el icono MongoDB (no usar la carpeta MongoDB ReplicaSet).



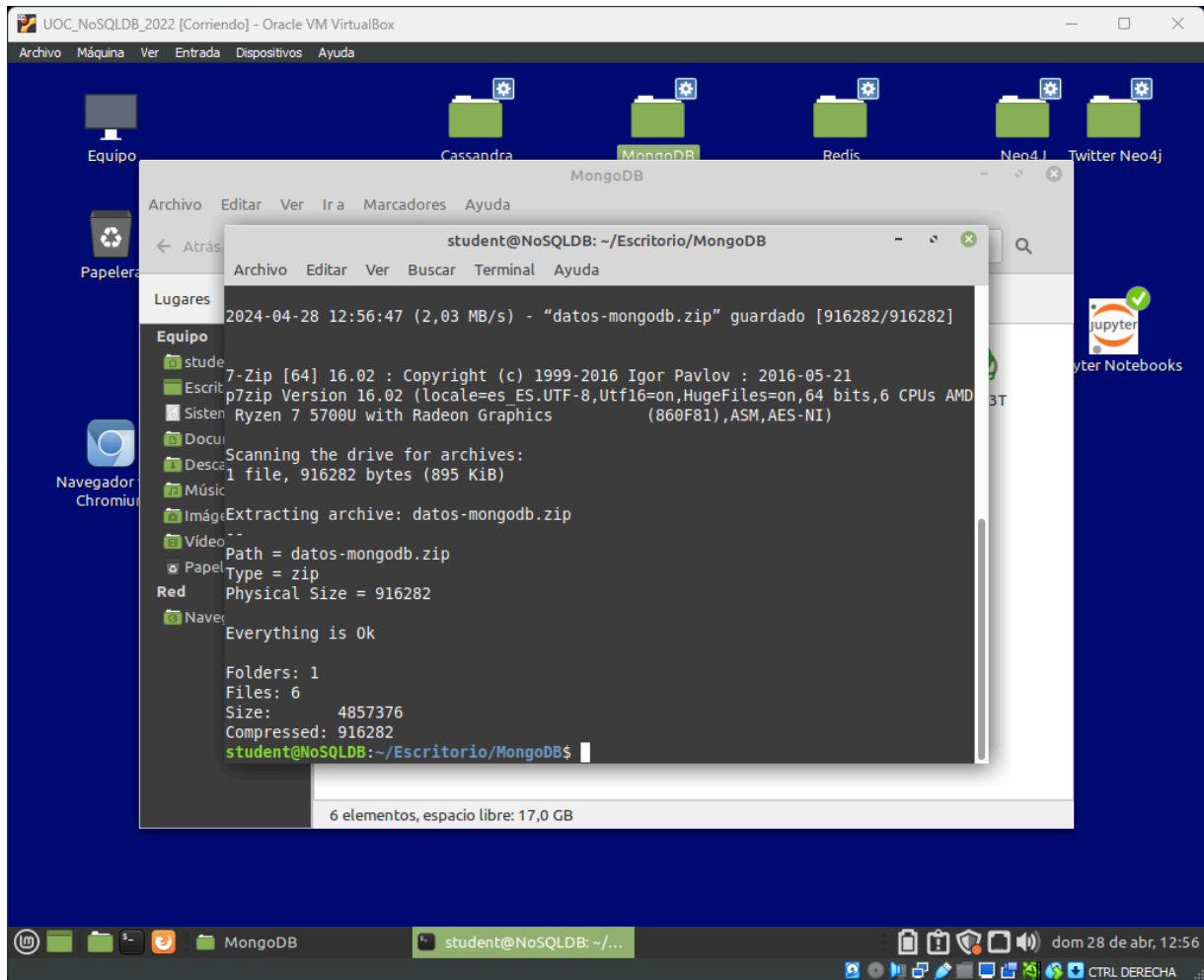
Hacemos clic con el botón derecho en la carpeta y en Abrir en una terminal.



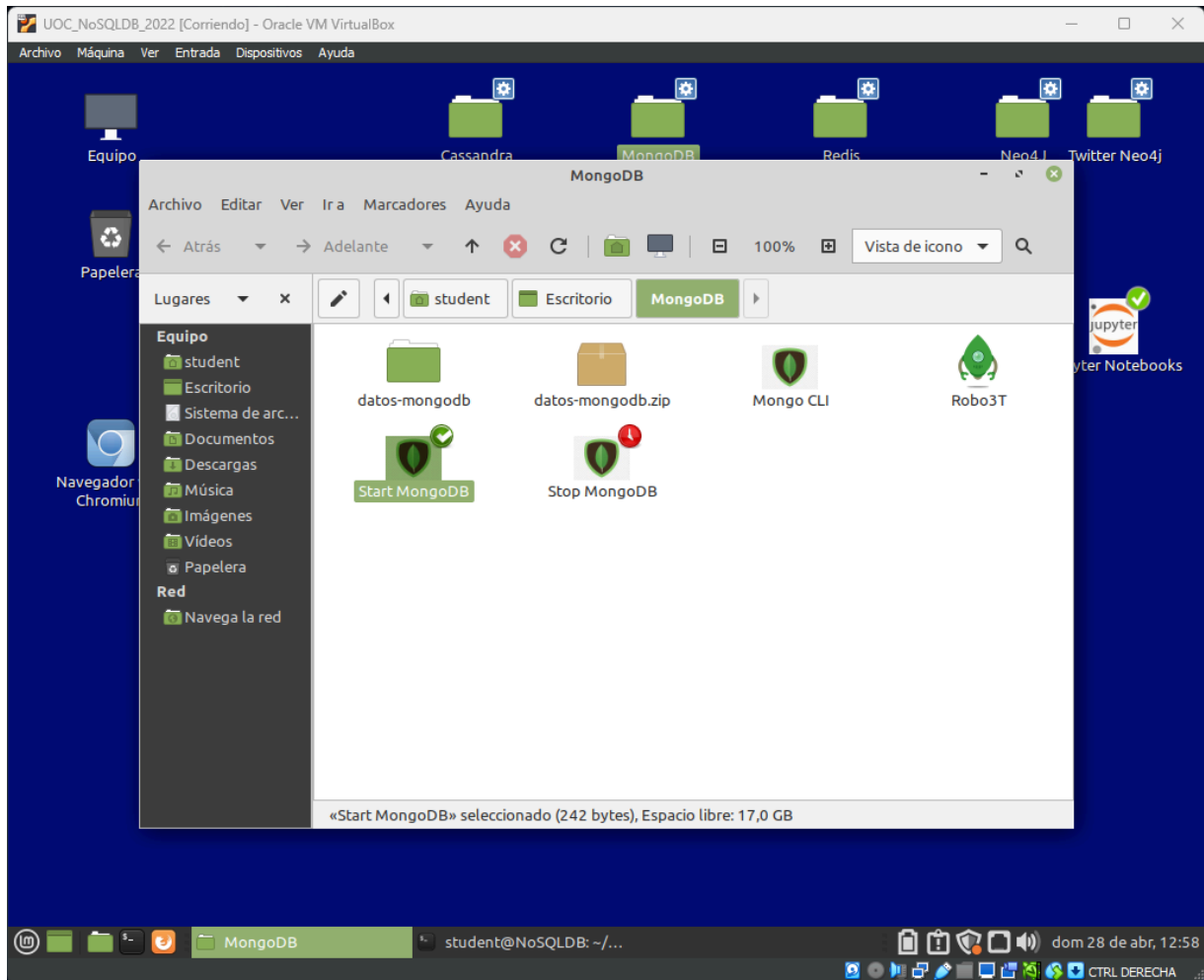
Copiamos y ejecutamos el siguiente comando:

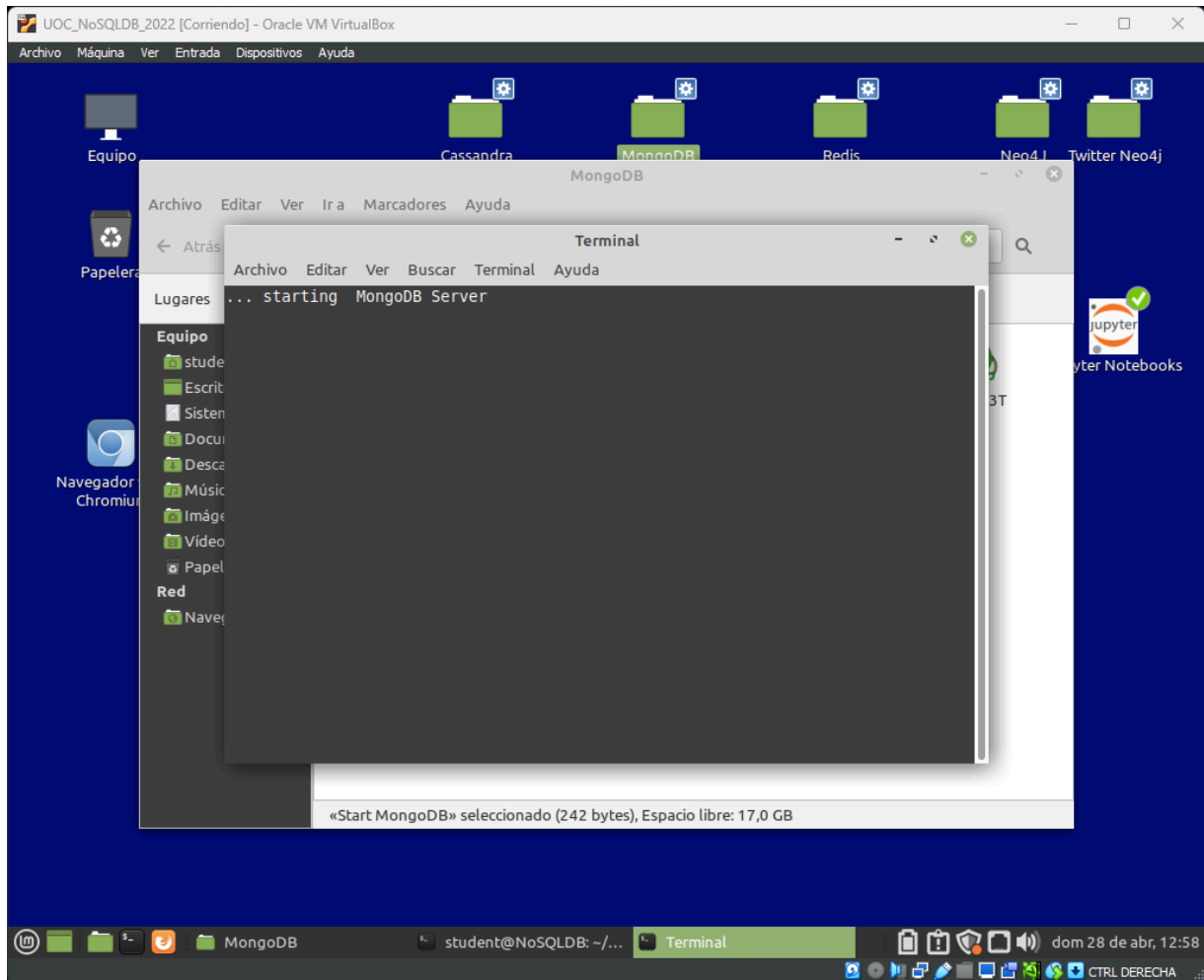
```
wget https://github.com/uoc-bbdd-no-tradicionales/\
docker-compose-replicaset/\
raw/pr-202302/datos-mongodb.zip && 7z x datos-mongodb.zip
```

El resultado final será:



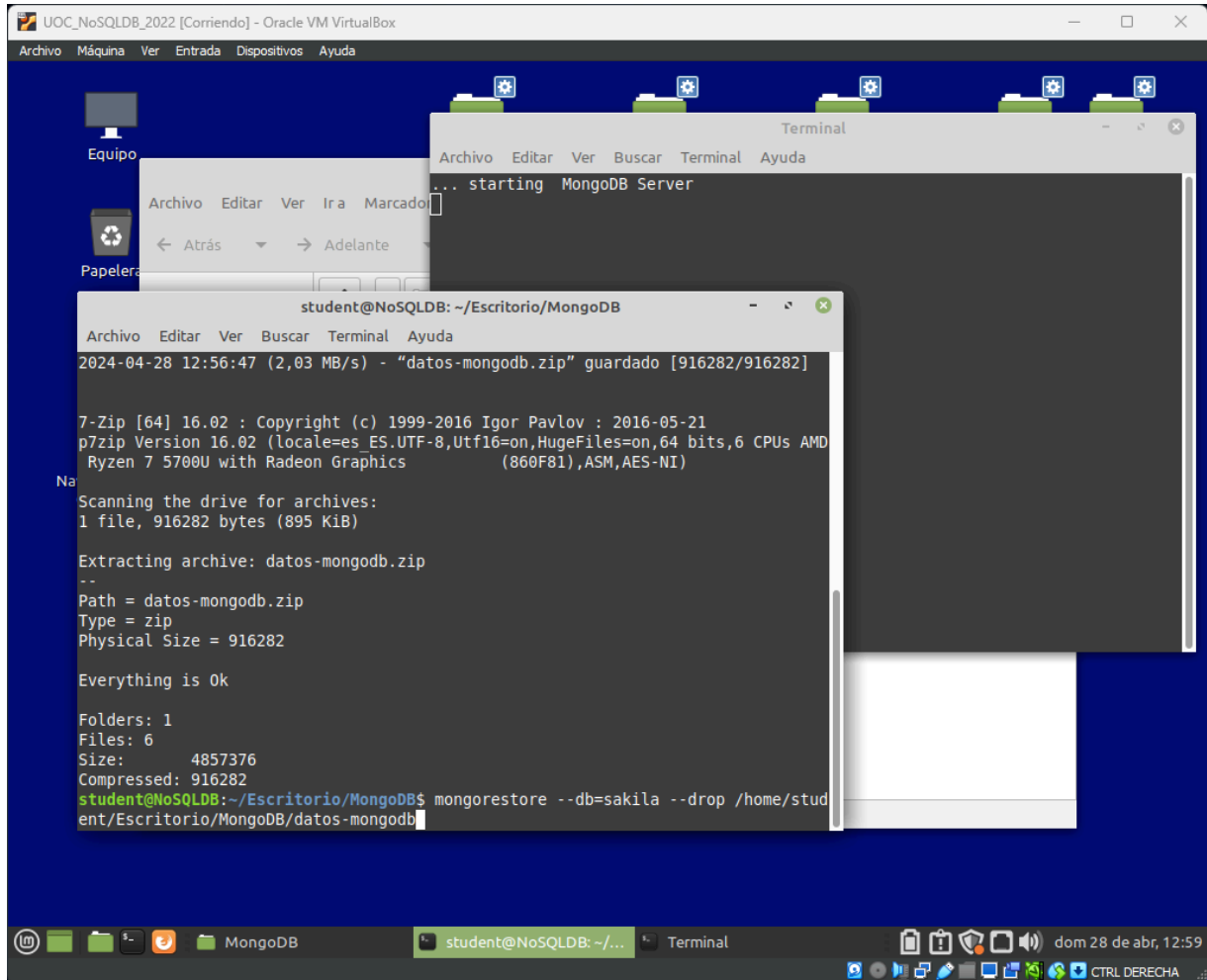
Hacemos doble clic sobre el icono de la misma carpeta “Start MongoDB” que abrirá otra ventana de terminal. **No cerréis esta ventana de terminal pues esto detendría la instancia de MongoDB**, pero sí que lo podemos minimizar.



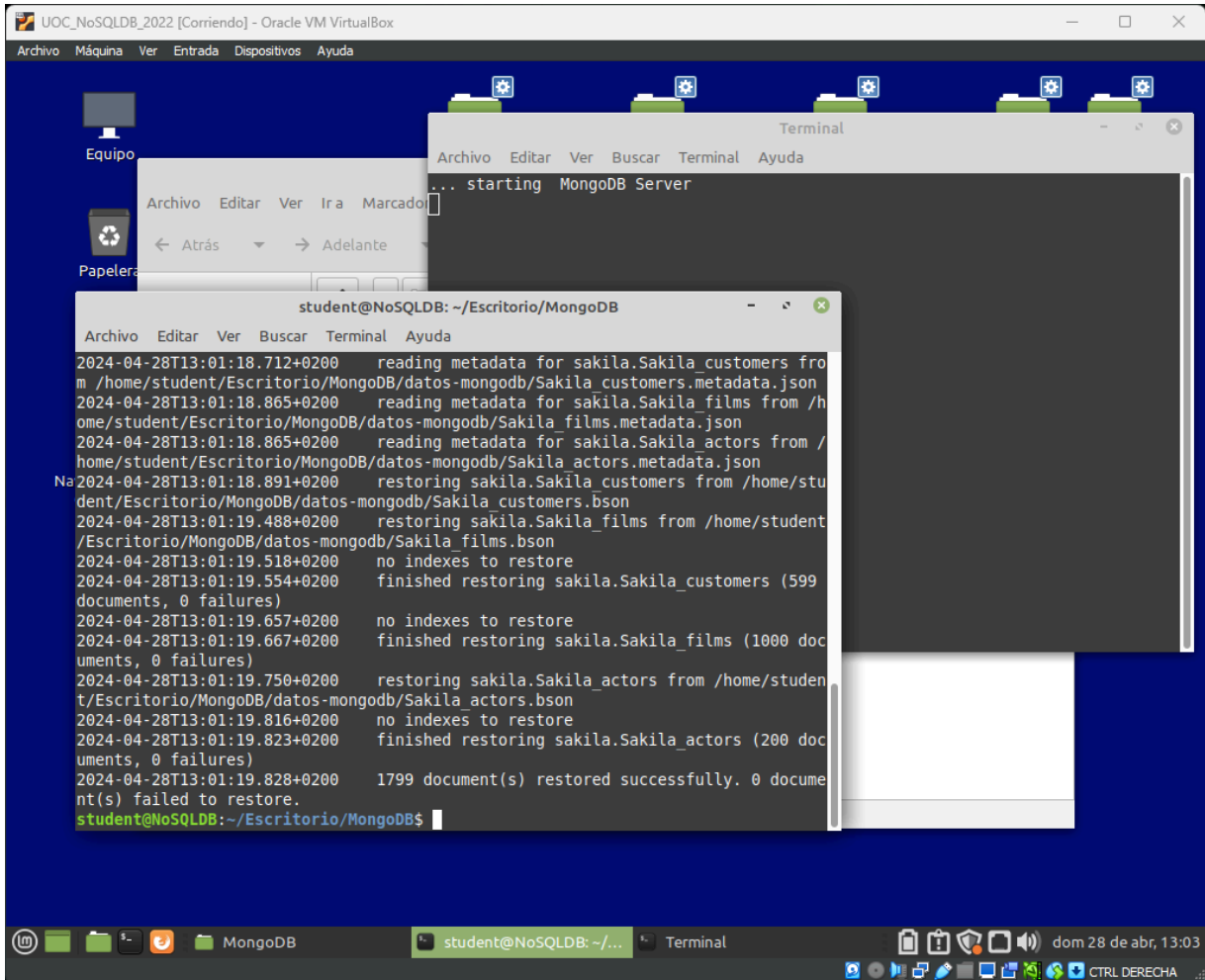


En la ventana del terminal anterior (donde hemos ejecutado el comando para descargar y descomprimir los archivos) copiamos y pegamos el siguiente comando:

```
mongorestore --db=sakila --drop
/home/student/Escritorio/MongoDB/datos-mongodb
```



Que nos dará como resultado:



```

... starting MongoDB Server

student@NoSQLDB: ~/Escritorio/MongoDB
2024-04-28T13:01:18.712+0200 reading metadata for sakila.Sakila_customers from /home/student/Escritorio/MongoDB/datos-mongodb/Sakila_customers.metadata.json
2024-04-28T13:01:18.865+0200 reading metadata for sakila.Sakila_films from /home/student/Escritorio/MongoDB/datos-mongodb/Sakila_films.metadata.json
2024-04-28T13:01:18.865+0200 reading metadata for sakila.Sakila_actors from /home/student/Escritorio/MongoDB/datos-mongodb/Sakila_actors.metadata.json
2024-04-28T13:01:18.891+0200 restoring sakila.Sakila_customers from /home/student/Escritorio/MongoDB/datos-mongodb/Sakila_customers.bson
2024-04-28T13:01:19.488+0200 restoring sakila.Sakila_films from /home/student/Escritorio/MongoDB/datos-mongodb/Sakila_films.bson
2024-04-28T13:01:19.518+0200 no indexes to restore
2024-04-28T13:01:19.554+0200 finished restoring sakila.Sakila_customers (599 documents, 0 failures)
2024-04-28T13:01:19.657+0200 no indexes to restore
2024-04-28T13:01:19.667+0200 finished restoring sakila.Sakila_films (1000 documents, 0 failures)
2024-04-28T13:01:19.750+0200 restoring sakila.Sakila_actors from /home/student/Escritorio/MongoDB/datos-mongodb/Sakila_actors.bson
2024-04-28T13:01:19.816+0200 no indexes to restore
2024-04-28T13:01:19.823+0200 finished restoring sakila.Sakila_actors (200 documents, 0 failures)
2024-04-28T13:01:19.828+0200 1799 document(s) restored successfully. 0 document(s) failed to restore.
student@NoSQLDB:~/Escritorio/MongoDB$
  
```

Una vez terminada la carga de la base de datos, en el mismo terminal podemos ejecutar el comando:

```
mongo
```

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

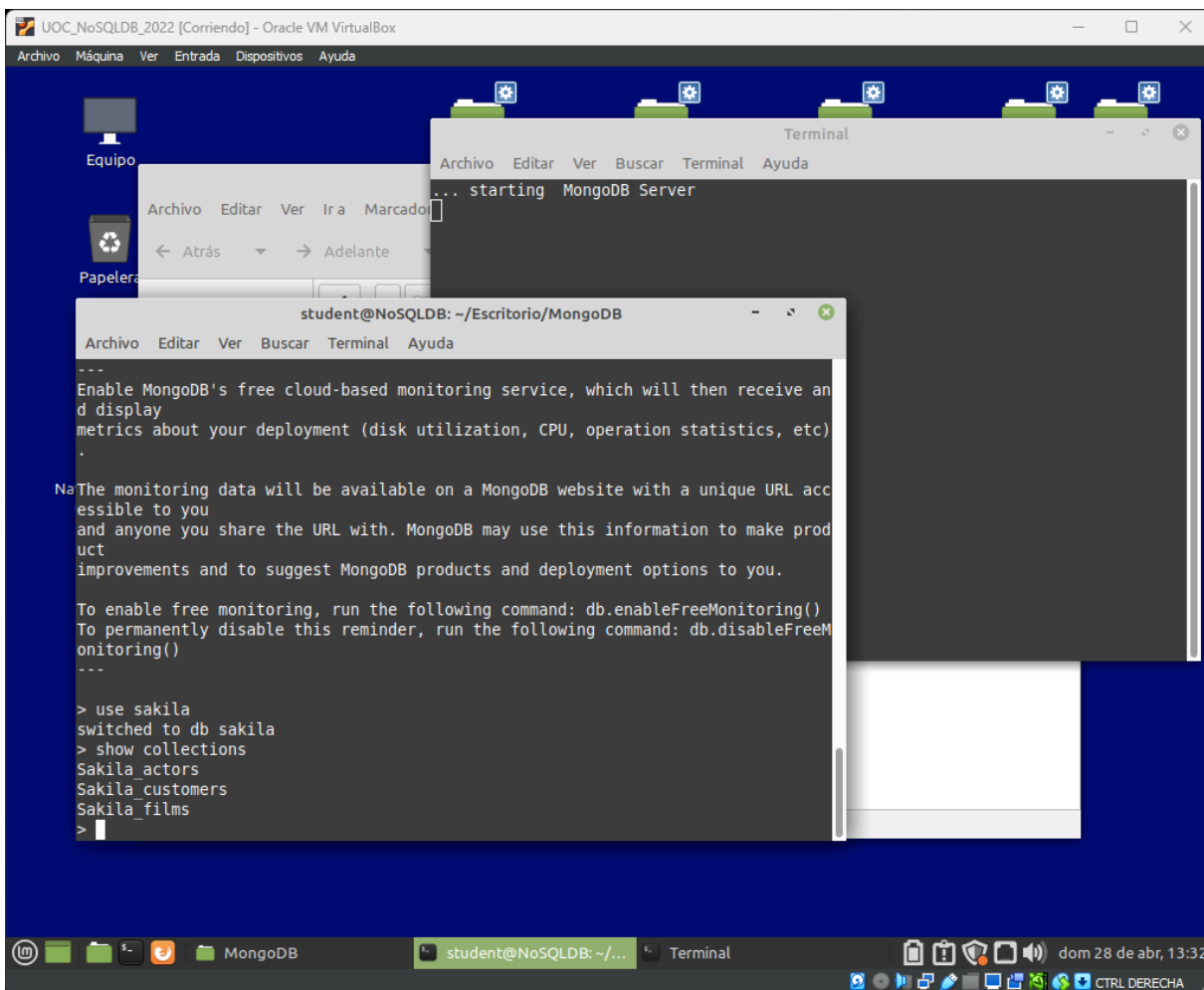
```
show collections
```

Que nos debe retornar:

```
> show collections
Sakila_actors
```

```
Sakila_customers
Sakila_films
>
```

O como se ve en la siguiente captura de pantalla:



También podéis hacer doble clic en el icono con nombre “Mongo CLI” que nos proporciona la shell desde la cual podremos acceder a MongoDB. Recordad que antes de hacer doble clic en “Mongo CLI” se tiene que haber iniciado el servicio haciendo clic en el icono “Start MongoDB” y minimizando el terminal que se abre.

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

Ya estamos en la base de datos recién cargada y ya podemos hacer las consultas.

En todo momento podéis comprobar si estáis en la base de datos correcta escribiendo el comando `db` que indica en qué base de datos se está trabajando actualmente. Por ejemplo:

```
> db
```

Retorna:

```
sakila
>
```

Por lo tanto estamos en la base de datos correcta. Si el comando retorna otra base de datos como por ejemplo:

```
> db
test
>
```

Indica que estáis en la base de datos de test, por lo tanto no encontraréis las colecciones cargadas en la base de datos sakila. **Tened en cuenta que MongoDB es *case sensitive* (sensible a mayúsculas y minúsculas) en todos los comandos que se ejecutan, incluyendo los nombres de los atributos cuando se especifican en las consultas.** Por lo tanto, la base de datos `Sakila` (incorrecta) no es la misma que `sakila` (correcta). Si tecleáis por error `use Sakila` en lugar de `use sakila` os habréis situado en una base de datos vacía llamada `Sakila`.

Finalmente, recordad que después de reiniciar no es necesario repetir los pasos para cargar la base de datos porque los datos ya se guardan en la máquina virtual. **Recordad también que al volver a realizar la carga perderéis los cambios realizados.** En otras palabras, si volvéis a ejecutar el comando `mongorestore` la base de datos volverá a restaurarse y perderéis todo el trabajo hecho, aunque es útil en aquellos casos en los que se quiera empezar de cero.

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" : "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"
}
>
```

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.

SOLUCIÓN:

Hay dos maneras de hacerlo, con la instrucción `distinct` y con el `aggregation pipeline`. Según como se haga con el `distinct` no se puede ordenar el resultado, de manera que la consulta no será correcta.

La consulta con la instrucción `distinct`:

```
db.Sakila_films.distinct("Rating");
```

Retorna:

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


Si intentamos ordenar el resultado de manera descendente con la función `sort` al final de la expresión, nos dará error.

```
db.Sakila_films.distinct("Rating").sort({"Rating":-1});
```

Retorna:

```
2024-05-05T00:11:12.992+0200 E QUERY [js] uncaught exception:
TypeError: invalid Array.prototype.sort argument :
@(shell):1:1
```

La consulta con la función `distinct()` concatenándola con la función `reverse()` es válida, pero si no se revierte el orden con `reverse()`, la consulta no estará en el orden especificado en el enunciado, por lo que sería incorrecta. La consulta:

```
db.Sakila_films.distinct("Rating").sort().reverse();
```

Retorna:

```
[ "R", "PG-13", "PG", "NC-17", "G" ]
```

Y es una solución correcta.

Por otro lado, se podría crear un script en la interfaz de comandos para resolver el problema, pero dependiendo del volumen de datos, es demasiado procesado para la interfaz de comandos ya que los datos se tendrían que cargar en memoria para procesarlos. La solución más eficiente y consistente es que la consulta sea procesada por la base de datos y retorne los datos de la manera deseada, por lo que el aggregation pipeline es la mejor solución. La consulta con el aggregation pipeline es:

```
db.Sakila_films.aggregate([
  { $project: { _id: 0, Rating:1 } },
  { $group: { _id: "$Rating" } },
  { $sort: { _id: -1 } }
]);
```

Que retorna:

```
{ "_id" : "R" }
{ "_id" : "PG-13" }
{ "_id" : "PG" }
{ "_id" : "NC-17" }
{ "_id" : "G" }
>
```

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.

SOLUCIÓN:

Esta consulta se puede ejecutar directamente sin necesidad del aggregation pipeline. Ejecutamos la consulta:

```
db.Sakila_films.find({ Rating: "NC-17" }).count();
```

Que retorna:

210

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.

SOLUCIÓN:

Como indica el enunciado, para obtener una agrupación inequívoca se debe agrupar por el campo actorId, puesto que si se intenta agrupar por los campos nombre y apellido la agrupación no es correcta (son campos separados que no forman parte de un registro). Tampoco es correcta si se intenta agrupar por el documento del Array completo, es un error que comete la inteligencia artificial al generar la consulta (sí, nosotros también la hemos probado para resolver los ejercicios y ver cuáles son los fallos). Al final de la resolución del ejercicio se adjuntan dos ejemplos con un mal recuento.

Primero obtenemos el recuento de los tres actores que aparecen en más películas junto con su identificador. La consulta:

```
db.Sakila_films.aggregate([
  { $unwind: "$Actors" },
```

```
{ $group: { _id: {"actorId": "$Actors.actorId"}, count: { $sum: 1 }
} },
{ $sort: { count: -1 } },
{ $limit: 3 }
]);
```

Que retorna:

```
{ "_id" : { "actorId" : 107 }, "count" : 42 }
{ "_id" : { "actorId" : 102 }, "count" : 41 }
{ "_id" : { "actorId" : 198 }, "count" : 40 }
>
```

Procedemos a buscar los actores en base a su identificador en la colección de actores con la consulta:

```
db.Sakila_actors.find({
  $or:[{_id:107}, {_id:102}, {_id:198}],
  {"FirstName":1, "LastName":1});
```

Que retorna:

```
{ "_id" : 102, "FirstName" : "WALTER", "LastName" : "TORN" }
{ "_id" : 107, "FirstName" : "GINA", "LastName" : "DEGENERES" }
{ "_id" : 198, "FirstName" : "MARY", "LastName" : "KEITEL" }
```

Esta es una de las dificultades que presentan algunas de las bases de datos no relacionales NoSQL. Si se tienen que relacionar datos entre varias colecciones para una consulta (hacer una *join*), tendremos que relacionar nosotros mismos los datos con varias consultas. La base de datos no implementa operaciones de *join* entre tablas.

Al principio de la solución se ha mencionado que si se agrupa por nombre y apellidos se obtiene un resultado incorrecto. Esta es la consulta:

```
db.Sakila_films.aggregate([
  { $unwind: "$Actors" },
  { $group: { _id: {"First Name": "$Actors.First name", "Last
name": "$Actors.Last name"}, count: { $sum: 1 } } },
  { $sort: { count: -1 } },
  { $limit: 3 }
]);
```

Que retorna:

```
{ "_id" : { "First Name" : "SUSAN", "Last name" : "DAVIS" }, "count"
: 54 }
```

```
{ "_id" : { "First Name" : "GINA", "Last name" : "DEGENERES" },
"count" : 42 }
{ "_id" : { "First Name" : "WALTER", "Last name" : "TORN" }, "count"
: 41 }
>
```

Como se puede apreciar el recuento no es correcto, puesto que los dos campos agrupados no forman parte de la misma tupla como pasaría en una base de datos con datos estructurados en tablas.

Si intentamos agrupar por el array completo con una consulta generada por inteligencia artificial:

```
db.Sakila_films.aggregate([
  { $unwind: "$Actors" },
  { $group: { _id: "$Actors", count: { $sum: 1 } } },
  { $sort: { count: -1 } },
  { $limit: 3 }
]);
```

Retorna:

```
{ "_id" : { "First name" : "WALTER", "Last name" : "TORN", "actorId"
: 102 }, "count" : 35 }
{ "_id" : { "First name" : "MARY", "Last name" : "KEITEL", "actorId"
: 198 }, "count" : 32 }
{ "_id" : { "First name" : "SCARLETT", "Last name" : "DAMON",
"actorId" : 81 }, "count" : 32 }
>
```

Que como podemos ver es otro resultado totalmente diferente de los anteriores y tampoco es correcto porque agrupa por el objeto array completo. La inteligencia artificial no siempre retorna el resultado correcto y siempre se tienen que supervisar las respuestas que proporciona.

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.

SOLUCIÓN:

La consulta puede ejecutarse de dos maneras, agrupando por título o agrupando por identificador de película. Al tratarse de un solo campo y no haber dos películas con el mismo título, ambas consultas retornan el mismo recuento. Sin embargo, especialmente en el caso de los *remakes*, hay películas que son diferentes y que tienen el mismo título. Es un caso

plausible que debemos considerar. Por lo que la consulta es más consistente si se agrupa por Identificador y después se obtiene el nombre en otra consulta.

En el caso de buscar por título ejecutaríamos:

```
db.Sakila_films.aggregate([
  { $unwind: "$Actors" },
  { $group: { _id: "$Title", count: { $sum: 1 } } },
  { $sort: { count: -1 } },
  { $limit: 1},
]);
```

Que retorna directamente el resultado esperado:

```
{ "_id" : "LAMBS CINCINATTI", "count" : 15 }
```

Si optamos por agrupar por identificador tendremos que la siguiente consulta:

```
db.Sakila_films.aggregate([
  { $unwind: "$Actors" },
  { $group: { _id: "$_id", count: { $sum: 1 } } },
  { $sort: { count: -1 } },
  { $limit: 1},
]);
```

Retorna:

```
{ "_id" : 508, "count" : 15 }
```

Por lo tanto, tendremos que hacer la siguiente consulta para obtener el título de la película:

```
db.Sakila_films.find({_id:508}, {"Title" :1, "_id":0});
```

Que retorna:

```
{ "Title" : "LAMBS CINCINATTI" }
```

Esta segunda opción es más consistente ya que contempla todos los casos de uso.

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.
- 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.

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.

SOLUCIÓN:

Esta consulta o ejercicio puede resolverse de dos maneras.

OPCIÓN A:

Una opción sería indicar y justificar que no hace falta hacer nada y se puede pasar directamente al ejercicio siguiente. Si observáis un detalle del enunciado de la pregunta siguiente se puede obtener una “pista” que os lleve a esta conclusión:

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.

La justificación es que la base de datos es Schemaless. Por lo tanto, no es necesario inicializar ningún atributo en los alquileres. Aquellos alquileres que tienen valoración tendrán un campo con una valoración y aquellos que no tienen valoración no será necesario que tengan ningún campo inicializado a -1. Todos los documentos pueden coexistir con diferentes atributos sin problemas. MongoDB es Schemaless y esta es una de sus ventajas.

En tal caso, también se debe indicar que las aplicaciones que consultan y operan con esta base de datos deberán estar preparadas para recibir y realizar operaciones con documentos de diferentes estructuras. Dependiendo de la consulta, se recibirán documentos que pueden contener el campo informado y otros documentos que pueden no tenerlo. En este caso, las aplicaciones deberán contemplar esta posibilidad y no dar error en caso de recibir campos inesperados o inicializar los campos cuando no estén presentes. Por lo tanto, los programas/usuarios que acceden a la base de datos deberán adaptar los programas para que tengan en cuenta que los dos casos (con valor -1 o sin atributo) son equivalentes. Esta es una de las desventajas, contrapartidas o *tradeoffs* de las bases de datos Schemaless.

Por otro lado, si se busca una mayor estabilidad y que la estructura de los datos sea más predecible para las aplicaciones, se tiene que proceder a una inicialización en toda la base

de datos (la otra solución). Como la base de datos es Schemaless, no será necesario alterar ningún esquema y se podrá inicializar el valor sin hacer ningún otro cambio previo.

Por lo tanto, el ejercicio siguiente puede resolverse perfectamente sin hacer nada en este. Si se justifica correctamente, no hacer nada es una opción válida.

De todos modos, el enunciado pedía la inicialización del atributo de manera explícita. Por lo tanto, la siguiente opción es perfectamente válida y es la que se espera en la respuesta.

OPCIÓN B:

Para la actualización de valores podemos utilizar las funciones `update`, `updateOne` y `updateMany`. Aunque para la resolución de la práctica no tiene ningún impacto en la valoración haber usado `update`, se debe tener en cuenta que esta función está marcada como *deprecated*. Eso significa que aunque funcione, no es recomendable usarla porque será retirada en futuras versiones. La función `updateOne` no nos sirve, pues tendremos que actualizar múltiples documentos. Por lo tanto, utilizaremos la función `updateMany`.

La consulta de modificación será:

```
db.Sakila_customers.updateMany({}, { $set: { "Rentals.$[].Rate": -1 }
});
```

Que retona:

```
{ "acknowledged" : true, "matchedCount" : 599, "modifiedCount" : 599
}
```

Seleccionamos el décimo cliente de la colección para comprobar que el campo se ha añadido correctamente. Como se indica en la advertencia del enunciado, la instrucción `skip` es en base 0. Por lo tanto, si utilizamos la función `skip(0)` obtendremos el primer cliente. Si utilizamos `skip(1)` obtendremos el segundo cliente y así sucesivamente, por lo que si queremos obtener el décimo cliente deberemos utilizar `skip(9)`.

La consulta no tiene filtro, pero se ordena la colección por el orden especificado y se limita el resultado a un solo elemento con la función `limit()`.

Finalmente usamos la función `pretty()` para que el resultado sea más legible.

```
db.Sakila_customers.find(
  {},
  {"Last Name":1, "Rentals":1, "_id":0}).sort(
  {"Last Name":1}).skip(9).limit(1).pretty();
```

Podemos ver cómo el atributo “Rate” se ha añadido en todos los elementos del array “Rentals” del cliente.

```
{
  "Last Name" : "ANDREWS",
  "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" : 2.990000009536743,
          "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" : 0.9900000095367432
            }
        ],
        "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" : 2.990000009536743,
            "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" : 0.9900000095367432,
            "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" : 1.9900000095367432,
                "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,

```

```

        "Payment Date" : "2005-07-12 01:33:25.0",
        "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" : 5.989999771118164,
            "Payment Date" : "2005-07-31 06:34:06.0",
            "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" : 2.990000009536743,
            "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
}
]
}

```

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?

SOLUCIÓN:

La colección ideal para guardar la media de las valoraciones es Sakila_films. En los documentos de esta colección podremos crear y mantener actualizado un campo cada vez que haya una modificación en las valoraciones del array de Rentals. El campo podría llamarse (por ejemplo) RateAvg.

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.
- 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.

SOLUCIÓN:

Para obtener la valoración media de la película con identificador 4 ejecutamos la consulta:

```

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

```

Que retorna:

```

[
  {
    "_id": 4,

```

```

    "avgRate": 3
  }
]

```

Lógicamente la media de una sola valoración de 3 es 3. Si os ha dado un valor diferente es que la consulta no está filtrando los valores -1 o está calculando la media de otras películas.

Para actualizar el valor de la media estableceremos el valor medio en un campo llamado RateAvg en la colección Sakila_films. Si el campo no existe en el documento se creará y si ya existe se actualizará con el valor que establezcamos. No hace falta actualizar todos los demás documentos de la colección, pues mongoDB no nos obliga a que este campo esté presente en todos los documentos (es schemaless). De esta manera, a medida que se vayan produciendo valoraciones se irán añadiendo y actualizando los campos con la media calculada según sea el caso. La misma consulta nos sirve tanto para crear el campo si no existe como para actualizarlo.

En el caso de una base de datos tradicional o con esquema definido, deberíamos modificar la estructura de la tabla antes de insertar este valor.

```

db.Sakila_films.updateOne({
  "_id": 4
}, {
  $set: {"RateAvg": 3}
});

```

Comprobamos con la consulta:

```

db.Sakila_films.find({"_id": 4});

```

Que retorna:

```

{
  "_id": 4,
  "Actors": [
    {
      "First name": "JODIE",
      "Last name": "DEGENERES",
      "actorId": 41
    },
    {
      "First name": "SCARLETT",
      "Last name": "DAMON",
      "actorId": 81
    },
    {
      "Last name": "PESCI",
      "actorId": 88,
      "First name": "KENNETH"
    }
  ]
}

```

```

    },
    {
      "First name": "FAY",
      "Last name": "WINSLET",
      "actorId": 147
    },
    {
      "First name": "OPRAH",
      "Last name": "KILMER",
      "actorId": 162
    }
  ],
  "Category": "Horror",
  "Description": "A Fanciful Documentary of a Frisbee And a
Lumberjack who must Chase a Monkey in A Shark Tank",
  "Length": "117",
  "Rating": "G",
  "Rental Duration": "5",
  "Replacement Cost": "26.99",
  "Special Features": "Commentaries,Behind the Scenes",
  "Title": "AFFAIR PREJUDICE",
  "RateAvg": 3
}

```

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 **16/06/24**.

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.