Uso de bases de datos NoSQL

PRA1

**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.

# 

# 

# 

# 

# 

# Estudiante: Lukaz Martin Doehne

# Ejercicio 1: Cassandra (30%)

## Descripción de la base de datos y enunciado

Dada la importancia del cambio climático y el impacto que pueden tener los coches eléctricos en la reducción de emisiones, el gobierno del Estado de Washington (no confundir con la ciudad de Washington DC) ha publicado datos sobre los coches híbridos y eléctricos. El conjunto original de datos muestra los vehículos eléctricos con batería (BEV) y los vehículos eléctricos híbridos enchufables (PHEV) que están actualmente registrados en el Departamento de Licencias del Estado de Washington.

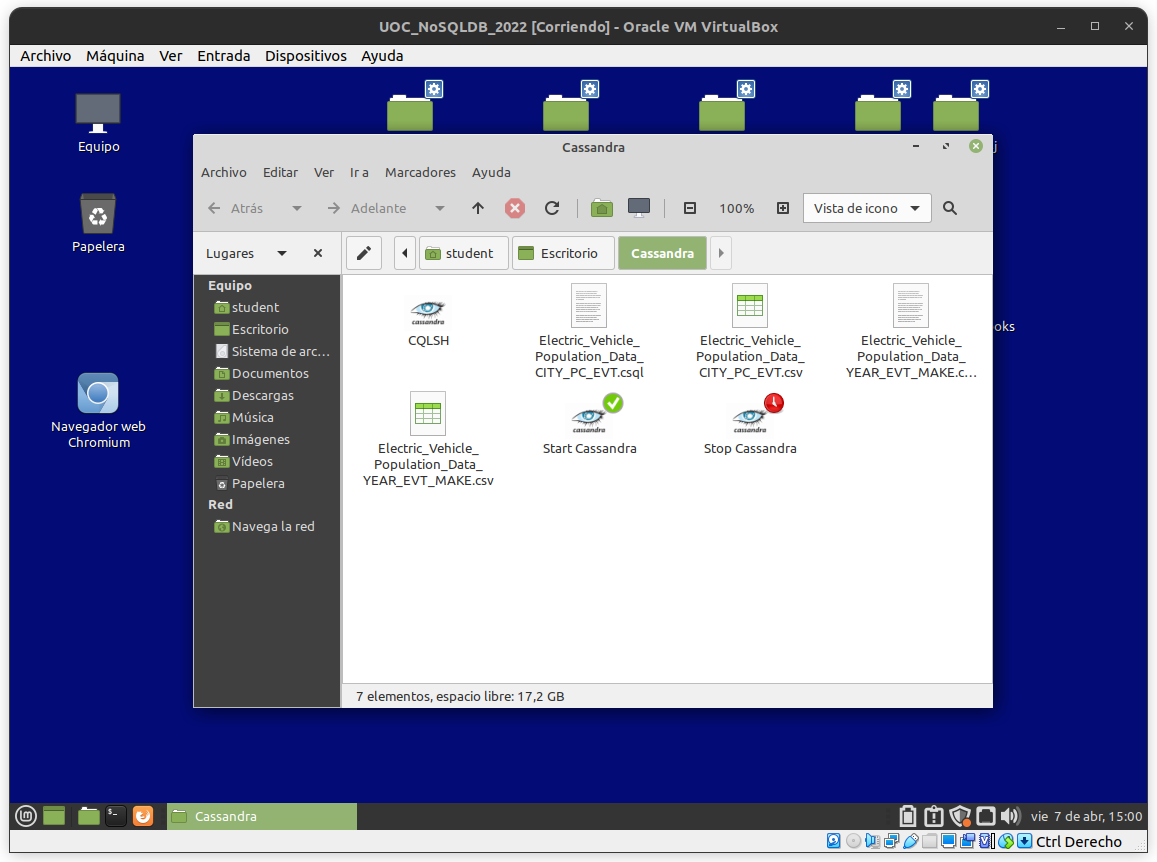
Con el objetivo de analizar los datos y escalar la base de datos a nivel de todos los Estados Unidos, se han construido unos agregados con toda la información disponible. Los agregados se pueden ver en la siguiente sección de carga de datos y con ellos se realizará el apartado de Cassandra de esta práctica.

## Carga de datos

Para comenzar con la carga, tenéis que cargar los siguientes archivos en la máquina virtual:

* Electric\_Vehicle\_Population\_Data\_CITY\_PC\_EVT.csv
* Electric\_Vehicle\_Population\_Data\_CITY\_PC\_EVT.csql
* Electric\_Vehicle\_Population\_Data\_YEAR\_EVT\_MAKE.csql
* Electric\_Vehicle\_Population\_Data\_YEAR\_EVT\_MAKE.csv

Estos archivos están disponibles junto al enunciado en la carpeta Cassandra, y tendrán que quedar ubicados tal y como se ve en la imagen siguiente. En los vídeos detallados al principio del enunciado encontraréis indicaciones sobre cómo copiar y pegar archivos dentro de la máquina virtual. Dado el tamaño de los archivos, es posible que solamente funcione la técnica de “arrastrar y soltar”.



Después, se deberán ejecutar los siguientes comandos CQL para crear y rellenar las tablas o familias de columnas. Las instrucciones siguientes parten de una conexión a la línea de comandos de la base de datos y con el servicio de Cassandra ya arrancado. Partiremos del siguiente símbolo del sistema (*command prompt*) en la línea de comandos. Para llegar a este punto podéis seguir el manual “Uso de máquina virtual - Bases de datos NoSQL”.

cqlsh>

Primero creamos un keyspace llamado ELECTRIC\_CARS con el siguiente comando:

CREATE KEYSPACE ELECTRIC\_CARS WITH REPLICATION ={ 'class': 'SimpleStrategy','replication\_factor': 1};

Indicamos que queremos utilizar el keyspace creado ejecutando:

USE ELECTRIC\_CARS;

Que nos cambiarà el *command prompt* a:

cqlsh:electric\_cars>

Una vez estamos en el keyspace correcto, creamos la primera agregación Electric\_Vehicle\_Population\_Data\_CITY\_PC\_EVT ejecutando el archivo Electric\_Vehicle\_Population\_Data\_CITY\_PC\_EVT.csql que encontraréis junto al enunciado. Las instrucciones para ejecutar archivos de comandos las encontraréis en los vídeos del principio de este mismo enunciado. Tened en cuenta que **este archivo borra y crea de nuevo la tabla o agregación, por lo que se borrarán todos los datos y cambios que hayáis podido realizar**. Si queréis guardar los cambios hechos en esta tabla, podéis cambiarle el nombre y de esta manera el script no la borrará.

El siguiente paso es rellenar los datos en la tabla Electric\_Vehicle\_Population\_Data\_YEAR\_EVT\_MAKE. Para ello, tenéis que ejecutar el archivo Electric\_Vehicle\_Population\_Data\_YEAR\_EVT\_MAKE.csql igual que se ha ejecutado el paso anterior. Este archivo también borra por completo la tabla si existe en el momento de ejecutarlo.

Para comprobar que las filas se han insertado correctamente podemos ejecutar para la familia de columnas ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT la consulta:

SELECT \*

FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT

LIMIT 10;

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

La consulta retorna:

cqlsh:electric\_cars> SELECT \* FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT LIMIT 10;

city | postal\_code | electric\_vehicle\_type | total\_count

----------------+-------------+----------------------------------------+-------------

Chattaroy | 99021 | Battery Electric Vehicle (BEV) | 1

Chattaroy | 99021 | Plug-in Hybrid Electric Vehicle (PHEV) | 1

Olympia | 98504 | Battery Electric Vehicle (BEV) | 8

Clarkston | 99403 | Battery Electric Vehicle (BEV) | 20

Clarkston | 99403 | Plug-in Hybrid Electric Vehicle (PHEV) | 18

Ravensdale | 98038 | Battery Electric Vehicle (BEV) | 2

Ravensdale | 98038 | Plug-in Hybrid Electric Vehicle (PHEV) | 4

East Wenatchee | 98802 | Battery Electric Vehicle (BEV) | 125

East Wenatchee | 98802 | Plug-in Hybrid Electric Vehicle (PHEV) | 50

Lacey | 98509 | Battery Electric Vehicle (BEV) | 1

(10 rows)

cqlsh:electric\_cars>

Y para la otra tabla ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE podemos ejecutar la consulta:

SELECT \*

FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE

LIMIT 10;

Que retorna:

cqlsh:electric\_cars> SELECT \* FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE LIMIT 10;

model\_year | electric\_vehicle\_type | make | total\_count

------------+--------------------------------+---------------+-------------

2017 | Battery Electric Vehicle (BEV) | BMW | 105

2017 | Battery Electric Vehicle (BEV) | CHEVROLET | 1286

2017 | Battery Electric Vehicle (BEV) | FIAT | 165

2017 | Battery Electric Vehicle (BEV) | FORD | 25

2017 | Battery Electric Vehicle (BEV) | HYUNDAI | 17

2017 | Battery Electric Vehicle (BEV) | KIA | 162

2017 | Battery Electric Vehicle (BEV) | MERCEDES-BENZ | 22

2017 | Battery Electric Vehicle (BEV) | MITSUBISHI | 1

2017 | Battery Electric Vehicle (BEV) | NISSAN | 932

2017 | Battery Electric Vehicle (BEV) | SMART | 34

(10 rows)

cqlsh:electric\_cars>

Este formato 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, adjuntad los resultados de este apartado en modo Texto, no capturas de pantalla.

## Descripción de los agregados

Como habéis podido comprobar, habéis cargado dos tablas que son dos agregados. Ninguno de estos agregados contiene todos los datos en detalle como podrían encontrarse en la base de datos original, sino que la información ya está agrupada.

El primer agregado (ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT) agrupa los vehículos por ciudad, código postal y tipo de vehículo eléctrico. Estas serían las dimensiones del agregado. El recuento de vehículos sería el único dato que no forma parte de ninguna clasificación (también conocido como *fact*).

El segundo agregado (ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE) agrupa los vehículos por año, tipo de vehículo y por fabricante. Estas serían las dimensiones del agregado. Igual que en el agregado anterior, el recuento de vehículos sería el único *fact* de la tabla.

Aunque provengan de la misma base de datos, ambos agregados tienen propósitos diferentes y están optimizados para realizar consultas con diferentes dimensiones y condiciones.

## Consultas

Con los datos que habéis cargado 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. En tal caso, debéis adjuntar las consultas que os dan error y los comandos o acciones que habéis utilizado para resolver este error. **Se valorará todo el procedimiento que habéis seguido para poder ejecutar las consultas y obtener la información que se pide en el enunciado,** especialmente para aquellas consultas que no son directas. Si hay más de una opción, también se tendrá en cuenta que la solución elegida sea la que ofrezca mejor rendimiento.

En algunos casos, puede que tengáis que obtener datos adicionales para ejecutar las consultas, puede que tengáis que crear un índice, o puede que tengáis que crear otra tabla. **Tenéis total libertad para realizar todas aquellas acciones necesarias para que la consulta retorne los datos solicitados en el enunciado, pero siempre teniendo en cuenta cuál es la opción más eficiente, que se valorará con mejor puntuación.**

Por cuestiones de espacio disponible, el volumen de datos no es muy elevado, pero debéis considerar que esta base de datos en un entorno real tendrá un gran volumen de datos. Por lo tanto, si se requieren consultas previas para recuperar los datos necesarios para otra consulta, **no se puede consultar toda la tabla y obtener los datos a simple vista.** Se deben consultar los datos de manera filtrada para que la consulta previa retorne **la información concreta** que se necesita para resolver el ejercicio. En un escenario real con muchos registros no es viable seleccionar toda una tabla entera y buscar los datos manualmente en pantalla. Por último, se deben recuperar los datos solicitados o escribir las consultas a partir de la información facilitada en el enunciado o de otras consultas realizadas durante la elaboración del ejercicio.

Por último, si se pide realizar alguna modificación de datos, **se valorará que la base de datos quede en un estado consistente**. Además, tendréis que adjuntar una consulta que demuestre que todos los datos han quedado correctamente actualizados.

### 1.1 Consulta para sumas totales (10%):

Con el objetivo de comprobar la validez de los dos agregados, se pide la suma total de vehículos de ambos. Las dos sumas deben retornar el mismo resultado.

Consulta:

SELECT SUM(total\_count) AS total\_city FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT;

SELECT SUM(total\_count) AS total\_year FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE;

Resultado:

Para cada tabla calculamos la suma de total\_count.

total\_city

------------

114312

total\_year

------------

114312

### 1.2 Pregunta sobre la consulta anterior (10%)

El ejercicio devuelve un warning ¿podrías explicar por qué utilizando tus propias palabras?

Warning:

Aggregation query used without partition key

Respuesta:

Este warning nos indica que no estamos optimizando la consulta. Una ventaja de Cassandra es que puede dividir los datos utilizando una *partition key*, lo que permite agrupar (nodos) y procesar eficientemente los registros que tienen la misma llave. Para aprovechar esta ventaja, hemos de añadir la cláusula *WHERE* con la clave de partición. De esta manera, Cassandra sólo buscará en la partición deseada en lugar de examinar toda la tabla.

### 1.3 Consulta recuento de datos (20%):

Total de vehículos eléctricos del código postal 99403 desglosado por tipo de vehículo. La consulta debe retornar dos columnas: electric\_vehicle\_type y total\_count, y solamente mostrar los vehículos del código postal indicado.

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

Consulta:

CREATE INDEX idx\_postal\_code ON ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT (postal\_code);

SELECT electric\_vehicle\_type, total\_count FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT

WHERE postal\_code = '99403';

Resultado:

Si intentamos realizar directamente la segunda consulta nos da un error diciendo que la consulta puede involucrar filtración de datos y hemos de activar *ALLOW FILTERING*. Cómo la consulta en cuestión utiliza la cláusula *WHERE* para filtrar códigos postales, podemos crear un índice (secundario) sobre postal\_code ya que no usamos la clave de partición y Cassandra habría de examinar todos los nodos.

electric\_vehicle\_type | total\_count

----------------------------------------+-------------

Battery Electric Vehicle (BEV) | 20

Plug-in Hybrid Electric Vehicle (PHEV) | 18

### 1.4 Consulta recuento de datos (20%):

### Suma de todos los vehículos de la marca BMW.

**Restricción: igual que antes, no se puede utilizar la cláusula ALLOW FILTERING.**

Consulta:

CREATE INDEX idx\_make ON ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE (make);

SELECT SUM(total\_count) AS total\_bmw FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE

WHERE make = 'BMW';

Resultado:

Similar al ejercicio anterior primero hemos de crear un índice sobre la columna usada en la cláusula *WHERE*. En este caso, sobre *make* para filtrar por marcas.

total\_bmw

-----------

4742

### 1.5 Consulta de recuento de datos (20%)

Total de vehículos agrupado por ciudad y código postal para la ciudad de Chattaroy y su código postal 99021 y para la ciudad de Olympia y su código postal 98504.

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

**Precaución**: los filtros son sensibles a las mayúsculas y minúsculas. No es lo mismo Olympia que OLYMPIA.

Consulta:

SELECT city, postal\_code, SUM(total\_count) AS total FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT WHERE city = 'Chattaroy' AND postal\_code = '99021'GROUP BY city, postal\_code;

SELECT city, postal\_code, SUM(total\_count) AS total FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT

WHERE city = 'Olympia' AND postal\_code = '98504' GROUP BY city, postal\_code;

Resultado:

La query inicial que he intentado ha sido agrupar por ciudad y código postal en una consulta tal que así:

SELECT city, postal\_code, SUM(total\_count) AS total FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT

WHERE (city, postal\_code) IN (('Chattaroy', '99021'), ('Olympia', '98504'))

GROUP BY city, postal\_code;

Pero he obtenido un error de Cassandra diciendo que no soporta multi-column relations por lo que he optado por hacer dos consultas individuales. El resultado agrupado es:

city | postal\_code | total

-----------+-------------+-------

Chattaroy | 99021 | 2

Olympia | 98504 | 8

### 1.6 Comportamiento de actualización de datos en Cassandra (20%)

Seguid los siguientes pasos:

1. Ejecutad un **recuento** de las filas de los dos agregados (no la suma de vehículos, **el recuento** de filas). Adjuntad las consultas y ambos recuentos.

Consultas:

SELECT COUNT(\*) AS CITY FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT;

SELECT COUNT(\*) AS YEAR FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE;

Resultados:

city

------

1248

year

------

264

1. Ejecutad los siguientes dos comandos. **Notad que son comandos update, no insert:**

UPDATE ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT

SET TOTAL\_COUNT = 10

WHERE city = 'Alabama'

AND POSTAL\_CODE = '89760'

AND ELECTRIC\_VEHICLE\_TYPE = 'Battery Electric Vehicle (BEV)';

Y seguidamente:

UPDATE ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE

SET TOTAL\_COUNT = 10

WHERE MODEL\_YEAR = '2020'

AND ELECTRIC\_VEHICLE\_TYPE = 'Battery Electric Vehicle (BEV)'

AND MAKE = 'MITSUBISHI';

1. Volved a hacer el recuento de las dos familias de columnas. Adjuntad nuevamente las consultas y ambos recuentos.

Consultas:

SELECT COUNT(\*) AS CITY FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT;

SELECT COUNT(\*) AS YEAR FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_YEAR\_EVT\_MAKE;

Resultados:

city

------

1249

year

------

265

1. ¿Qué ha pasado? ¿Tienen sentido los resultados? Puede que tengáis que buscar información adicional en internet para responder la pregunta.

Una característica de Cassandra es que para garantizar la consistencia y recuperación de fallos cada versión del registro se mantiene en la tabla. Por ese motivo, la primera consulta ha dado los resultados 1248 y 264. Al realizar la actualización, se ha añadido el registro actualizado y se mantiene en reserva la versión anterior. Es por ello, que al volver a realizar la consulta de reencuentro se han incrementado en 1 ambos valores. Hay que tener en cuenta que este versionado es interno y cuando realizamos una consulta como:

SELECT \* FROM ELECTRIC\_VEHICLE\_POPULATION\_DATA\_CITY\_PC\_EVT

WHERE city = 'Alabama' AND POSTAL\_CODE = '89760'

AND ELECTRIC\_VEHICLE\_TYPE = 'Battery Electric Vehicle (BEV)';

Solo aparecerá la última versión del registro y no ambas.

# 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 y recordad que el usuario / contraseña para acceder a Neo4j son: **neo4j / uoc**

La propuesta de solución en todos los apartados debe proporcionar:

- Las consultas Cypher **en formato texto**,

- Una captura de pantalla de las consultas Cypher con los resultados obtenidos.

## 2.1 Carga de datos (no puntúa):

A continuación se presentan las instrucciones para 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 como 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

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 el gestor de grafos, **y que deberéis eliminar**:

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. Adaptadlos para eliminar todos los índices y restricciones de la salida del comando anterior:

DROP INDEX index\_f7700477;

DROP CONSTRAINT constraint\_59c89b17;

**CREACIÓN DE LOS NUEVOS NODOS Y SUS RELACIONES**

**Nodos de los Personajes principales y sus relaciones**

CALL apoc.load.jsonArray('https://gist.githubusercontent.com/JosepMeseguer/f7a7df6fce3b10d77ff57e82e1e9d075/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) | [toLower(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;

Que después de unos segundos de ejecución retornará algo parecido a lo siguiente (al ingerir datos que están disponible 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**

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) | [toLower(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'],['',[''],[],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 2602 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);

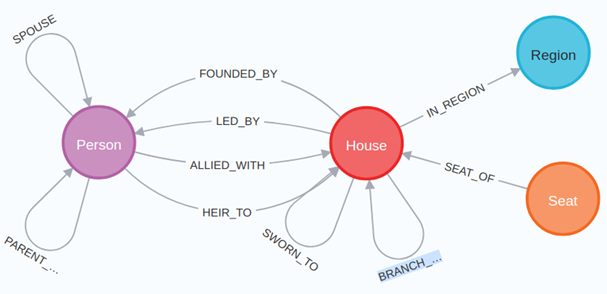
create index for (r:Region) on (r.name);

## 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 - -> persona que ha fundado la casa

· House LED\_BY Person - -> gobernante de la casa

· Person ALIED\_WITH House - -> persona perteneciente o aliada a una casa

· Person HEIR\_TO House - -> heredero de la casa

· Person PARENT\_TO Person - -> padres de la persona

· Person SPOUSE Person - -> pareja consorte de la persona heredera

## 2.3 Consultas básicas (20%):

Se propone responder a las siguientes consultas una vez realizada la carga de datos, utilizando Cypher y su interfaz web. Es necesario adjuntar en modo texto la query y una captura de pantalla de su resultado:

**Consulta 1 (25%):**

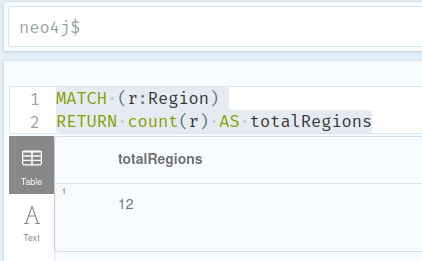
Contar cuántos nodos de tipo Región se han creado en la carga de datos.

Consulta:

MATCH (r:Region)

RETURN count(r) AS totalRegions

Resultado: 12



**Consulta 2 (35%):**

Contar el número de personajes femeninos que se han creado en la carga de datos.

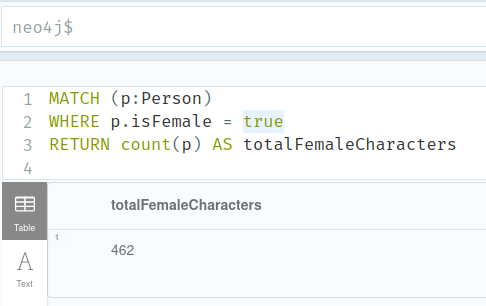
Consulta:

MATCH (p:Person)

WHERE p.isFemale = true

RETURN count(p) AS totalFemaleCharacters

Resultado: 462



**Consulta 3 (40%):**

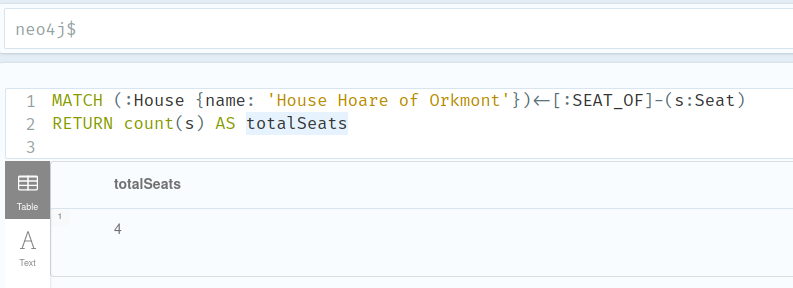
Contar el número de fortalezas donde se ha establecido la casa ’House Hoare of Orkmont’.

Consulta:

MATCH (:House {name: 'House Hoare of Orkmont'})<-[:SEAT\_OF]-(s:Seat)

RETURN count(s) AS totalSeats

Resultado: 4



## 2.4 Consultas avanzadas (60%):

Resolver las siguientes consultas en Cypher presentando la query en modo texto y una captura de pantalla con los resultados obtenidos:

**Consulta 1 (20%):**

Se desean conocer los nombres de las regiones donde pueden localizarse más de 50 casas a lo largo de la historia. Presentar el nombre de la región y el número de casas encontradas en la misma, ordenando los registros descendentemente en función del número de casas localizadas.

Consulta:

MATCH (r:Region)<-[:IN\_REGION]-(h:House)

WITH r, count(DISTINCT h) AS numCasas

WHERE numCasas > 50

RETURN r.name AS nombreRegion, numCasas

ORDER BY numCasas DESC

Resultado:



**Consulta 2 (20%):**

Queremos conocer las casas ubicadas en la región ‘The Reach” que ya han desaparecido. Mostrar el nombre y fechas de extinción de las casas.

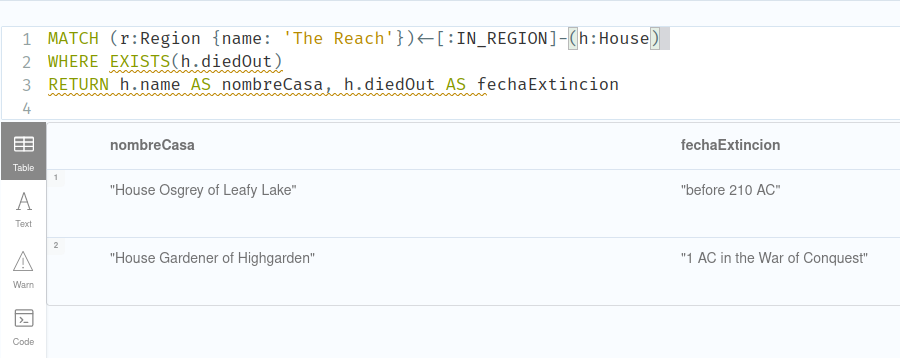
Consulta:

MATCH (r:Region {name: 'The Reach'})<-[:IN\_REGION]-(h:House)

WHERE EXISTS(h.diedOut)

RETURN h.name AS nombreCasa, h.diedOut AS fechaExtincion

Resultado:



**Consulta 3 (30%):**

Deseamos obtener en una única fila el número actual de casas y el número actual de personas que son fieles a la casa “House Stark of Winterfell”.

Consulta:

MATCH (h:House {name: 'House Stark of Winterfell'})

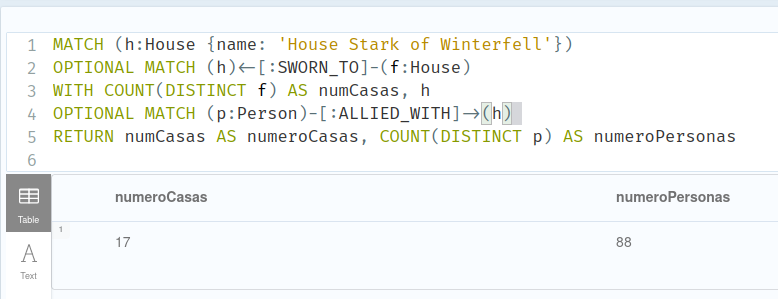
OPTIONAL MATCH (h)<-[:SWORN\_TO]-(f:House)

WITH COUNT(DISTINCT f) AS numCasas, h

OPTIONAL MATCH (p:Person)-[:ALLIED\_WITH]->(h)

RETURN numCasas AS numeroCasas, COUNT(DISTINCT p) AS numeroPersonas

Resultado:



**Consulta 4 (30%):**

Necesitamos conocer a las personas que heredarán el gobierno de su casa y que no están casadas. La consulta retornará el nombre de la casa, el nombre de la persona heredera, la cantidad de casas fieles y un campo de nombre “Gender” que presentará el valor “Women” si se trata de una heredera o “Man” si se trata de un heredero. Se presentarán únicamente los 4 registros con mayor número de casas fieles ordenados descendentemente por dicho número.

NOTA: Las sentencias condicionales de Cypher os pueden ayudar a realizar este ejercicio.

Consulta:

MATCH (p:Person)-[:HEIR\_TO]->(h:House)

WHERE NOT EXISTS((p)-[:SPOUSE]-())

MATCH (h)<-[:SWORN\_TO]-(f:House)

WITH h, p, COUNT(DISTINCT f) AS numCasas,

CASE WHEN p.isFemale THEN "Women" ELSE "Man" END AS Gender

ORDER BY numCasas DESC

LIMIT 4

RETURN h.name AS nombreCasa, p.name AS nombreHeredera, numCasas, Gender

Resultado:



## 2.5 Restricciones en Neo4j (20%):

Un problema de las BBDD NoSQL es que, en general, no garantizan la integridad de los datos. Sin embargo, en neo4j existen mecanismos para garantizar la unicidad de nodos en función de algunas de sus propiedades, es decir, que todos los nodos de un tipo deben tener valores distintos para aquellas propiedades definidas como únicas. Por ejemplo, esto impediría crear un nodo de tipo **Region** con un valor en el atributo ***name*** que ya existe en otro nodo de tipo **Region**.

Se pide adjuntar las consultas de creación y de comprobación, que demuestran que:

1. **(30%)** Actualmente podemos crear diferentes nodos de tipo *Region* con el mismo valor en su atributo *name*, en este contexto representaría un indicador de inconsistencia .
2. **(70%)** Podemos reconfigurar neo4j para que garantice que todos los nodos de tipo Region tengan un nombre (name) distinto.

Ayuda:

- Para mostrar el ID de un nodo: MATCH (r:Region)

WHERE r.name = "The Reach"

RETURN r.name, ID(r)

- Para borrar nodos usar: MATCH … DELETE …

- Para listar los índices del grafo: SHOW indexes

- Para borrar un índice que ya existe: DROP INDEX index\_name

1. **Creación diferentes nodos con el mismo nombre**

Primero vamos a comprobar que efectivamente podemos añadir 2 nodos con el mismo valor *name* para un tipo de *Region*. Ejecutando la siguiente consulta vemos si hay valores repetidos:

MATCH (r:Region)

WITH r.name AS name, collect(ID(r)) AS ids, count(\*) AS count

WHERE count > 1

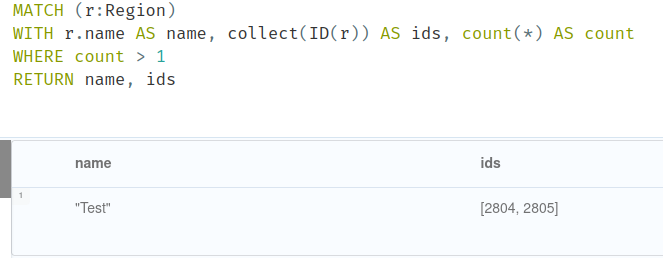
RETURN name, ids

Al no haber registros duplicados, nos devuelve un mensaje con “no records”. A continuación probamos añadir dos regiones con el mismo nombre, realizar la misma consulta y ver si realmente se guardan. Para ello realizamos las siguientes querys:

CREATE (:Region {name: "Test"})

CREATE (:Region {name: "Test"})

Realizamos otra vez la consulta inicial y obtenemos:



Por lo tanto, se han guardado dos regiones con el mismo nombre. Eliminamos los dos registros que acabamos de crear con la query:

MATCH (r:Region {name: "The Reach"})

DELETE r

1. **Restricciones para evitar nombre repetidos en una región**

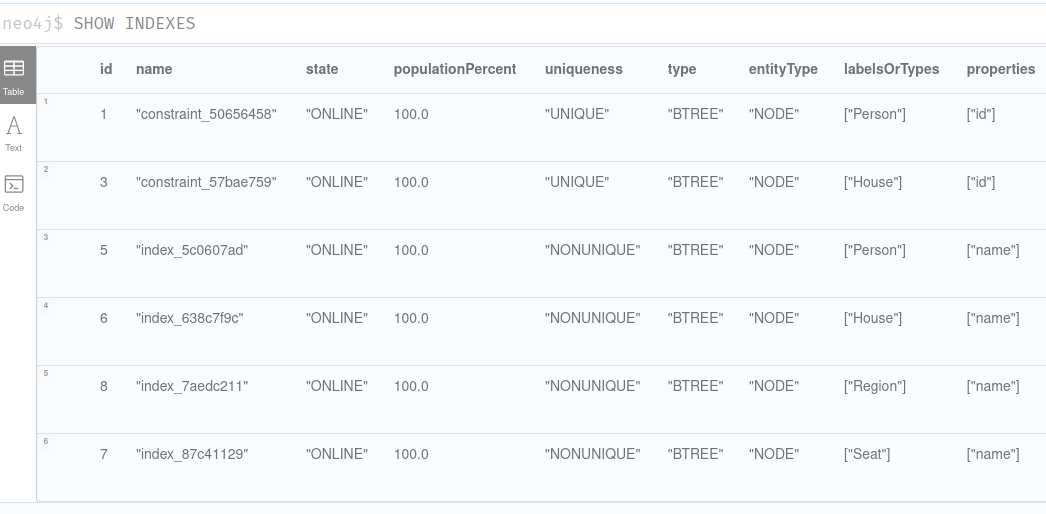
Para crear una restricción sobre el atributo *name* de la Region realizaremos la siguiente query:

CREATE CONSTRAINT ON (r:Region) ASSERT r.name IS UNIQUE

Al realizar esta consulta Neo4j se queja de que ya existe un índice sobre el atributo por lo que tenemos que eliminarlo primero (drop)

SHOW indexes

Utilizamos Show indexes para ver los que hay actualmente.



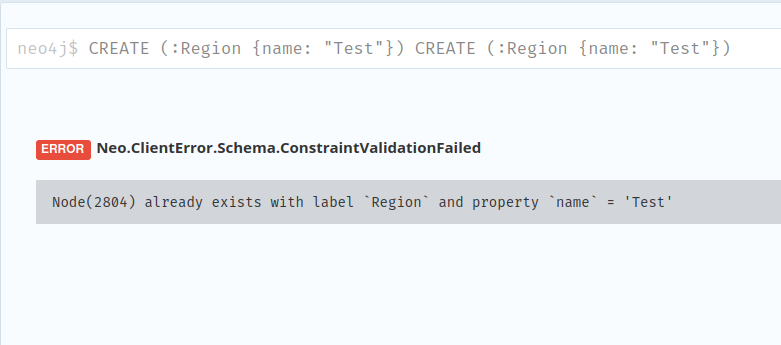
Podemos ver en el registro 8, que ya hay un índice que hace referencia a Region.name. Lo eliminamos.

DROP INDEX index\_7aedc211

Ahora volvemos a crear el CONSTRAINT:

CREATE CONSTRAINT ON (r:Region) ASSERT r.name IS UNIQUE

Finalmente, probamos nuevamente insertar 2 registros de tipo Region con el mismo nombre:



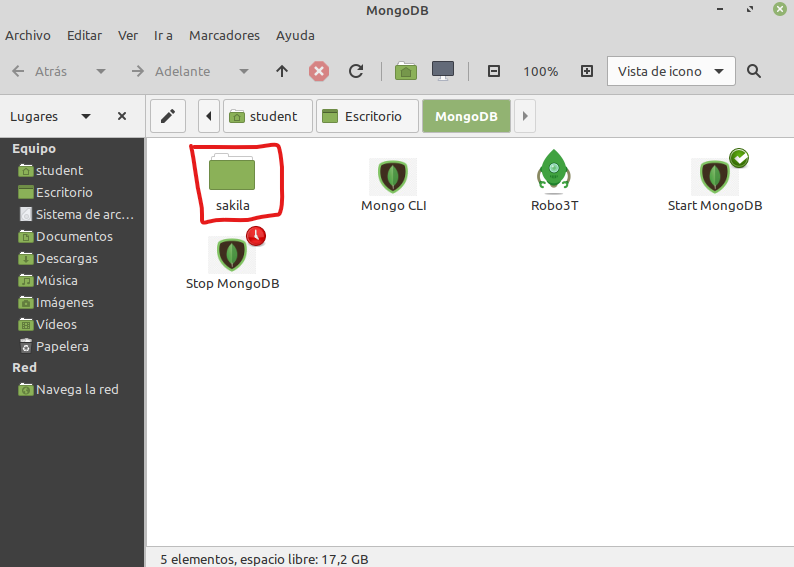
Observamos que ha devuelto un error y que ya existe una Region con ese nombre.

# 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

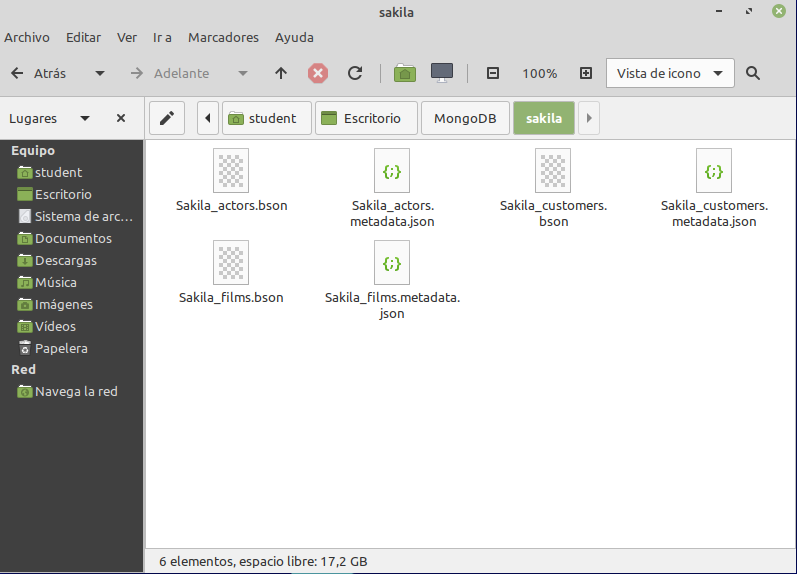
Para cargar los datos deberéis crear una carpeta llamada sakila (en minúsculas) dentro de la carpeta MongoDB que se muestra en el tutorial, quedando de la siguiente manera:



Dentro de la carpeta creada (sakila) copiar y pegar los archivos sin comprimir proporcionados junto con el enunciado:

* Sakila\_actors.bson
* Sakila\_actors.metadata.json
* Sakila\_customers.bson
* Sakila\_customers.metadata.json
* Sakila\_films.bson
* Sakila\_films.metadata.json

El contenido de la carpeta debe ser:



## Después de haber arrancado el servicio de MongoDB (haciendo doble clic en el icono *Start MongoDB*, tal y como se indica en el manual), abrid un terminal de comandos desde la barra de tareas del sistema y ejecutad el siguiente comando:

mongorestore --db=sakila --drop /home/student/Escritorio/MongoDB/sakila

Ya nos podemos conectar a MongoDB y comprobar que los datos se han cargado correctamente. Desde un terminal de comandos podemos ejecutar:

mongo

O bien podéis hacer doble clic en el icono con nombre “Mongo CLI”:

El command prompt cambiará a:

student@NoSQLDB:~$ mongo

MongoDB shell version v4.2.17

connecting to: mongodb://127.0.0.1:mongo27017/?compressors=disabled&gssapiServiceName=mongodb

Implicit session: session { "id" : UUID("48b256a8-d32c-459e-9fdd-c646d0027374") }

MongoDB server version: 4.2.17

Server has startup warnings:

(...)

To enable free monitoring, run the following command: db.enableFreeMonitoring()

To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

---

>

Ejecutamos:

use sakila

Que retorna:

> use sakila

switched to db sakila

Ejecutamos

show collections

Que retorna:

> show collections

Sakila\_actors

Sakila\_customers

Sakila\_films

>

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

Recordad que 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).

Para saber si estáis en la base de datos correcta, escribiendo el comando db sabréis en qué base de datos estáis 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). 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 todos los cambios realizados.

## 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.

### Consulta de ejemplo

Retorna un actor de la colección Sakila\_actors en un formato de 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" : "GUINESS",

"phone" : "XXX-XXXX-XXX",

"address" : "XXXXXXXXXXXXXXXXXXXXXXXXXXXX"

}

>

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

### 3.1 Consulta (5%)

Recuento de los clientes que han alquilado la película "RACER EGG". La consulta debe retornar un solo número.

Consulta:

db.Sakila\_customers.aggregate([

{

$match: {

"Rentals.Film Title": "RACER EGG"

}

},

{

$count: "Total"

}

])

Resultado:

{ "Total" : 18 }

### 

### 

### 3.2 Consulta (20%)

Para esta consulta, [la referencia de MongoDB](https://www.mongodb.com/docs/v6.0/meta/aggregation-quick-reference/) sobre la Aggregation Pipeline os puede ser útil.

Los cinco primeros clientes que más películas han alquilado. La consulta debe retornar el nombre del cliente (“First Name”) y el recuento. Opcionalmente puede retornar el identificador.

Consulta:

db.Sakila\_customers.aggregate([

{

$project: {

"First Name": 1,

rentalCount: { $size: "$Rentals" }

}

},

{

$sort: { rentalCount: -1 }

},

{

$limit: 5

}

])

Resultado:

{ "\_id" : 148, "First Name" : "ELEANOR", "rentalCount" : 46 }

{ "\_id" : 526, "First Name" : "KARL", "rentalCount" : 45 }

{ "\_id" : 236, "First Name" : "MARCIA", "rentalCount" : 42 }

{ "\_id" : 144, "First Name" : "CLARA", "rentalCount" : 42 }

{ "\_id" : 75, "First Name" : "TAMMY", "rentalCount" : 41 }

### 3.3 Consulta (20%)

Top 7 de las películas más alquiladas. La consulta debe retornar solamente el nombre de la película y el número de veces que ha sido alquilada. El resultado debe estar ordenado por el número de veces descendente y el nombre de la película ascendente.

Consulta:

db.Sakila\_customers.aggregate([

{

$unwind: "$Rentals"

},

{

$group: {

\_id: "$Rentals.Film Title",

rentalCount: { $sum: 1 }

}

},

{

$sort: {

rentalCount: -1,

\_id: 1

}

},

{

$limit: 7

},

{

$project: {

\_id: 0,

"Film Title": "$\_id",

rentalCount: 1

}

}

])

Resultado:

{ "rentalCount" : 34, "Film Title" : "BUCKET BROTHERHOOD" }

{ "rentalCount" : 33, "Film Title" : "ROCKETEER MOTHER" }

{ "rentalCount" : 32, "Film Title" : "FORWARD TEMPLE" }

{ "rentalCount" : 32, "Film Title" : "GRIT CLOCKWORK" }

{ "rentalCount" : 32, "Film Title" : "JUGGLER HARDLY" }

{ "rentalCount" : 32, "Film Title" : "RIDGEMONT SUBMARINE" }

{ "rentalCount" : 32, "Film Title" : "SCALAWAG DUCK" }

### 3.4 Modificación de datos (40%)

A tener en cuenta para este ejercicio:

* Para resolver este ejercicio necesitaréis más de una consulta. Es posible que la salida de algunas consultas necesarias para poder resolver el ejercicio sea muy larga. Para este ejercicio podéis “truncar” el resultado remarcando o dejando la parte más relevante.
* Recordad que la base de datos tiene que quedar en un estado consistente.
* Adicionalmente a la información que tenéis disponible en los apuntes, deberéis conocer la dot notation para acceder a los elementos de un array en MongoDB. [Aquí encontraréis la documentación necesaria](https://www.mongodb.com/docs/manual/core/document/#dot-notation) para realizar el ejercicio acerca de la dot notation y [aquí documentación al respecto del operador $set](https://www.mongodb.com/docs/manual/reference/operator/update/set/). Con esta notación, el acceso a un campo de un determinado elemento de un array es exactamente igual que el acceso a un atributo tanto para el filtro como para el operador de actualización. Recordad que la numeración de un array (tanto en dot notation como si no) siempre es en base 0 (el primer elemento es el 0).
* Alternativamente al punto anterior, [también podéis mirar la documentación de los operadores de actualización de un array](https://www.mongodb.com/docs/manual/reference/operator/update/positional/).

**Ejercicio:**

Actualiza el título de la película "MIXED DOORS" a "LOST DOORS".

Consulta:

db.Sakila\_films.updateMany(

{ Title: "MIXED DOORS" },

{ $set: { Title: "LOST DOORS" } }

)

db.Sakila\_customers.updateMany(

{ "Rentals.Film Title": "MIXED DOORS" },

{ $set: { "Rentals.$.Film Title": "LOST DOORS" } }

)

Resultado:

Sakila\_films:

{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }

Sakila\_customers:

{ "acknowledged" : true, "matchedCount" : 4, "modifiedCount" : 4 }

Esta es la manera que encuentro más fácil para la actualización del nombre de la película. Primero actualizamos el nombre de la película de la colección Sakila\_films y seguidamente actualizamos la información de los consumidores que han alquilado esa película por el nuevo nombre en la colección Sakila\_customers. Otra manera hubiese sido actualizar la colección Sakila\_films como hemos hecho, y actualizar la colección Sakila\_customers en base del identificador de la película “LOST DOORS”.

### 3.5 Indexación (15%)

Se requiere crear un índice que agilice las consultas a la colección Sakila\_films y el campo "Tape status" para ir anotando en qué estado se encuentra la cinta de vídeo. El índice tendrá que adaptarse a las siguientes características de uso de la colección:

* Se ha comprobado que el rendimiento no se verá penalizado porque hay muchas más lecturas que escrituras en la tabla.
* Habrá muchos documentos que no tendrán el atributo porque se irán introduciendo los datos a medida que se vayan retornando las cintas alquiladas. Aquellas cintas que ya estén descatalogadas o que no se alquilen no tendrán este atributo en el documento.
* No se requiere realizar ningún tipo de consultas sobre los documentos que no tienen el atributo.
* Se quiere buscar por coincidencia entera (no búsquedas parciales de texto).

Escribe cuál es el comando para crear el índice más adecuado para el escenario descrito.

Primeramente, se ha de añadir el campo Tape status a algún registro de la colección Sakila\_films ya que de momento no se encuentra en ningun registro:

db.Sakila\_films.find({ "Tape status": { $exists: true } })

Se nos especifica que no se han de realizar consultas sobre los documentos que no tienen Tape status y que muchos no tendrán este atributo. Por lo que incluiremos:

{ partialFilterExpression: { "Tape status": { $exists: true } }

Así, solo filtraremos los documentos que contengan el atributo. El comando final será:

db.Sakila\_films.createIndex(

{ "Tape status": 1 },

{ partialFilterExpression: { "Tape status": { $exists: true } } }

)