

Uso de bases de datos NoSQL

PRA1

Propuesta de solución

Instrucciones

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

- *Máquina virtual Linux Mint* suministrada.
- Documento “*Máquina virtual Linux Mint (Manual)*”

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 las máquinas virtuales o estas bases de datos, os recomendamos su visionado, ya que podéis encontrar alguna sugerencia o idea nueva que os facilite el desarrollo de las actividades:

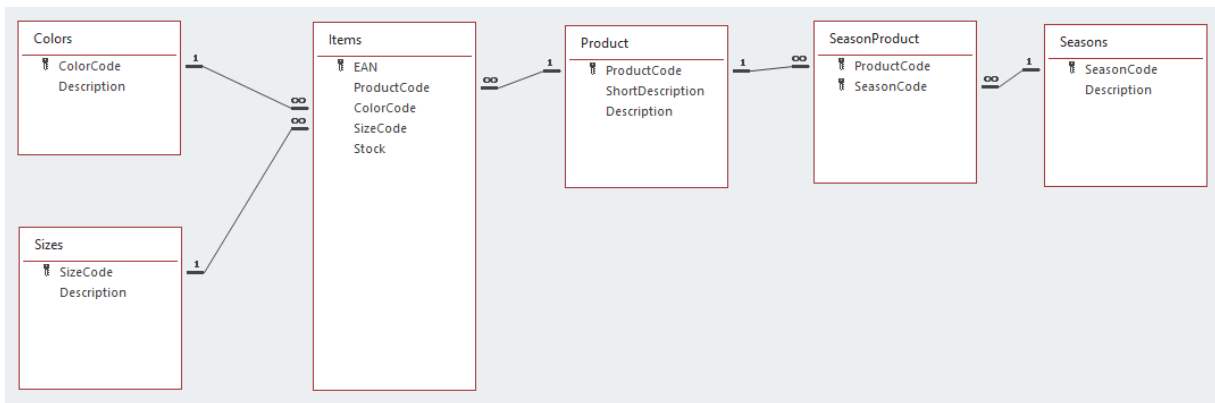
- Consejos sobre el uso de la máquina virtual: <https://vimeo.com/539103098>
- Consejos sobre el uso de Cassandra: <https://vimeo.com/539103072>

En algunos casos se utilizarán las bases de datos ya instaladas en la máquina virtual. En el caso de que sea necesario cargar nuevos datos, se indicará en cada ejercicio y se detallarán las instrucciones de cómo hacerlo.

Ejercicio 1: Cassandra (30%)

Los datos necesarios para este ejercicio los tendréis que cargar vosotros. Encontraréis las instrucciones en el siguiente enunciado.

El sistema de gestión de una empresa distribuidora de artículos textiles tiene las siguientes tablas en un sistema relacional y normalizado:



Como se puede observar en el diagrama de relaciones, los datos están estructurados de la siguiente manera.

Por una parte, las temporadas (primavera, verano, otoño e invierno) se almacenan en una tabla llamada *Seasons*. Los colores están en la tabla *Colors* y finalmente las tallas están en la tabla *Sizes*.

La tabla *Product* contiene el código de los productos, una descripción corta para los tickets de compra y albaranes, y una descripción más detallada para la web comercial. Un producto puede estar disponible en diferentes tallas y colores.

Como un producto puede pertenecer a más de una temporada, hay una tabla intermedia llamada *SeasonProduct* que relaciona el producto con todas las temporadas a las que pertenece. Es una relación de muchos a muchos entre temporadas y productos.

En la tabla *Item* es donde se establecen las unidades disponibles de cada producto. Cada fila de la tabla representa una composición de un producto (ProductCode), una talla (SizeCode) y un color (ColorCode). Por reglas de integridad, esta composición es única, no puede repetirse. Al ser el nivel más bajo de detalle, cuenta con el stock disponible y una referencia única llamada EAN que es la clave primaria de la tabla *Item*. Por lo tanto, pueden haber diferentes colores y tallas de un mismo producto, es decir el campo ProductCode puede repetirse en la tabla *Item*. Sin embargo, un EAN no puede repetirse en la tabla *Item* ya que es clave primaria.

La base de datos dispone de la integridad típica de un sistema relacional: claves primarias, índices únicos y todas las validaciones disponibles en una base de datos tradicional. Esto implica que los datos que se importarán a nuestro sistema tienen la integridad referencial garantizada, asegura que los datos son consistentes y ahorra tiempo en validaciones. Por ejemplo, no podremos introducir una fila en la tabla *Items* con un código de color que no esté en la tabla de *Colors*.

La tabla colores tiene el siguiente contenido (es posible que no todos los colores se utilicen en la tabla ítem):

ColorCode	Description
BE	Beige
BL	Black
CB	Cobalt blue
RE	Red
WH	White
YE	Yellow

La tabla tallas tiene las siguientes filas (evidentemente no es una lista completa de tallas posibles y puede que algunas no se utilicen):

SizeCode	Description
31	Trousers 31
32	Trousers 32
33	Trousers 33
34	Trousers 34
35	Trousers 35
L	Large
M	Medium
S	Small

XL	Extra large
XS	Extra small
XXL	Extra extra large
XXS	Extra extra small

La tabla *Seasons* tiene las cuatro temporadas típicas del sector textil. Al estar relacionada con las cuatro estaciones del año, **es una tabla que siempre tendrá los mismos valores**.

SeasonCode	Description
AU	Autumn
SP	Spring
SU	Summer
WI	Winter

La tabla *SeasonProduct* relaciona las temporadas con los siguientes artículos. Destacar que el artículo referencia CASTR pertenece a dos temporadas, AU y WI.

ProductCode	SeasonCode
BEATS	SU
CASTR	AU
CASTR	WI
RUNTS	SU

La tabla *Products* contiene los siguientes productos. **Considerar que esta tabla en un entorno real tendrá muchos más registros y referencias**.

ProductCode	ShortDescription	Description
BEATS	Beach T Shirt	T Shirt to go to the beach
CASTR	Casual trousers	Trousers for casual dress

RUNTS	Running T Shirt	T Shirt for running
-------	-----------------	---------------------

Y la tabla *Items* contiene los siguientes ítems:

EAN	ProductCode	ColorCode	SizeCode	Stock
843245599300	BEATS	BL	L	2
843245599301	BEATS	BL	XL	1
843245599302	BEATS	BL	M	2
843245599303	BEATS	BL	S	1
843245599304	BEATS	BL	XS	0
843245599305	BEATS	BL	XXL	1
843245599306	BEATS	BL	XXS	0
843245599309	CASTR	CB	33	10
843245599310	CASTR	CB	34	15
843245599311	CASTR	CB	35	2
843245599312	CASTR	WH	31	0
843245599313	CASTR	WH	32	0
843245599314	CASTR	WH	33	30
843245599315	CASTR	WH	34	10
843245599316	RUNTS	YE	L	5
843245599317	RUNTS	YE	XL	7
843245599318	RUNTS	YE	M	8
843245599319	RUNTS	YE	S	2

Al tener un volumen de datos creciente, para el análisis de la información y para mejorar el rendimiento de algunas aplicaciones operacionales, la empresa se plantea utilizar una base de datos que implemente un modelo agrupado como Cassandra.

Las primeras agrupaciones en tablas (o familias de columnas) ya están realizadas y se proporcionan a continuación. Los datos no están cargados en la base de datos Cassandra de la máquina virtual proporcionada, pero a continuación encontraréis detallados los pasos para cargarlos.

Debéis considerar que esta base de datos en un entorno real tendrá mucho volumen de datos, salvo la excepción de la tabla *Seasons* ya mencionada anteriormente. Las consultas deberán escribirse con este escenario en mente.

Ejercicio 1.1 (no puntúa):

Carga de datos. Esta parte de la práctica no puntúa porque están todos los pasos detallados, pero es necesaria para la elaboración de este ejercicio. Se tendrán que 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. Para llegar a este punto tendréis que haber visto y leído los vídeos mencionados al principio del enunciado y en el documento “Uso de máquina virtual_ Bases de datos no convencionales.pdf”. Partiremos del siguiente símbolo del sistema (*command prompt*) en la línea de comandos:

```
cqlsh>
```

Primero creamos un keyspace llamado *wear_dealer* con el siguiente comando:

```
create keyspace wear_dealer with replication ={ 'class':  
'SimpleStrategy','replication_factor': 1};
```

Indicamos que queremos utilizar el keyspace creado ejecutando:

```
use wear_dealer;
```

Que nos cambiarà el *command prompt* a:

```
cqlsh:wear_dealer>
```

Creamos la primera agrupación *product_color_size_stock* con el siguiente comando:

```
CREATE TABLE product_color_size_stock (ProductCode TEXT, ColorCode  
TEXT, SizeCode TEXT, EAN TEXT, ShortDescription TEXT, Stock INT,  
PRIMARY KEY (ProductCode, ColorCode, SizeCode));
```

Rellenamos la tabla con las siguientes instrucciones. Podéis copiarlas y pegarlas en la interfaz de línea de comandos, pero es más práctico que ejecutéis el archivo de texto “insert_into_product_color_size_stock.cql” que encontraréis junto con el enunciado, siguiendo las instrucciones que se indican en uno de los vídeos detallados al principio.

```
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('BEATS', 'BL', 'L',
'843245599300', 'Beach T Shirt', 2);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('BEATS', 'BL',
'XL', '843245599301', 'Beach T Shirt', 1);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('BEATS', 'BL', 'M',
'843245599302', 'Beach T Shirt', 2);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('BEATS', 'BL', 'S',
'843245599303', 'Beach T Shirt', 1);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('BEATS', 'BL',
'XS', '843245599304', 'Beach T Shirt', 0);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('BEATS', 'BL',
'XXL', '843245599305', 'Beach T Shirt', 1);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('BEATS', 'BL',
'XXS', '843245599306', 'Beach T Shirt', 0);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('CASTR', 'CB',
'33', '843245599309', 'Casual trousers', 10);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('CASTR', 'CB',
'34', '843245599310', 'Casual trousers', 15);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('CASTR', 'CB',
'35', '843245599311', 'Casual trousers', 2);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('CASTR', 'WH',
'31', '843245599312', 'Casual trousers', 0);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('CASTR', 'WH',
'32', '843245599313', 'Casual trousers', 0);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('CASTR', 'WH',
'33', '843245599314', 'Casual trousers', 30);
```

```
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('CASTR', 'WH',
'34', '843245599315', 'Casual trousers', 10);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('RUNTS', 'YE', 'L',
'843245599316', 'Running T Shirt', 5);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('RUNTS', 'YE',
'XL', '843245599317', 'Running T Shirt', 7);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('RUNTS', 'YE', 'M',
'843245599318', 'Running T Shirt', 8);
INSERT INTO product_color_size_stock (ProductCode, ColorCode,
SizeCode, EAN, ShortDescription, Stock) VALUES ('RUNTS', 'YE', 'S',
'843245599319', 'Running T Shirt', 2);
```

Para comprobar que las filas se han insertado correctamente podemos ejecutar:

```
select * from product_color_size_stock;
```

El uso de “select *” no es una buena práctica en entornos de producción con muchas columnas. En este caso sabemos que hay pocas columnas y queremos revisarlas todas. El comando nos retornará:

productcode	colorcode	sizecode	ean	shortdescription	stock
BEATS	BL	L	843245599300	Beach T Shirt	2
BEATS	BL	M	843245599302	Beach T Shirt	2
BEATS	BL	S	843245599303	Beach T Shirt	1
BEATS	BL	XL	843245599301	Beach T Shirt	1
BEATS	BL	XS	843245599304	Beach T Shirt	0
BEATS	BL	XXL	843245599305	Beach T Shirt	1
BEATS	BL	XXS	843245599306	Beach T Shirt	0
CASTR	CB	33	843245599309	Casual trousers	10
CASTR	CB	34	843245599310	Casual trousers	15
CASTR	CB	35	843245599311	Casual trousers	2
CASTR	WH	31	843245599312	Casual trousers	0
CASTR	WH	32	843245599313	Casual trousers	0
CASTR	WH	33	843245599314	Casual trousers	30
CASTR	WH	34	843245599315	Casual trousers	10
RUNTS	YE	L	843245599316	Running T Shirt	5
RUNTS	YE	M	843245599318	Running T Shirt	8
RUNTS	YE	S	843245599319	Running T Shirt	2
RUNTS	YE	XL	843245599317	Running T Shirt	7

```
(18 rows)
cqlsh:wear_dealer>
```

Este es el formato recomendado para adjuntar los resultados de Cassandra en vuestras respuestas a los ejercicios.

A continuació, creamos la segunda agrupación `seasons_product_color_size` con el siguiente comando:

```
CREATE TABLE seasons_product_color_size (SeasonCode TEXT,  
ProductCode TEXT, ColorCode TEXT, SizeCode TEXT, EAN TEXT,  
ShortDescription TEXT, PRIMARY KEY ((SeasonCode, ProductCode),  
ColorCode, SizeCode));
```

A continuació, rellenamos la tabla con las siguientes instrucciones. Igual que con la anterior tabla, es más práctico ejecutar el archivo `insert_into_seasons_product_color_size.cql` que encontraréis junto con el enunciado siguiendo las instrucciones que se indican en uno de los vídeos detallados al principio.

```
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('AU', 'CASTR',  
'CB', '33', '843245599309', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('AU', 'CASTR',  
'CB', '34', '843245599310', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('AU', 'CASTR',  
'CB', '35', '843245599311', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('AU', 'CASTR',  
'WH', '31', '843245599312', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('AU', 'CASTR',  
'WH', '32', '843245599313', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('AU', 'CASTR',  
'WH', '33', '843245599314', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('AU', 'CASTR',  
'WH', '34', '843245599315', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('WI', 'CASTR',  
'CB', '33', '843245599309', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('WI', 'CASTR',  
'CB', '34', '843245599310', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('WI', 'CASTR',  
'CB', '35', '843245599311', 'Casual trousers');  
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,  
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('WI', 'CASTR',  
'WH', '31', '843245599312', 'Casual trousers');
```

```
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('WI', 'CASTR',
'WH', '32', '843245599313', 'Casual trousers');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('WI', 'CASTR',
'WH', '33', '843245599314', 'Casual trousers');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('WI', 'CASTR',
'WH', '34', '843245599315', 'Casual trousers');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'BEATS',
'BL', 'L', '843245599300', 'Beach T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'BEATS',
'BL', 'XL', '843245599301', 'Beach T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'BEATS',
'BL', 'M', '843245599302', 'Beach T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'BEATS',
'BL', 'S', '843245599303', 'Beach T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'BEATS',
'BL', 'XS', '843245599304', 'Beach T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'BEATS',
'BL', 'XXL', '843245599305', 'Beach T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'BEATS',
'BL', 'XXS', '843245599306', 'Beach T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'RUNTS',
'YE', 'L', '843245599316', 'Running T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'RUNTS',
'YE', 'XL', '843245599317', 'Running T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'RUNTS',
'YE', 'M', '843245599318', 'Running T Shirt');
INSERT INTO seasons_product_color_size (SeasonCode, ProductCode,
ColorCode, SizeCode, EAN, ShortDescription) VALUES ('SU', 'RUNTS',
'YE', 'S', '843245599319', 'Running T Shirt');
```

Para comprobar que las filas se han insertado correctamente podemos ejecutar:

```
select * from seasons_product_color_size;
```

Que retorna:

seasoncode	productcode	colorcode	sizecode	ean	shortdescription
WI	CASTR	CB	33	843245599309	Casual trousers
WI	CASTR	CB	34	843245599310	Casual trousers
WI	CASTR	CB	35	843245599311	Casual trousers
WI	CASTR	WH	31	843245599312	Casual trousers
WI	CASTR	WH	32	843245599313	Casual trousers
WI	CASTR	WH	33	843245599314	Casual trousers
WI	CASTR	WH	34	843245599315	Casual trousers
AU	CASTR	CB	33	843245599309	Casual trousers
AU	CASTR	CB	34	843245599310	Casual trousers
AU	CASTR	CB	35	843245599311	Casual trousers
AU	CASTR	WH	31	843245599312	Casual trousers
AU	CASTR	WH	32	843245599313	Casual trousers
AU	CASTR	WH	33	843245599314	Casual trousers
AU	CASTR	WH	34	843245599315	Casual trousers
SU	BEATS	BL	L	843245599300	Beach T Shirt
SU	BEATS	BL	M	843245599302	Beach T Shirt
SU	BEATS	BL	S	843245599303	Beach T Shirt
SU	BEATS	BL	XL	843245599301	Beach T Shirt
SU	BEATS	BL	XS	843245599304	Beach T Shirt
SU	BEATS	BL	XXL	843245599305	Beach T Shirt
SU	BEATS	BL	XXS	843245599306	Beach T Shirt
SU	RUNTS	YE	L	843245599316	Running T Shirt
SU	RUNTS	YE	M	843245599318	Running T Shirt
SU	RUNTS	YE	S	843245599319	Running T Shirt
SU	RUNTS	YE	XL	843245599317	Running T Shirt

```
(25 rows)
cqlsh:wear_dealer>
```

Tal y como indican los nombres y podéis comprobar en los datos, la familia de columnas `product_color_size_stock` agrupa los datos por producto, color y talla, y también detalla el stock disponible. La familia de columnas `seasons_product_color_size` agrupa los datos por temporadas, productos, color y talla.

Ejercicio 1.2 (80%):

Con los datos que habéis cargado en el apartado anterior, se requiere realizar las siguientes consultas y adjuntar su resultado.

Limitaciones para las consultas:

- Para cada consulta deberéis elegir la tabla más adecuada.
- No se pueden buscar manualmente en la base de datos los códigos necesarios para las consultas de actualización. Se deben recuperar los datos solicitados a partir de la información facilitada en el enunciado. En un escenario real con muchos registros no es viable seleccionar toda una tabla entera y buscar los datos manualmente en pantalla.
- No se puede utilizar la cláusula `ALLOW FILTERING`.

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. También se tendrá en cuenta que la solución sea la más óptima posible.

Para adjuntar los comandos que vayáis ejecutando en la práctica es mejor hacerlo en modo texto, tal y como se ha realizado en el ejemplo de carga de datos. Igualmente, para documentar el resultado de la ejecución de los comandos, se puede adjuntar la salida de texto de la consola de comandos como se ha visto en la carga de datos.

Si el texto no cabe en el documento, también podéis adjuntar una captura de pantalla **en la que el texto se pueda leer claramente**.

Consulta 1 (10%):

El departamento de logística quiere saber el stock total de todos los productos que hay en el almacén. La consulta debe retornar una sola cifra. El ejercicio devuelve un warning. ¿Podrías explicar por qué?

SOLUCIÓN:

```
SELECT sum (stock) FROM product_color_size_stock;
```

Que retorna:

```
system.sum(stock)
-----
                96
```

(1 rows)

Warnings :

Aggregation query used without partition key

```
cqlsh:wear_dealer>
```

El comando SELECT 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.

Consulta 2 (15%):

El mismo departamento quiere saber el stock del artículo con referencia 'CASTR' agrupado por colores. La consulta debe detallar el código del producto, el código del color y el stock.

SOLUCIÓN:

```
SELECT productCode, colorcode, SUM(Stock) as Stock
FROM product_color_size_stock
WHERE productCode = 'CASTR'
GROUP BY productCode, colorcode;
```

Que retorna:

productcode	colorcode	stock
CASTR	CB	27
CASTR	WH	40

```
(2 rows)
cqlsh:wear_dealer>
```

Consulta 3 (20%):

Listar los productos sin stock para hacer nuevos pedidos a fábrica. Se requieren el código del producto, el código del color, el código de talla y el EAN.

SOLUCIÓN:

Si ejecutamos la consulta:

```
SELECT ProductCode, ColorCode, SizeCode, EAN
FROM product_color_size_stock
WHERE stock = 0;
```

Nos retorna un error:

```
InvalidRequest: Error from server: code=2200 [Invalid query]
message="Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you want
to execute this query despite the performance unpredictability, use
ALLOW FILTERING"
cqlsh:wear_dealer>
```

Por lo tanto no podemos ejecutar la consulta directamente a la tabla. Como la cláusula ALLOW FILTERING no está permitida, inicialmente tenemos dos posibilidades. Una, crear

una tabla con los datos y una clave de agrupación que permita realizar la consulta. La otra, indexar la columna stock para poderla incluir en la cláusula *where*.

La segunda es más óptima pues crear una nueva tabla requiere de mantenimiento posterior, reservar mucho espacio en disco y requerirá mucho tiempo de ejecución para rellenar todos los datos (no olvidemos que en la realidad esta base de datos tiene un volumen muy elevado). Por estos motivos se considera como la solución más óptima la creación de un índice en la tabla ya existente. Ejecutamos:

```
CREATE INDEX idx_product_color_size_stock_stock ON
product_color_size_stock(stock);
```

Y con el índice creado podemos ejecutar la misma consulta sin errores:

```
SELECT ProductCode, ColorCode, SizeCode, EAN
FROM product_color_size_stock
WHERE stock = 0;
```

Que retorna:

productcode	colorcode	sizecode	ean
BEATS	BL	XS	843245599304
BEATS	BL	XXS	843245599306
CASTR	WH	31	843245599312
CASTR	WH	32	843245599313

(4 rows)

```
cqlsh:wear_dealer>
```

De manera opcional (no es necesario para el ejercicio), si la consulta no es frecuente y no se necesita para otras consultas, se puede eliminar el índice con el comando:

```
DROP INDEX idx_product_color_size_stock_stock;
```

Consulta 4 (15%):

El departamento de marketing quiere revisar el catálogo de productos. Necesitan responder a la pregunta ¿Cuáles son los productos que hay en cada temporada? Se necesita un listado que proporcione Código de temporada, código de producto y la descripción corta. No necesitan ningún recuento de stock.

SOLUCIÓN:

Esta consulta es directa:

```
SELECT SeasonCode, productCode, shortdescription
```

```
FROM seasons_product_color_size
GROUP BY SeasonCode, productCode;
```

Retorna:

seasoncode	productcode	shortdescription
WI	CASTR	Casual trousers
AU	CASTR	Casual trousers
SU	BEATS	Beach T Shirt
SU	RUNTS	Running T Shirt

(4 rows)

Warnings :

Aggregation query used without partition key

cqlsh:wear_dealer>

Consulta 5 (40%):

Actualizar la descripción del producto con código referencia RUNTS para todas las tallas de manera consistente para toda la base de datos. La nueva descripción será 'Running T-Shirt High Performance'.

Al obtener los datos necesarios para realizar las consultas de actualización no se puede hacer un "select * from" de cualquier tabla sin ningún filtro y obtener los datos con un listado de todos los registros de la tabla. En una base de datos real habrá muchos registros y resultará imposible hacerlo, por lo que todos los datos necesarios para realizar las modificaciones se tienen que obtener con consultas debidamente filtradas.

Una vez obtenidos los datos filtrados, como obtendremos una parte pequeña de la tabla y debido a las características de Cassandra, es posible que tengáis que escribir varias instrucciones manualmente para realizar la actualización.

SOLUCIÓN:

Los datos están en dos tablas, por lo que para mantener la base de datos en un estado consistente deben actualizarse ambas. Hacerlo en una sola tabla implicaría dejar la base de datos en un estado inconsistente, requisito obvio y detallado en el enunciado. Comenzaremos por la tabla seasons_product_color_size.

Para poder hacer un update en Cassandra se necesita proporcionar todas las columnas que componen la clave primaria. Como solamente tenemos la referencia, primero deberemos obtener los valores de las columnas que tenemos que utilizar para la sentencia de actualización.

Para esta tabla necesitamos los valores de las columnas SeasonCode, ProductCode, ColorCode y SizeCode. Ejecutamos:

```
select *
from seasons_product_color_size
where productCode = 'RUNTS';
```

Pero igual que en una consulta anterior nos retorna un error:

```
InvalidRequest: Error from server: code=2200 [Invalid query]
message="Cannot execute this query as it might involve data
filtering and thus may have unpredictable performance. If you want
to execute this query despite the performance unpredictability, use
ALLOW FILTERING"
cqlsh:wear_dealer>
```

Por lo que tenemos que indexar la columna productCode. Ejecutamos:

```
create index idx_seasons_product_color_size_productCode on
seasons_product_color_size(productCode);
```

Y ya podemos volver a ejecutar la misma consulta que retornará:

```
cqlsh:wear_dealer> select *
... from seasons_product_color_size
... where productCode = 'RUNTS';
```

seasoncode	productcode	colorcode	sizecode	ean	shortdescription
SU	RUNTS	YE	L	843245599316	Running T Shirt
SU	RUNTS	YE	M	843245599318	Running T Shirt
SU	RUNTS	YE	S	843245599319	Running T Shirt
SU	RUNTS	YE	XL	843245599317	Running T Shirt

```
(4 rows)
cqlsh:wear_dealer>
```

Ya tenemos todos los datos necesarios para escribir el update y así actualizar las filas de la tabla. Al poder utilizar solamente una clave primaria en la cláusula where de un update, este comando solo puede actualizar una única fila. Por lo tanto, tendremos que escribir cuatro updates.

```
update seasons_product_color_size
set shortdescription = 'Running T-Shirt High Performance'
where seasonCode = 'SU' AND productCode = 'RUNTS' AND colorcode =
'YE' AND sizecode = 'L';
```

```
update seasons_product_color_size
set shortdescription = 'Running T-Shirt High Performance'
```



```
where seasonCode = 'SU' AND productCode = 'RUNTS' AND colorcode =
'YE' AND sizecode = 'M';
```

```
update seasons_product_color_size
set shortdescription = 'Running T-Shirt High Performance'
where seasonCode = 'SU' AND productCode = 'RUNTS' AND colorcode =
'YE' AND sizecode = 'S';
```

```
update seasons_product_color_size
set shortdescription = 'Running T-Shirt High Performance'
where seasonCode = 'SU' AND productCode = 'RUNTS' AND colorcode =
'YE' AND sizecode = 'XL';
```

Comprobamos:

```
select *
from seasons_product_color_size
where productCode = 'RUNTS';
```

Retorna:

seasoncode	productcode	colorcode	sizecode	ean	shortdescription
SU	RUNTS	YE	L	843245599316	Running T-Shirt High Performance
SU	RUNTS	YE	M	843245599318	Running T-Shirt High Performance
SU	RUNTS	YE	S	843245599319	Running T-Shirt High Performance
SU	RUNTS	YE	XL	843245599317	Running T-Shirt High Performance

(4 rows)
cqlsh:wear_dealer>

Opcionalmente se puede eliminar el índice:

```
drop index idx_seasons_product_color_size_productCode;
```

Hasta aquí la tabla más elaborada, por lo que puntúa un 60% del ejercicio. Sin embargo, para obtener el 100% se debe actualizar la tabla product_color_size_stock, procedimiento algo menos elaborado porque no es necesario crear ningún índice (la clave de partición es la misma columna que tenemos que especificar en la cláusula WHERE).

Podemos consultar los valores para el where con la consulta:

```
select *
from product_color_size_stock
where productCode = 'RUNTS';
```

Que retorna directamente:

productcode	colorcode	sizecode	ean	shortdescription	stock
RUNTS	YE	L	843245599316	Running T Shirt	5

```
(4 rows)
cqlsh:wear dealer>
```

Igual que anteriormente podemos escribir los updates:

```
update product_color_size_stock
set shortdescription = 'Running T-Shirt High Performance'
where productCode = 'RUNTS' AND colorcode = 'YE' AND sizecode =
'S';
```

```
update product_color_size_stock
set shortdescription = 'Running T-Shirt High Performance'
where productCode = 'RUNTS' AND colorcode = 'YE' AND sizecode =
'L';
```

```
update product_color_size_stock
set shortdescription = 'Running T-Shirt High Performance'
where productCode = 'RUNTS' AND colorcode = 'YE' AND sizecode =
'M';
```

```
update product_color_size_stock
set shortdescription = 'Running T-Shirt High Performance'
where productCode = 'RUNTS' AND colorcode = 'YE' AND sizecode =
'XL';
```

Comprobamos:

```
cqlsh:wear_dealer> select *
... from product_color_size_stock
... where productCode = 'RUNTS';
```

productcode	colorcode	sizecode	ean	shortdescription
stock				
-----+	-----+	-----+	-----+	-----+

5	RUNTS	YE	L 843245599316	Running T-Shirt High Performance
8	RUNTS	YE	M 843245599318	Running T-Shirt High Performance
2	RUNTS	YE	S 843245599319	Running T-Shirt High Performance
7	RUNTS	YE	XL 843245599317	Running T-Shirt High Performance

```
(4 rows)
cqlsh:wear dealer>
```

Ejercicio 1.3 (20%)

Por defecto Cassandra utiliza la primera columna como clave de partición. Sin embargo, se pueden especificar más columnas poniéndolas entre paréntesis al principio de la definición de la clave primaria como se puede ver en la sentencia utilizada:

```
CREATE TABLE seasons_product_color_size (SeasonCode TEXT,
ProductCode TEXT, ColorCode TEXT, SizeCode TEXT, EAN TEXT,
ShortDescription TEXT, PRIMARY KEY ((SeasonCode, ProductCode),
ColorCode, SizeCode));
```

En la tabla `seasons_product_color_size` ¿por qué se ha elegido como clave de partición la combinación de columnas `SeasonCode` y `ProductCode`, en lugar de elegir solamente `SeasonCode`?

SOLUCIÓN:

Primero de todo, si no creáramos esta familia tendríamos que hacer el *group by* de la consulta 4 buscando los datos en diferentes familias y relacionándolos en la aplicación que accede a la base de datos. O bien, hacer el *group by* con la cláusula `ALLOW FILTERING` que no está permitido en el enunciado e implica procesar la consulta en todas las particiones de la familia de columnas (procesado de consulta ineficiente).

Cassandra solamente permite realizar agrupaciones con las columnas que forman parte de la Primary Key en el mismo orden en el que están creadas. Para agrupar los datos en la consulta 4 con la cláusula `GROUP BY SeasonCode y productCode`, será necesario definir estas dos columnas en la clave primaria y en el mismo orden. Por lo tanto, será imprescindible que la columna `SeasonCode` esté al principio.

Por otro lado, **Cassandra utiliza por defecto la primera columna de la clave primaria como clave de dispersión** y el número de valores posibles de la columna `SeasonCode` es de 4 (tal y como se indica en el enunciado, solamente hay cuatro temporadas posibles). Esto nos puede llevar a cometer un error de diseño con consecuencias en el rendimiento y la dispersión de datos. **Si season code fuera la clave de dispersión, los datos de esta tabla solamente podrían dispersarse en un máximo de cuatro particiones.** Por lo tanto, si la base de datos crece y tenemos que añadir más de cuatro particiones para mejorar el rendimiento y poder almacenar todos los datos, los datos de esta tabla no se dispersarán en todas las particiones. Para que la dispersión sea eficiente, los valores de las columnas elegidas deben ser lo suficientemente diferentes como para poder asignarlos en distintas particiones.

Además, es posible que haya más productos de ciertas temporadas que de otras, por lo que no tendríamos los datos distribuidos equitativamente en las particiones (habría particiones de diferentes tamaños).

Del documento Tutorial de Cassandra.pdf: “Una clave compuesta en Cassandra está formada por una clave de partición y una clave de agrupación. La clave de partición se utiliza para ubicar el nodo donde se almacenará el registro. La clave de agrupación indica el orden en que se ordenarán los datos en disco dentro de cada nodo. Por defecto, el primer campo de una clave primaria se utilizará como clave de partición y el resto como clave de agrupación, pero puede configurarse de forma distinta.”. (...) “Todas aquellas [filas] que tengan la misma clave de partición se almacenan juntas. En cambio, la clave de agrupamiento influye en el ordenamiento de las filas que se encuentran almacenadas juntas.”. [Más información.](#)

Ejercicio 2: MongoDB (30%)

Para este ejercicio considera la base de datos descrita en el documento “*Diseño de una base de datos para una app de mensajería instantánea*” que se encuentra en los materiales del curso.

También se necesitará la máquina virtual LinuxMint que contiene una instalación de MongoDB y la base de datos ya cargada. **Para este ejercicio no es necesario cargar ningún dato adicional.**

Inicia el servicio de MongoDB y accede a los datos con los siguientes comandos:

```
sudo systemctl start mongod
```

Deja pasar unos segundos para que el servicio arranque y después ejecuta:

```
mongo
```

Y finalmente deberás entrar en la base de datos en cuestión:

```
use mensajeria
```

Para más información, lee el manual de la máquina virtual que encontrarás en los recursos de la asignatura.

Se pide proporcionar las sentencias (en texto) para el shell de MongoDB y los resultados que se obtienen (haciendo una captura de pantalla o adjuntando el texto retornado) para las siguientes consultas:

Consulta 1 (20%):

De la colección Contactos listar los documentos que tengan un contacto cuya edad sea igual o mayor que 52 años ordenado por identificador (no el campo “_id”) ascendente. El listado debe contener el identificador (no el campo “_id”), el nombre y los apellidos del usuario, y el nombre y los apellidos del contacto.

SOLUCIÓN:

Ejecutamos la consulta:

```
db.Contactos.find({
  "Contacto.Edad": { $gte: 52 }
},
{
  "_id": 0,
```

```
"Identificador" : 1,
"Usuario.Nombre": 1,
"Usuario.Apellidos": 1,
"Contacto.Nombre": 1,
"Contacto.Apellidos": 1
}).sort({"Identificador" : 1}).pretty()
```

Que retorna:

```
{
  "Identificador" : "Contacto-35",
  "Usuario" : {
    "Nombre" : "Alfonso",
    "Apellidos" : "Martínez Osorio"
  },
  "Contacto" : {
    "Nombre" : "Amelia",
    "Apellidos" : "Martín Bosques"
  }
}
{
  "Identificador" : "Contacto-42",
  "Usuario" : {
    "Nombre" : "Ramón",
    "Apellidos" : "Pérez Amigo"
  },
  "Contacto" : {
    "Nombre" : "Amelia",
    "Apellidos" : "Martín Bosques"
  }
}
{
  "Identificador" : "Contacto-50",
  "Usuario" : {
    "Nombre" : "Carmen",
    "Apellidos" : "Aragón Cebrián"
  },
  "Contacto" : {
    "Nombre" : "Amelia",
    "Apellidos" : "Martín Bosques"
  }
}
{
  "Identificador" : "Contacto-86",
  "Usuario" : {
    "Nombre" : "Elsa",
    "Apellidos" : "Serna Risco"
  }
}
```

```

    },
    "Contacto" : {
        "Nombre" : "Amelia",
        "Apellidos" : "Martín Bosques"
    }
}

```

Consulta 2 (20%):

De la colección Desbloquesos listar los desbloquesos realizados por el usuario con email "jsanzrobles@hotmail.es". El listado debe mostrar el nombre y el email del usuario desbloqueado, y el identificador del desbloqueo (no el campo "_id").

SOLUCIÓN:

Ejecutamos la consulta:

```

db.Desbloquesos.find(
    {"Usuario_desbloqueador.Email": "jsanzrobles@hotmail.es"},
    {"_id" : 0, "Usuario_desbloqueado.Nombre":1,
"Usuario_desbloqueado.Email":1, "Identificador": 1});

```

Que retorna:

```

{ "Usuario_desbloqueado" : { "Nombre" : "Mercedes", "Email" :
"mreysordo@gmail.es" }, "Identificador" : "Desbloqueo-1" }
{ "Usuario_desbloqueado" : { "Nombre" : "Isabel", "Email" :
"ilopezmena@gmail.es" }, "Identificador" : "Desbloqueo-2" }

```

Consulta 3 (30%):

De la colección Usuarios_grupos listar los tres usuarios que son propietarios de más grupos en orden descendente (el usuario que tiene más grupos en propiedad debe aparecer el primero). La consulta debe retornar solamente los tres primeros, mostrar el correo electrónico del propietario y el recuento de los grupos de los que es propietario. Se puede asumir que no hay correos electrónicos repetidos en esta colección, no es necesario realizar ninguna comprobación adicional.

SOLUCIÓN:

Ejecutamos la consulta:

```

db.Usuarios_grupos.aggregate([
    {$project: {"_id": "$Email", "Total_grupos":{$size:"$Grupos_propietario"} }},
    {$sort: {"Total_grupos":-1}},
    {$limit: 3}
]);

```

Que retorna:

```
{ "_id" : "mreysordo@gmail.es", "Total_grupos" : 5 }
{ "_id" : "efrancolopez@gmail.es", "Total_grupos" : 4 }
{ "_id" : "lsagradosanzen@gmail.es", "Total_grupos" : 3 }
>
```

Consulta 4 (30%):

De la colección `Contactos_usuarios` y del usuario con email mgarciasanz@gmail.es, seleccionar solamente los contactos de este usuario que tengan una edad igual o mayor que 36 años. La consulta debe retornar el email del usuario (mgarciasanz@gmail.es), y el email y edad de sus contactos. Los contactos de este usuario que no estén en el rango de edad especificado no deben salir en la consulta.

Para esta consulta y a modo de ayuda, se adjunta el resultado esperado a continuación. Por favor, adjuntar la consulta en modo texto para que se pueda comprobar que retorna el resultado indicado. El resultado de la consulta debe ser:

```
{   "_id"       :   "mgarciasanz@gmail.es",   "EmailContacto"       :
"vtiernocrespo@hotmail.es", "EdadContacto" : 41 }
{   "_id"       :   "mgarciasanz@gmail.es",   "EmailContacto"       :
"adelgadosanchez@hotmail.es", "EdadContacto" : 37 }
{   "_id"       :   "mgarciasanz@gmail.es",   "EmailContacto"       :
"tjazzmintablas@hotmail.es", "EdadContacto" : 36 }
>
```

SOLUCIÓN:

Se puede pensar que poniendo el filtro en un *find* funcionará con una consulta como la que sigue:

```
db.Contactos_usuarios.find(
  {"Email" : "mgarciasanz@gmail.es",
  "Contactos.Usuario_contacto.Edad":{$gte:36}},
  {"_id" :0, "Contactos.Usuario_contacto.Email":1,
  "Contactos.Usuario_contacto.Edad":1});
```

Sin embargo, esta consulta no retorna los datos que se solicitan. Sí que retorna el usuario que se solicita, pero también todos los contactos del array y el enunciado requiere solamente los contactos de edad mayor o igual que 36 años.

Esto sucede porque cuando se filtra por un valor que forma parte de un array en la función `find`, el filtro selecciona los documentos que cumplen la condición, **pero retorna los documentos enteros**. Explicado funcionalmente: si un solo elemento del array cumple con la condición, la consulta retornará todo el documento entero incluyendo todos los elementos

del array que está dentro del documento (tanto los elementos del array que cumplen con el filtro como los que no).

El resultado de la consulta incorrecta es el siguiente (en amarillo están resaltados los elementos que no deben estar):

```
{  "Contactos"  :  [  {  "Usuario_contacto"  :  {  "Email"  :  
"bbellovalero@gmail.es", "Edad" : 29 } }, { "Usuario_contacto" : {  
"Email"  :  "vtiernocrespo@hotmail.es",  "Edad"  :  41  }  }, {  
"Usuario_contacto"  :  {  "Email"  :  "adelgadosanchez@hotmail.es",  
"Edad"  :  37  }  }, {  "Usuario_contacto"  :  {  "Email"  :  
"lgutierrezamor@gmail.es", "Edad" : 28 } }, { "Usuario_contacto" :  
{ "Email" : "tjazmintablas@hotmail.es", "Edad" : 36 } } ] }  
>
```

Para poder ejecutar la consulta correctamente es necesario utilizar una *aggregation pipeline* y principalmente la cláusula \$unwind. El primer paso de la agregación recibe como entrada toda la colección y establece una salida de datos. El paso siguiente tendrá como entrada la salida del paso anterior y así sucesivamente.

El primer paso es filtrar por el documento que queremos (\$match) así los siguientes pasos no tendrán que lidiar con todos los datos de la colección, lo harán con un solo documento.

El segundo paso es convertir el array en documentos individuales (\$unwind). Como recibimos un solo documento de la salida anterior, no convertiremos en documentos todos los arrays de toda la colección proporcionando una salida con muchos datos, sino que expandimos solamente el array de un solo documento (esta secuencia es eficiente).

El siguiente paso será filtrar los elementos expandidos del array (ahora documentos) con el criterio que indica el enunciado (otra cláusula \$match)

Y por último, proyectar las columnas que queremos mostrar (\$project).

La consulta es:

```
db.Contactos_usuarios.aggregate([  
  {$match: {"Email" : "mgarciasanz@gmail.es"}},  
  {$unwind: "$Contactos"},  
  {$match: {"Contactos.Usuario_contacto.Edad":{$gte:36}}},  
  {$project: { "_id": "$Email",  
"EmailContacto": "$Contactos.Usuario_contacto.Email",  
"EdadContacto": "$Contactos.Usuario_contacto.Edad" } }  
]);
```

Que retorna:

```
{   "_id"      :   "mgarciasanz@gmail.es",   "EmailContacto"   :
"vtiernocrespo@hotmail.es", "EdadContacto" : 41 }
{   "_id"      :   "mgarciasanz@gmail.es",   "EmailContacto"   :
"adelgadosanchez@hotmail.es", "EdadContacto" : 37 }
{   "_id"      :   "mgarciasanz@gmail.es",   "EmailContacto"   :
"tjazzmintablas@hotmail.es", "EdadContacto" : 36 }
>
```

Ejercicio 3: Neo4j (20 %)

Para la realización de este ejercicio se seguirán las instrucciones del caso de estudio ubicado en la siguiente URL: <https://neo4j.com/developer/guide-importing-data-and-etl/>. Este caso se aborda también en el documento de ejercicios titulado “Transformación de una base de datos relacional a un modelo en grafo” pero a un nivel más superficial. Se recomienda leer dicho documento antes de realizar este ejercicio.

Se recomienda leer con atención la página web, ya que se introducen conceptos y buenas prácticas a seguir en ‘production’.

En la máquina virtual ya hay una base de datos precargada (twitter) que no es necesaria para realizar los ejercicios de esta práctica. Los nodos y relaciones que hay creados no interfieren en el ejercicio, los podéis ignorar tranquilamente.

Encontraréis las instrucciones para arrancar el servicio de Neo4j en la página 11 del documento con nombre “Uso de máquina virtual_ Bases de datos no convencionales.pdf”. Recordad que el usuario / contraseña para acceder a Neo4j son: **neo4j / uoc**

Ejercicio 3.1 (no puntúa):

Carga de datos. A continuación, se añaden las instrucciones (queries) para crear nuevos grafos en neo4j. Aunque se puede hacer la carga mediante un *:play northwind-graph*, se propone realizar las sentencias una a una para una mejor comprensión de las mismas y de los datos se cargan en cada caso.

NODOS

```
// Create orders
```

```
LOAD CSV WITH HEADERS FROM
```

```
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/orders.csv' AS row
```

```
MERGE (order:Order {orderID: row.OrderID})
```

```
ON CREATE SET order.shipName = row.ShipName;
```

Que retornará:

```
0 rows available after 1781 ms, consumed after another 0 ms
```

```
Added 830 nodes, Set 1660 properties, Added 830 labels
```

```
// Create products
```

```
LOAD CSV WITH HEADERS FROM
```

```
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/products.csv' AS row
```

```
MERGE (product:Product {productID: row.ProductID})
```

```
ON CREATE SET product.productName = row.ProductName, product.unitPrice =  
toFloat(row.UnitPrice);
```

Que retornará:

```
0 rows available after 332 ms, consumed after another 0 ms
```

```
Added 77 nodes, Set 231 properties, Added 77 labels
```

```
// Create suppliers
```

```
LOAD CSV WITH HEADERS FROM
```

```
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/suppliers.csv' AS row
```

```
MERGE (supplier:Supplier {supplierID: row.SupplierID})
```

```
ON CREATE SET supplier.companyName = row.CompanyName;
```

Que retornará:

```
0 rows available after 258 ms, consumed after another 0 ms
```

```
Added 29 nodes, Set 58 properties, Added 29 labels
```

```
// Create employees
```

```
LOAD CSV WITH HEADERS FROM
```

```
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/employees.csv' AS row
```

```
MERGE (e:Employee {employeeID:row.EmployeeID})
```

```
ON CREATE SET e.firstName = row.FirstName, e.lastName = row.LastName, e.title =  
row.Title;
```

Que retornará:

```
0 rows available after 244 ms, consumed after another 0 ms
```

```
Added 9 nodes, Set 36 properties, Added 9 labels
```

```
// Create categories
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36df
c88381995e6823ff3f419b5a0cb8ac4f/categories.csv' AS row
MERGE (c:Category {categoryID: row.CategoryID})
ON CREATE SET c.categoryName = row.CategoryName, c.description = row.Description;
```

Que retornará:

```
0 rows available after 232 ms, consumed after another 0 ms
Added 8 nodes, Set 24 properties, Added 8 labels
```

RELACIONES

```
// Create relationships between orders and products
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36df
c88381995e6823ff3f419b5a0cb8ac4f/orders.csv' AS row
MATCH (order:Order {orderID: row.OrderID})
MATCH (product:Product {productID: row.ProductID})
MERGE (order)-[op:CONTAINS]->(product)
ON CREATE SET op.unitPrice = toFloat(row.UnitPrice), op.quantity = toFloat(row.Quantity);
```

Que retornará:

```
0 rows available after 1832 ms, consumed after another 0 ms
Created 2155 relationships, Set 4310 properties
```

```
// Create relationships between orders and employees
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36df
c88381995e6823ff3f419b5a0cb8ac4f/orders.csv' AS row
MATCH (order:Order {orderID: row.OrderID})
MATCH (employee:Employee {employeeID: row.EmployeeID})
MERGE (employee)-[:SOLD]->(order);
```

Que retornará:

```
0 rows available after 1285 ms, consumed after another 0 ms
Created 830 relationships
```

```
// Create relationships between products and suppliers
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36df
c88381995e6823ff3f419b5a0cb8ac4f/products.csv'
' AS row
```

```
MATCH (product:Product {productID: row.ProductID})
MATCH (supplier:Supplier {supplierID: row.SupplierID})
MERGE (supplier)-[:SUPPLIES]->(product);
```

Que retornará:

```
0 rows available after 131 ms, consumed after another 0 ms
Created 77 relationships
```

```
// Create relationships between products and categories
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/products.csv'
AS row
MATCH (product:Product {productID: row.ProductID})
MATCH (category:Category {categoryID: row.CategoryID})
MERGE (product)-[:PART_OF]->(category);
```

Que retornará:

```
0 rows available after 264 ms, consumed after another 0 ms
Created 77 relationships
```

```
// Create relationships between employees (reporting hierarchy)
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/employees.csv' AS row
MATCH (employee:Employee {employeeID: row.EmployeeID})
MATCH (manager:Employee {employeeID: row.ReportsTo})
MERGE (employee)-[:REPORTS_TO]->(manager);
```

Que retornará:

```
0 rows available after 86 ms, consumed after another 0 ms
Created 8 relationships
```

Ejercicio 3.2 (30%):

Se propone responder las siguientes cuestiones:

1. Una vez realizada la carga de datos, utilizando cypher y su interfaz web, se pide:
 - a. obtener una visualización del esquema de los nodos/relaciones que se han creado.
 - b. contar cuántos nodos de tipo `Supplier` se han creado (adjuntar una captura de pantalla de las query y resultado).
2. Explicar brevemente (máx 3 líneas) la funcionalidad implementada de la línea:

```
ON CREATE SET      c.categoryName = row.CategoryName,
```

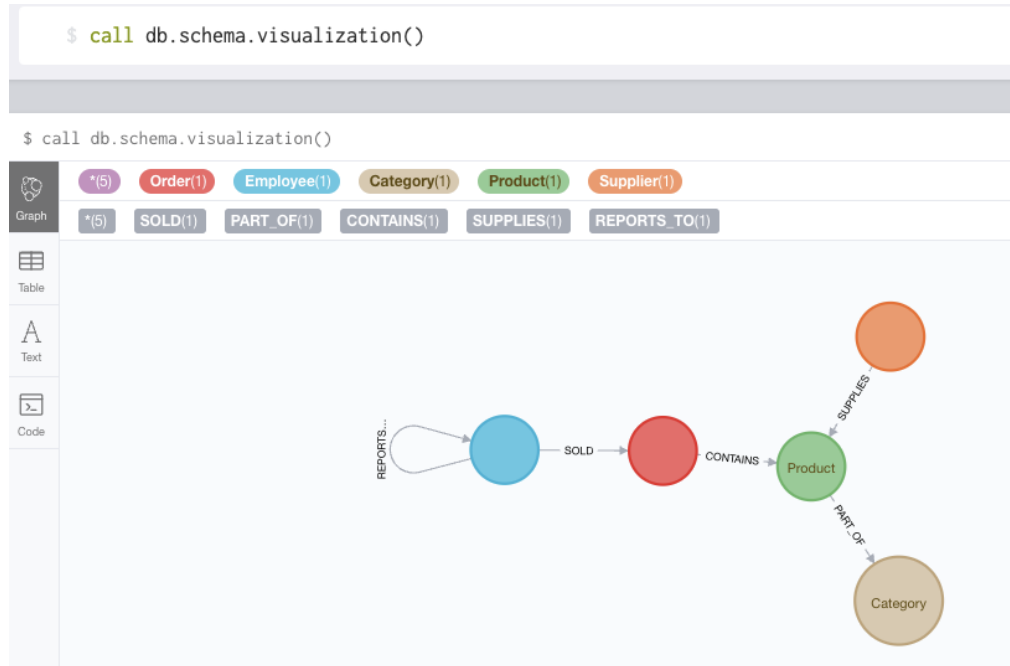
```
c.description = row.Description;
```

- En la página web, se crea un índice para cada tipo de nodo y una restricción de unicidad para los nodos de tipo `Order`. Explicar brevemente (máx 3 líneas): ¿Por qué se recomienda crear un índice para cada tipo de nodo?

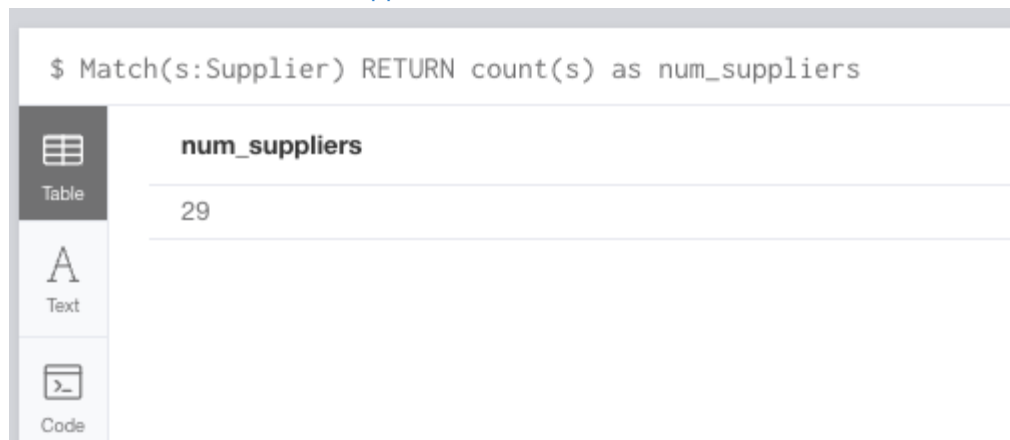
SOLUCIÓN:

- Para crear nodos y relaciones, solo hay que copiar y pegar las diferentes queries.

Para visualizar el esquema en neo4j



El número total de nodos `Supplier` es 29.



The image shows a Neo4j query result. At the top, there is a command bar with the text: `$ Match(s:Supplier) RETURN count(s) as num_suppliers`. Below this, a toolbar contains icons for Table, Text, and Code. The main area displays a table with the following content:

num_suppliers
29

- SET ... indica cómo asignar el valor de las propiedades de los nodos que se crean usando el valor de una columna del .csv de datos 'a rellenar'. La sentencia permite asignar a los atributos `categoryname` y `description` los valores homónimos del fichero de entrada.

3. Los índices creados para cada tipo de nodo mejoran el rendimiento a la hora de crear las relaciones, ya que durante este proceso se debe acceder de forma rápida a todos los nodos creados.

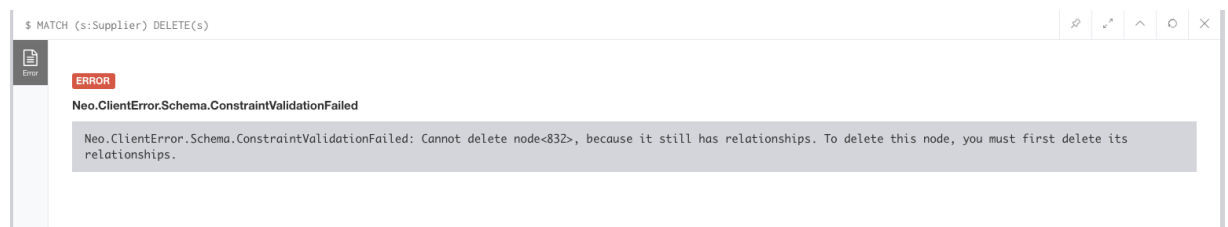
Ejercicio 3.3 (30%):

Se pide borrar todos los nodos de tipo **Supplier**: Adjuntar una captura de pantalla con las queries utilizadas para eliminar todos estos nodos. Ayuda: no es posible borrar nodos que tengan relaciones con otros.

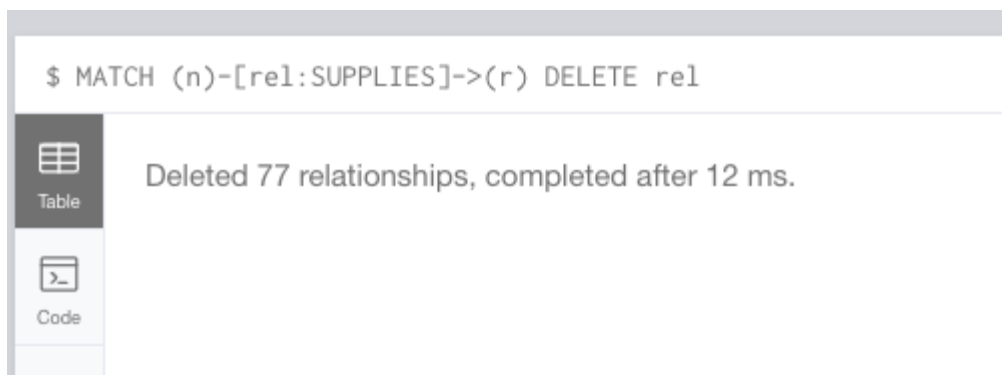
SOLUCIÓN:

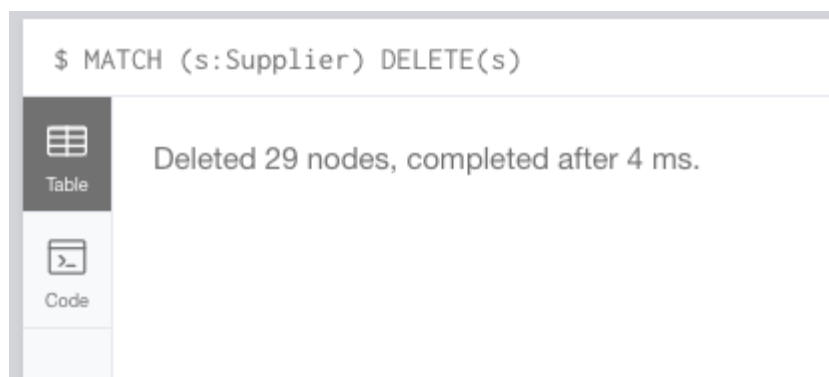
No es posible borrar todos los **Supplier** de manera directa (MATCH (s:Supplier) DELETE s), ya que primero hay que borrar las relaciones a las cuales los nodos pertenecen. Al intentar, neo4j alerta con un el mensaje:

```
Neo.ClientError.Schema.ConstraintValidationFailed: Cannot delete node<832>, because it still has relationships. To delete this node, you must first delete its relationships.
```



Los **Supplier** están relacionados en **SUPPLIES**: por lo que primero hay que borrar **SUPPLIES** y luego ejecutar la query anterior (MATCH (s:Supplier) DELETE s).





También se podría realizar con un *optional match* o con un *detach* (en función de la versión de Neo4j con la que se trabaje):

```
MATCH (s:Supplier) OPTIONAL MATCH (s:Supplier)-[rel:SUPPLIES]->() DELETE s, rel
```

```
MATCH (s:Supplier) DETACH DELETE
```

Ejercicio 3.4 (40%):

Se pide recrear los nodos `Supplier` y las relaciones `SUPPLIES` para restablecer el estado de la BBDD¹ y poder responder a las siguientes preguntas:

- Cuantos `Supplier` provienen de Francia? (ayuda: buscar 'France' ya que la información es en inglés).
- Cuales son los `Supplier` cuya web de referencia apunta a una URL absoluta por cada país? Es decir, que contenga una página que empiece por "http".

Se deberá proporcionar las sentencias de creación utilizadas, las dos consultas planteadas y los resultados de las mismas.

¹ Para hacerlo podéis volver a ejecutar las operaciones de carga que se especifican al principio de este ejercicio, pero añadiendo los datos necesarios para resolver las preguntas planteadas. Encontraréis información al respecto en la página web de referencia.

SOLUCIÓN:

La query para crear los nodos **Supplier** debe incluir más información (columnas) extraída por los ficheros .csv

```
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/suppliers.csv' AS row
MERGE (s:Supplier {supplierID: row.SupplierID})
ON CREATE SET s.companyName = row.CompanyName, s.estado = row.Country, s.web = row.HomePage;
```

```
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/products.csv' AS row MATCH (product:Product {productID: row.ProductID}) MATCH (supplier:Supplier {supplierID: row.SupplierID}) MERGE (supplier)-[:SUPPLIES]->(product);
```

```
1 LOAD CSV WITH HEADERS FROM
  'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/suppliers.csv' AS row
2 MERGE (s:Supplier {supplierID: row.SupplierID})
3 ON CREATE SET s.companyName = row.CompanyName, s.estado = row.Country, s.web = row.HomePage;
```

To enjoy the full Neo4j Browser experience, we advise you to use [Neo4j Browser Sync](#).

\$ LOAD CSV WITH HEADERS FROM 'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/suppliers.csv' AS r...

Table

Code

Added 29 labels, created 29 nodes, set 92 properties, completed after 294 ms.

Hay 3 **Supplier** que provienen de Francia (France en inglés)

\$ match (s:Supplier) where s.estado = 'France' return count(s) as num_fr_suppliers;

Table

Text




Code

num_fr_suppliers
3

Hay **Supplier** con un url bien formateado: para descubrirlos, se usa la clausula WHERE y CONTAINS, ya que al no ponerla, la mayoría de **Supplier** no tienen el campo informado, o tienen solo 'páginas' HTML que no son accesibles por internet (p.e. #CAJUN.HTM#).

```
1 match (s:Supplier)
2 with s.companyName as company
3 where s.web CONTAINS 'http'
4 return company;
```

```
$ match (s:Supplier) with s.companyName as company where s.web CONTAINS 'http' return company;
```

	company
Table	"Mayumi's"
	"Plutzer Lebensmittelgroßmärkte AG"
Text	"G'day, Mate"
	
Code	

Ejercicio 4: Neo4j (20 %)

Este ejercicio asume que se ha resuelto el ejercicio anterior. En caso que hayas resuelto el ejercicio 3.B (borrar nodos `Supplier`) pero no hayas recuperado el estado inicial del DDBB (resuelto el ejercicio 3.C), antes de seguir con este ejercicio crea de nuevo los nodos `Supplier` y las relaciones `SUPPLIES` ejecutando de nuevo estos dos comandos:

```
// Create suppliers
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/suppliers.csv'
AS row
MERGE (supplier:Supplier {supplierID: row.SupplierID})
ON CREATE SET supplier.companyName = row.CompanyName;

// Create relationships between products and suppliers
LOAD CSV WITH HEADERS FROM
'https://gist.githubusercontent.com/jexp/054bc6baf36604061bf407aa8cd08608/raw/8bdd36dfc88381995e6823ff3f419b5a0cb8ac4f/products.csv'
AS row
MATCH (product:Product {productID: row.ProductID})
MATCH (supplier:Supplier {supplierID: row.SupplierID})
MERGE (supplier)-[:SUPPLIES]->(product);
```

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

Consulta 1 (20%):

Encontrar el empleado con título “Sales Representative” que ha vendido (SOLD) más pedidos que contienen la palabra “White”. Listar nombre y apellido del empleado y número de pedidos vendidos

SOLUCIÓN:

```
MATCH(e:Employee{title:"Sales Representative"})-[r:SOLD]->(o)
```

WHERE o.shipName CONTAINS "White"
WITH e, count(r) as num_orders
RETURN e.firstName, e.lastName, num_orders
ORDER BY num_orders DESC LIMIT 1

```
$ MATCH(e:Employee{title:"Sales Representative"})-[r:SOLD]->(o) where o.shipName CONTAINS "White" with e, count(r) as num_orders return e.firstName, e.lastName, num_orders ORDER BY num_orders DESC LIMIT 1
```

```
$ MATCH(e:Employee{title:"Sales Representative"})-[r:SOLD]->(o) where o.shipName CONTAINS "White" with e, count(r) as num_o...
```

e.firstName	e.lastName	num_orders
"Margaret"	"Peacock"	4

Consulta 2 (20%):

Listar el nombre de la categoría asociada (PART_OF) a 6 productos

SOLUCIÓN:

MATCH (p)-[r:PART_OF]->(c)
WITH c.categoryName as category, count(p) as num_products
WHERE num_products = 6
RETURN category

```
1 MATCH (p)-[r:PART_OF]->(c) with c.categoryName as category, count(p) as num_products where num_products = 6
2 RETURN category
```

```
$ MATCH (p)-[r:PART_OF]->(c) with c.categoryName as category, count(p) as num_products where num_products = 6 RETURN catego...
```

category
"Meat/Poultry"

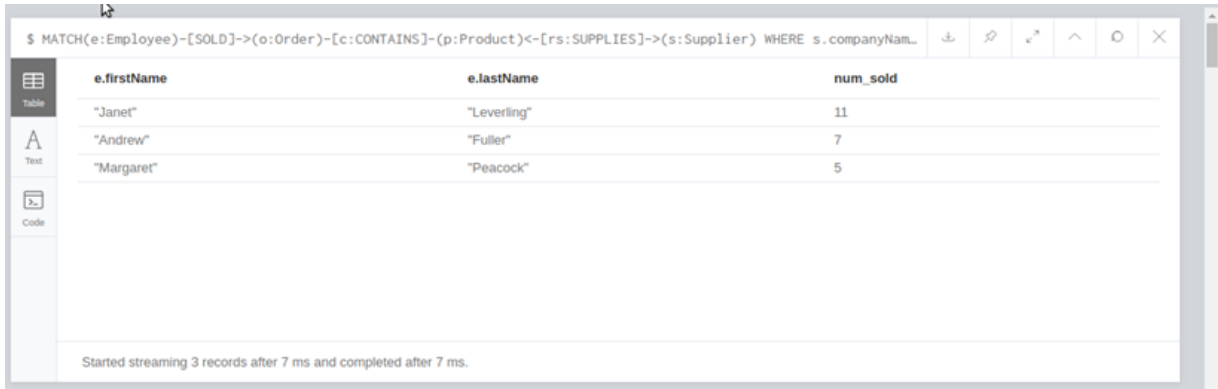
Consulta 3 (30%):

Obtener el segundo, tercer y cuarto mejores vendedores de "Tokyo Traders". Los mejores vendedores son aquellos que han vendido más productos de "Tokyo Traders". Listar sólo el nombre y apellido de los vendedores y el número de unidades vendidas por vendedor.

SOLUCIÓN:

MATCH(e:Employee)-[SOLD]->(o:Order)-[c:CONTAINS]-(p:Product)<-[rs:SUPPLIES]-

```
>(s:Supplier)
WHERE s.companyName = 'Tokyo Traders'
WITH e, count(p) as num_sold
RETURN e.firstName, e.lastName, num_sold
ORDER BY num_sold DESC
SKIP 1 LIMIT 3
```



The screenshot shows a query execution interface with a table view. The table has three columns: e.firstName, e.lastName, and num_sold. It contains three rows of data. Below the table, a status message indicates that 3 records were streamed after 7 ms and completed after 7 ms.

e.firstName	e.lastName	num_sold
"Janet"	"Leverling"	11
"Andrew"	"Fuller"	7
"Margaret"	"Peacock"	5

Started streaming 3 records after 7 ms and completed after 7 ms.

Consulta 4 (30%):

El propietario de las empresas "Leka Trading", "Karkki Oy", "Pavlova, Ltd." quiere centrarse en el mercado del marisco ("Seafood" en la base de datos). Para ello, ha adquirido las empresas "Lyngbysild", "Ma Maison", "Tokyo Traders". Después de hacerlo, el propietario nos pide que calculemos cuál ha sido el impacto de dichas adquisiciones en el número de productos de marisco vendidos. Para dar respuesta a esta pregunta deberemos calcular el marisco vendido por las empresas originales (antes de la adquisición) y el marisco vendido por todas sus empresas después de su adquisición.

SOLUCIÓN:

```
match (s:Supplier)-[SUPPLIES]-(p:Product)-[PART_OF]->(c:Category)
where s.companyName IN ["Leka Trading", "Karkki Oy", "Pavlova, Ltd."] and
      c.categoryName = "Seafood"
with count(p) as num_product_seafood_cong_A
```

```
match (s:Supplier)-[SUPPLIES]-(p:Product)-[PART_OF]->(c:Category)
where s.companyName IN ["Lyngbysild", "Ma Maison", "Tokyo Traders"] and
      c.categoryName = "Seafood"
with count(p) as num_product_seafood_cong_B, num_product_seafood_cong_A
```

```
return      num_product_seafood_cong_A,      num_product_seafood_cong_B      +
            num_product_seafood_cong_A as new_cong_num_products
```

```
2 where s.companyName IN ['Leka Trading', 'Karkki Oy', 'Pavlova, Ltd.'] and c.categoryName = 'Seafood'
3 with count(p) as num_product_seafood_cong_A
4 match (s:Supplier)-[SUPPLIES]-(p:Product)-[PART_OF]->(c:Category)
5 where s.companyName IN ['Lyngbysild', 'Ma Maison', 'Tokyo Traders'] and c.categoryName = 'Seafood'
6 with count(p) as num_product_seafood_cong_B, num_product_seafood_cong_A
7 return num_product_seafood_cong_A, num_product_seafood_cong_B + num_product_seafood_cong_A as
    new_cong_num_products
```

```
$ match (s:Supplier)-[SUPPLIES]-(p:Product)-[PART_OF]->(c:Category) where s.companyName IN ["Leka Trading", "Karkki Oy", "P...
```



num_product_seafood_cong_A	new_cong_num_products
1	4

A

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 al buzón de Entrega y registro de EC disponible en el aula (apartado Evaluación). 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 apellido1_nombreCompleto_nosql_pra1.pdf

La fecha límite para entregar la PRA1 es el **2 de enero de 2022**.

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.