



**Universitat Oberta
de Catalunya**

Máster universitario de Ciencia de Datos

Práctica 1 – PRA1

**Uso de base de datos NoSQL - Cassandra, MongoDB y
Neo4j.**

Autor:

Mario Ubierna San Mamés

Índice de Contenido

Índice de Contenido	3
Índice de ilustraciones	5
1. Ejercicio 1: Cassandra	7
1.1. Ejercicio 1.1 (no puntúa)	8
1.2. Ejercicio 1.2 (80%)	8
1.2.1. Consulta 1 (10%)	8
1.2.2. Consulta 2 (15%)	9
1.2.3. Consulta 3 (20%)	10
1.2.4. Consulta 4 (15%)	10
1.2.5. Consulta 5 (40%)	11
1.3. Ejercicio 1.3 (20%)	15
2. Ejercicio 2: MongoDB.....	17
2.1. Consulta 1 (20%).....	17
2.2. Consulta 2 (20%).....	18
2.3. Consulta 3 (30%).....	19
2.4. Consulta 4 (30%).....	21
3. Ejercicio 3: Neo4j	23
3.1. Ejercicio 3.1 (no puntúa)	23
3.2. Ejercicio 3.2 (30%)	23
3.2.1. Comprobaciones de la carga de datos	24
3.2.2. Explicar la funcionalidad implementada	25
3.2.3. ¿Por qué se recomienda crear un índice para cada tipo de nodo? ..	25
3.3. Ejercicio 3.3 (30%)	25

3.4.	Ejercicio 3.4.....	26
3.4.1.	Recreación de los nodos Supplier y las relaciones SUPPLIES.....	27
3.4.2.	¿Cuántos Supplier provienen de Francia?.....	27
3.4.3.	¿Cuántos Supplier hay por cada país con una URL absoluta?	27
4.	Ejercicio 4: Neo4j	29
4.1.	Consulta 1 (20%).....	29
4.2.	Consulta 2 (20%).....	30
4.3.	Consulta 3 (30%).....	30
4.4.	Consulta 4 (30%).....	31
5.	Bibliografía	32

Índice de ilustraciones

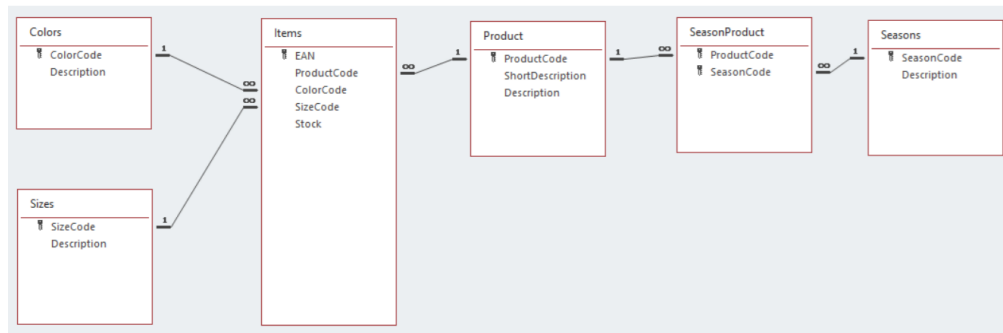
Ilustración 1 - Cassandra consulta y resultado consulta 1.....	8
Ilustración 2 - Cassandra consulta y resultado consulta 2.....	9
Ilustración 3 - Cassandra consulta y resultado consulta 3.....	10
Ilustración 4 - Cassandra consulta y resultado consulta 4.....	11
Ilustración 5 - Cassandra consulta y resultado consulta 5.....	13
Ilustración 6 - Cassandra consulta y resultado consulta 5.....	15
Ilustración 7 - MongoDB consulta y resultado de la consulta 1.	18
Ilustración 8 - MongoDB consulta y resultado de la consulta 2.	19
Ilustración 9 - MongoDB consulta y resultado de la consulta 3.	20
Ilustración 10 - MongoDB consulta y resultado de la consulta 4.	22
Ilustración 11 – función para obtener el esquema.....	24
Ilustración 12 - Visualización del esquema.....	24
Ilustración 13 - Consulta para obtener el número de nodos de tipo Supplier.	25
Ilustración 14 - Resultado del número de nodos de tipo Supplier.	25
Ilustración 15 - Borrado de los nodos tipo Supplier.	26
Ilustración 16 - Resultado del borrado de los nodos tipo Supplier.	26
Ilustración 17 - Creación de los nodos de tipo Supplier.	27
Ilustración 18 - Creación de las relaciones entre Supplier y Product.	27
Ilustración 19 - Consulta cuántos Supplier provienen de Francia.	27
Ilustración 20 - Resultado cuántos Supplier provienen de Francia.	27
Ilustración 21 - Consulta cuántos países con URL absoluta.....	28
Ilustración 22 - Resultado cuántos países con URL absoluta.....	28
Ilustración 23 – Neo4j consulta 1.	29
Ilustración 24 - Neo4j resultado consulta 1.	29
Ilustración 25 - Neo4j consulta 2.	30

Ilustración 26 - Neo4j resultado consulta 2.....	30
Ilustración 27 - Neo4j consulta 3.	30
Ilustración 28 - Neo4j resultado consulta 3.....	31
Ilustración 29 - Neo4j consulta 4.	31
Ilustración 30 - Neo4j resultado consulta 4.....	31

1. Ejercicio 1: Cassandra

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.

1.1. Ejercicio 1.1 (no puntúa)

Mirar el enunciado de la práctica (carga de los datos).

1.2. 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.

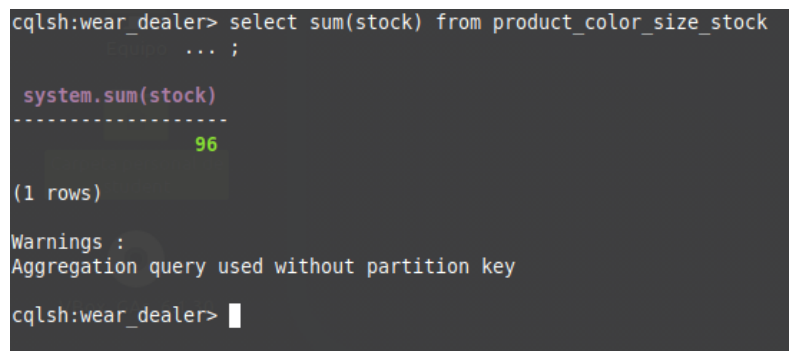
1.2.1. Consulta 1 (10%)

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?

La consulta es:

```
select sum(stock) from product_color_size_stock ;
```

El resultado de la consulta:



```
cqlsh:wear_dealer> select sum(stock) from product_color_size_stock
... ;

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

(1 rows)

Warnings :
Aggregation query used without partition key

cqlsh:wear_dealer> █
```

Ilustración 1 - Cassandra consulta y resultado consulta 1.

En este caso al ejecutar la consulta se produce un *warning*, esto se debe a que estamos haciendo uso de una *select* en la que el agregado es definido por nosotros sin una clave de partición, es decir, estamos haciendo una suma de todos los valores que hay en *stock*. Esto puede ser contraproducente si hay muchas filas, ya que para resolver esta consulta se tiene que leer todas las filas que hay en la tabla, pudiendo llegar al tiempo máximo establecido para una consulta y afectar al rendimiento.

Por lo tanto, lo que nos dice este *warning* es que es mejor hacer uso de una clave de partición, para así solo leer los datos de la clave de partición que se indique, mejorando el rendimiento de la consulta y del servidor de *Cassandra*.

1.2.2. 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.

La consulta realizada es:

```
SELECT productcode, colorcode, SUM(stock)
```

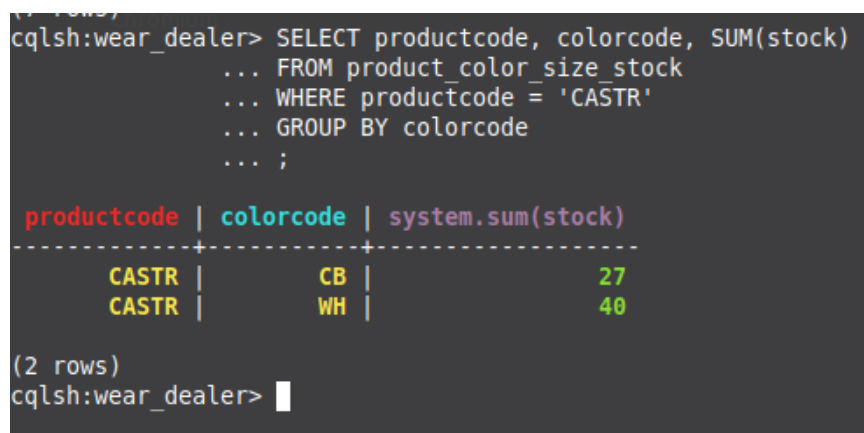
```
FROM product_color_size_stock
```

```
WHERE productcode = 'CASTR'
```

```
GROUP BY colorcode
```

```
;
```

El resultado de la consulta es:



```
cqlsh:wear_dealer> SELECT productcode, colorcode, SUM(stock)
... FROM product_color_size_stock
... WHERE productcode = 'CASTR'
... GROUP BY colorcode
... ;
```

productcode	colorcode	system.sum(stock)
CASTR	CB	27
CASTR	WH	40

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

Ilustración 2 - Cassandra consulta y resultado consulta 2.

1.2.3. 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.

La consulta es:

```
SELECT productcode, colorcode, sizecode, ean
```

```
FROM product_color_size_stock
```

```
WHERE stock = 0
```

```
;
```

El resultado que nos genera:

```
(18 rows)
cqlsh:wear_dealer> SELECT productcode, colorcode, sizecode, ean
... FROM product_color_size_stock
... WHERE stock = 0
... ;
InvalidRequest: Status 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 implications, use ALLOW FILTERING
```

Como podemos apreciar nos devuelve un error del tipo `code=2200 Invalid query`. Para solventar este problema hemos realizado lo siguiente, crear un índice para esa columna y volver a ejecutar la consulta de antes. Creamos el índice:

```
CREATE INDEX iStock on product_color_size_stock(stock);
```

Ejecutamos la consulta anterior de nuevo, proporcionando el siguiente resultado:

```
cqlsh:wear_dealer> CREATE INDEX iStock on product_color_size_stock(stock);
cqlsh:wear_dealer> SELECT productcode, colorcode, sizecode, ean
... FROM product_color_size_stock
... WHERE stock = 0
... ;
```

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>
```

Ilustración 3 - Cassandra consulta y resultado consulta 3.

1.2.4. 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.

La consulta realizada es:

```
SELECT seasoncode, productcode, shortdescription
```

```
FROM seasons_product_color_size
```

```
GROUP BY seasoncode, productcode
```

```
;
```

El resultado que nos genera es el siguiente:

```
cqlsh:wear_dealer> SELECT seasoncode, productcode, shortdescription
... FROM seasons_product_color_size
... GROUP BY seasoncode, productcode
... ;
```

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

(4 rows)

Ilustración 4 - Cassandra consulta y resultado consulta 4.

1.2.5. 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'.

La consulta realizada para la tabla `product_color_size_stock` es:

```
SELECT productcode, colorcode, sizecode, shortdescription
```

```
FROM product_color_size_stock
```

```
WHERE productcode = 'RUNTS'
```

```
;
```

```
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 = 'M'  
  
;
```

```
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 = 'XL'  
  
;
```

La cual genera como resultado:

```

cqlsh:wear_dealer> SELECT productcode, colorcode, sizecode, shortdescription
... FROM product_color_size stock
... WHERE productcode = 'RUNTS'
... ;

productcode | colorcode | sizecode | shortdescription
-----+-----+-----+-----
RUNTS      | YE       | L       | Running T Shirt
RUNTS      | YE       | M       | Running T Shirt
RUNTS      | YE       | S       | Running T Shirt
RUNTS      | YE       | XL      | Running T Shirt

(4 rows)
cqlsh:wear_dealer> UPDATE product_color_size stock
... SET shortdescription = 'Running T-Shirt High Performance'
... WHERE productcode = 'RUNTS' and colorcode = 'YE' and sizecode = 'S'
... ;
cqlsh:wear_dealer> UPDATE product_color_size stock
... SET shortdescription = 'Running T-Shirt High Performance'
... WHERE productcode = 'RUNTS' and colorcode = 'YE' and sizecode = 'M'
... ;
cqlsh:wear_dealer> UPDATE product_color_size stock
... SET shortdescription = 'Running T-Shirt High Performance'
... WHERE productcode = 'RUNTS' and colorcode = 'YE' and sizecode = 'L'
... ;
cqlsh:wear_dealer> UPDATE product_color_size stock
... SET shortdescription = 'Running T-Shirt High Performance'
... WHERE productcode = 'RUNTS' and colorcode = 'YE' and sizecode = 'XL'
... ;
cqlsh:wear_dealer> SELECT productcode, colorcode, sizecode, shortdescription
... FROM product_color_size stock
... WHERE productcode = 'RUNTS'
... ;

productcode | colorcode | sizecode | shortdescription
-----+-----+-----+-----
RUNTS      | YE       | L       | Running T-Shirt High Performance
RUNTS      | YE       | M       | Running T-Shirt High Performance
RUNTS      | YE       | S       | Running T-Shirt High Performance
RUNTS      | YE       | XL      | Running T-Shirt High Performance

(4 rows)
cqlsh:wear_dealer> 

```

Ilustración 5 - Cassandra consulta y resultado consulta 5.

La consulta realizada para la tabla *seasons_product_color_size* es, cabe destacar que se ha tenido que crear un índice para la columna *productcode*, ya que de lo contrario no se podía hacer un filtro sobre el código del producto (saltaba un error), por lo que la consulta final quedaría:

```

SELECT seasoncode, productcode, colorcode, sizecode, shortdescription

FROM seasons_product_color_size

WHERE productcode = 'RUNTS'

;

```

```
CREATE INDEX iProductoCode on seasons_product_color_size(productcode);
```

```
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 = 'M'
```

```
;
```

```
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 = 'XL'
```

;

El resultado que nos generaría sería:

```
cqlsh:wear_dealer> SELECT seasoncode, productcode, colorcode, sizecode, shortdescription
... FROM seasons_product_color_size
... WHERE productcode = 'RUNTS'
... ;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it might involve data
mutability, use ALLOW FILTERING"
cqlsh:wear_dealer> CREATE INDEX iProductCode on seasons_product_color_size(productcode) ;
cqlsh:wear_dealer> SELECT seasoncode, productcode, colorcode, sizecode, shortdescription
... FROM seasons_product_color_size
... WHERE productcode = 'RUNTS'
... ;
```

seasoncode	productcode	colorcode	sizecode	shortdescription
SU	RUNTS	YE	L	Running T Shirt
SU	RUNTS	YE	M	Running T Shirt
SU	RUNTS	YE	S	Running T Shirt
SU	RUNTS	YE	XL	Running T Shirt

```
(4 rows)
cqlsh:wear_dealer> UPDATE seasons_product_color_size
... SET shortdescription = 'Running T-Shirt High Performance'
... WHERE seasoncode = 'SU' and productcode = 'RUNTS' and colorcode = 'YE' and sizecode = 'S'
... ;
cqlsh:wear_dealer> UPDATE seasons_product_color_size
... SET shortdescription = 'Running T-Shirt High Performance'
... WHERE seasoncode = 'SU' and productcode = 'RUNTS' and colorcode = 'YE' and sizecode = 'M'
... ;
cqlsh:wear_dealer> UPDATE seasons_product_color_size
... SET shortdescription = 'Running T-Shirt High Performance'
... WHERE seasoncode = 'SU' and productcode = 'RUNTS' and colorcode = 'YE' and sizecode = 'L'
... ;
cqlsh:wear_dealer> UPDATE seasons_product_color_size
... SET shortdescription = 'Running T-Shirt High Performance'
... WHERE seasoncode = 'SU' and productcode = 'RUNTS' and colorcode = 'YE' and sizecode = 'XL'
... ;
cqlsh:wear_dealer> SELECT seasoncode, productcode, colorcode, sizecode, shortdescription
... FROM seasons_product_color_size
... WHERE productcode = 'RUNTS'
... ;
```

seasoncode	productcode	colorcode	sizecode	shortdescription
SU	RUNTS	YE	L	Running T-Shirt High Performance
SU	RUNTS	YE	M	Running T-Shirt High Performance
SU	RUNTS	YE	S	Running T-Shirt High Performance
SU	RUNTS	YE	XL	Running T-Shirt High Performance

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

Ilustración 6 - Cassandra consulta y resultado consulta 5.

1.3. 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?

Lo primero de todo es tener claro para qué sirve la clave de partición, no es más que la forma en la que se van a distribuir los datos en los diferentes nodos de la red. Si la clave de partición fuera solo el código de temporada, podríamos tener solamente 4 particiones de nuestros datos, una partición por temporada, haciendo que la búsqueda de los productos dentro de cada temporada fuera muy ineficiente, en este caso porque no hay muchos productos pero en una compañía real esta tabla sería muy grande y en consecuencia sus particiones también.

Por lo que es mejor hacer la partición por la temporada y el producto en sí, de esta forma vamos a tener muchas más particiones, las cuales con menos datos, y su búsqueda de información será eficiente.

2. Ejercicio 2: MongoDB

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.

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:

2.1. 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.

La consulta es:

```
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

    }

)

```

Y el resultado de 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
...   }
... )
{ "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 Cebrian", "Contacto" : { "Nombre" : "Amelia", "Apellidos" : "Martín Bosques" } } }
{ "Identificador" : "Contacto-86", "Usuario" : { "Nombre" : "Elsa", "Apellidos" : "Serna Risco", "Contacto" : { "Nombre" : "Amelia", "Apellidos" : "Martín Bosques" } } }

```

Ilustración 7 - MongoDB consulta y resultado de la consulta 1.

2.2. Consulta 2 (20%)

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

La consulta es:

```

db.Desbloques.find(

    {

        "Usuario_desbloqueador.Email": "jsanzroble@hotmail.es"

    }
)

```

```

    },
    {
      _id: 0,
      Identificador: 1,
      "Usuario_desbloqueado.Nombre": 1,
      "Usuario_desbloqueado.Email": 1
    }
  )

```

El resultado de la consulta es:

```

> db.Desbloques.find(
...   {
...     "Usuario_desbloqueador.Email": "jsanzrobles@hotmail.es"
...   },
...   {
...     id: 0,
...     Identificador: 1,
...     "Usuario_desbloqueado.Nombre": 1,
...     "Usuario_desbloqueado.Email": 1
...   }
... )
{ "Usuario_desbloqueado" : { "Nombre" : "Mercedes", "Email" : "mreysordo@gmail.es" }, "Identificador" : "Desbloqueo-1" }
{ "Usuario_desbloqueado" : { "Nombre" : "Isabel", "Email" : "ilopezmena@gmail.es" }, "Identificador" : "Desbloqueo-2" }
>

```

Ilustración 8 - MongoDB consulta y resultado de la consulta 2.

2.3. 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.

La consulta es:

```

db.Usuarios_grupos.aggregate([
  {
    $project: {

```

```
    _id: 0,  
    Email: 1,  
    Count: {$size: "$Grupos_propietario"}  
  }  
},  
{  
  $sort: {"Count": -1}  
},  
{  
  $limit: 3  
}  
})
```

El resultado de la consulta es:

```
> db.Usuarios_grupos.aggregate([  
...   {  
...     $project: {  
...       _id: 0,  
...       Email: 1,  
...       Count: {$size: "$Grupos_propietario"}  
...     }  
...   },  
...   {  
...     $sort: {"Count": -1}  
...   },  
...   {  
...     $limit: 3  
...   }  
... ])  
{ "Email" : "mreysordo@gmail.es", "Count" : 5 }  
{ "Email" : "efrancolopez@gmail.es", "Count" : 4 }  
{ "Email" : "lsagradosanz@gmail.es", "Count" : 3 }  
> 
```

Ilustración 9 - MongoDB consulta y resultado de la consulta 3.

2.4. 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.

La consulta hecha es:

```
db.Contactos_usuarios.aggregate([  
  
  {  
  
    $match: {"Email": "mgarciasanz@gmail.es"}  
  
  },  
  
  {  
  
    $unwind: "$Contactos"  
  
  },  
  
  {  
  
    $match: {"Contactos.Usuario_contacto.Edad": {$gte: 36}}  
  
  },  
  
  {  
  
    $project: {  
  
      _id: 0,  
  
      Email: 1,  
  
      "Contactos.Usuario_contacto.Email": 1,  
  
      "Contactos.Usuario_contacto.Edad": 1  
  
    }  
  
  }  
])
```

))

El resultado que proporciona:

```
> db.Contactos_usuarios.aggregate([
...   {
...     $match: {"Email": "mgarciasanz@gmail.es"}
...   },
...   {
...     $unwind: "$Contactos"
...   },
...   {
...     $match: {"Contactos.Usuario_contacto.Edad": {$gte: 36}}
...   },
...   {
...     $project: {
...       id: 0,
...       Email: 1,
...       "Contactos.Usuario_contacto.Email": 1,
...       "Contactos.Usuario_contacto.Edad": 1
...     }
...   }
... ])
{ "Email" : "mgarciasanz@gmail.es", "Contactos" : { "Usuario_contacto" : { "Email" : "vtiernocrespo@hotmail.es", "Edad" : 41 } } }
{ "Email" : "mgarciasanz@gmail.es", "Contactos" : { "Usuario_contacto" : { "Email" : "adelgadosanchez@hotmail.es", "Edad" : 37 } } }
{ "Email" : "mgarciasanz@gmail.es", "Contactos" : { "Usuario_contacto" : { "Email" : "tjazmintablas@hotmail.es", "Edad" : 36 } } }
>
```

Ilustración 10 - MongoDB consulta y resultado de la consulta 4.

3. Ejercicio 3: Neo4j

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.

3.1. Ejercicio 3.1 (no puntúa)

Mirar el enunciado de la práctica (carga de los datos).

3.2. 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;
```

3. 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?

3.2.1. Comprobaciones de la carga de datos

Obtener una visualización del esquema

Para obtener una visualización del esquema de los nodos/relaciones que se han creado usamos “`call db.schema.visualization()`”.

```
$ call db.schema.visualization()
```

Ilustración 11 – función para obtener el esquema.



Ilustración 12 - Visualización del esquema.

Cuántos nodos tipo *Supplier* se han creado

Para obtener el número de nodos hay que ejecutar la siguiente consulta y obtendremos dicho resultado:

```
$ MATCH (n:Supplier) Return COUNT(n)
```

Ilustración 13 - Consulta para obtener el número de nodos de tipo *Supplier*.

\$ MATCH (n:Supplier) Return COUNT(n)	
Table	COUNT(n)
	29
Text	

Ilustración 14 - Resultado del número de nodos de tipo *Supplier*.

3.2.2. Explicar la funcionalidad implementada

```
ON CREATE SET      c.categoryName = row.CategoryName,  
                  c.description = row.Description;
```

Lo que se está haciendo es definir los valores de las dos propiedades que tiene cada nodo del tipo *Category*, es decir, por cada línea del fichero CSV leído, coge los valores de las columnas (*CategoryName* y *Description*) y los añade a las propiedades correspondientes.

3.2.3. ¿Por qué se recomienda crear un índice para cada tipo de nodo?

Básicamente se recomienda crear un índice para cada tipo de nodo para garantizar la búsqueda de nodos de forma óptima, dando así un valor único para cada nodo del mismo tipo.

3.3. 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.

Para resolver este ejercicio se puede hacer de dos formas, la primera es eliminando primero las relaciones y luego los nodos de tipo *Supplier* (2 consultas), y la segunda es eliminar directamente los nodos pero haciendo uso de la palabra reservada *DETACH* (elimina primero las relaciones y luego los nodos pero todo de una).

En este caso, como no se indica que no se pueda hacer uso de *DETACH*, se ha elegido esta opción ya que es una consulta menos que hay que hacer, dando lugar al siguiente resultado:

```
1 MATCH (n:Supplier)
2 DETACH DELETE n
```

Ilustración 15 - Borrado de los nodos tipo *Supplier*.



Ilustración 16 - Resultado del borrado de los nodos tipo *Supplier*.

3.4. Ejercicio 3.4

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

- ¿Cuántos *Supplier* provienen de Francia? (ayuda: buscar 'France' ya que la información es en inglés).

- ¿Cuáles 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 pide identificar cuántos *Supplier* hay por cada país con una URL absoluta.

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

3.4.1. Recreación de los nodos Supplier y las relaciones SUPPLIES

Lo primero de todo es crear los nodos de tipo *Supplier*, para ello hay que almacenar nuevos campos (*Country*, *HomePage*) para así satisfacer luego las consultas:

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

Ilustración 17 - Creación de los nodos de tipo Supplier.

Una vez creado los nodos, creamos las diferentes relaciones, solo se producen con producto:

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

Ilustración 18 - Creación de las relaciones entre Supplier y Product.

3.4.2. ¿Cuántos Supplier provienen de Francia?

La consulta que tenemos que ejecutar es:

```
$ MATCH(n:Supplier) WHERE n.country = "France" RETURN n.country, COUNT(n.country)
```

Ilustración 19 - Consulta cuántos Supplier provienen de Francia.

Proporciona el siguiente resultado:

\$ MATCH(n:Supplier) WHERE n.country = "France" RETURN n.country, COUNT(n.country)		
Table	n.country	COUNT(n.country)
	"France"	3

Ilustración 20 - Resultado cuántos Supplier provienen de Francia.

3.4.3. ¿Cuántos Supplier hay por cada país con una URL absoluta?

La consulta a ejecutar es:

```
$ MATCH(n:Supplier) WHERE n.homePage CONTAINS "http" RETURN n.country, COUNT(n)
```

Ilustración 21 - Consulta cuántos países con URL absoluta.

Esta consulta proporciona el siguiente resultado:

\$ MATCH(n:Supplier) WHERE n.homePage CONTAINS "http" RETURN n.country, COUNT(n)		
<div><div></div><div>Table</div></div>	n.country	COUNT(n)
	"Germany"	1
	"Australia"	1
	"Japan"	1

Ilustración 22 - Resultado cuántos países con URL absoluta.

4. Ejercicio 4: Neo4j

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.

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

4.1. 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.

La consulta es la siguiente:

```
1 MATCH (e:Employee)-[:SOLD]->(o:Order)
2 WHERE e.title = "Sales Representative" AND o.shipName CONTAINS "White"
3 RETURN e.firstName, e.lastName, COUNT(*) as NumPedidosVendidos
4 ORDER BY NumPedidosVendidos DESC LIMIT 1
```

Ilustración 23 – Neo4j consulta 1.

Resultado de la consulta:

\$ MATCH (e:Employee)-[:SOLD]->(o:Order) WHERE e.title = "Sales Representative" AND o.shipName CONTAINS "White" RETURN e.firstName, e.lastName, COUNT(*) as NumPedidosVendidos ORDER BY NumPedidosVendidos DESC LIMIT 1			
Table	e.firstName	e.lastName	NumPedidosVendidos
	"Margaret"	"Peacock"	4
Text			

Ilustración 24 - Neo4j resultado consulta 1.

4.2. Consulta 2 (20%)

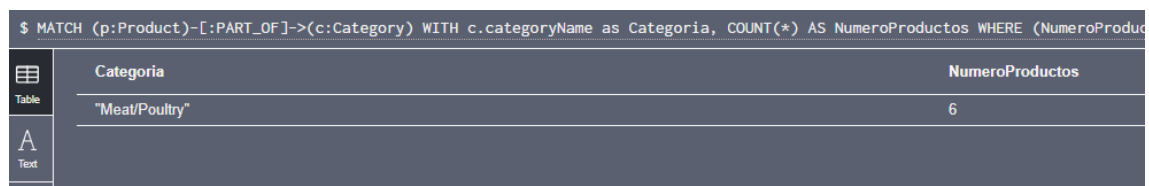
Listar el nombre de la categoría asociada (PART_OF) a 6 productos.

La consulta realizada es:

```
1 MATCH (p:Product)-[:PART_OF]->(c:Category)
2 WITH c.categoryName as Categoria, COUNT(*) AS NumeroProductos
3 WHERE (NumeroProductos = 6)
4 RETURN Categoria, NumeroProductos
```

Ilustración 25 - Neo4j consulta 2.

El resultado es:



\$ MATCH (p:Product)-[:PART_OF]->(c:Category) WITH c.categoryName as Categoria, COUNT(*) AS NumeroProductos WHERE (NumeroProductos = 6)	
Categoria	NumeroProductos
"Meat/Poultry"	6

Ilustración 26 - Neo4j resultado consulta 2.

4.3. 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.

La consulta es:

```
1 MATCH (e:Employee)-[:SOLD]->(o:Order)-[:CONTAINS]->(p:Product)-[:SUPPLIES]->(s:Supplier)
2 WHERE s.companyName = "Tokyo Traders"
3
4 WITH e.firstName as Nombre, e.lastName as Apellido, COUNT(*) as UnidadesVendidas
5 ORDER BY UnidadesVendidas DESC
6
7 WITH COLLECT([Nombre, Apellido, UnidadesVendidas]) as lstUnidadesVendidas
8 UNWIND lstUnidadesVendidas[1..4] as lst
9
10 RETURN lst[0] as Nombre, lst[1] as Apellido, lst[2] as UnidadesVendidas
```

Ilustración 27 - Neo4j consulta 3.

El resultado obtenido:

```
$ MATCH (e:Employee)-[:SOLD]->(o:Order)-[:CONTAINS]->(p:Product)<-[:SUPPLIES]-(s:Supplier) WHERE s.companyName = "Tokyo Traders"
```

	Nombre	Apellido	UnidadesVendidas
Table	"Nancy"	"Davolio"	11
Text	"Andrew"	"Fuller"	7
Code	"Margaret"	"Peacock"	5

Ilustración 28 - Neo4j resultado consulta 3.

4.4. 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.

La consulta es:

```
1 MATCH (s:Supplier)-[:SUPPLIES]->(p:Product)-[:PART_OF]->(c:Category {categoryName: "Seafood"})
2 WHERE (s.companyName = "Leka Trading" OR s.companyName = "Karkki Oy" OR s.companyName = "Pavlova, Ltd." OR
3 s.companyName = "Lyngbysild" OR s.companyName = "Ma Maison" OR s.companyName = "Tokyo Traders")
4 MATCH (o:Order)-[:CONTAINS]->(p)
5 WITH s.companyName AS Compania, COUNT(s.companyName) as UnidadesVendidas
6 RETURN Compania, UnidadesVendidas
```

Ilustración 29 - Neo4j consulta 4.

El resultado de la consulta es:

```
$ MATCH (s:Supplier)-[:SUPPLIES]->(p:Product)-[:PART_OF]->(c:Category {categoryName: "Seafood"}) WHERE (s.companyName = "Leka Trading" OR s.companyName = "Karkki Oy" OR s.companyName = "Pavlova, Ltd." OR s.companyName = "Lyngbysild" OR s.companyName = "Ma Maison" OR s.companyName = "Tokyo Traders")
```

	Compania	UnidadesVendidas
Table	"Lyngbysild"	41
Text	"Tokyo Traders"	33
Code	"Pavlova, Ltd."	27

Ilustración 30 - Neo4j resultado consulta 4.

5. Bibliografía

Apuntes proporcionados en la asignatura.