



**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 tablas .....	4
Índice de ilustraciones .....	5
1. Ejercicio 1: Cassandra .....	6
2. Ejercicio 2: MongoDB.....	7
3. Ejercicio 3: Neo4j .....	13
4. Ejercicio 4: Neo4j .....	19
5. Bibliografía .....	22

---

## Índice de tablas

---

No se encuentran elementos de tabla de ilustraciones.

---

## Índice de ilustraciones

---

No se encuentran elementos de tabla de ilustraciones.

---

## 1.Ejercicio 1: Cassandra

---

A

---

## 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 1 - MongoDB consulta y resultado de la consulta 1.*

## 2.2. 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").*

La consulta es:

```

db.Desbloquesos.find(

  {

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

  }
)

```

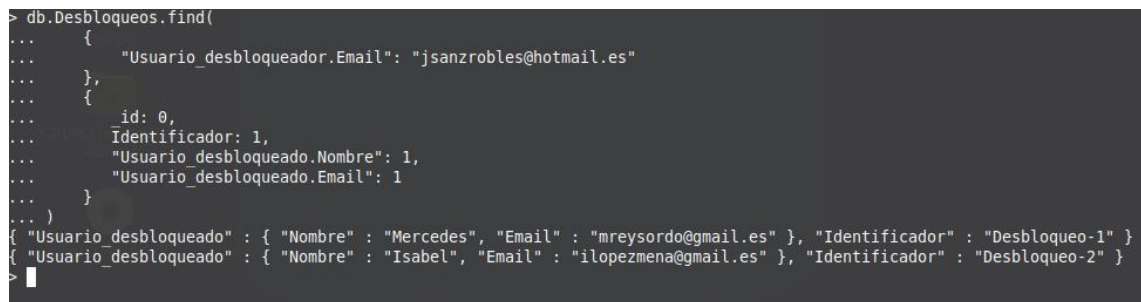


```

    },
    {
      _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 2 - 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 3 - 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 4 - 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 5 – función para obtener el esquema.



Ilustración 6 - 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 7 - Consulta para obtener el número de nodos de tipo *Supplier*.



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

Ilustración 8 - 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%)

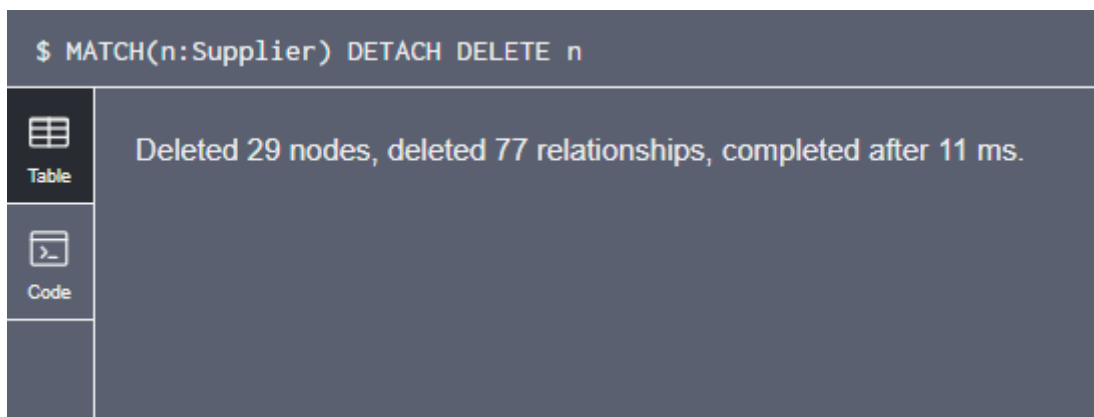
*Se pide borrar todos los nodos de tipo Supplier: Adjuntar una captura de pantalla con las queries utilizadas para eliminar todos estos nodos. Ayuda: no es posible borrar nodos que tengan relaciones con otros.*

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 9 - Borrado de los nodos tipo Supplier.*



*Ilustración 10 - 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 11 - 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 12 - 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 13 - 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 14 - 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 15 - 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 16 - 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 17 – 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,			
Table	e.firstName	e.lastName	NumPedidosVendidos
	"Margaret"	"Peacock"	4
Text			

*Ilustración 18 - 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 19 - 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 20 - 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 21 - 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
"Nancy"	"Davolio"	11
"Andrew"	"Fuller"	7
"Margaret"	"Peacock"	5

Ilustración 22 - 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:

---

## 5. Bibliografía

---

A