Bases de Datos No SQL

4. Bases de Datos orientadas a Grafos. **Neo4J.**

Universidad
Rey Juan Carlos

Índice

- *Introducción a las Bases de Datos NoSQL*
  - *1.1 Limitaciones de las Bases de Datos Relacionales*
  - *1.2 Clasificación de las Bases de Datos NoSQL*
  - *1.3 Elección de una Base de Datos NoSQL*
- *BD Clave-Valor*
- *BD orientadas a Documentos*
- **BD orientadas a Grafos**
- BD Familia de Columnas

Bibliografía

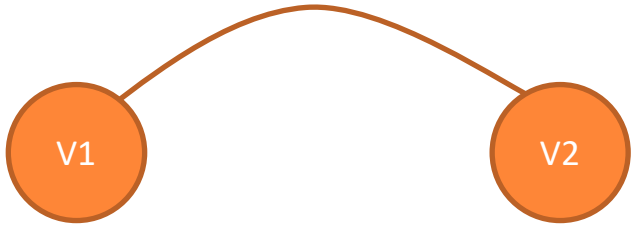
- https://neo4j.com/docs/
- https://neo4j.com/developer/cypher/
- Ian Robinson, Jim Webber y Emil Eifrem (2015) **Graph Databases**, 2nd Edition. O'Reilly Media, Inc.
- Thomas Frisendal (2016). **Graph Data Modeling for NoSQL and SQL.** Visualize Structure and Meaning. Technics Publications
- Dan Sullivan (2015). **NoSQL for Mere Mortals.** Addison-Wesley Professional
- Guy Harrison (2016) **Next Generation Databases: NoSQL, NewSQL, and Big Data.** Apress

Índice

- **Introducción a Neo4J**
- Instalación y puesta en marcha
- Creación de una BD orientada a grafos
- Lenguaje Cypher para crear nodos y relaciones
- Consultas en Cypher
- Actualizaciones en Cypher
- Importar y Exportar Nodos

Introducción a Neo4J

- Las BD orientadas a grafos modelan los datos utilizando nodos y relaciones entre ellos (vértices y aristas).



Introducción a Neo4J

- Neo4J es una **base de datos orientada a grafos**.
- Neo4J es una de las bases de datos orientadas a grafos **más usadas**.
- Se trata de un **proyecto open source** (bajo licencia GNU public license GPL v3.0).
- **Versión gratuita**: **Neo4j Community Server**.
- También tiene una **versión comercial** distribuida por Neo Technology (GNU AGPL v3.0 and commercial licensing): **Neo4j Enterprise Server**
- https://neo4j.com/licensing/

Introducción a Neo4J

- Las **bases de datos** orientadas a **grafos,** como Neo4j, tienen **mejor rendimiento** que las relacionales (SQL) y las no relacionales (NoSQL) cuando los **datos están muy relacionados/conectados**.
- El **rendimiento de Neo4j no desciende** con el número de relaciones de las consultas (*joins*); esto sí sucede con las BD relacionales.

Introducción a Neo4J.

Características

- **Neo4j** puede almacenar **miles de millones de nodos** (con la versión 3.0 deja de existir el límite de 34.000 millones de nodos).
- **Neo4j** soporta el **desarrollo rápido** de sistemas basados en grafos, aprovechando que los **datos están altamente conectados**.
- **Neo4J** es una base de datos orientada a grafos **nativa:** Se creó desde el inicio pensada como una base de datos orientada a grafos. Su arquitectura está diseñada para **optimizar** la gestión, almacenamiento y recorrido de **nodos y relaciones**.

Introducción a Neo4J.

Características

- En Neo4j las **relaciones son ciudadanos de primer orden** que representan las conexiones pre-materializadas entre entidades.
  - Una operación conocida en la terminología de las bases de datos relacionales como un ***join***, cuyo rendimiento se *reduce exponencialmente* con el número de relaciones, en Neo4j se lleva a cabo como una **navegación de un nodo a otro,** con un **rendimiento lineal**.
- Esta aproximación para el almacenamiento y la consulta de las conexiones entre entidades proporciona un rendimiento transversal de hasta **4 millones de saltos por segundo** (sin influir el tamaño de la BD).

Introducción a Neo4J.

Características

- **Schema Optional: no** es obligatorio tener un **esquema** de la base de datos **a priori**. De esta forma, el modelo de dominio puede **evolucionar continuamente** si hay un cambio en los requisitos, sin ningún coste debido a un cambio de esquema o una migración de esquema.
- **Sin tipado de datos**, por lo que es altamente flexible.

Introducción a Neo4J.

Características

- **Seguridad de los datos por medio de transacciones ACID:** Neo4j usa transacciones para garantizar la persistencia de los datos en caso de un fallo de sistema.
- **Neo4j** está diseñado para soportar **operaciones de negocio críticas y de alto rendimiento.**
- **Neo4J** ofrece **alta disponibilidad** mediante clustering.

Introducción a Neo4J.

Aplicaciones

Aplicaciones:

- Redes sociales
- Clasificación en dominios médicos o biológicos
- Sistemas de recomendación
- Detección de patrones

Índice

- *Introducción a Neo4J*
- **Instalación y puesta en marcha**
- Creación de una BD orientada a grafos
- Lenguaje Cypher para crear nodos y relaciones
- Consultas en Cypher
- Actualizaciones en Cypher
- Importar y Exportar Nodos

Instalación y puesta en marcha

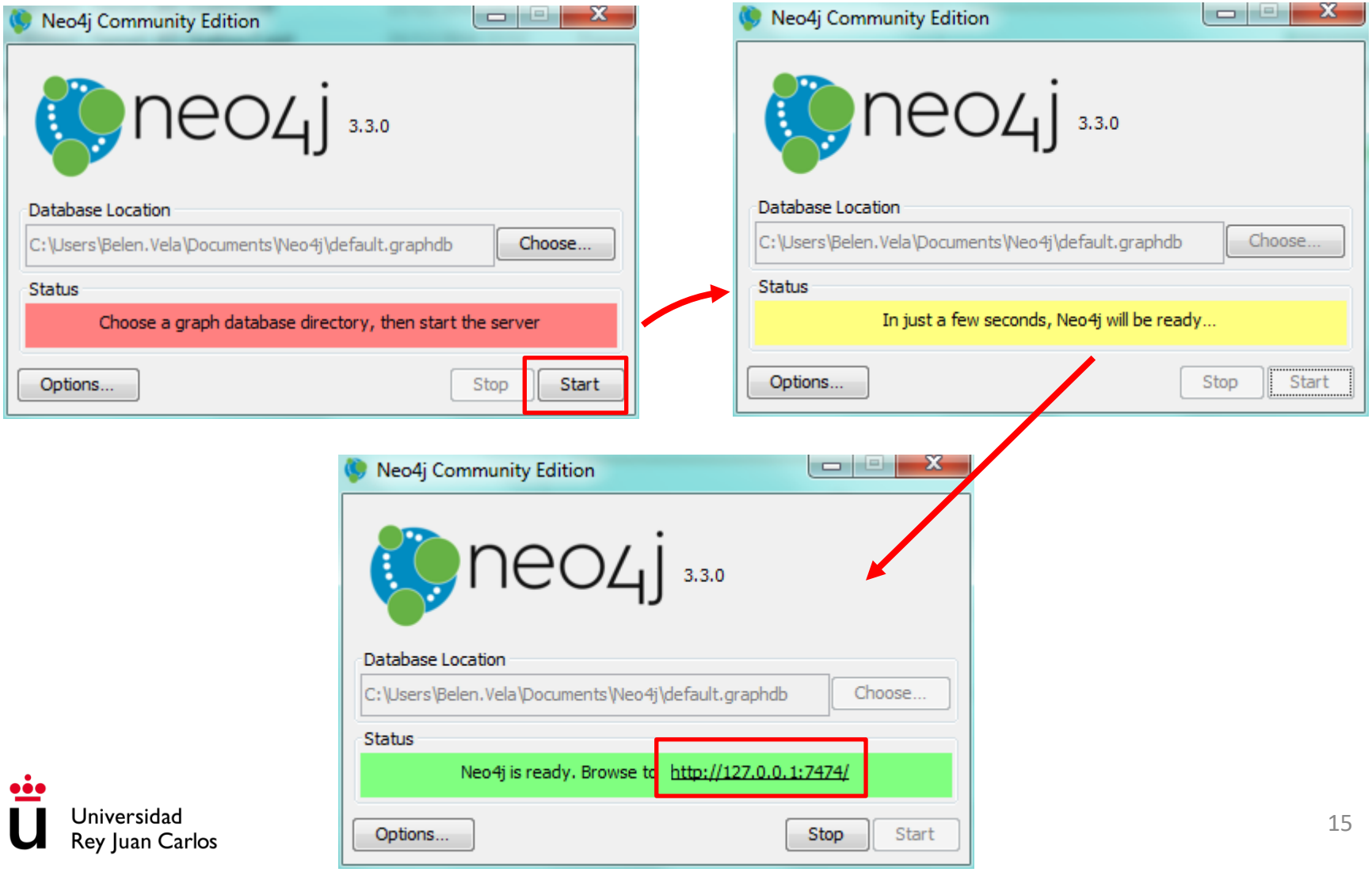
Neo4j Server 3.X (Neo4J Community Edition)
https://neo4j.com/download/other-releases/#releases
El instalador incluye la versión de Java necesaria para ejecución de Neo4j.

Paso 1: Descargar del instalador.
Paso 2: Lanzar la instalación y dar permisos para realizar cambios en el Sistema.

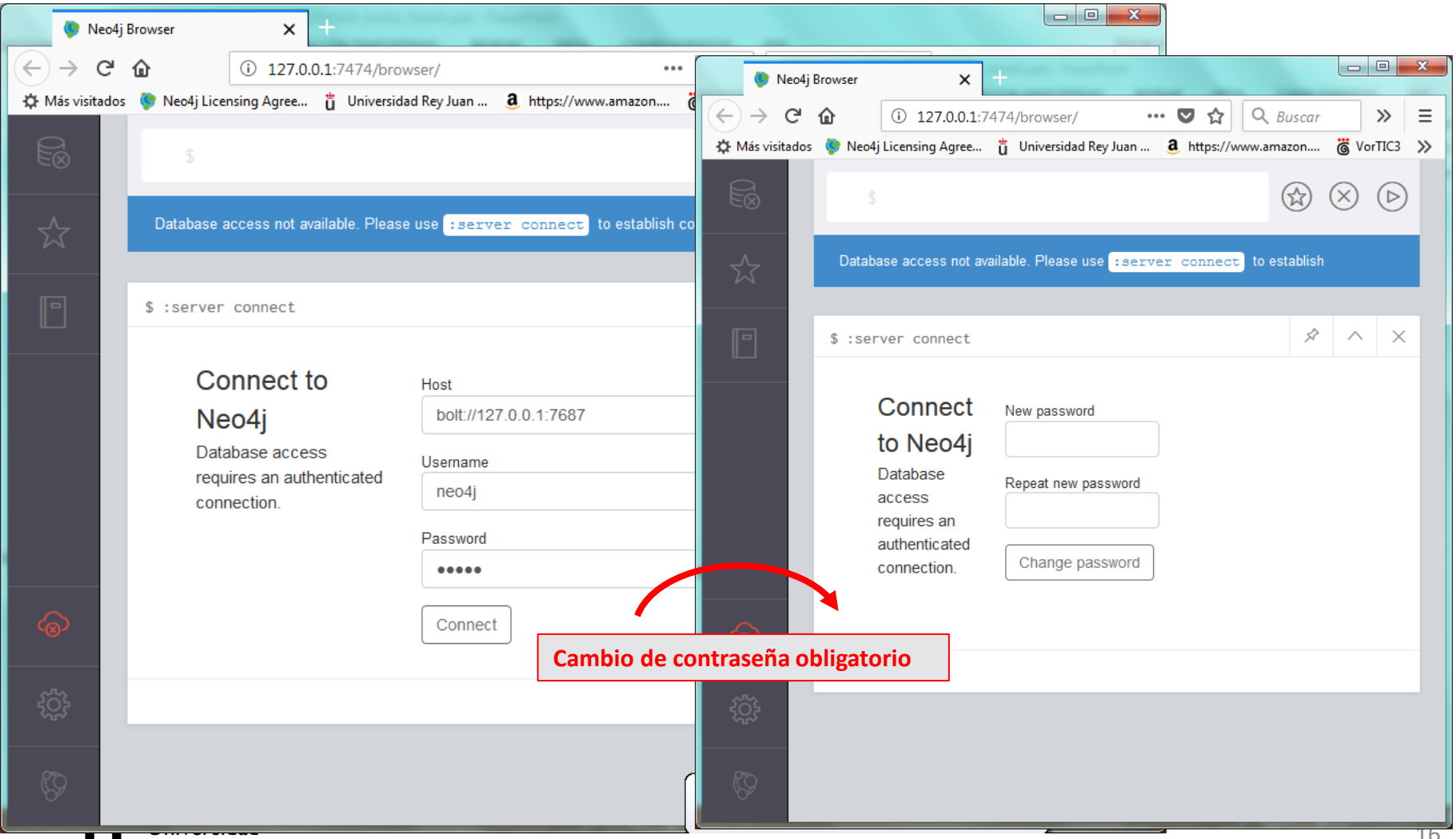
Una vez instalado:

1) Elegir la opción **Run Neo4j**.
2) Hacer click en el botón **'Start'** para arrancar el servidor de Neo4j.
3) Abrir en el navegador web la URL **http://localhost:7474/**
4) Cambiar la clave de la cuenta (login: neo4j password: neo4j)

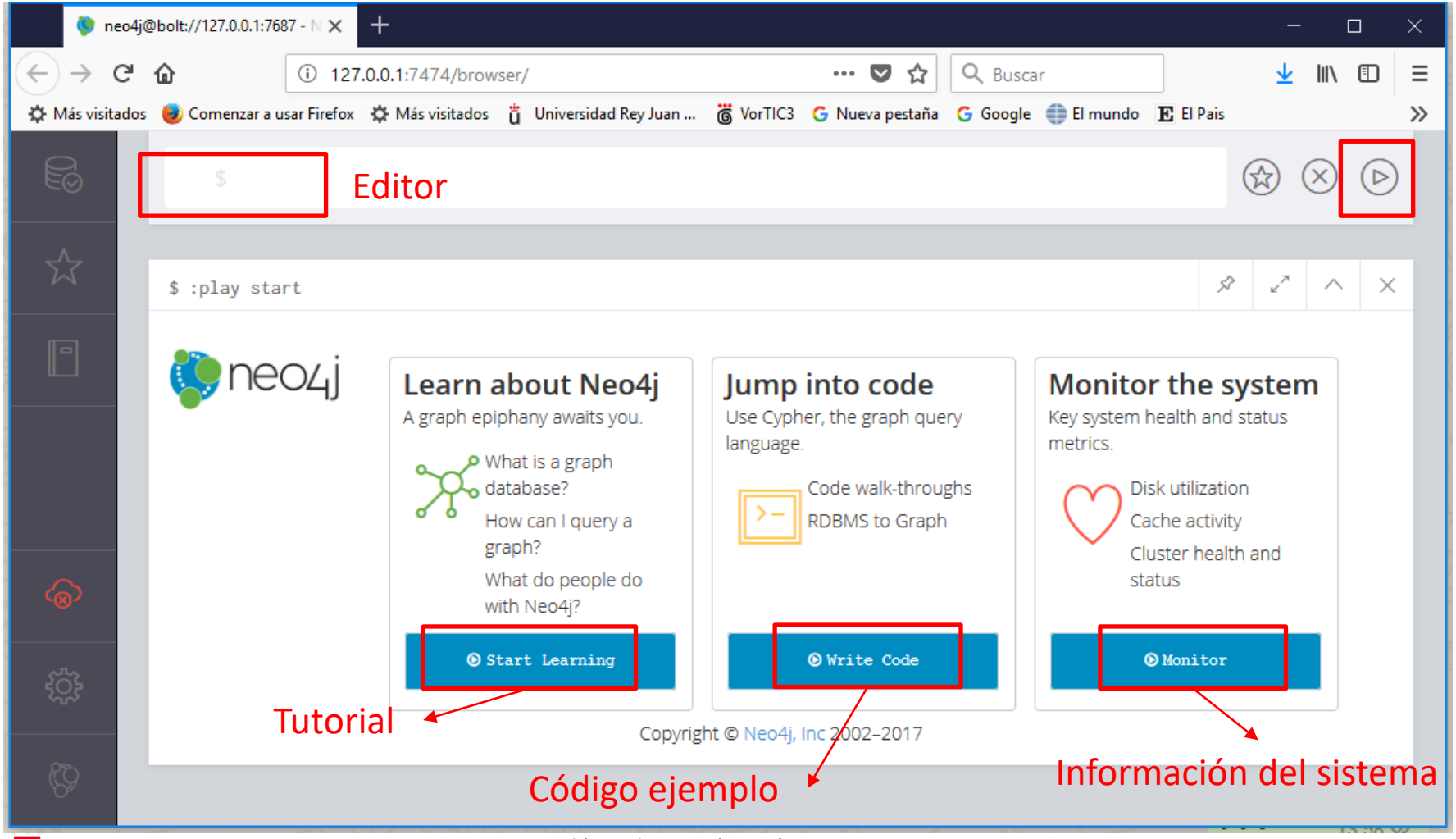
Instalación y puesta en marcha



Instalación y puesta en marcha



Instalación y puesta en marcha



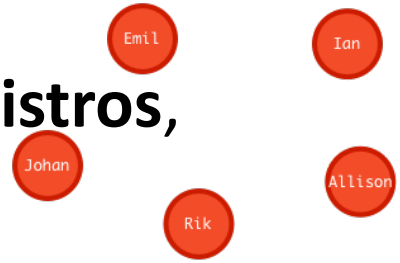
Índice

- *Introducción a Neo4J*
- *Instalación y puesta en marcha*
- **Creación de una BD orientada a grafos**
- Lenguaje Cypher para crear nodos y relaciones
- Consultas en Cypher
- Actualizaciones en Cypher
- Importar y Exportar Nodos

Creación de una BD orientada a grafos

- *Neo4J es **schema-free**:* No es necesario tener un esquema predefinido.

- Neo4j almacena los datos en un **grafo con registros**, denominados **nodos**.



- Un nodo puede tener un conjunto de **propiedades**.
- Nodos similares pueden tener propiedades diferentes.
- Propiedades pueden ser cadenas de caracteres (entre comillas simples o dobles), números o booleanos (true, false, TRUE, FALSE).

Creación de una BD orientada a grafos

- El grafo más sencillo únicamente tiene **un nodo con propiedades con nombre** (pares clave:valor).
- A cada nodo, el sistema le asigna un **identificador interno único** (id).

Ejemplo:
Un grafo social de amigos:

- Nodo de un amigo con nombre (name) “Emil” de (from) “Sweden”

Creación de una BD orientada a grafos

Labels (o etiquetas)

- Asocia un **conjunto de nodos** del mismo tipo.
- Aplicando un *label* a un conjunto de nodos estos se pueden agrupar.
- Un nodo puede tener una o varias etiquetas (*labels*).
- Una etiqueta no puede tener propiedades.
- Ejemplo: Grafo social de amigos
  - Aplicamos el label “Amigo" al nodo que creamos para “Emil”
- Se puede modificar la apariencia de un tipo de *label de nodo.* Para ello hay que seleccionar previamente los nodos del tipo de *label correspondiente.*

Creación de una BD orientada a grafos

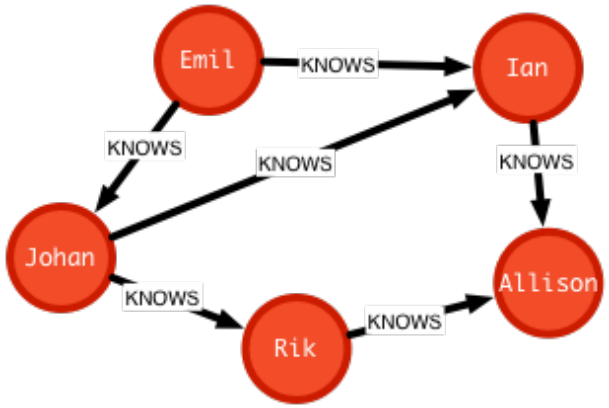
La potencia real de Neo4J es posibilidad de **conectar datos**.

Para conectar los nodos en un grafo hay que usar **relaciones** que describan qué dos registros (nodos) están relacionados.

- Las relaciones pueden tener **una dirección** (*se pueden recorrer en los dos sentidos*).
- Los nodos pueden estar relacionados consigo mismos.
- Las relaciones pueden tener un tipo (*label*).
- A cada relación el sistema le asigna un **identificador interno único** (id).

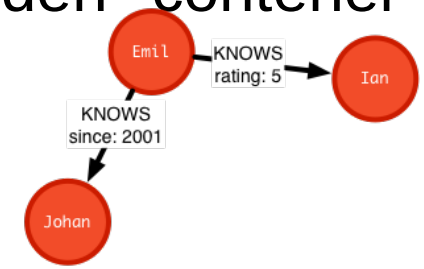
Ejemplo:

Permite representar quién conoce a quien (KNOWS):



Creación de una BD orientada a grafos

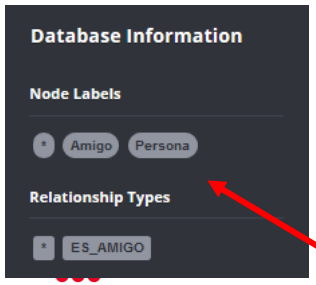
Las relaciones en un grafo también pueden contener **propiedades**.



Ejemplo:

- Fecha desde en la que conoce un amigo a otro (*since*)
- Valoración que le da un amigo a otro (*rating*)

Se puede cambiar la apariencia de un tipo de relación. Para ello es necesario seleccionar previamente el tipo de relación a cambiar.



Índice

- *Introducción a Neo4J*
- *Instalación y puesta en marcha*
- *Creación de una BD orientada a grafos*
- **Lenguaje Cypher para crear nodos y relaciones**
- Consultas en Cypher
- Actualizaciones en Cypher
- Importar y Exportar Nodos

Lenguaje Cypher para crear nodos y relaciones

Lenguaje Cypher

- Cypher es un **lenguaje declarativo sencillo** para trabajar con grafos.
- Usa sentencias similares a las del SQL.
- Permite crear la estructura de un grafo (nodos y relaciones), así como realizar consultas filtrando con argumentos, ordenar, etc.

Lenguaje Cypher para crear nodos y relaciones

- Declarativo: **Qué** encontrar, pero **no cómo**
- Usa patrones para describir los datos del grafo

- Comentarios: //
- CASE - Sensitive

Lenguaje Cypher para crear nodos y relaciones

Creación de un NODO de una BD orientada a grafos en Cypher:

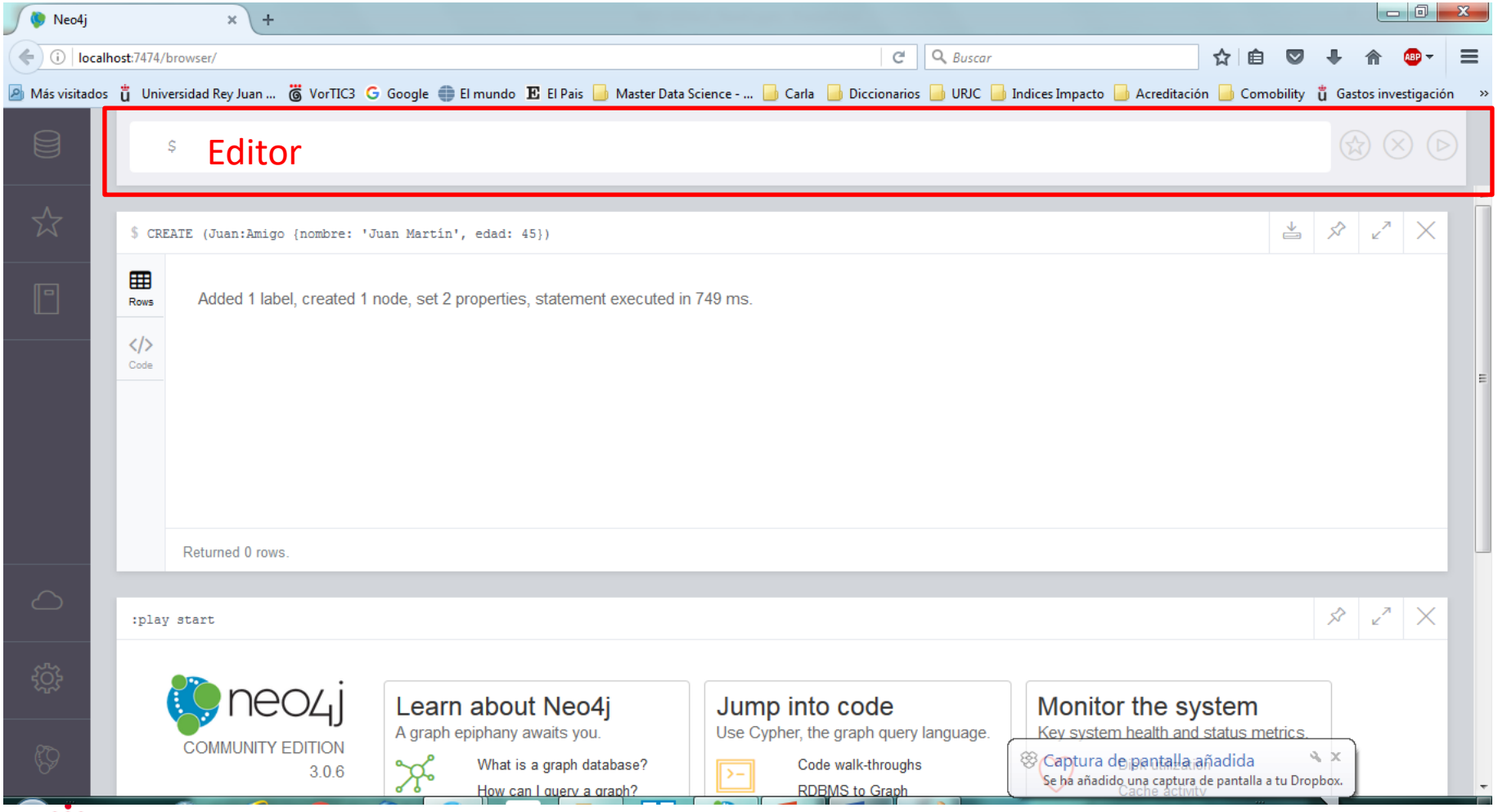
- Cypher usa paréntesis para representar un nodo (que suele contener una cadena de caracteres): (), (Juan).

- CREATE () -> Nodo anónimo, sin características (únicamente id)
- CREATE (Juan) -> Nodo denominado “Juan”

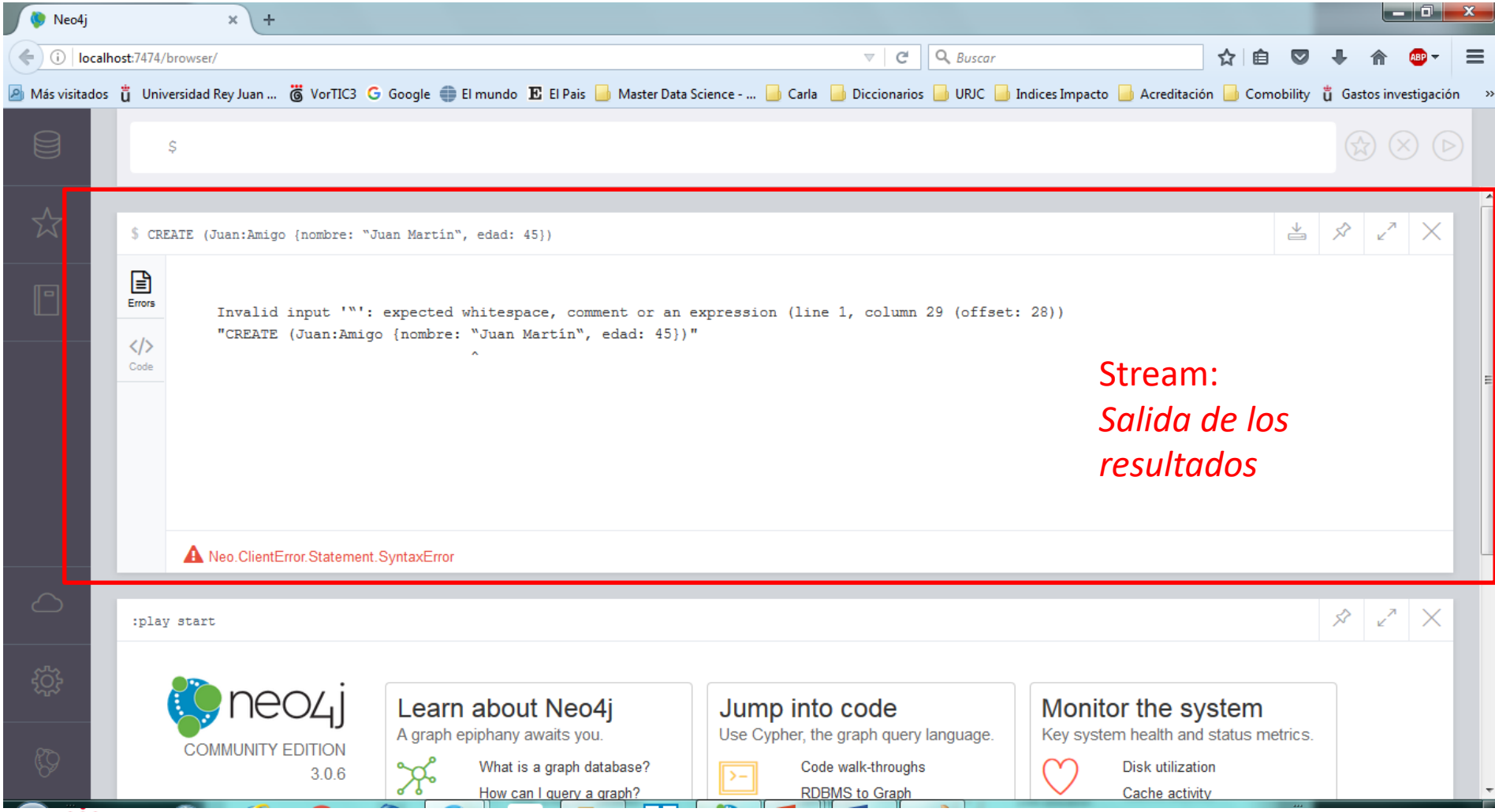
- CREATE (:Amigo) -> Nodo de tipo Amigo (Label)
- CREATE (Juan:Amigo) -> Nodo Juan de tipo Amigo
- CREATE (Juan:Amigo {nombre: ‘Juan Martín’}) -> Nodo Juan de tipo Amigo con una propiedad (format JSON)
- CREATE (Juan:Amigo {nombre: ‘Juan Martín’, edad: 45}) -> Nodo Juan de tipo Amigo con dos propiedades (format JSON)
- **CREATE** (Maria:Amigo {nombre: ‘María Alonso’, edad: 27, ciudad:’Madrid’})

  return Maria -> Devuelve el nodo María que acabamos de crear

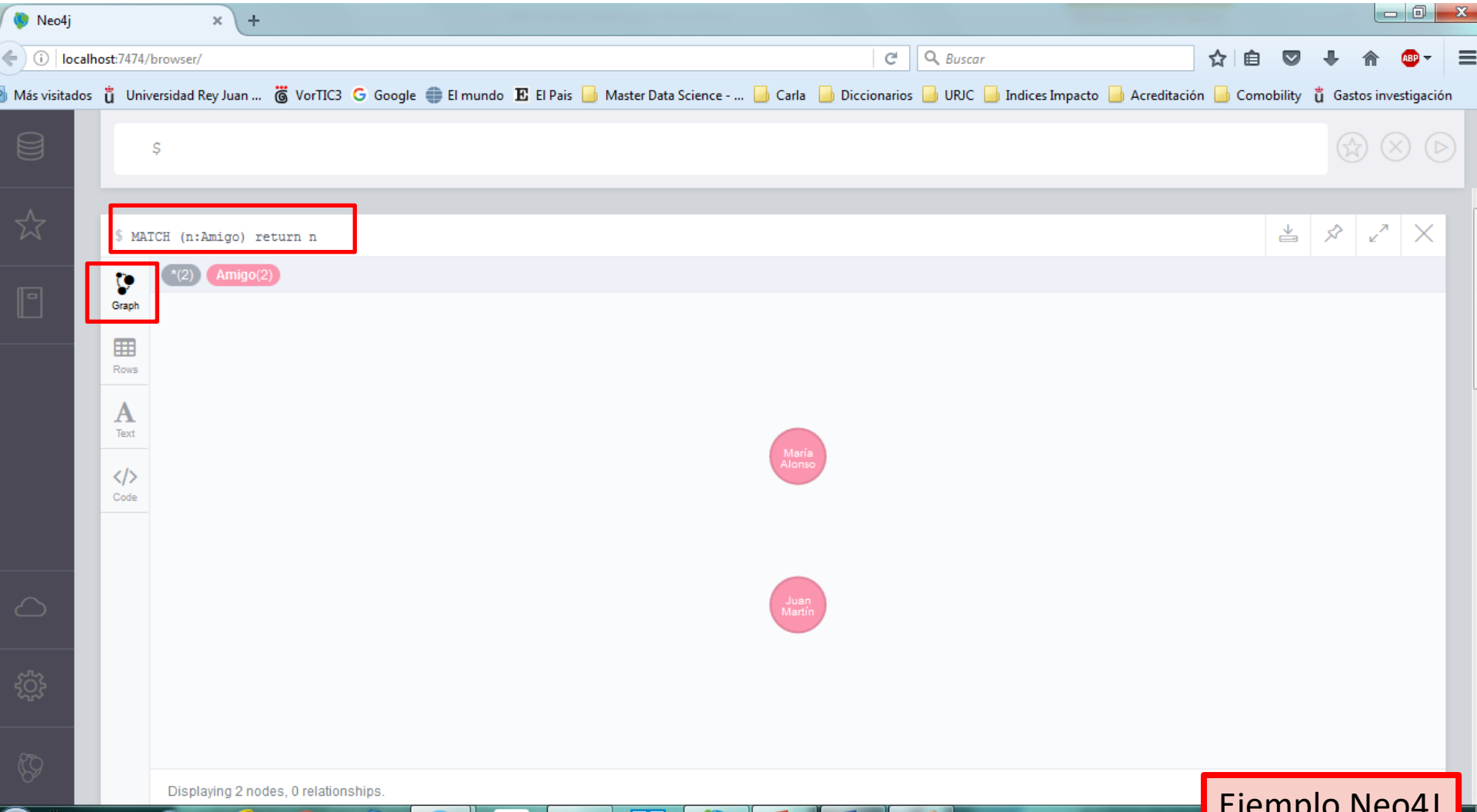
Lenguaje Cypher para crear nodos y relaciones



Lenguaje Cypher para crear nodos y relaciones



Lenguaje Cypher para crear nodos y relaciones



Ejemplo Neo4J

Lenguaje Cypher para crear nodos y relaciones

- Estos nodos no forman un grafo, para que lo formen es necesario crear las **relaciones**.
- Las relaciones se crean de forma muy similar a los nodos:
  - Mediante **CREATE** indicamos a través de las variables que identifican los nodos las relaciones con otros nodos.
  - La relación puede tener nombre y propiedades.

Lenguaje Cypher para crear nodos y relaciones

Sintaxis de las relaciones

- Cypher usa dos guiones (--) para representar una relación
- Relaciones direccionales se representan de la siguiente forma : <--, -->.
  - Es **obligatorio especificar su dirección** en la creación. **-->** : relación direccional

- Para añadir detalles se pueden usar expresiones entre corchetes -[…]->. Pueden ser variables, propiedades y/o información de tipo:
  - -[rol]-> : relación direccional con un nombre del "rol"
  - -[:ES_AMIGO]-> : relación direccional con una etiqueta (label) para el tipo de relación "ES_AMIGO"
  - -[rol:ES_AMIGO]-> : relación direccional con nombre "rol" de tipo "ES_AMIGO"
  - -[rol:ES_AMIGO {fecha:'2/07/2007'}]-> : relación direccional incluyendo un atributo "fecha"

Lenguaje Cypher para crear nodos y relaciones

```
//Nodo Juan de tipo Amigo con dos propiedades (formato JSON)
CREATE (Juan:Amigo {nombre: 'Juan Martín', edad: 45})

//Nodo María de tipo Amigo con tres propiedades (formato JSON)
CREATE (Maria:Amigo {nombre: 'María Alonso', edad: 27, ciudad: 'Madrid'})

// Relación entre Juan y María (variables, no nodos)
CREATE (Juan)-[:ES_AMIGO]->(Maria)
```

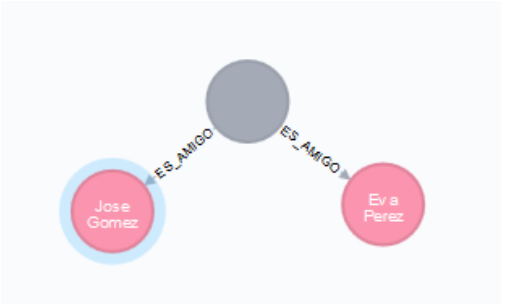
¿Cuántos nodos se crean?

Lenguaje Cypher para crear nodos y relaciones

- Con una **única sentencia** se pueden crear varios nodos y relaciones.
- Ejemplo:

// Creación de dos nodos y una relación (conectados a través de nodo intermedio)

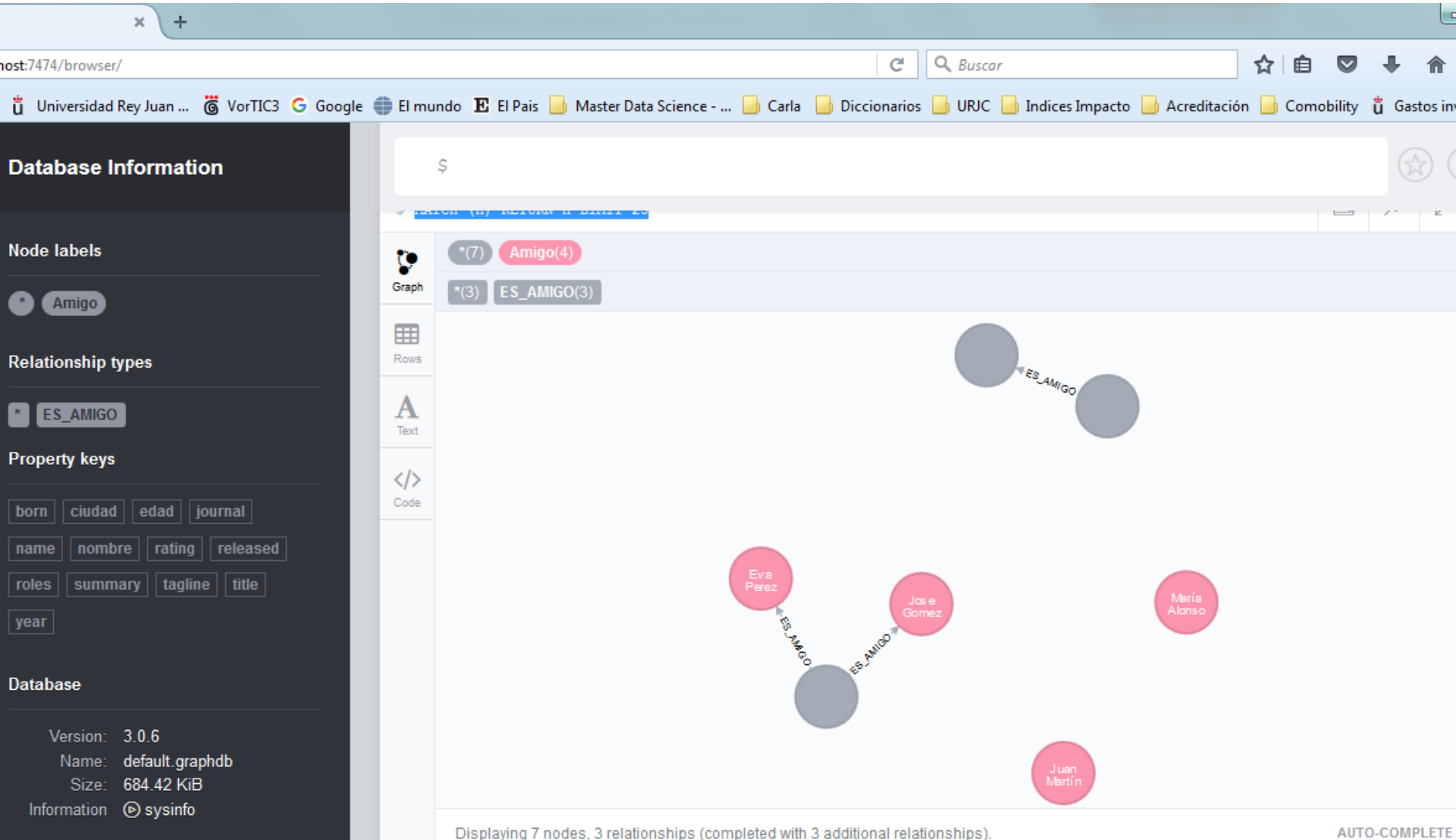
```
CREATE (Jose:Amigo {nombre: 'Jose Gomez', edad: 25}),
(Eva:Amigo {nombre: 'Eva Perez', edad: 33, ciudad:'Avila'}),
(Juan)-[:ES_AMIGO]->(Jose),
(Juan)-[:ES_AMIGO {desde:'31/12/2015'}]->(Eva)
return Jose, Eva, Juan
```



- Para ver todos los nodos creados en la Base de Datos:

```
MATCH (n)
RETURN n
```

Lenguaje Cypher para crear nodos y relaciones



Lenguaje Cypher para crear nodos y relaciones

También se pueden crear las relaciones ***una vez creados los nodos*** (sentencia MATCH):

```
//Nodo de tipo Amigo con dos propiedades (format JSON)
CREATE (:Amigo {nombre: 'Pepe', edad: 70})

//Nodo de tipo Amigo con tres propiedades (format JSON)
CREATE (:Amigo {nombre: 'Carmen', edad: 65})
```

```
MATCH (a:Amigo),(b:Amigo)
WHERE a.nombre= 'Pepe' AND b.nombre = 'Carmen'
    CREATE (a)-[r:ES_AMIGO]->(b)
RETURN r
```

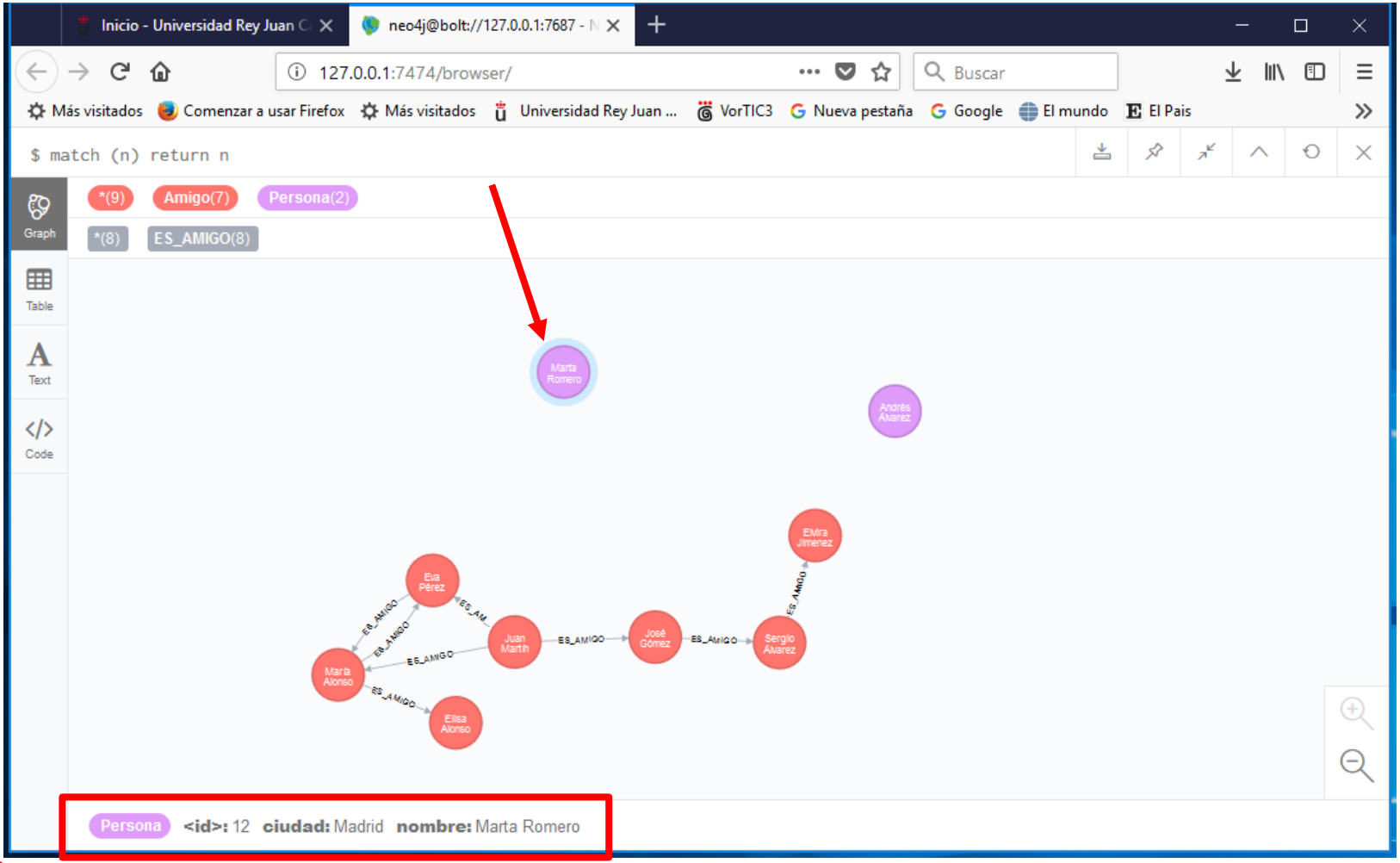
Lenguaje Cypher para crear nodos y relaciones

```
CREATE (Juan:Amigo {nombre: 'Juan Martín', edad: 45}),
(Maria:Amigo {nombre: 'María Alonso', edad: 27, ciudad: 'Madrid'}),
(Jose:Amigo {nombre: 'Jose Gomez', edad: 25}),
(Eva:Amigo {nombre: 'Eva Perez', edad: 33, ciudad:'Barcelona'}),
(Elisa:Amigo {nombre: 'Elisa Alonso', edad: 34, ciudad: 'Barcelona'}),
(Sergio:Amigo {nombre: 'Sergio Alvarez', edad: 26}),
(Juan)-[:ES_AMIGO {desde:'31/12/2015'}]->(Maria),
(Juan)-[:ES_AMIGO]->(Eva),
(Juan)-[:ES_AMIGO]->(Jose),
(Jose)-[:ES_AMIGO]->(Sergio),
(Eva)-[:ES_AMIGO]->(Maria),
(Maria)-[:ES_AMIGO]->(Eva),
(Maria)-[:ES_AMIGO]->(Elisa),
(Sergio)-[:ES_AMIGO]->(Elvira)

//Nodos de tipo Persona con propiedades (format JSON)
CREATE (:Persona {nombre: 'Andrés Álvarez', edad:21, ciudad:‘Avila'})
CREATE (:Persona {nombre: 'Marta Romero', ciudad:'Madrid'})
```

Lenguaje Cypher para crear nodos y relaciones

Lenguaje Cypher para crear nodos y relaciones



Lenguaje Cypher para crear nodos y relaciones

- Para **borrar un nodo**, se puede usar la sentencia DELETE:

  MATCH (n:Etiqueta) DELETE n

  o

  MATCH (n) DELETE n

- ¿Qué ocurre si el nodo a borrar tiene relaciones?

Lenguaje Cypher para crear nodos y relaciones

- Para borrar todos los nodos y sus relaciones, se puede usar la sentencia **DETACH DELETE** (esto no es apropiado para grandes volúmenes de datos):

```
MATCH (n)
DETACH DELETE n

MATCH (n { nombre:'Sergio' })
DETACH DELETE n
```

- Para **borrar grandes volúmenes de datos** de la BD es recomendable realizarlo **por lotes**:

```
MATCH (n:Amigo) where n.edad > 18
// Elimina los 10000 primeros nodos de tipo Amigo (mayores de 18 años) y sus relaciones.
//Si existen más de 100 relaciones por nodo mejor disminuir el tamaño de los nodos.
WITH n LIMIT 10000
DETACH DELETE n
RETURN count(*);
```

Esta operación se repite hasta que el número de nodos devueltos sea 0.

Restricciones en Cypher. UNIQUE

- **Restricción de Unicidad**

Para asegurar que **no** van a existir **dos nodos** de un determinado tipo que tengan el mismo valor para una propiedad. Para crearla:

CREATE CONSTRAINT ON (a:Persona) **ASSERT** a.DNI **IS UNIQUE**

Para borrar una restricción:

DROP CONSTRAINT ON (a:Persona) **ASSERT** a.DNI IS UNIQUE

Restricciones en Cypher. EXISTS

- **Restricción de Existencia**

Para asegurar que **todos los nodos** de un determinado tipo tienen una propiedad. Para crearla:

CREATE CONSTRAINT ON (a:Persona) **ASSERT exists** (a.**DNI**)

⇨ No se creará si existe algún nodo que lo incumple

⇨ Property existence constraint requires Neo4j Enterprise Edition

Para borrar una restricción:

DROP CONSTRAINT ON (a:Persona) **ASSERT exists** (a.DNI)

- **Para ver las restricciones existentes en la Base de Datos:**

CALL db.constraints

Índice

- *Introducción a Neo4J*
- *Instalación y puesta en marcha*
- *Creación de una BD orientada a grafos*
- *Lenguaje Cypher para crear nodos y relaciones*
- **Consultas en Cypher**
- Actualizaciones en Cypher
- Importar y Exportar Nodos

Consultas en Cypher

- Estructura similar al **SQL.**
- Cláusulas se van encadenando y el resultado sirve de entrada a la siguiente parte.
- Las variables del MATCH sirven de entrada a la siguiente parte de la sentencia.
- Para consultar un grafo tenemos las siguientes cláusulas:
  - **MATCH:** Patrón que debe cumplir el grafo
  - **WHERE:** Añade restricciones/condiciones a un patrón
  - **RETURN:** Resultado de la consulta

Consultas en Cypher.
Nodos

Recupera todos los nodos de la BD:

```
MATCH (n)
RETURN n
// 8 nodos (7 tipo Amigo y 2 tipo Persona)
```

Recupera solo las propiedades de todos los nodos de la BD (formato json):

```
MATCH (n)
RETURN properties(n)
```

Recupera los nodos de la BD que tengan en la propiedad edad un valor comprendido entre 21 y 30:

```
MATCH (n)
WHERE 20 < n.edad <= 30
RETURN n
```

Recupera los nodos de **tipo Amigo** de la BD que tengan en la propiedad edad un valor comprendido entre 21 y 30:

```
MATCH (n:Amigo)
WHERE 20 < n.edad <= 30
RETURN n
```

Recupera los id de los nodos de la BD:

```
MATCH (n)
RETURN id(n)
```

Recupera la propiedad nombre de los nodos de la BD en los que su id sea inferior a 10:

```
MATCH (n)
WHERE id(n)<10
RETURN n.nombre
//para los nodos que no tengan nombre saldrá un valor nulo
```

Consultas en Cypher.
Order By

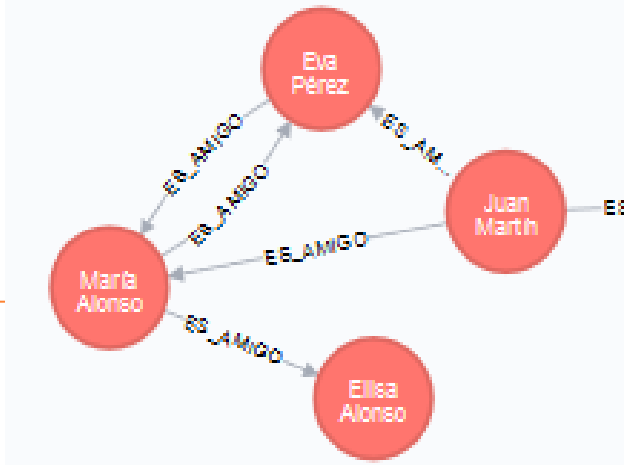
- Se puede ordenar el resultado de una consulta por una o varias propiedades usando ORDER BY.
- Por defecto, ASC (nulos al final).

Ejemplo:

Recupera los nodos de tipo Amigo de la BD que sean de Madrid y que tengan una edad comprendida entre 21 y 30 ordenados por nombre:

```
MATCH (n:Amigo {ciudad:'Madrid'})
WHERE 20 < n.edad <= 30
RETURN n
ORDER BY n.nombre DESC //nulos al principio
```

Consultas en Cypher. **Relaciones**



Ejemplo:

Recuperar el nombre y la edad de los amigos de María Alonso

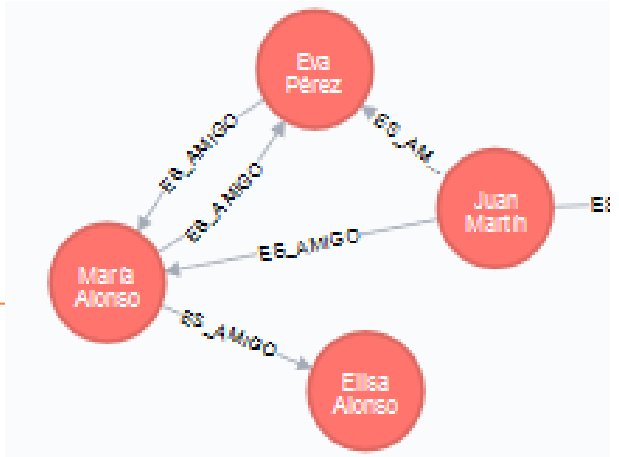
```
MATCH (Maria{nombre:'María Alonso'})-[:ES_AMIGO]->(amigoM)
RETURN Maria.nombre,Maria.edad,"es amigo de",
amigoM.nombre,amigoM.edad
```

```
$ MATCH (Maria{nombre:'María Alonso'})-[:ES_AMIGO]->(amigoM) RETURN Maria.nombre,Maria.edad,"es amigo de", amigoM.nombre,amigoM.edad
```

Table | Text | Code

"Maria.nombre"	"Maria.edad"	"\"es amigo de\""	"amigoM.nombre"	"amigoM.edad"
"María Alonso"	27	"es amigo de"	"Eva Pérez"	33
"María Alonso"	27	"es amigo de"	"Elisa Alonso"	34

Consultas en Cypher. Relaciones



Ejemplo:

Recuperar el nombre y la edad de los amigos de María Alonso y de los que ella es amiga

```
MATCH (Maria{nombre:'María Alonso'})-[:ES_AMIGO]-(amigoM)
RETURN Maria.nombre,Maria.edad,"es amigo de",
amigoM.nombre,amigoM.edad
```

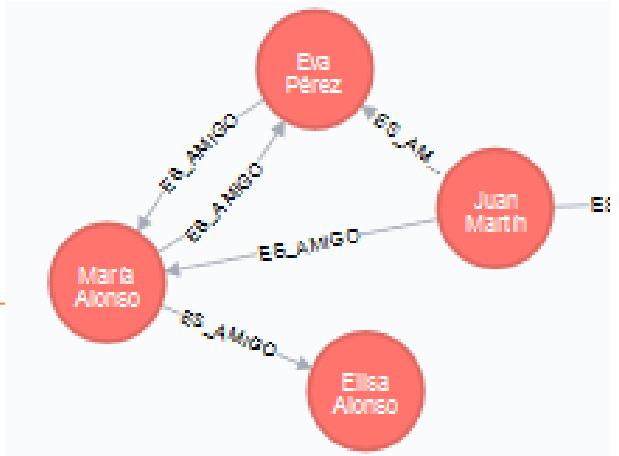
```
$ MATCH (Maria{nombre:'María Alonso'})-[:ES_AMIGO]-(amigoM) RETURN Maria.nombre,Maria.edad,"es amigo de", amigoM.nombre,amigoM.edad
```

Table | Text | Code

"Maria.nombre"	"Maria.edad"	"\"es amigo de\""	"amigoM.nombre"	"amigoM.edad"
"María Alonso"	27	"es amigo de"	"Eva Pérez"	33
"María Alonso"	27	"es amigo de"	"Eva Pérez"	33
"María Alonso"	27	"es amigo de"	"Juan Martín"	45
"María Alonso"	27	"es amigo de"	"Elisa Alonso"	34

→ Aparecen repetidos si no usamos **DISTINCT**

Consultas en Cypher. Relaciones



Ejemplo:

Recuperar el nombre y la edad de los amigos de María Alonso y de los que ella es amiga

```
MATCH (Maria{nombre:'María Alonso'})-[:ES_AMIGO]-(amigoM)
RETURN DISTINCT Maria.nombre,Maria.edad,"es amigo de",
amigoM.nombre,amigoM.edad
```

```
$ MATCH (Maria{nombre:'María Alonso'})-[:ES_AMIGO]-(amigoM) RETURN distinct Maria.nombre,Maria.edad,"es amigo de", amigoM.nombre,amigoM.edad
```

Table | Text | Code

"Maria.nombre"	"Maria.edad"	"\"es amigo de\""	"amigoM.nombre"	"amigoM.edad"
"María Alonso"	27	"es amigo de"	"Eva Pérez"	33
"María Alonso"	27	"es amigo de"	"Juan Martín"	45
"María Alonso"	27	"es amigo de"	"Elisa Alonso"	34

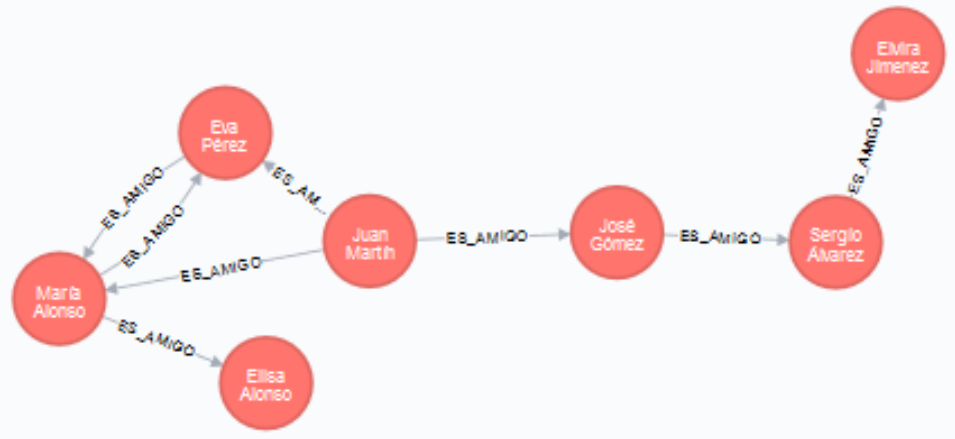
Consultas en Cypher. Relaciones

```
MATCH (a:Amigo)-[]->(c:Amigo)-[]->(d:Amigo)
WHERE a<>d
RETURN a.nombre, c.nombre, d.nombre
```

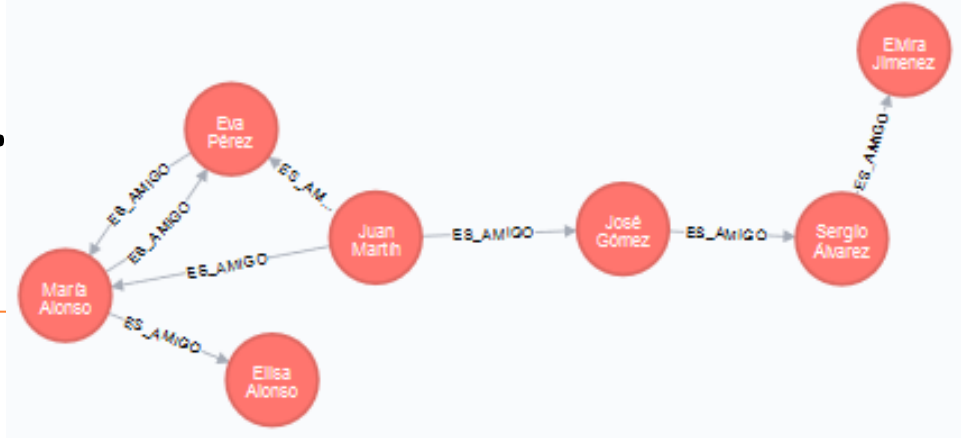
```
$ MATCH (a:Amigo)-[]->(c:Amigo)-[]->(d:Amigo) WHERE a<>d RETURN a.nombre, c.nombre, d.nombre
```

Table | Text | Code

"a.nombre"	"c.nombre"	"d.nombre"
"Juan Martín"	"Eva Pérez"	"María Alonso"
"Juan Martín"	"María Alonso"	"Eva Pérez"
"Eva Pérez"	"María Alonso"	"Elisa Alonso"
"Juan Martín"	"María Alonso"	"Elisa Alonso"
"Juan Martín"	"José Gómez"	"Sergio Álvarez"
"José Gómez"	"Sergio Álvarez"	"Elvira Jimenez"



Consultas en Cypher. Relaciones




```
MATCH (a:Amigo) -[]- (c:Amigo) -[]- (d:Amigo)
WHERE a<>d
RETURN DISTINCT a.nombre, c.nombre, d.nombre
```

```
$ MATCH (a:Amigo) -[]- (c:Amigo) -[]- (d:Amigo) WHERE a<>d RETURN distinct a.nombre, c.nombre, d.nombre
```

Table | Text | Code

a.nombre	c.nombre	d.nombre
"Sergio Álvarez"	"José Gómez"	"Juan Martín"
"María Alonso"	"Eva Pérez"	"Juan Martín"
"Eva Pérez"	"María Alonso"	"Juan Martín"
"Elisa Alonso"	"María Alonso"	"Juan Martín"
"Juan Martín"	"Eva Pérez"	"María Alonso"
"José Gómez"	"Juan Martín"	"María Alonso"
"Eva Pérez"	"Juan Martín"	"María Alonso"
"Elvira Jimenez"	"Sergio Álvarez"	"José Gómez"
"Eva Pérez"	"Juan Martín"	"José Gómez"
"María Alonso"	"Juan Martín"	"José Gómez"
"Juan Martín"	"María Alonso"	"Eva Pérez"
"Elisa Alonso"	"María Alonso"	"Eva Pérez"
"José Gómez"	"Juan Martín"	"Eva Pérez"
"María Alonso"	"Juan Martín"	"Eva Pérez"
"Eva Pérez"	"María Alonso"	"Elisa Alonso"
"Juan Martín"	"María Alonso"	"Elisa Alonso"
"Juan Martín"	"José Gómez"	"Sergio Álvarez"
"José Gómez"	"Sergio Álvarez"	"Elvira Jimenez"

Sin especificar la dirección.

Consultas en Cypher. Relaciones

```
MATCH (a)-[*2]->(b)
WHERE a<>b
RETURN b.nombre
```

→ Describe un grafo de 3 nodos y dos relaciones, es decir un **camino de longitud 2**.

- Es equivalente a :

```
MATCH(a)-->()-->(b)
WHERE a<>b
RETURN b.nombre
```

Consultas en Cypher. Relaciones

- **Relaciones de longitud variable:**

  (a)-[*3..5]->(b)

  -> Longitud mínima de 3 y máxima de 5. Describe un grafo de 4 nodos y **3 relaciones** o 5 nodos y 4 relaciones o 6 nodos y **5 relaciones** conectados.

  (a)-[*3..]->(b)

  -> Caminos de longitud 3 o más (3 o más relaciones).

  (a)-[*..5]->(b)

  -> Caminos de como máximo longitud 5 (como máximo 5 relaciones)

  (a)-[*]->(b)

  -> Caminos de a – b de cualquier longitud

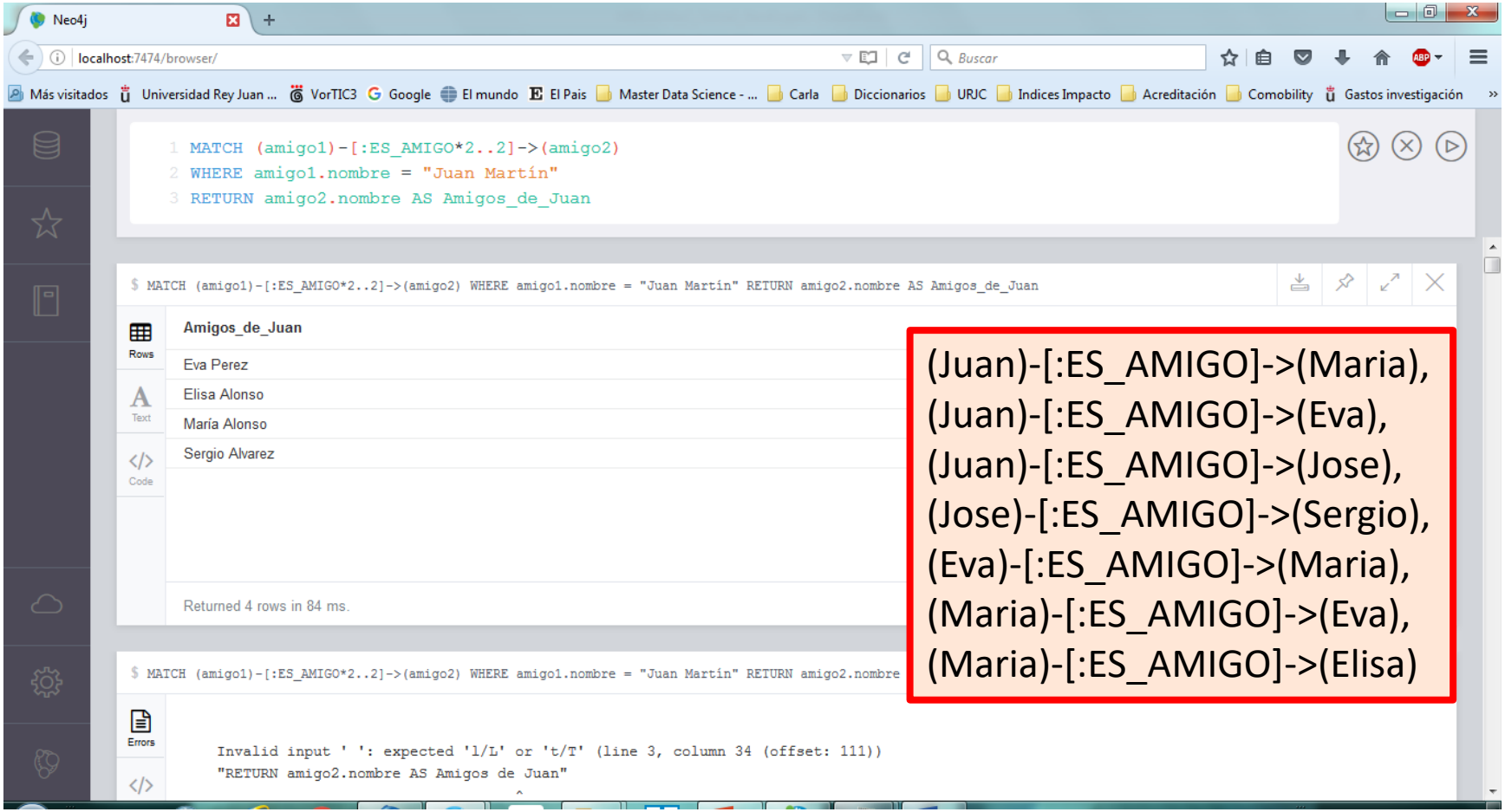
Consultas en Cypher. Relaciones

- Ejemplo:

```
MATCH (amigo1)-[:ES_AMIGO*2..2]->(amigo2)
WHERE amigo1.nombre = "Juan Martín"
RETURN amigo2.nombre AS Amigos_de_Juan
```

Equivalente a: `)-[:ES_AMIGO*2]->(`

Consultas en Cypher. Relaciones



Consultas en Cypher

CONDICIONES EN EL WHERE

Comparación de cadenas de caracteres (case-sensitive):

- STARTS WITH

```
MATCH (n)
WHERE n.nombre STARTS WITH ‘Mar'
RETURN n
```

- ENDS WITH

```
MATCH (n)
WHERE n.nombre ENDS WITH 'ta'
RETURN n
```

- CONTAINS

```
MATCH (n)
WHERE n.nombre CONTAINS ‘ar'
RETURN n
```

Consultas en Cypher

```
MATCH (amigo1)-[:ES_AMIGO]->(amigo2)
WHERE amigo1.nombre IN ['Juan Martín','María Alonso']
AND (amigo2.nombre =~ 'E.*' OR amigo2.nombre=~ 'J.*')
RETURN amigo1.nombre, amigo2.nombre
```

Similar al operador LIKE del SQL.

Consultas en Cypher

- Funciones

```
// Contar todos los nodos
MATCH (n)
RETURN count(n)

// Contar todas las relaciones
MATCH ()-->() RETURN count(*);
```

Consultas en Cypher

- Ejemplo:

```
MATCH (a1:Amigo)-->(a2:Amigo)-->
(amigo_de_amigo:Amigo)
WHERE a1.nombre= 'Juan Martín'
RETURN count(DISTINCT amigo_de_amigo),
count(amigo_de_amigo)
```

Consultas en Cypher

- Funciones CYPHER

https://neo4j.com/docs/cypher-manual/current/functions/

Consultas en Cypher. Índices

Creación de un índice:

Se crea para todos los nodos de un tipo etiqueta

```
//   ‘Amigo' nodos con etiqueta ‘Amigo’ a indexar
//   ‘nombre' propiedad a indexar
CREATE INDEX ON :Amigo(nombre)
CREATE INDEX ON :Amigo(nombre,edad)
//Si no existen las dos propiedades, no se añadirá el nodo al índice.
```

Borrar un índice:

```
DROP INDEX ON :Amigo(nombre)
```

Consultas en Cypher. Índices

Uso de los índices

- Neo4J automáticamente selecciona el índice
- USING INDEX :

```
MATCH (n:Amigo)
USING INDEX n:Amigo(nombre)
WHERE n.nombre= 'María Alonso'
RETURN n as amiguito
```

Consultas en Cypher. Transacciones

- Cualquier consulta que actualice un grafo se ejecutará bajo una transacción.
- Esta puede o no terminar de forma existosa.

Índice

- *Introducción a Neo4J*
- *Instalación y puesta en marcha*
- *Creación de una BD orientada a grafos*
- *Lenguaje Cypher para crear nodos y relaciones*
- *Consultas en Cypher*
- **Actualizaciones en Cypher**
- Importar y Exportar Nodos

Actualizaciones en Cypher

- Añadir una propiedad o varias a un nodo

```
MATCH (a:Amigo)
WHERE a.nombre = 'José Gómez'
SET a.ciudad='Madrid', a.provincia='Madrid'
RETURN a
```

- Eliminar una propiedad de un nodo

```
MATCH (a:Amigo)
WHERE a.nombre = 'José Gómez'
SET a.edad=NULL
RETURN a

MATCH (a {nombre = 'José Gómez' })
REMOVE a.edad
RETURN a
```

Actualizaciones en Cypher

- Copiar una propiedad entre nodos

```
MATCH (a:Amigo {nombre:'José Gómez'}),(b:Amigo{nombre: 'Elvira Jimenez'})
SET b.ciudad=a.ciudad
RETURN a,b
```

- Copiar todas las propiedades de un nodo a otro

```
MATCH (a:Amigo {nombre:'José Gómez'}),(b:Amigo{nombre: 'Elvira Jimenez'})
SET b=a
RETURN a,b
```

Actualizaciones en Cypher

- Añadir una etiqueta a un nodo

```
MATCH (a:Amigo {nombre: 'Eva Pérez'})
SET a :Persona
RETURN a
```

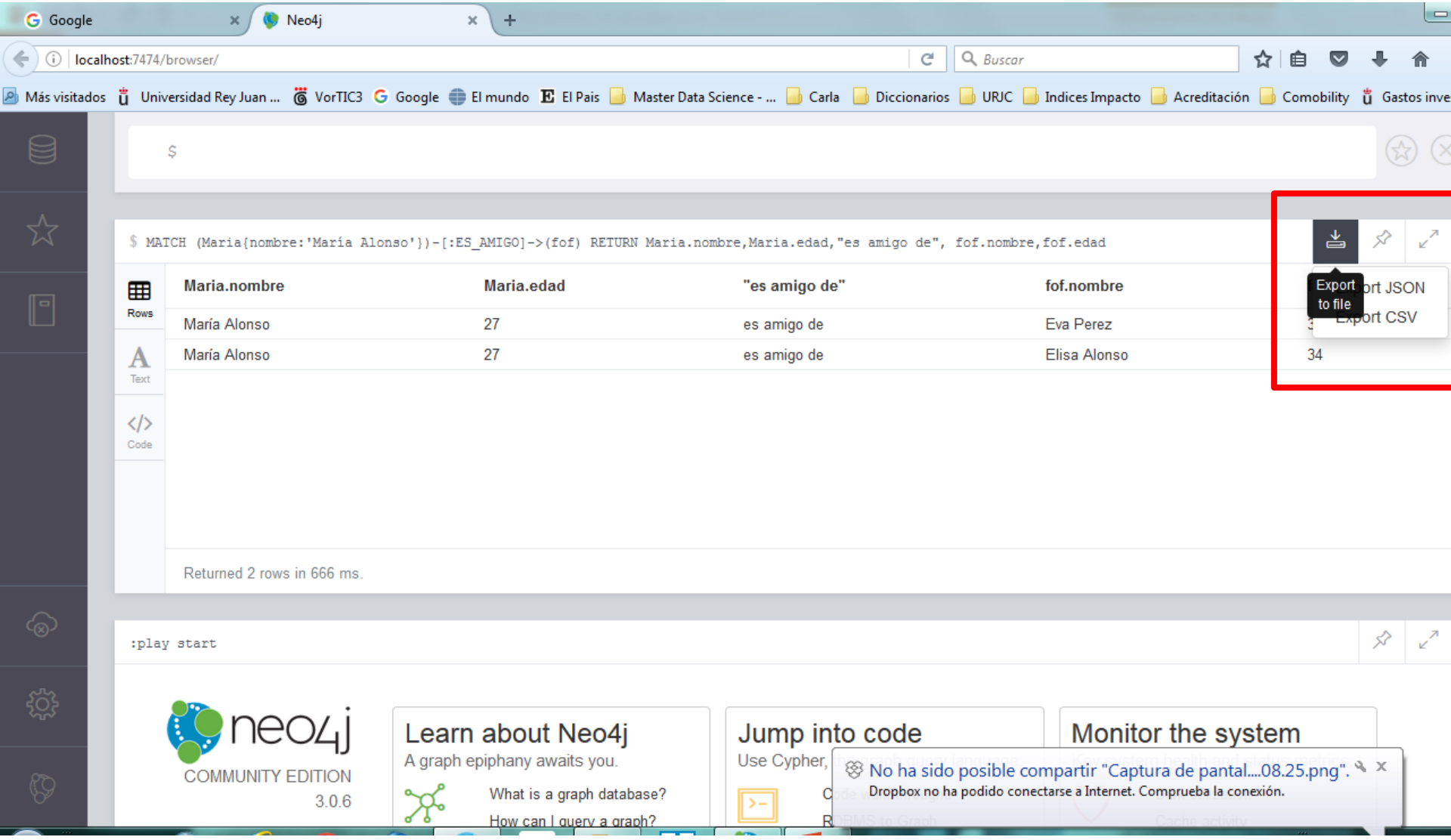
- Eliminar una etiqueta de un nodo

```
MATCH (a:Amigo {nombre: 'Eva Pérez'})
REMOVE a :Persona
RETURN a, labels(a)
```

Índice

- *Introducción a Neo4J*
- *Instalación y puesta en marcha*
- *Creación de una BD orientada a grafos*
- *Lenguaje Cypher para crear nodos y relaciones*
- *Consultas en Cypher*
- *Actualizaciones en Cypher*
- **Importar y Exportar Nodos**

Exportar resultados



Importar ficheros CSV con Cypher. Nodos

- Funcionalidad **neo4j-admin import**

```
neo4j-admin import [--mode=csv] [--database=<name>]
                   [--nodes[:Label1:Label2]=<"file1,file2,...">]
                   [--
relationships[:RELATIONSHIP_TYPE]=<"file1,file2,...">] ….
```

- LOAD CSV se usa para importar datos desde ficheros CSV.

```
LOAD CSV WITH HEADERS FROM “file:///persons.csv" AS csvReg
CREATE (p:Persona { id: toInt(csvReg.id), nombre: csvReg.name })
```

Si el fichero tiene cabecera y se podrá acceder a cada campo con el nombre de columna

file:///fichero.csv

<NEO4J_HOME>/import/fichero.csv

Importar ficheros CSV con Cypher. Relaciones

```
USING PERIODIC COMMIT 500
//Si se opera con ficheros CSV de gran tamaño
LOAD CSV WITH HEADERS FROM “roles.csv" AS csvP
MATCH (persona:Persona { id: toInt(csvP.personId)}),
(pelicula:Pelicula { id: toInt(csvP.movieId)})
CREATE (persona)-[:PLAYED { role: csvP.role }]->(pelicula)
```

Índice

- *Introducción a las Bases de Datos NoSQL*
  - *1.1 Limitaciones de las Bases de Datos Relacionales*
  - *1.2 Clasificación de las Bases de Datos NoSQL*
  - *1.3 Elección de una Base de Datos NoSQL*
- *BD Clave-Valor*
- *BD orientadas a Documentos*
- ***BD orientadas a Grafos***
- BD Familia de Columnas