

BBDD orientadas a grafos

Neo4j

Ampliación a Bases de Datos

Profesor: Pablo Ramos

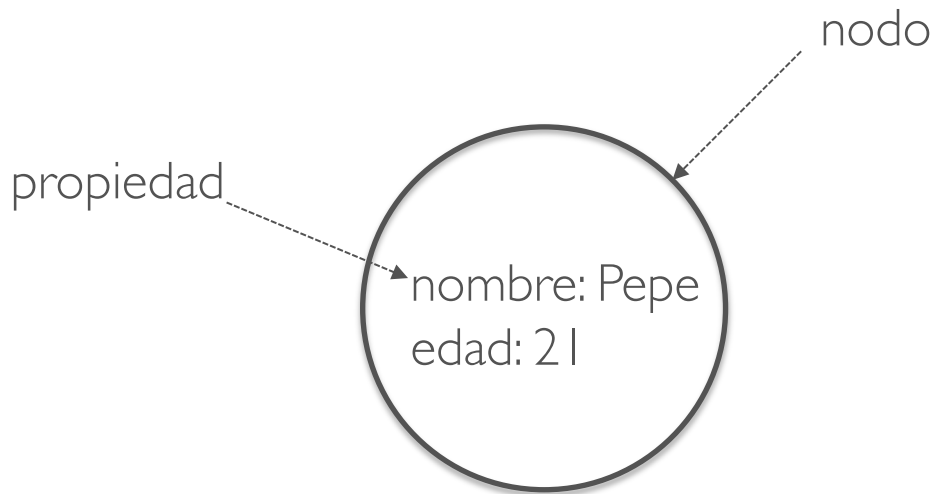
pablo.ramos@u-tad.com

INTRODUCCIÓN

- Uso de grafos para almacenar la información:
 - Nodos: representa entidades de información
 - Aristas: representan relaciones entre nodos.
 - Propiedades: tanto los nodos como las aristas tienen campos asociados que proporcionan información adicional.
- Patrones
 - Los grafos permiten describir patrones de una forma parecida a la que los humanos estructuran la información.

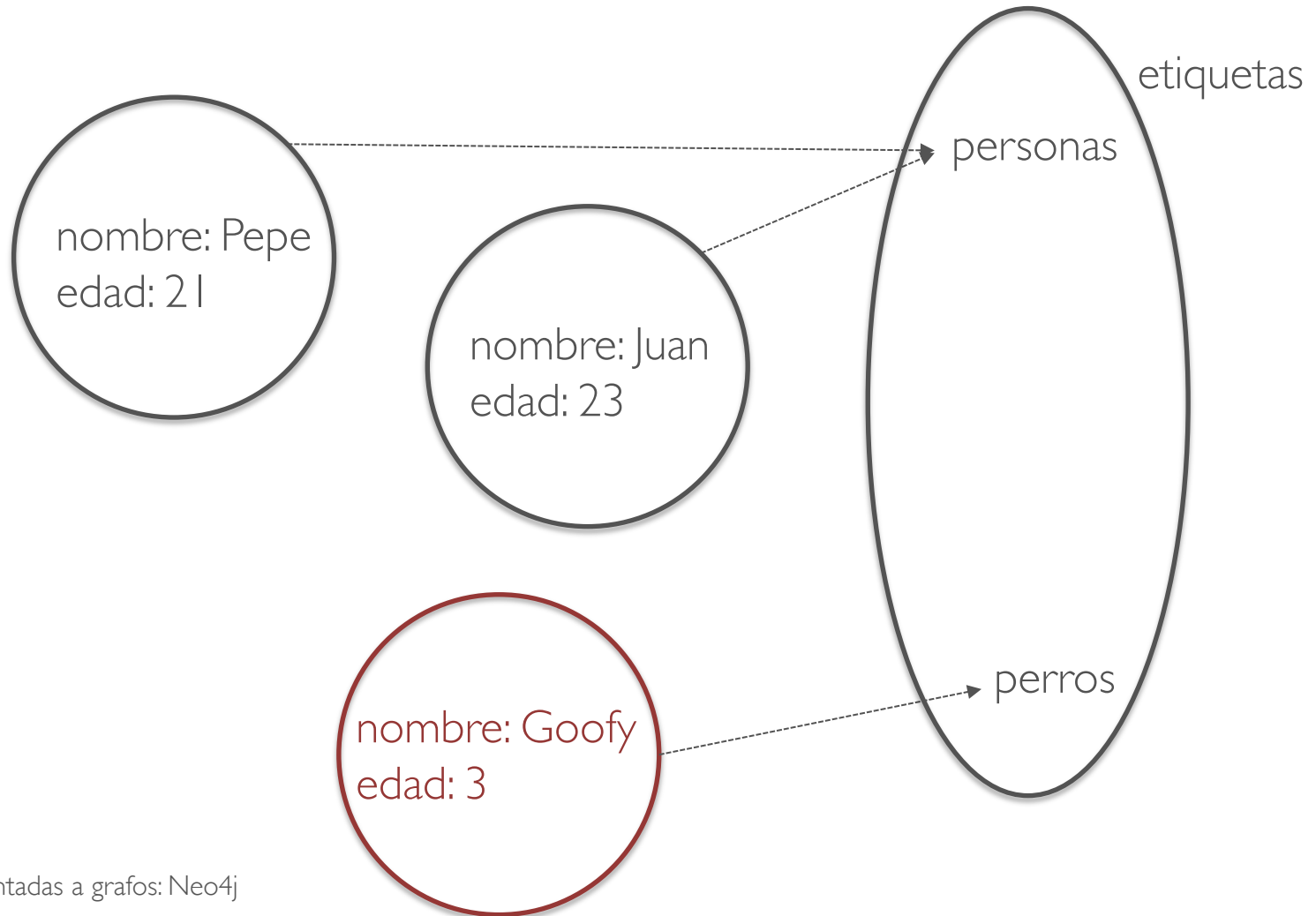
ELEMENTOS: Nodo y propiedades

- Un nodo representa una entidad de información.
- Los nodos tienen un conjunto de propiedades que contienen información del nodo.



ELEMENTOS: Etiquetas

- Los nodos son clasificados por etiquetas
 - Cada nodo puede tener cero o varias etiquetas.

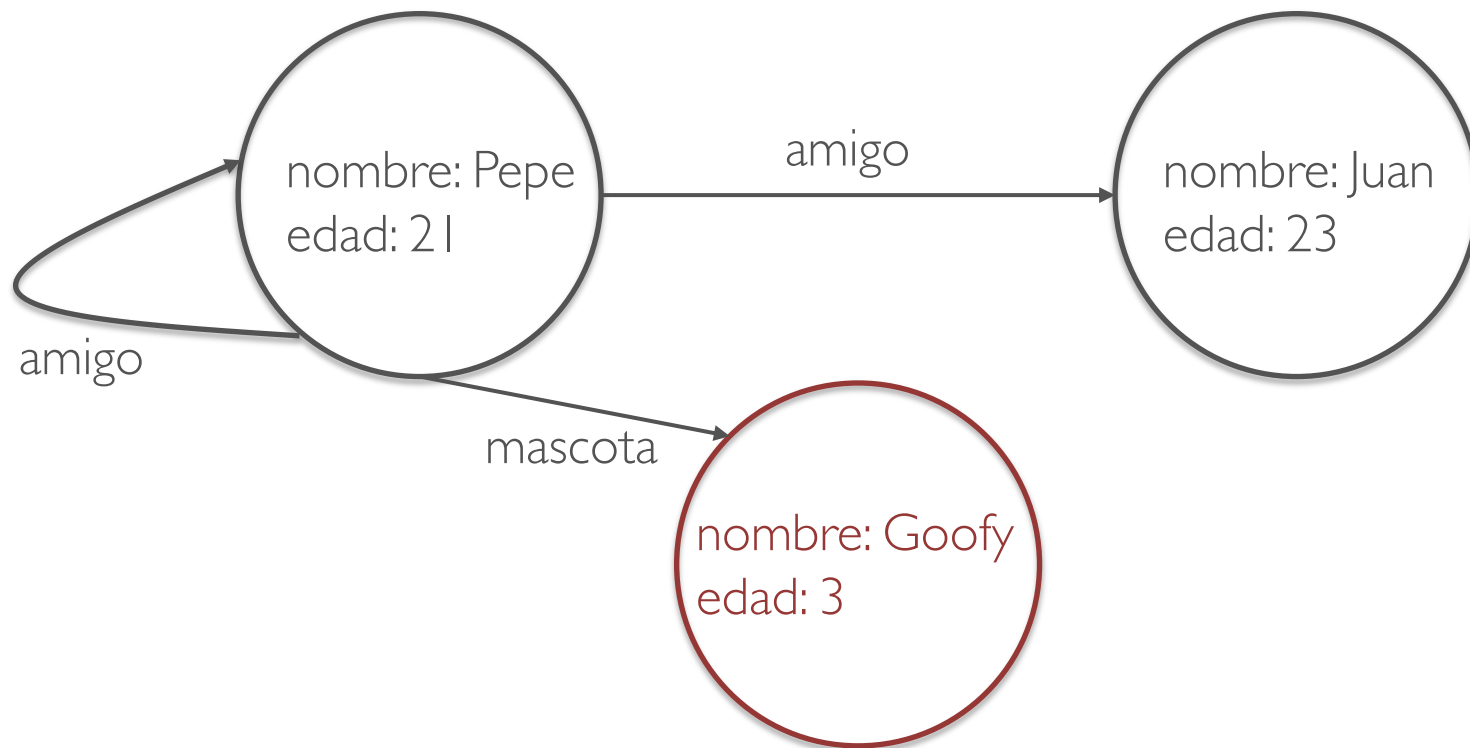


ELEMENTOS: Etiquetas

- Los nodos son clasificados por etiquetas
 - Cada nodo puede tener cero o varias etiquetas.
 - Las etiquetas se pueden añadir o quitar en tiempo de ejecución.
 - Los nodos son agrupados en conjuntos en función de las etiquetas que tengan.
 - Personas = {Pepe, Juan}
 - Perros = {Goofy}
 - Se pueden realizar consultas buscando por etiqueta.

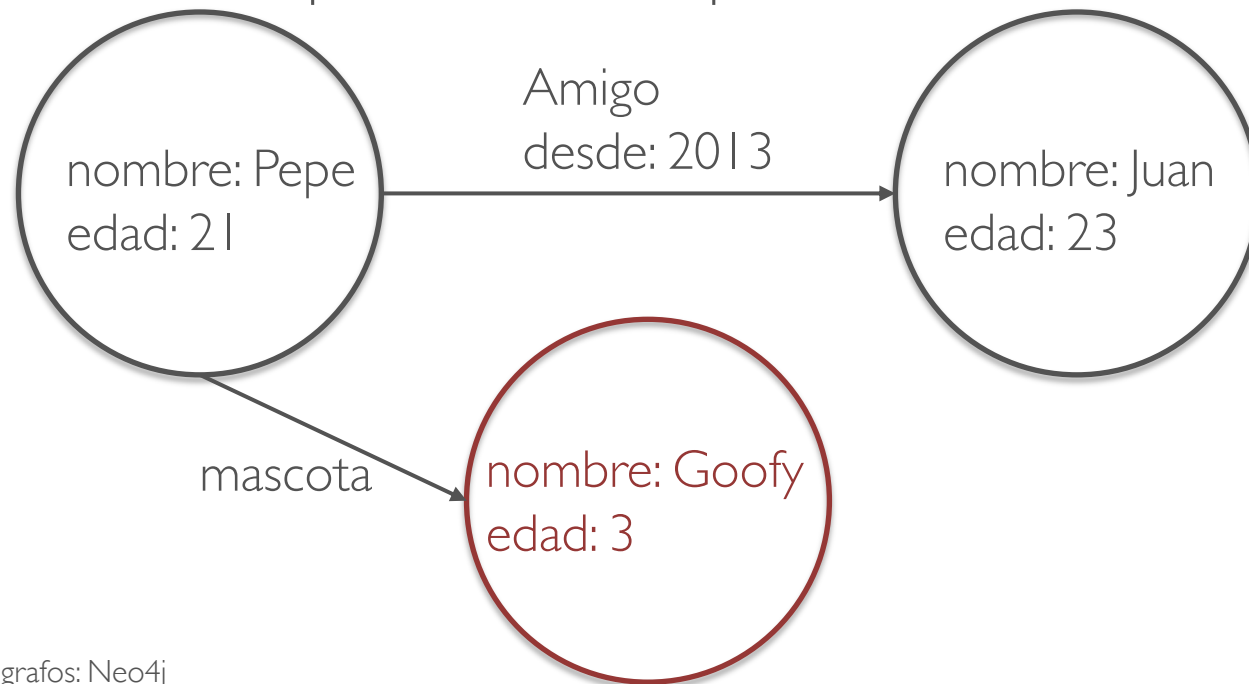
ELEMENTOS: Relaciones

- Las relaciones son aristas entre nodos que conectan nodos y expresan algún tipo de relación entre las entidades que enlaza.
 - Las relaciones tienen dirección.
 - Las relaciones son de un tipo.



ELEMENTOS: Relaciones

- Las relaciones son aristas entre nodos que expresan algún tipo de relación entre las entidades que enlaza
 - Las relaciones tienen dirección.
 - Las relaciones son de un tipo.
 - Las relaciones tienen propiedades que aportan más información al tipo de relación que tienen.

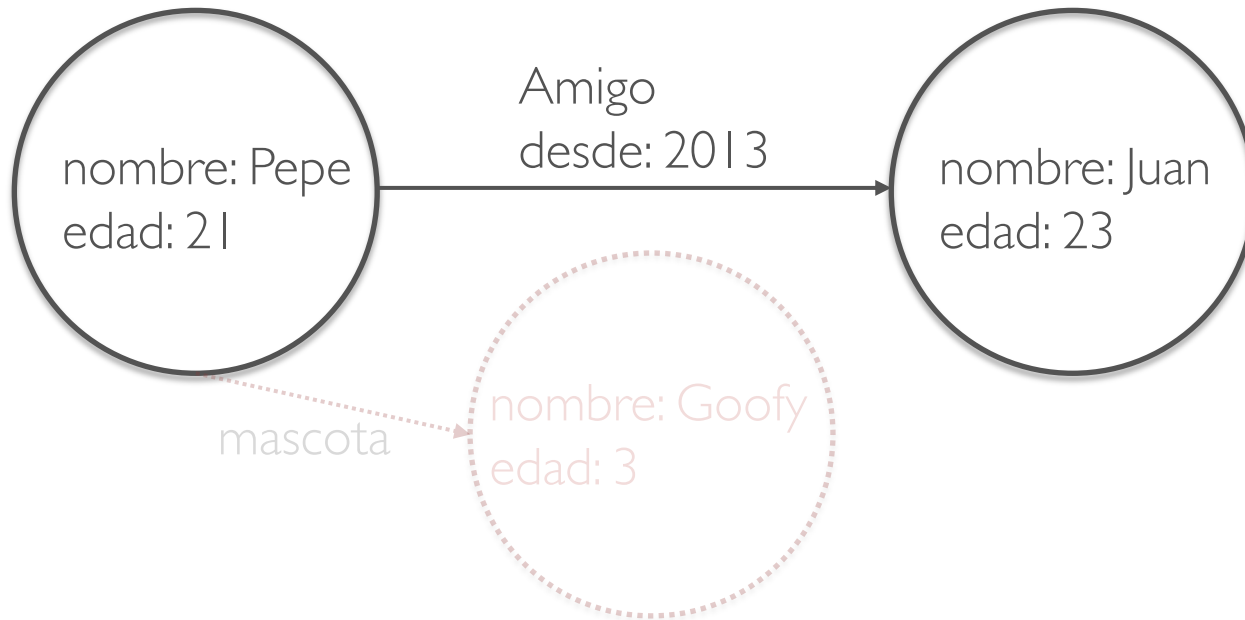


ELEMENTOS: Propiedades

- Las propiedades de los nodos y relaciones son pares clave valor.
 - Las claves son cadenas de texto
 - Los valores pueden ser:
 - Números
 - Cadenas de texto
 - Booleanos
 - Colecciones de números, cadenas de texto o booleanos.

CONSULTAS: *Traversal* (Recorrido)

- *Traversal*, es el modo en que se realizan consultas en Neo4j.
- Se realiza un recorrido a través del grafo siguiendo las reglas especificadas.
 - Buscar amigos de Pepe



- Cypher: Lenguaje de consultas para Neo4j

CONSULTAS: *Traversal* (Recorrido)

- *Traversal*, es el modo en que se realizan consultas en Neo4j.
- Se realiza un recorrido a través del grafo siguiendo las reglas especificadas.
- **Cypher**: Lenguaje de consultas para Neo4j
 - CheatSheet:
 - <https://neo4j.com/docs/cypher-cheat-sheet/5/auradb-enterprise>
- Cuando se realiza un recorrido:
 - Un nodo puede ser recorrido **varias veces**
 - Una relación solo puede ser recorrida **una vez**

CONSULTAS: Path (Camino)

- Path, es el resultado de una consulta en Neo4j.
- Aquellas relaciones que cumplan con las restricciones son devueltas junto con los nodos que interconecta.
 - Amigos de Pepe



CYPHER: Nodos

- Los nodos se representan por medio de ()
()
(identificador)
(:etiqueta)
(identificador:etiqueta)
(identificador:etiqueta {propiedad:valor})
(identificador:etiqueta WHERE ...)
- Ejemplo:
(a:persona {nombre: “Pepe”, edad:21})
(a:persona
WHERE a.nombre=“Pepe” AND a.edad=21})

CYPHER: Relaciones

- Las relaciones se representan mediante `-->`, `--`, `<--`
`-->`
 - [identificador]->
 - [:etiqueta]->
 - [identificador:etiqueta]->
 - [identificador:etiqueta {propiedad:valor}]->
 - [identificador:etiqueta WHERE ...]->
- Ejemplo:
 - [r:amigo {durante:5}]->
 - [r:amigo WHERE r.durante=5]->

CYPHER: Patrón

- Los nodos y combinaciones de nodos y relaciones crean estructuras que llamados patrones.

```
(identificador:etiqueta {propiedad:valor})  
-[identificador:etiqueta {propiedad:valor}]->  
(identificador:etiqueta {propiedad:valor})
```

- Ejemplo:

```
(p:persona {nombre: "Pepe", edad:21})  
-[r:amigo {desde:2013}]->  
(j:persona {nombre: "Juan", edad:23})
```

CYPHER: Patrón

- Un patrón puede tener identificador
`identificador = (:etiqueta)-->(:etiqueta)`
- Ejemplo:
`mejor_amigo = (:persona {nombre: Pepe,
edad:21})
-[:amigo {desde:2013}]->
(:persona {nombre: Juan,
edad:23})`

CYPHER: Patrón

- Los patrones también son utilizados para identificar estructuras específicas en la base de datos
 - Cualquier nodo

```
(n) // Utilizamos n como identificador
      // del nodo para poder operar con él.
```
 - Nodos con algún tipo de relación:

```
(a) --> (b)
(a) -- (b) - [r:REL_TYPE] -> ()
(a) - [r:TYPE_1|TYPE_2] -> () // tipo 1 o 2
```


CYPHER: Patrón

- Los patrones también son utilizados para identificar estructuras específicas en la base de datos
 - Caminos de longitud definida:
 - (a) - [*2] -> (b) // longitud 2
 - (a) - [*2..5] -> (b) // longitud 2 a 5
 - (a) - [*] -> (c) // cualquier longitud

Variante	Descripción
*	1 o más iteraciones.
*n	Exactamente n iteraciones.
*m..n	Entre m y n iteraciones.
*m..	m o más iteraciones.
*..n	Entre 1 y n iteraciones.

CYPHER: Patrón

- Los patrones también son utilizados para identificar estructuras específicas en la base de datos
 - Repeticiones de patrón:

(a) - [r] -> (b) {2} // 2 veces el patrón

(a) - [r] -> {2} (b) // 2 veces la relación

Variante	Forma canonica	Descripción
{m,n}	{m,n}	Entre m y n iteraciones.
+	{1,}	1 o más iteraciones.
*	{0,}	0 o más iteraciones.
{n}	{n,n}	Exactamente n iteraciones.
{m,}	{m,}	m or more iteraciones.
{,n}	{0,n}	Entre 0 y n iteraciones.
{,}	{0,}	0 o más iteraciones.

CYPHER: Consultas

- RETURN define la salida que va a tener una query
 - Identificador:
 - Nodo
 - Relación
 - propiedad

`RETURN identificador[.propiedad]`

- Todos los elementos

`RETURN *`

- Resultados únicos

`RETURN DISTINCT identificador[.propiedad]`

- Evaluar expresiones

`RETURN expresión // e.g. mayores de 30`

CYPHER: Consultas de escritura

- CREATE permite crear nuevos patrones.

```
CREATE patrón  
[RETURN identificador]
```

- Ejemplos:

```
CREATE (p:persona {nombre:"Pepe",edad:21})  
RETURN p
```

```
CREATE (:persona {nombre: "Pepe", edad:21})  
-[:amigo {desde:2013}]->  
(:persona {nombre: "Juan", edad:23})
```

CYPHER: Consultas de lectura

- MATCH permite realizar consultas que cumplan con el patrón especificado.

`MATCH patrón`

- Ejemplos:

`MATCH (n) RETURN n`

`//Todos los nodos`

`MATCH (p:Persona) RETURN p`

`//Todos los nodos de tipo persona`

`MATCH (p:Persona {edad: 21}) RETURN p.edad`

`// Edad de los nodos tipo persona de 21años`

`MATCH (p)-[r:amigo]->(s) RETURN r.desde`

`// Año desde con son amigos cualquier nodo`

`MATCH (p)-[*]-() //Cualquier longitud`

CYPHER: Consultas de lectura

- OPTIONAL MATCH permite incluir patrones adicionales a la consultas. Si el patrón no se cumple devuelve null, en vez de nada.

`MATCH patrón`

`OPTIONAL MATCH patrón`

- Ejemplos:

`MATCH (n:persona)`

`OPTIONAL MATCH (n) - [r*1] -> ()`

`RETURN r`

CYPHER: Consultas de lectura

- WHERE permite incluir restricciones en la consulta del patrón especificado.

```
WHERE restricciones
```

- Ejemplo:

```
MATCH (p)                //Filtro por etiqueta
WHERE p:Persona
```

```
MATCH (p)                //Filtro por propiedad
WHERE EXISTS(p.edad) AND // Tiene propiedad
      (p.edad < 30 AND p.edad > 20) AND
      p.nombre =~ "Pe.*" // Expresión regular
```

- Es posible utilizar expresiones regulares para filtrar propiedades

CYPHER: Consultas de lectura

- WHERE permite incluir restricciones en la consulta del patrón especificado.

`WHERE restricciones`

- Ejemplo:

```
MATCH (p) //Filtro por patrón
WHERE p.name IN [Pepe, Pedro] AND //Colección
      (p)-->({nombre:Juan}) //Patrón
```

```
MATCH ()-[r]->() //Filtro por patrón
WHERE type(r)=~"A.*" //Relaciones de tipo
                      //que empiece por A
```


CYPHER: Consultas de escritura

- MATCH CREATE permite crear nuevos patrones a partir de patrones existentes.

```
MATCH patrón  
[WHERE restricciones]  
CREATE patrón  
[RETURN identificador]
```

- Ejemplos

```
MATCH (a:persona), (b:persona)  
WHERE a.nombre = "Pepe" AND b.nombre = "Juan"  
CREATE (a)-[:amigo {desde:2013} ]->(b)
```

CYPHER: Consultas de escritura

- MERGE asegura que un patrón existe, si no, lo crea.

```
MERGE patrón
```

```
RETURN identificador
```

- Ejemplos

```
CREATE (:persona {nombre: "Pepe"})
```

```
// Crea un nuevo nodo
```

```
MERGE (a:persona {nombre: "Pepe", edad: "21"})
```

```
RETURN a
```

```
// Devuelve las dos personas con nombre Pepe
```

```
MERGE (b:persona {nombre: "Pepe"})
```

```
RETURN b
```

CYPHER: Consultas de escritura

- MERGE asegura que un patrón existe, si no, lo crea.

```
MERGE patrón
```

```
[ON CREATE SET propiedades]
```

```
[ON MATCH SET propiedades]
```

```
RETURN identificador
```

- ON CREATE permite asignar propiedades si se crea el patrón
- ON MATCH permite asignar propiedades si el patrón existe

- Ejemplo

```
MERGE (a:persona {nombre: "Pepe", edad: "21"})
```

```
ON CREATE SET a.creado = timestamp(),
```

```
                a.nuevo = True
```

```
RETURN a
```

CYPHER: Consultas de escritura

- MATCH MERGE partiendo de patrones existentes, asegura que un patrón existe que lo/s incluye, si no, lo/s crea.

`MATCH patrón`

`MERGE patrón`

`RETURN identificador`

- Ejemplos

`MATCH (a:persona), (b:persona)`

`WHERE a.nombre = "Pepe" AND b.nombre = "Juan"`

`MERGE (a)-[:amigo]-(b) //Podemos no especificar la dirección de la relación, de modo que busca en cualquier tipo de relación, si no existe crea una cualquiera.`

CYPHER: Consultas de lectura

- ORDER BY especifica en orden en que deben ser mostrados los resultados

```
ORDER BY identificador.propriedad
```

- Ejemplo:

```
MATCH (p)
```

```
RETURN p
```

```
ORDER BY p.edad
```

```
MATCH (p)
```

```
RETURN p
```

```
ORDER BY p.edad, p.nombre DESC
```

CYPHER: Consultas de lectura

- LIMIT limita el número de elementos a mostrar

`LIMIT número_elementos`

- Ejemplo:

`MATCH (p)`

`RETURN p`

`LIMIT 2 // Muestra primeros 2 elementos`

CYPHER: Consultas de lectura

- SKIP especifica desde que fila empezar a mostrar el resultado
SKIP número_elementos

- Ejemplo:

```
MATCH (p)
```

```
RETURN p
```

```
SKIP 2           // Salta primeros 2 elementos
```

```
MATCH (p)
```

```
RETURN p
```

```
SKIP 1
```

```
LIMIT 2         // Muestra segundo elemento
```

CYPHER: Consultas de lectura, *Aggregation*

- COUNT permite contar el número de elementos existentes en el resultado de una consulta.

- Número de filas

- `count(*)`

- Número de elementos no nulos con el identificador especificado

- `count(identificador)`

- Ejemplo:

- `MATCH (p) - [r] -> (s)`

- `RETURN count(*)`

CYPHER: Consultas de lectura, *Aggregation*

- SUM suma todos los valores con el identificador especificado. Los valores NULL son ignorados.

`sum(identificador)`

- AVG devuelve la media de todos los valores con el identificador especificado. Los valores NULL son ignorados.

`avg(identificador)`

- MAX devuelve el máximo de todos los valores con el identificador especificado.

`max(identificador)`

- MIN devuelve el mínimo de todos los valores con el identificador especificado.

`min(identificador)`

CYPHER: Consultas de lectura, *Aggregation*

- COLLECT permite crear una colección con todos los elementos resultantes de una consulta. Los valores NULL son ignorados.

```
collect(identificador)
```

- Ejemplo:

```
MATCH (p:Persona)  
RETURN collect(p.edad)
```

- DISTINCT elimina valores duplicados para un identificador dado del resultado de una consulta

```
DISTINCT identificador
```

- Ejemplo:

```
MATCH (p:Persona)  
RETURN collect(DISTINCT p.edad)
```

CYPHER: Consultas de lectura, *Aggregation*

- Claves de agrupación (*grouping keys*)
 - Permite agregar por clave de agrupación
 - Agregación para cada elemento K
- Ejemplo: tipos de relación y número de relaciones para cada tipo

```
MATCH (p) - [r] -> (s)
RETURN type(r), count(*)
```

CYPHER: Consultas de lectura

- WITH permite especificar que elementos y como se van a pasar a la siguiente parte de la consulta (pipeline)

`WITH identificadores [AS alias]`

`WITH función AS alias`

- Ejemplo:

`MATCH (p:persona)`

`WITH collect(p) as personas`

`RETURN personas`

CYPHER: Consultas de lectura

- UNWIND expande una colección a un conjunto de filas
UNWIND colección AS iterador

- Ejemplo:

```
UNWIND [1,2,3,4] AS numero  
RETURN numero
```

```
UNWIND [1,1,2,3,4,4] AS numero  
WITH DISTINCT numero  
RETURN collect(numero) AS conjunto
```

CYPHER: Consultas de lectura

- UNION combina resultados de varias consultas

- Combina todos los resultados

UNION ALL

- Combina los resultados eliminando duplicados

UNION

- Ejemplo:

```
MATCH (n:persona)
```

```
RETURN n.nombre AS nombre
```

```
UNION
```

```
MATCH (n:animal)
```

```
RETURN n.nombre AS nombre
```

CYPHER: Consultas de lectura, Funciones

- PREDICADOS:

- ALL: Comprueba si todos los elementos de una colección cumplen un predicado.

`all(identificador IN colección WHERE predicado)`

- ANY: Comprueba si alguno de los elementos de una colección cumple un predicado.

`any(identificador IN colección WHERE predicado)`

- NONE: Comprueba si ninguno de los elementos de una colección cumple un predicado.

`none(identificador IN colección WHERE predicado)`

- SINGLE: Comprueba si un elemento de una colección cumple un predicado.

`single(identificador IN colección WHERE predicado)`

- EXISTS: Comprueba si el patrón o identificador existe.

`exists(patrón o propiedad)`

- Ejemplo: Todas las relaciones en que son mayores de edad

`MATCH relación = ()-[r:amigo]-()`

`WHERE all(n IN nodes(relación) WHERE n.edad > 18)`

`RETURN relación`

CYPHER: Consultas de lectura, Funciones

- FUNCIONES ESCALARES (devuelven un solo valor):

NOMBRE_FUNCION(identificador)

- **length**: Longitud de un camino (*path*).
- **size**: Longitud de una lista o string.
- **type**: Tipo de relación del identificador especificado.
- **head, last**: Devuelve el primer o último elemento de una colección.
- **timestamp** : Devuelve el timestamp.
- **toInt, toFloat, toString**: Convierte una variable en entero, real o texto.
- **startNode, endNode**: Devuelve el nodo de comienzo o fin de una relación.
- **coalesce**: Devuelve el primer valor distinto de NULL de una lista.
- **elementId**: id de un nodo o relación.
- **properties**: Devuelve las propiedades de un nodo o relación.

CYPHER: Consultas de lectura, Funciones

- COLECCIONES (devuelven colecciones):
`NOMBRE_FUNCION(identificador)`
 - **nodes**: Devuelve todos los nodos de una ruta.
 - **relationships**: Devuelve todas las relaciones de una ruta.
 - **labels**: Listado de todas las etiquetas de un nodo.
 - **keys**: Listado con todas las claves de las propiedades de un nodo o relación.
 - **reduce**: Devuelve el resultado acumulado al aplicar una expresión a todos los elementos de una colección.
 - **tail**: Devuelve todos los elementos de una colección excepto el primero.
 - **range**: Devuelve todos los elementos de una colección en un intervalo especificado y con un paso especificado.

CYPHER: Consultas de lectura, Funciones

- DOCUMENTACION COMPLETA
 - <https://neo4j.com/docs/cypher-manual/current/functions/>
- MATEMÁTICAS:
 - <https://neo4j.com/docs/cypher-manual/current/functions/mathematical-numeric/>
- CADENAS
 - <https://neo4j.com/docs/cypher-manual/current/functions/string/>

CYPHER: Consultas de actualización

- SET permite actualizar las etiquetas los nodos y propiedades de nodos y relaciones.

```
MATCH patrón
```

```
SET etiqueta | propiedad
```

```
[RETURN identificador]
```

- Ejemplos

```
MATCH (a:persona {nombre: "Pepe"})
```

```
SET a :Extraterrestre:Marte //Nueva etiqueta
```

```
MATCH (a:persona {nombre: "Pepe"})
```

```
SET a.nombre = Juan, a.edad=29 //Nuevo valor
```

```
MATCH (a:persona {nombre: "Pepe"})
```

```
SET a.nombre = NULL //Elimina propiedad
```

CYPHER: Consultas de actualización

- SET se puede utilizar para copiar todas las propiedades de un nodo a otro. Las propiedades del nodo receptor serán borradas previamente.

`MATCH patrones`

```
SET identificador = properties(identificador)
[RETURN identificador]
```

- Ejemplos

```
MATCH (a:persona {nombre: "Pepe"}),
      (b:persona {nombre: "Juan"})
SET a = properties(b)
```

CYPHER: Consultas de actualización

- SET puede usarse también maps

- Ejemplos

```
MATCH (a:persona {nombre: "Pepe"})  
SET a += {altura: 1,80, peso: 75}  
//Añade propiedades
```

```
map = {altura: 1,80, peso: 75}
```

```
MATCH (a:persona {nombre: "Pepe"})  
SET a = $map //Reemplaza propiedades
```

```
MATCH (a:persona {nombre: "Pepe"})  
SET a = {} //Borra todas las propiedades
```

CYPHER: Consultas de actualización

- FOREACH permite actualizar datos en colecciones de datos, caminos y resultados de agregados.

`MATCH patrón`

`FOREACH (iterador | operador)`

- Ejemplos

`MATCH c = (:persona) - [*] - (:persona)`

`FOREACH (n IN nodes(c) | SET n:adulto)`

CYPHER: Consultas de eliminación

- DELETE permite eliminar nodos y relaciones.

`MATCH patrón`

`DELETE identificador`

- Ejemplos

`MATCH (a:persona {nombre: "Pepe"})`

`DELETE a`

CYPHER: Consultas de eliminación

- REMOVE permite eliminar etiquetas y propiedades.

`MATCH patrón`

`REMOVE id:etiqueta | id.propiedad`

- Ejemplos

`MATCH (a:persona {nombre: "Pepe"})`

`REMOVE a:persona, a.nombre`

SCHEMA: Índices

- CREATE INDEX crea un índice para una propiedad de una etiqueta o tipo determinado.
 - El tipo no es necesario si es un índice clásico (Range)
 - Tipos: RANGE, TEXT, POINT

```
CREATE [TIPO] INDEX nombre [IF NOT EXISTS]  
FOR etiqueta|tipo  
ON propiedad
```

- Ejemplos:

```
CREATE INDEX persona_nombre IF NOT EXISTS  
FOR (p:persona) ON p.nombre
```

```
SHOW INDEXES
```

SCHEMA: Índices

- DROP INDEX elimina el índice de una propiedad para una etiqueta determinada.

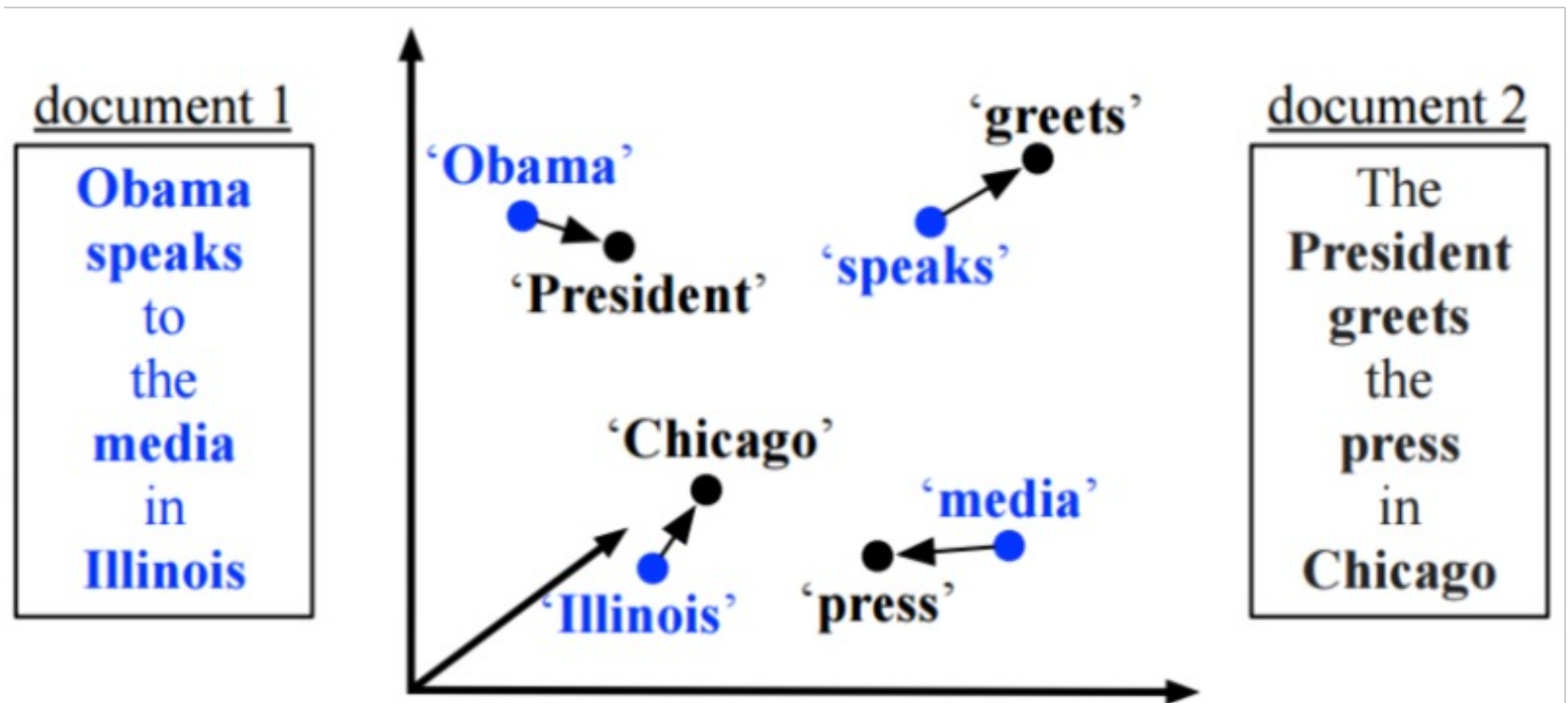
```
DROP INDEX nombre_indice
```

- Ejemplos

```
DROP INDEX persona_nombre
```

SCHEMA: Índices

- Capacidad para crear índices de vectores (*embeddings*)



SCHEMA: Restricciones

- CREATE CONSTRAINT crea una restricción del tipo indicado a la propiedad indicada. La restricción se cumplirá a nivel de etiqueta. Si se crea una restricción de tipo único, también se creará un índice sobre es propiedad.

```
CREATE CONSTRAINT nombre [IF NOT EXISTS]
FOR (etiqueta)
REQUIRE restricción
```

- Ejemplos

```
CREATE CONSTRAINT dni_único IF NOT EXISTS
FOR (p:persona)
REQUIRE p.dni IS UNIQUE
```

SCHEMA: Restricciones

- DROP CONSTRAINT elimina una restricción del tipo indicado a la propiedad indicada. Si se elimina una restricción de tipo único, también se eliminará el índice que se creó.

`DROP CONSTRAINT nombre_restricción`

- Ejemplos

`DROP CONSTRAINT dni_unico`