

# Bases de Datos orientadas a grafos

4.4.1 Características

4.4.2 Arquitecturas

4.4.3 Análisis de ventajas y desventajas

4.4.4 Aplicaciones

# ¿Por qué bases de datos de grafos?

- Hoy en día, la mayoría de los datos existen en forma de relación entre diferentes objetos y, más a menudo, la relación entre los datos es más valiosa que los datos en sí.
- Las bases de datos relacionales almacenan datos altamente estructurados con una gran cantidad de registros, que almacenan el mismo tipo de dato para cada columna.
- A diferencia de otras bases de datos, las bases de datos de grafos almacenan relaciones y conexiones como entidades.
- El modelo de datos para bases de datos de grafos es más simple en comparación con otras bases de datos y se pueden usar con sistemas OLTP. Proporcionan características como la integridad transaccional y la disponibilidad operativa.

# Base de Datos Orientada a Grafos

Una base de datos orientada a grafos es un sistema de administración de bases de datos en línea con operaciones de Crear, Leer, Actualizar y Eliminar (CRUD) que trabajan en un modelo de datos de grafos.

A diferencia de otras bases de datos, las relaciones tienen prioridad en las bases de datos orientadas a grafos.

- El modelo de datos para una BDOG también es significativamente más simple y más expresivo que los de bases de datos relacionales u otras bases de datos NoSQL.

Las bases de datos de grafos se crean para su uso con sistemas transaccionales (OLTP) y se diseñan teniendo en cuenta la integridad transaccional y la disponibilidad operativa.

<b>RDBMS</b>	<b>Base de datos de grafos</b>
Tablas	Grafos
Filas	Nodos
Columnas y datos	Propiedades y sus valores
Restricciones	Relaciones
Joins	Navegación

- A continuación, se muestra la tabla que compara las bases de datos relacionales y las bases de datos de grafos.

## RDBMS vs base de datos de grafos

Es importante tener en cuenta que el almacenamiento y el procesamiento de grafos nativo no son ni buenos ni malos, sino que simplemente presentan ciertas ventajas o desventajas dependiendo de la situación.

- La ventaja del almacenamiento nativo de grafos es que la infraestructura de distribución de los datos ha sido construida y diseñada especialmente para un buen rendimiento y una alta escalabilidad en el tratamiento de modelos de grafos.
- El almacenamiento de grafos no nativo generalmente depende de un backend no basado en grafos, con muchos años de experiencia (como MySQL), cuyas características de producción son bien conocidas por los equipos de administración.

El procesamiento de grafos nativo a través de índices beneficia el rendimiento de consulta de los grafos, ya que permite el recorrido de los nodos a través de sus relaciones (consulta por recorrido), pero a expensas de realizar algunas consultas con cierta complejidad o de alto consumo de memoria.

# Neo4j

- Neo4j fue desarrollado por Neo Technology, una empresa sueca con base en Malmö y San Francisco Bay Area en Estados Unidos.
- Es una base de datos orientada a grafos nativa de código abierto, NoSQL, que proporciona un backend transaccional compatible con ACID para sus aplicaciones.
  - El desarrollo inicial comenzó en 2003, pero ha estado disponible públicamente desde 2007. Su primera versión fue lanzada en febrero de 2010.
  - El código fuente, escrito en Java y Scala, está disponible de forma gratuita en GitHub o como una descarga de aplicaciones de escritorio fácil de usar.
- Neo4j tiene una edición de la comunidad y una edición empresarial. La versión Enterprise Edition incluye todo lo que la Community Edition, además de requisitos empresariales adicionales, como copias de seguridad, clústeres y capacidades de conmutación por error.
- Empresas como eBay, Walmart, Telenor, UBS, Cisco, Hewlett-Packard o Lufthansa han confiado en las cualidades de Neo4j para mejorar sus servicios.

- Neo4j se centra más en las relaciones entre valores que en los puntos en común entre conjuntos de valores (como colecciones de documentos o tablas de filas).
  - De esta manera, puede almacenar datos altamente variables de una manera natural y directa.
  - Neo4j es lo suficientemente pequeño como para integrarse en casi cualquier aplicación, pero también puede ejecutarse en grandes clústeres de servidores utilizando replicación maestro-esclavo, y almacenar decenas de miles de millones de nodos. Se considera una base de datos CA.
- En Neo4j, los nodos dentro de los grafos actúan como documentos que almacenan propiedades.

# Ventajas de Neo4j

- **Modelo de datos flexible:** Neo4j proporciona un modelo de datos flexible, simple y potente, que se puede cambiar fácilmente de acuerdo con las aplicaciones y las industrias.
- **Alta disponibilidad:** Neo4j es altamente disponible para aplicaciones en tiempo real con capacidades transaccionales.
- **Fácil recuperación:** con Neo4j, se puede representar y recuperar (atravesar / navegar) datos conectados más rápido en comparación con otras bases de datos.
- **Lenguaje de consulta Cypher** – Neo4j proporciona un lenguaje de consulta declarativo para representar el grafo visualmente. Los comandos de este idioma están en formato legible para humanos, similar al SQL.
- **Sin uniones:** con Neo4j, no se requieren uniones complejas para recuperar datos conectados / relacionados.



# Características de Neo4j

- **Escalabilidad y confiabilidad:** se puede escalar la base de datos aumentando el número de lecturas/escrituras y el volumen sin afectar la velocidad de procesamiento de consultas y la integridad de los datos. Neo4j también proporciona soporte para replicación para la seguridad y confiabilidad de los datos.
- **Aplicación web incorporada:** Neo4j proporciona una aplicación web *Neo4j Browser* incorporada. Con esto, se pueden crear y consultar los datos del grafo.
- **Controladores** – Neo4j puede trabajar con
  - API REST para trabajar con lenguajes de programación como Java, Spring, Scala, etc.
  - Java Script para trabajar con marcos de UI MVC como Node JS.
  - Es compatible con dos tipos de API de Java: API de Cypher y API de Java nativa para desarrollar aplicaciones Java. Además de estos, también puede trabajar con otras bases de datos como MongoDB, Cassandra, etc.
- **Indexación:** Neo4j admite índices mediante Apache Lucene.

# Neo4j - Bloques de construcción

- Neo4j tiene los siguientes bloques de construcción:
  - Nodos
  - Propiedades
  - Relaciones
  - Etiquetas
  - Navegador de datos
- **Nodo**
  - El nodo es una unidad fundamental de un grafo. Contiene propiedades con pares clave-valor como se muestra en la siguiente imagen.
  - Aquí, el nombre de nodo "*Empleado*" contiene un conjunto de propiedades como pares clave-valor.

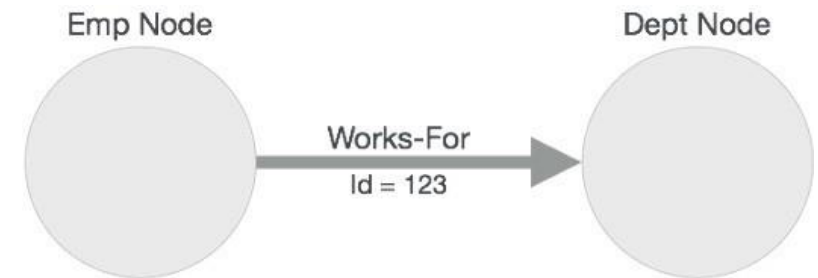


- Propiedades
  - La propiedad es un *par clave-valor* para describir los nodos y las relaciones del grafo.
- Clave = Valor
  - La **Clave** es una cadena y **Valor** se puede representar utilizando cualquier tipo de datos Neo4j.
- Relaciones
  - Conecta dos nodos como se muestra en la siguiente figura:



- Aquí, **Emp** y **Dept** son dos nodos diferentes. "**WORKS\_FOR**" es una relación entre los nodos.
- Como denota, la marca de flecha de *Emp* a *Dept*, esta relación describe:
  - *Emp WORKS\_FOR Dept*
  - Cada relación contiene un nodo de inicio y un nodo de extremo.
    - Aquí, "*Emp*" es un nodo de inicio, y "*Dept*" es un nodo final.

- La marca de flecha de una relación representa el sentido del nodo "*Emp*" al nodo "*Dept*"; "relación entrante" al nodo "*Dept*" y "relación saliente" del nodo "*Emp*".
- Al igual que los nodos, las relaciones también pueden contener propiedades como pares *clave-valor*.
- Aquí, la relación "*WORKS\_FOR*" tiene una propiedad como par clave-valor, *Id = 123*



# Etiquetas

Una etiqueta (Label) asocia un nombre común a un conjunto de nodos o relaciones.

- Un nodo o relación puede contener una o más etiquetas.
- Se denota con el nombre de la etiqueta con el símbolo `:` posterior al nombre del nodo o de la relación

`nodo:etiqueta1:etiqueta2:...`


`[relacion:etiqueta1:etiqueta2:...]`

- Se pueden crear nuevas etiquetas para nodos o relaciones existentes.
- Se pueden eliminar las etiquetas de los nodos o relaciones existentes.

Neo4j almacena datos en propiedades de nodos o relaciones.

# Descarga de Neo4j

<https://neo4j.com/download-center/#community>

ProductsSolutionsLearnDevelopersData ScientistsPricingContact UsGet Started Free

## Current Releases

On this page

- [Current Releases](#)
- [Neo4j Helm Charts](#)
- [Neo4j Ops Manager](#)
- [Pre-Releases](#)
- [Cypher Shell](#)
- [Neo4j Graph Data Science](#)
- [Neo4j Bloom](#)
- [Neo4j Integrations and Connectors](#)
- [Official Neo4j Drivers](#)
- [Community-Contributed Drivers](#)
- [Neo4j Add-Ons](#)
- [PGP Key](#)

Enterprise Server	Community Server	Neo4j Desktop
<b>Neo4j Community Edition 5.7.0</b> 20 April 2023 <a href="#">Release Notes</a>   <a href="#">Read More</a>		
Operating System / Platform	Link	
Red Hat Linux (rpm) Package	<a href="#">Neo4j 5.7.0 (rpm)</a> SHA-256	
Debian / Ubuntu (deb) Package	<a href="#">Neo4j 5.7.0 (deb)</a> SHA-256	
Windows Executable	<a href="#">Neo4j 5.7.0 (zip)</a> SHA-256	
Linux / Mac Executable	<a href="#">Neo4j 5.7.0 (tar)</a> SHA-256	

# Configuración

- Una vez descargado el archivo *neo4j-community-5.7.0-windows.zip*, se descomprime en alguna de las carpetas del SO (se recomienda crear alguna representativa).
- Es necesario para la ejecución del sistema contar con una instalación de *Java* (versión 17+), así como tener instalado el software *PowerShell* de Microsoft.
- Se abre una terminal del SO y se ejecutan las siguientes instrucciones (debe estar localizado en la carpeta *c:[ruta\_instalación]\neo4j-community-5.2.0\bin*)
- Posteriormente ejecutar el comando *> neo4j console* para levantar el servidor

```
C:\WINDOWS\system32\cmd.exe - neo4j console
Microsoft Windows [Versión 10.0.19042.1889]
(c) Microsoft Corporation. Todos los derechos reservados.

c:\temp\neo4j-community-5.2.0\bin>set JAVA_HOME=c:\temp\jdk-17.0.5

c:\temp\neo4j-community-5.2.0\bin>set CLASSPATH=c:\temp\jdk-17.0.5\lib

c:\temp\neo4j-community-5.2.0\bin>set path=c:\temp\jdk-17.0.5\bin;c:\windows\system32\windowspowershell\v1.0

c:\temp\neo4j-community-5.2.0\bin>neo4j
Usage: neo4j [-hV] [--expand-commands] [--verbose] [COMMAND]
A partial alias for 'neo4j-admin server'. Commands for working with DBMS process from 'neo4j-admin server' category can
be invoked using this command.
  --expand-commands  Allow command expansion in config value evaluation.
  -h, --help          Show this help message and exit.
  -V, --version        Print version information and exit.
  --verbose           Prints additional information.

Commands:
  version            Print version information and exit.
  help              Displays help information about the specified command
  console            Start server in console.
  restart            Restart the server daemon.
  start              Start server as a daemon.
  status             Get the status of the server.
  stop              Stop the server daemon.
  windows-service    Neo4j windows service commands.

Environment variables:
  NEO4J_CONF        Path to directory which contains neo4j.conf.
  NEO4J_DEBUG        Set to anything to enable debug output.
  NEO4J_HOME         Neo4j home directory.
  HEAP_SIZE         Set JVM maximum heap size during command execution. Takes a number and a unit, for example 512m.
  JAVA_OPTS          Used to pass custom setting to Java Virtual Machine executing the command. Refer to JVM documentation
about the exact format. This variable is incompatible with HEAP_SIZE and takes precedence over HEAP_SIZE.

c:\temp\neo4j-community-5.2.0\bin>neo4j console
Directories in use:
home:           C:\temp\neo4j-community-5.2.0
config:         C:\temp\neo4j-community-5.2.0\conf
logs:           C:\temp\neo4j-community-5.2.0\logs
plugins:        C:\temp\neo4j-community-5.2.0\plugins
import:         C:\temp\neo4j-community-5.2.0\import
data:           C:\temp\neo4j-community-5.2.0\data
certificates:   C:\temp\neo4j-community-5.2.0\certificates
licenses:       C:\temp\neo4j-community-5.2.0\licenses
run:            C:\temp\neo4j-community-5.2.0\run
Starting Neo4j.
2023-05-09 21:46:34.923+0000 WARN  Use of deprecated setting 'dbms.unmanaged_extension_classes'. It is replaced by 'serv
er.unmanaged_extension_classes'.
2023-05-09 21:46:34.992+0000 INFO  Starting...
2023-05-09 21:46:37.721+0000 INFO  This instance is ServerId{55bc5ffe} (55bc5ffe-4fb0-461f-aca5-904ae310f031)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
2023-05-09 21:46:41.138+0000 INFO  ===== Neo4j 5.2.0 =====
2023-05-09 21:46:53.744+0000 INFO  Bolt enabled on localhost:7687.
2023-05-09 21:46:53.850+0000 INFO  Mounted unmanaged extension [n10s.endpoint] at [/rdf]
2023-05-09 21:46:56.559+0000 INFO  Remote interface available at http://localhost:7474/
2023-05-09 21:46:56.565+0000 INFO  id: ADE2D1ACFA833F4539FE593A0DF4AAAF43D60386AF28E3302FA5BE4B57E162D88
2023-05-09 21:46:56.565+0000 INFO  name: system
2023-05-09 21:46:56.566+0000 INFO  creationDate: 2022-12-12T22:19:54.733Z
2023-05-09 21:46:56.567+0000 INFO  Started.
```

- Finalmente abrir un navegador web y escribir la dirección URL <http://localhost:7474> para interactuar con el sistema

The screenshot displays the Neo4j Browser interface. On the left, a sidebar contains 'Database Information' with sections for 'Use database' (showing 'neo4j'), 'Node labels' (listing categories like Customer, Product, Supplier), 'Relationship types' (listing types like ORDERS, PURCHASED), and 'Property keys' (listing various attributes like address, cargo, etc.). The main area shows a graph visualization of the 'neo4j' database. The graph contains nodes representing people (e.g., Paul Henriot, Karin Josep..., Carlos Diaz) and products (e.g., White Clover Mar..., Vins et alcools Che, Beate Vileid). Relationships are labeled with 'PURCHASED' and 'ORDERS'. A query bar at the top of the graph area contains the Cypher query: `neo4j$ match (n) return n`. On the right, an 'Overview' panel shows statistics: 300 nodes, 504 relationships, and a breakdown of node labels and relationship types. At the bottom, there are three informational cards: 'Getting started with Neo4j Browser', 'Try Neo4j with live data', and 'Cypher basics'.



# Neo4j CQL - Introducción

- CQL son las siglas de Cypher Query Language.
- Es un lenguaje declarativo de consulta para Neo4j.
- Sigue al SQL como sintaxis.
- La sintaxis es muy simple y en formato legible por humanos.
- Tiene comandos para realizar operaciones de base de datos.
- Soporta muchas cláusulas como WHERE, ORDER BY, etc., para escribir consultas muy complejas de una manera fácil.
- Soporta algunas funciones de tratamiento de datos

Cláusula de escritura	Uso
CREATE	Crea nodos, así como relaciones y propiedades de cada nodo.
MERGE	Verifica si el patrón especificado existe en el grafo. Si no, crea el patrón.
SET	Actualiza etiquetas en nodos, propiedades en nodos y relaciones.
DELETE	Elimina nodos y relaciones entre nodos.
REMOVE	Elimina propiedades y propiedades de nodos y relaciones.
FOREACH	Actualiza los datos de una lista.
CREATE UNIQUE	Similar al CREATE, pero evitando duplicados

- Las siguientes son las cláusulas de escritura de Neo4j CQL

# Neo4j CQL

## - Creación de nodos

Un nodo es un dato/registro en una base de datos de grafos.

Se utiliza la cláusula CREATE para:

- Crear un solo nodo
- Crear varios nodos
- Crear un nodo con propiedades
- Crear un nodo con una etiqueta
- Crear un nodo con varias etiquetas

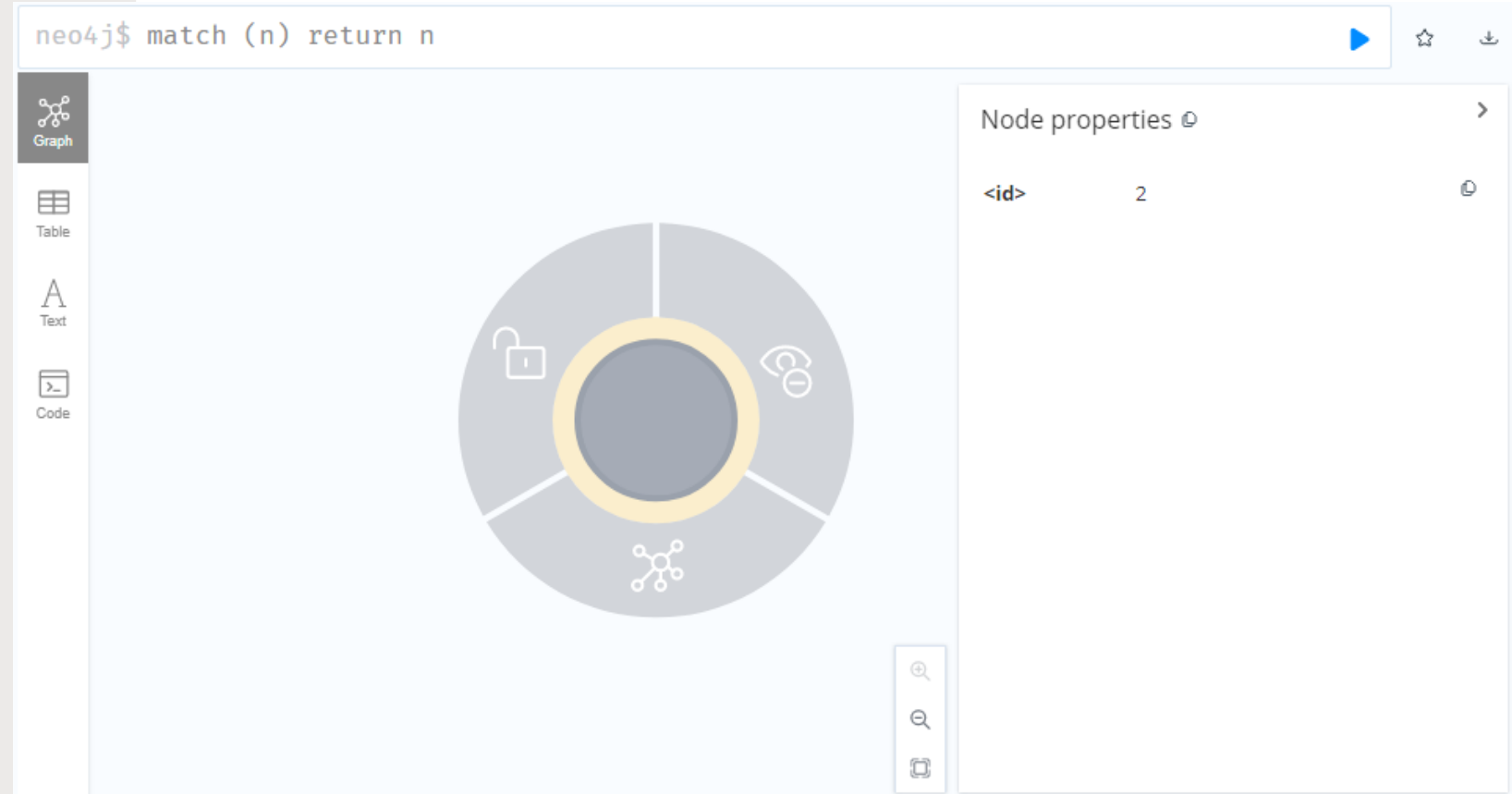
# Creación de un solo nodo

- Se puede crear un nodo especificando el nombre del nodo que se va a crear dentro de la cláusula **CREATE**.

**CREATE** (node\_name)

\$ **CREATE** (a11)

- Los paréntesis de la sintaxis dan idea de lo que representa un nodo (un círculo)



- Las propiedades son los pares clave-valor con los que un nodo almacena datos. Se deben especificar estas propiedades separadas por comas dentro de llaves {}.

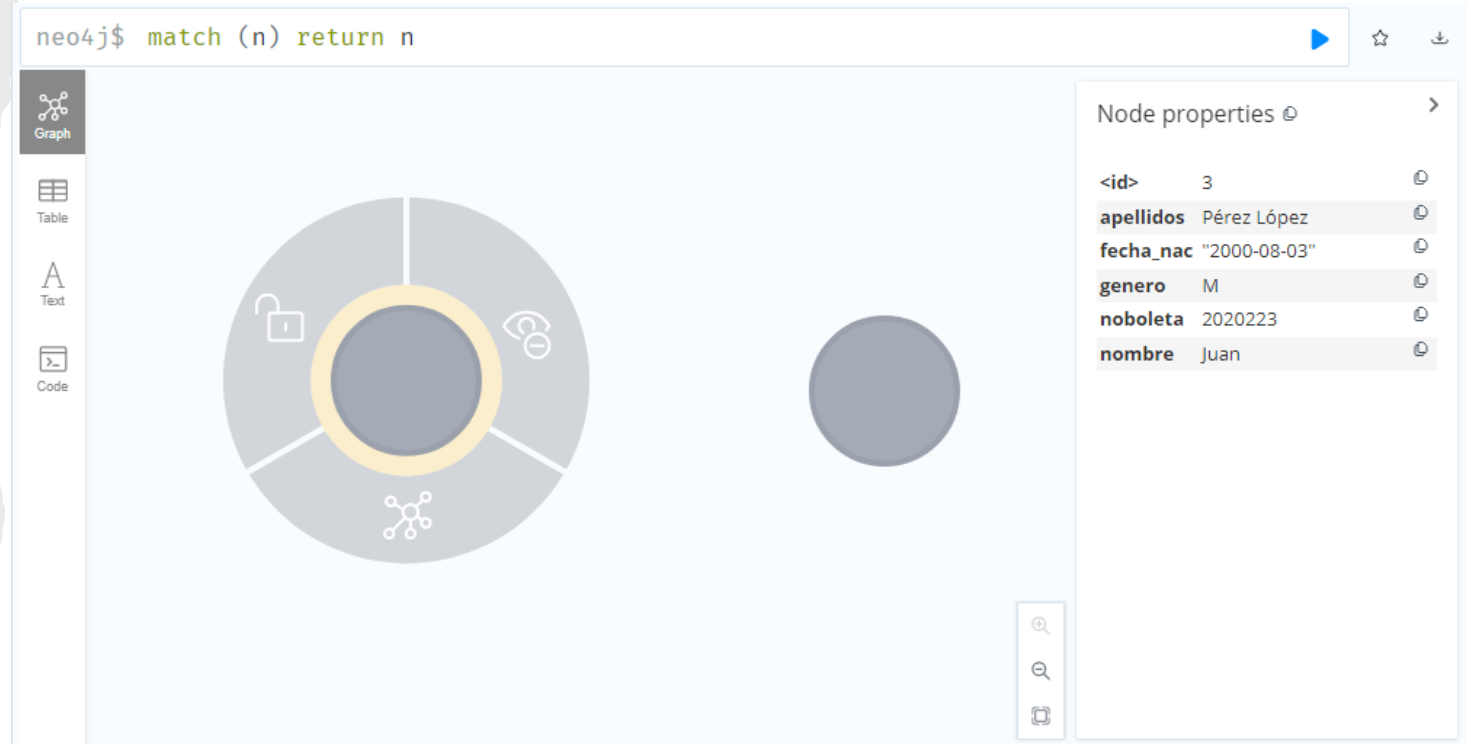
```
CREATE (node {key1: value,  
key2: value,... })
```

```
CREATE (a12{noboleta:2020223,  
nombre:'Juan', apellidos:'Pérez López',  
genero:'M', fecha_nac:date('2000-08-03')})
```

- Para comprobar la creación del nodo, se ejecuta la siguiente consulta en el símbolo del sistema (símbolo **neo4j\$**).

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

- Esta consulta devuelve todos los nodos de la base de datos.



# Etiquetas (Label)

---

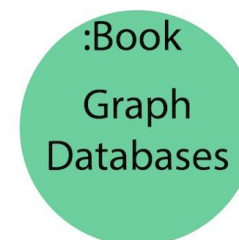
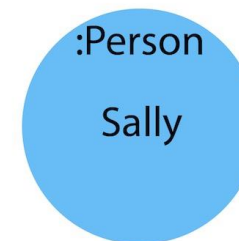
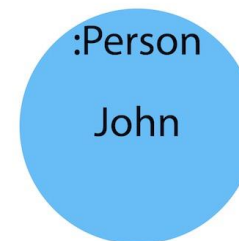
- Las etiquetas son un tipo de nomenclatura que se puede aplicar a cualquier nodo del gráfico. Son solo un nombre, por lo que las etiquetas están presentes o ausentes
- Se indican con el símbolo `:` después del nombre del nodo y antes de la definición de atributos
- Las etiquetas se utilizan para dar forma al dominio agrupando los nodos en conjuntos donde todos los nodos que tienen una determinada etiqueta pertenecen al mismo conjunto.

```
CREATE (per1:Person{name:'John'})
```

```
CREATE (per2:Person{name:'Sally'})
```

```
CREATE (book1:Book{title:'Graph Databases'})
```

- Una vez que se aplican etiquetas, se muestran con coloración automática



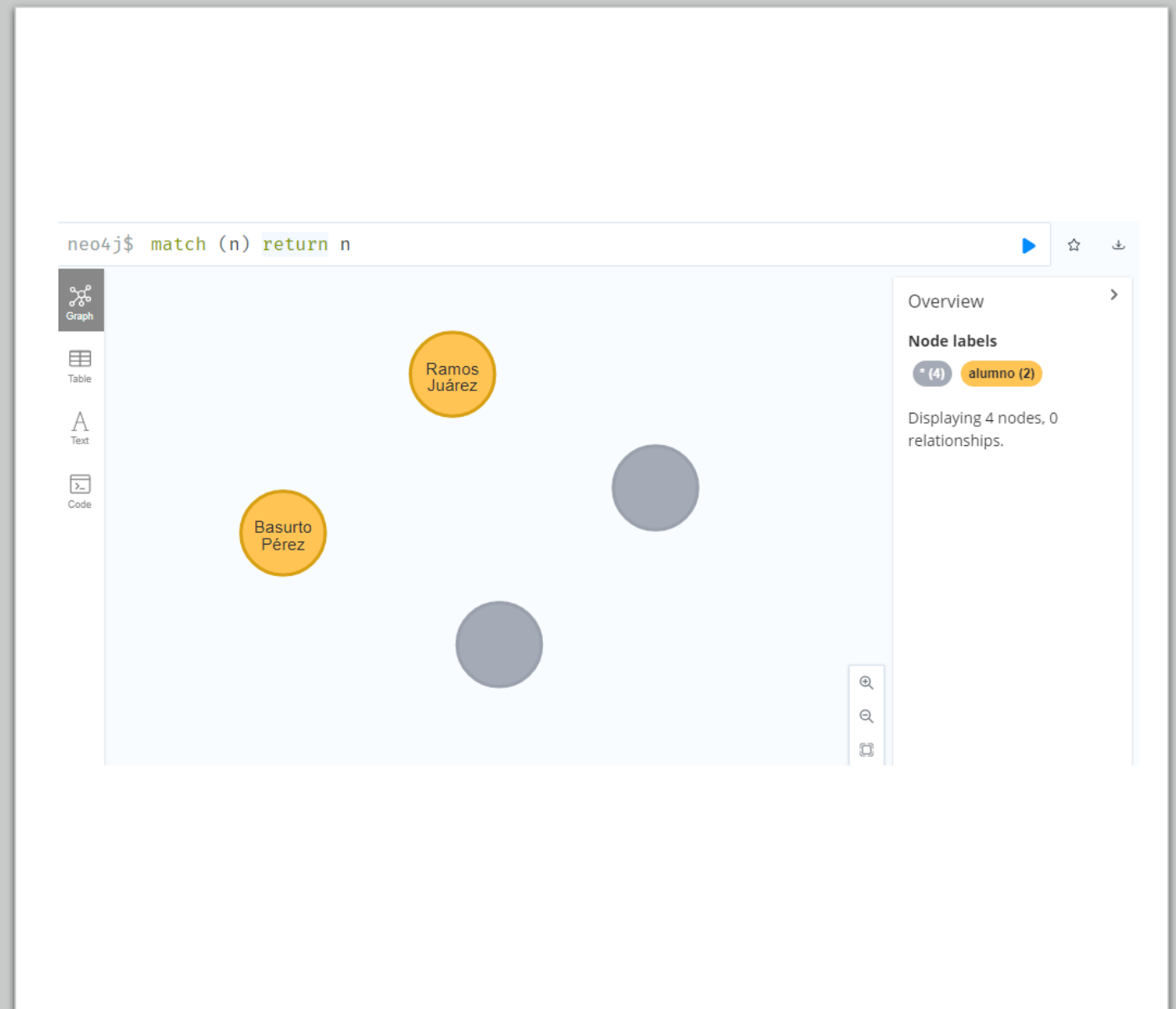
# Creación de varios nodos

- La cláusula CREATE también se utiliza para crear varios nodos al mismo tiempo. Se deben pasar los nodos que se crearán, separados por comas.

```
CREATE (nodo1), (nodo2),  
...
```

- Ejemplo:

```
$ CREATE (al3:alumno{noboleta:  
:2023456, nombre:'Martha', ap  
ellidos:'Ramos  
Juárez', género:'F'}),  
(al4:alumno{noboleta:2021678,  
nombre:'María', apellidos:'Bas  
urto Pérez', género:'F', fecha_  
nac:date('2020-03-15')})
```

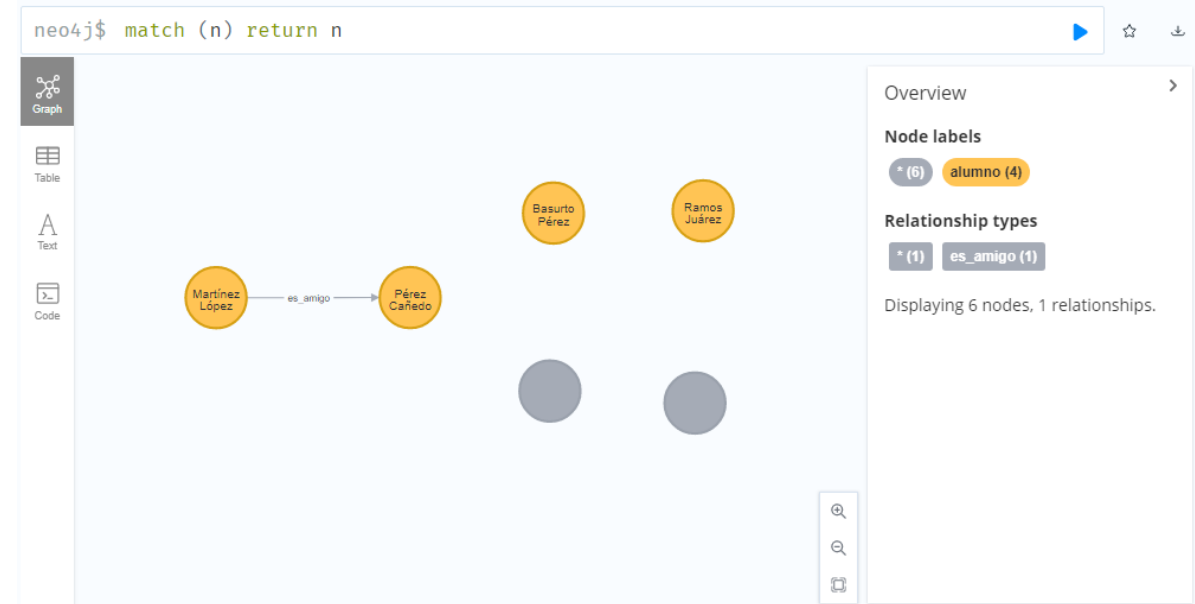


# Neo4j CQL - Creación de relaciones

- Las relaciones tienen patrones de sentido, tipo y formato de datos. Utilizando la cláusula CREATE podemos:
  - Crear relaciones
  - Crear una relación entre los nodos existentes
  - Crear una relación con etiqueta y propiedades
- Creación de relaciones con nodos no existentes
  - Las relaciones se especifican dentro de corchetes "[ ]" y dependiendo del sentido, se coloca entre los símbolos - y -> para simular una flecha dirigida

**CREATE** (node1) -[label:relation]-> (node2)

```
CREATE (a15:alumno{noboleta:2020987,nombre:'Isela',apellidos:'Martínez López',genero:'F',fecha_nac:date('2020-09-19')})-[amistad:es_amigo]->(a16:alumno{noboleta:20198765,nombre:'Martín',apellidos:'Pérez Cañedo',genero:'F'})
```

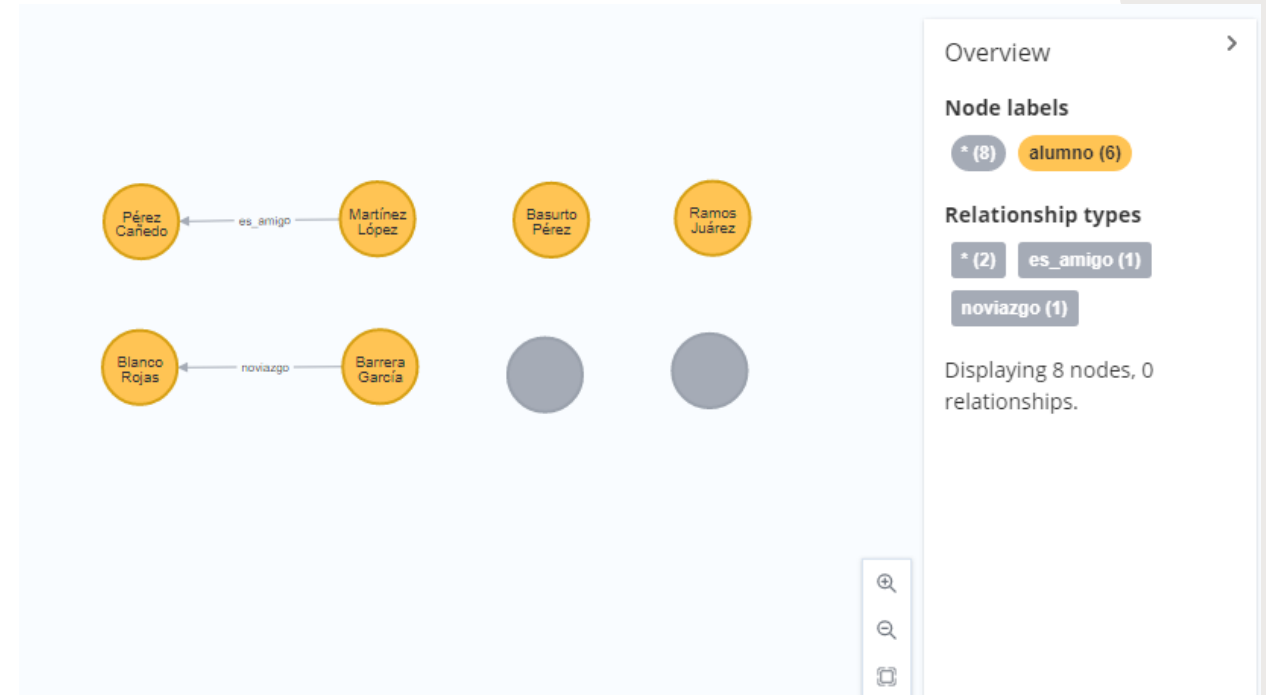




# Creación de una relación con etiqueta y propiedades

- Se puede crear una relación con etiquetas y propiedades, mediante la cláusula CREATE.

```
CREATE (node1)-[label:relation  
{prop1:value1, prop2:value2, . . .  
}]-> (node2)
```

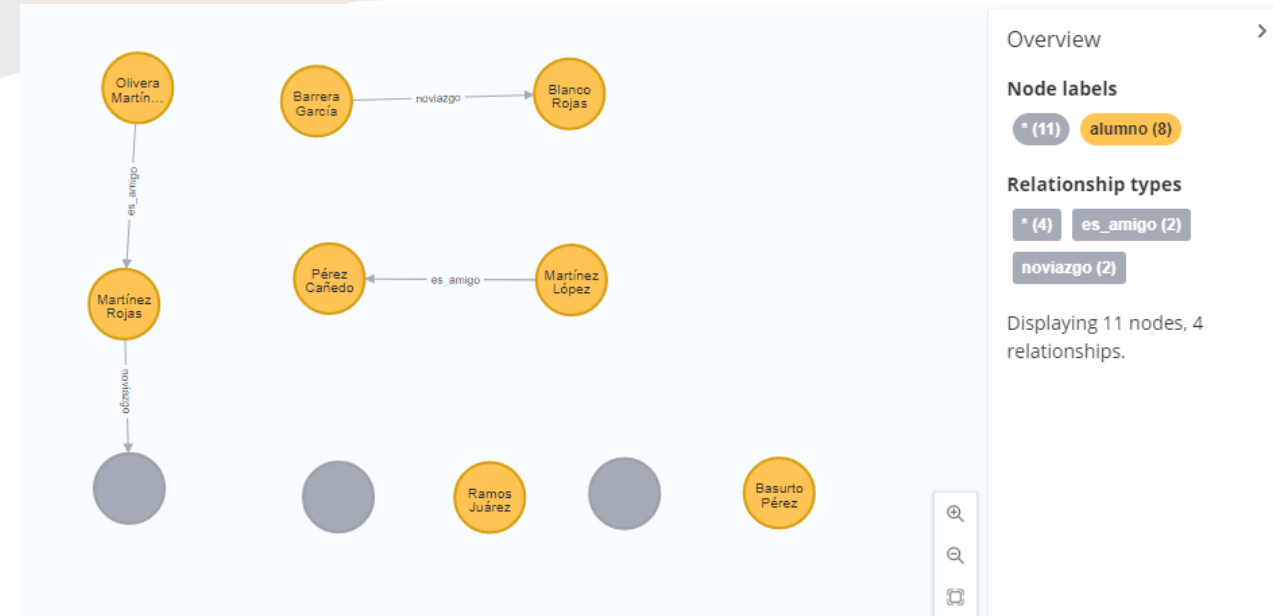


```
CREATE (a17:alumno{noboleta:1999001,nombre:'Guadalupe',apellidos:'Barrera  
García',genero:'F',fecha_nac:date('1997-03-28')})-  
[novia:noviazgo{inicio:date('2021-09-01'),lugar:'parque México'}]-  
>(a18:alumno{noboleta:2001657,nombre:'José',apellidos:'Blanco Rojas',gene  
ro:'M'})
```

# Creación de una ruta completa

- En Neo4j, se forma una ruta utilizando relaciones continuas. Se puede crear una ruta de acceso mediante la cláusula CREATE.

```
CREATE (node1 {props}) -  
[:relation1]->  
    (node2 {props}) -  
[:relation2]->(node3 {props})  
...
```



```
CREATE al9:alumno{noboleta:2000468, nombre:'Ruben', apellidos:  
'Olivera Martínez', genero:'M'}) -  
[am2:es_amigo{inicio:date('2015-06-07')}] -  
>(al10:alumno{noboleta:2000259, nombre:'José', apellidos:'Mart  
ínez Rojas', genero:'M'}) -[novio:noviazgo] -  
>(al11:alumno{noboleta:1999002, nombre:'Martha', apellidos:'Ram  
írez López', genero:'F'})
```

# Cláusulas de consulta en Neo4j CQL

cláusulas	Uso
<b>MATCH</b>	Esta cláusula se utiliza para buscar los datos con un patrón especificado.
<b>OPTIONAL MATCH</b>	Esto es lo mismo que MATCH, la única diferencia es que puede usar nulos en caso de que falten partes del patrón (similar a la operación de reunión externa en SQL)
<b>WHERE</b>	Este identificador de cláusula se utiliza para agregar contenido a las consultas CQL.

- Las instrucciones utilizadas para consultar gráficos Neo4j en Cypher suelen tener un aspecto similar al siguiente:

```
$ MATCH [nodos y/o relaciones]  
WHERE [condiciones a cumplir]  
RETURN [conjunto de nodos y/o relaciones de  
resultado]
```

# Operadores CQL

- Lista de operadores compatibles con Neo4j CQL

Tipo	Operadores
Matemático	+, -, *, /, %, ^
Comparación	+, <>, <, >, <=, >=
Booleano	AND, OR, XOR, NOT
Concatenación de String	+
Lista	+, IN, [X], [X..... Y]
Expresión regular	=-

# Tabla de verdad de operadores AND, OR, XOR y NOT

a	b	a AND b	a OR b	a XOR b	NOT a
false	false	false	false	false	true
false	null	false	null	null	true
false	true	false	true	true	true
true	false	false	true	true	false
true	null	null	true	null	false
true	true	true	true	false	false
null	false	false	null	null	null
null	null	null	null	null	null
null	true	null	true	null	null

# Creación de una relación entre nodos existentes

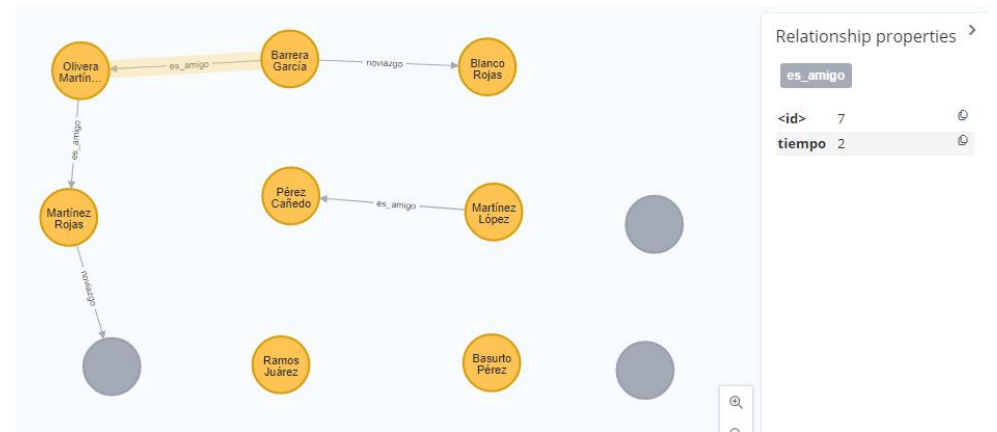
- También se pueden crear relaciones entre nodos existentes mediante la cláusula MATCH.

```
MATCH (a:label), (b:label)
WHERE a.name = "value" AND b.name =
"value"
CREATE (a)-[:relation]->(b)
RETURN a,b
```

- Por ejemplo:

```
MATCH (x:alumno), (y:alumno)
WHERE x.noboleta = 1999001 AND y.nobole
ta = 2000468

CREATE (x)-[am1:es_amigo{tiempo: 2}]-
> (y)
RETURN x, y
```



- Ejemplo

```
MATCH (x:alumno), (y:alumno)
WHERE x.noboleta IS NOT NULL
AND y.noboleta IS NOT NULL
AND x.noboleta <> y.noboleta

CREATE (x) -[:es_compañero]-
> (y)

RETURN x
```

- Para obtener todos los nodos con los cuales esté relacionado:

```
MATCH (x:etiqueta{propiedades}
) --> (n) RETURN n
```

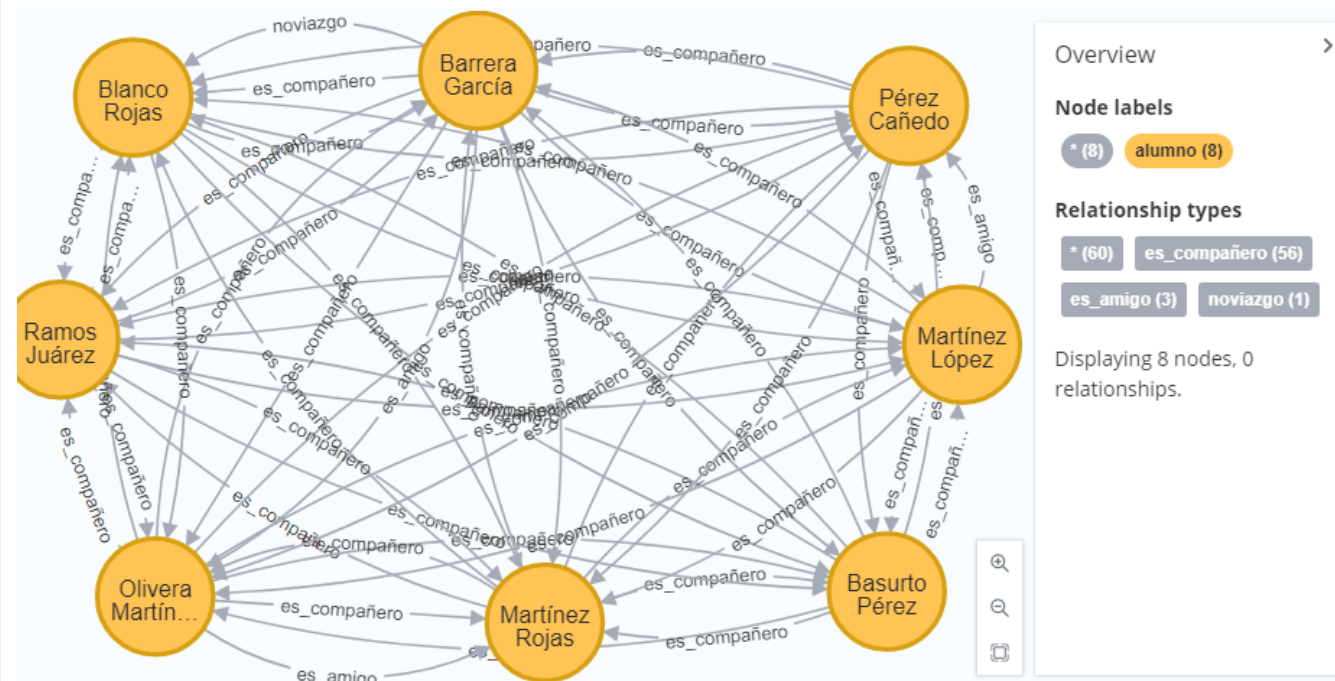
- Ejemplo

```
MATCH (a:alumno{nombre:'José'})
--> (x)

RETURN x
```

- Para obtener las distintas relaciones

```
MATCH () -[r]- () RETURN DISTINCT
TYPE (r)
```





- Cláusulas generales de Neo4j **CQL**

Cláusulas generales	Uso
RETURN	Se utiliza para definir qué incluir en el conjunto de resultados de la consulta.
ORDER BY	Se utiliza para organizar la salida de una consulta en orden. Se utiliza junto con las cláusulas <b>RETURN</b> o <b>WITH</b> .
LIMIT	Se utiliza para limitar las filas del resultado a un valor específico.
SKIP	Se utiliza para definir desde qué fila comenzar, incluidas las filas en la salida.
WITH	Se utiliza para encadenar las partes de consulta.
UNION	Se utiliza para combinar el resultado de dos consultas.

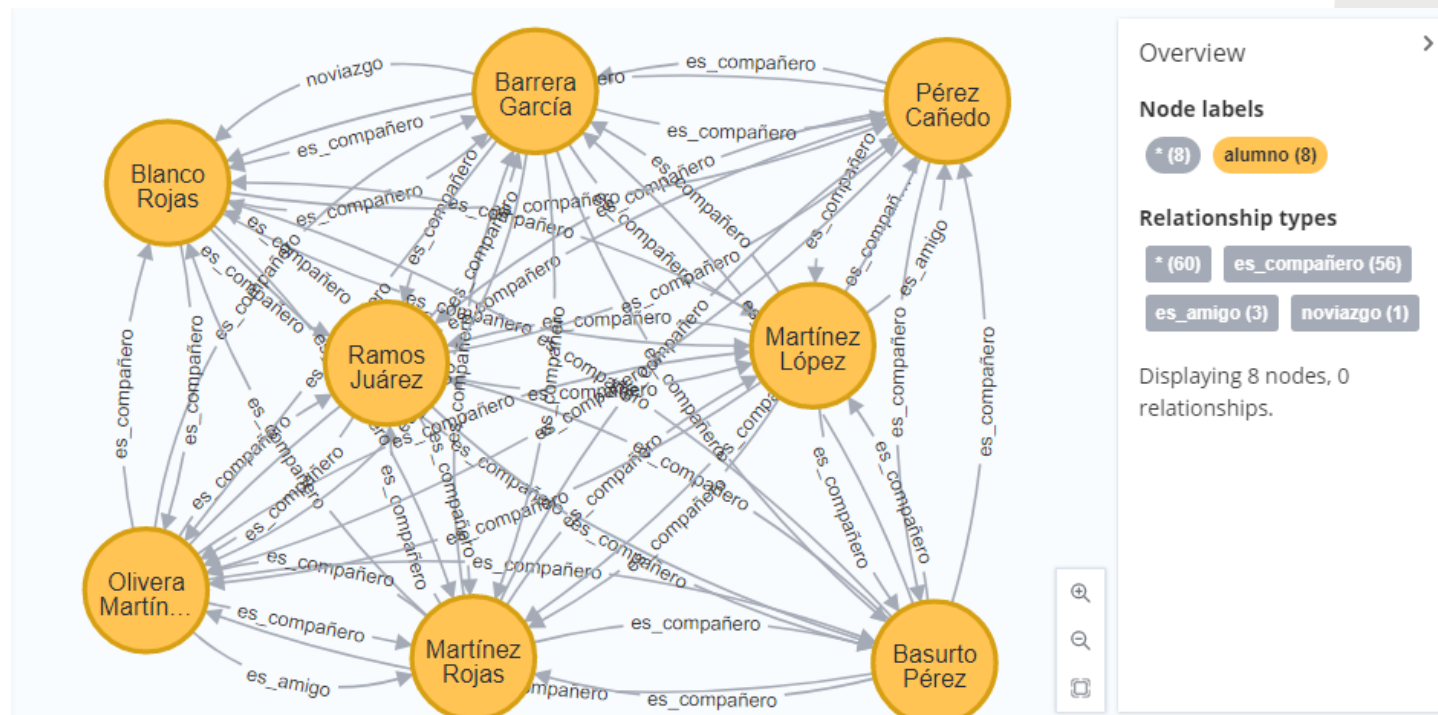
# Obtener todos los nodos bajo una etiqueta específica

- Usando la cláusula MATCH, se pueden obtener todos los nodos bajo una etiqueta específica.

```
MATCH (nodo:etiqueta)  
RETURN nodo
```

- Ejemplo

```
MATCH (a:alumno) RETURN a
```



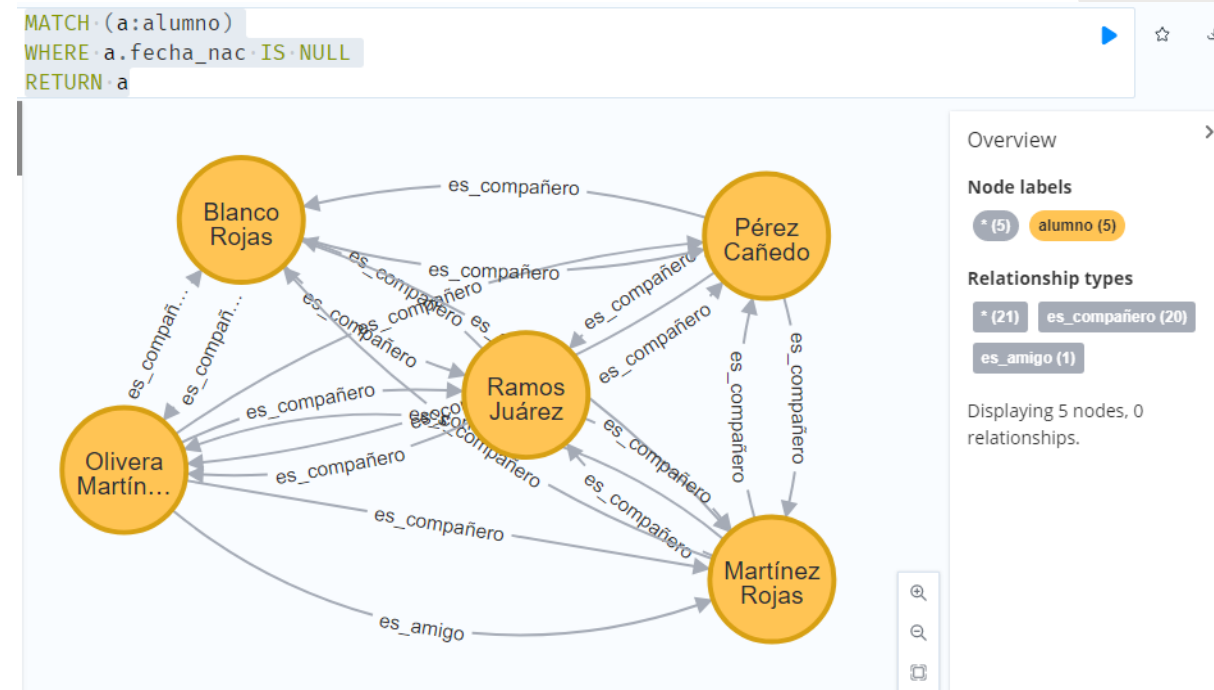
# Cláusula WHERE

- Al igual que en SQL, Neo4j CQL ha proporcionado la cláusula WHERE en el comando MATCH para filtrar los resultados de una consulta.
- A continuación, se encuentra la sintaxis de la cláusula WHERE.

```
MATCH (etiqueta)
WHERE etiqueta.propiedad = "valor"
RETURN etiqueta
```

## Ejemplo

```
MATCH (a:alumno)
WHERE a.fecha_nac IS NULL
RETURN a
```



# Cláusula WHERE con múltiples condiciones

- La cláusula WHERE puede verificar varias condiciones con los conectores AND y OR

```
MATCH (n:etiqueta)
WHERE n.propiedad1 = 'Abc'
AND/OR n.propiedad2 = 'Xyz'
RETURN n
```

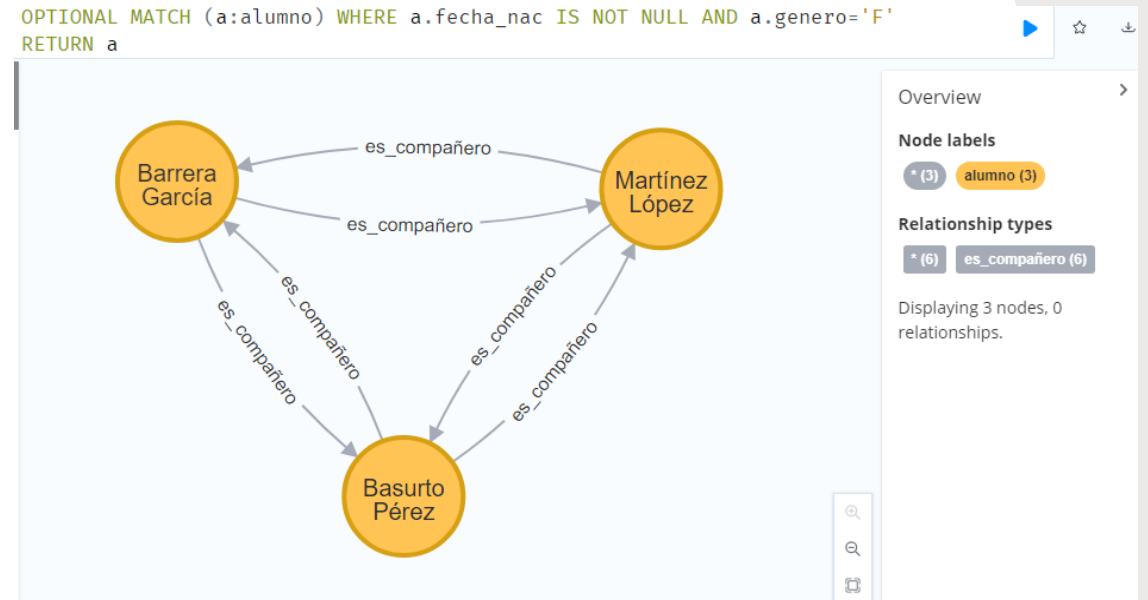
## Ejemplo

```
MATCH (a:alumno) WHERE a.fecha_nac IS NOT
NULL OR a.genero='F'
```

```
RETURN a
```

```
OPTIONAL MATCH (a:alumno) WHERE a.fecha_nac IS
NOT NULL AND a.genero='F'
```

```
RETURN a
```



# Consultas basadas en RegEx

- Para aplicar operaciones de búsqueda de patrones empleando expresiones regulares (RegEx), se emplea la siguiente sintaxis:

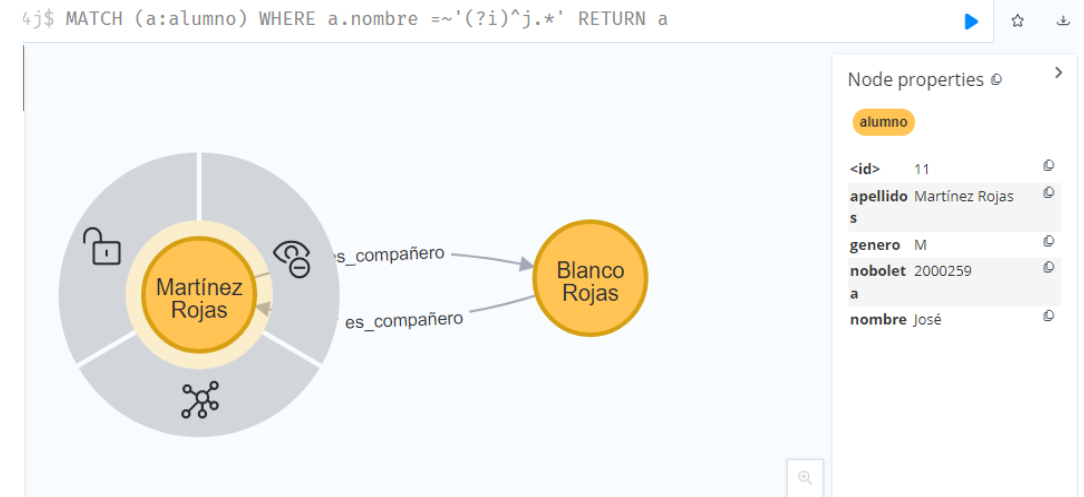
```
MATCH (n:etiqueta)
WHERE n.propiedad =~ 'RegEx'
RETURN n
```

- En donde el símbolo `~` indica que se va a emplear la expresión regular dada por **RegEx**

```
MATCH (a:alumno)
WHERE a.nombre =~ '^J.*'
RETURN a
```

- Incluyendo la expresión **(?i)** en el patrón de búsqueda RegEx, indica que la búsqueda es no sensitiva a mayúsculas y minúsculas

```
MATCH (a:alumno)
WHERE a.nombre =~ '(?i)^j.*'
RETURN a
```



# Caracteres predefinidos en RegEx

.	Cualquier carácter
\d	Un dígito: [0-9]
\D	Un no dígito: [^0-9]
\h	Un carácter de espacio en blanco horizontal: [ \t \xA0 \u1680 \u180e \u2000 -\u200a \u202f \u205f \u3000]
\H	Un carácter de no espacio en blanco horizontal: [^\h]
\s	Un carácter de espacio en blanco: [ \t \n \x0B \f \r]
\S	Un carácter de no espacio en blanco: [^\s]
\v	Un carácter de espacio en blanco vertical: [\n \x0B \f \r \x85 \u2028 \u2029]
\V	Un carácter de no espacio en blanco vertical: [^\v]
\w	Un carácter de texto: [a-zA-Z_0-9]
\W	Un carácter de no texto: [^\w]

# Coincidencia de inicio y fin de cadenas

- La búsqueda de un patrón al final de un texto se realiza con el sufijo **ENDS WITH**
- Realiza una búsqueda sensible (mayúsculas y minúsculas) al texto

```
MATCH (n:label) WHERE n.prop ENDS WITH 'pattern' RETURN n
```

- La búsqueda de un patrón al inicio de un texto se realiza con el sufijo **STARTS WITH**

```
MATCH (n:label) WHERE n.prop STARTS WITH 'pattern' RETURN n
```

Ejemplo:

```
MATCH (a:alumno) WHERE a.nombre STARTS WITH 'J' AND  
a.nombre ENDS WITH 'é'
```

```
RETURN a
```

```
MATCH (a:alumno) WHERE a.nombre STARTS WITH 'J' AND a.nombre ENDS WITH 'é'  
RETURN a
```



# Búsqueda mediante CONTAINS

- El operador CONTAINS se usa para realizar una búsqueda sensible sin importar la posición del patrón dentro de un texto.

```
MATCH (n) WHERE n.porp CONTAINS 'pattern'  
RETURN n
```

- Ejemplo:

```
MATCH (a:alumno) WHERE a.nombre CONTAINS 'h'  
RETURN a
```

```
$ MATCH (a:alumno) WHERE a.nombre CONTAINS 'h' RETURN a
```



Overview

Node labels

\* (1) alumno (1)

Displaying 1 nodes, 0 relationships.



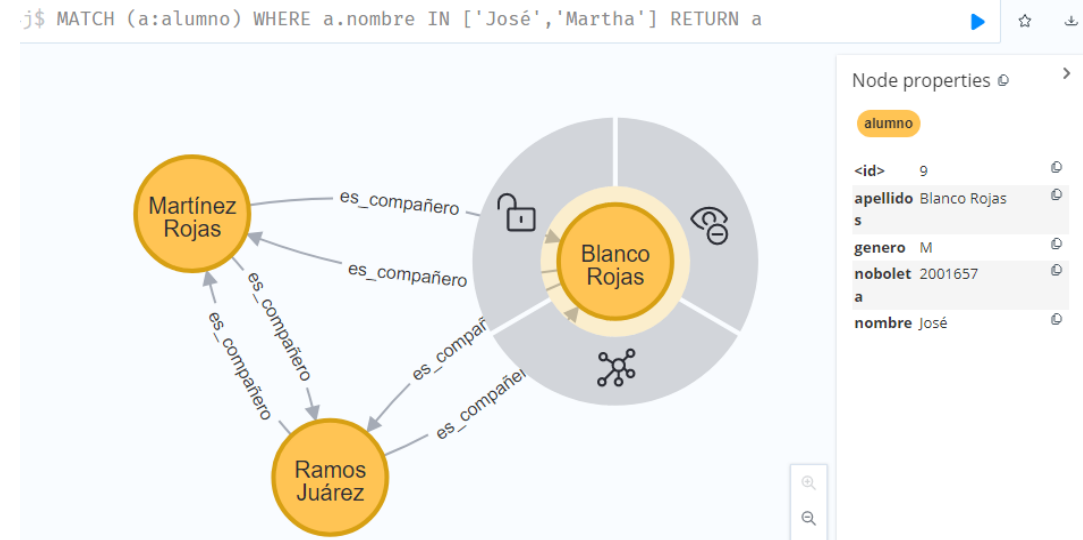
# Operador IN

- Para revisar si un elemento existe en una lista, se usa el operador **IN**:

```
MATCH (a:label) WHERE a.prop  
IN [patter1, pattern2  
,...] RETURN a
```

- Ejemplo:

```
MATCH (a:alumno) WHERE a.nombre  
IN ['José', 'Martha']  
RETURN a
```



# Operador UNION

- El operador UNION combina los resultados de dos o más consultas en un solo resultado que incluye todos los nodos de las consultas (eliminando duplicados)
- El nombre y número de las propiedades debe ser idéntico en todas las consultas empleadas en el operador
- Para obtener todos los nodos (incluyendo duplicados) se debe incluir el término ALL.

```
MATCH (n:label) RETURN n.prop AS name
```

```
UNION [ALL]
```

```
MATCH (m:label) RETURN m.prop AS name
```

- Ejemplo:

```
MATCH (a:alumno) WHERE a.nombre STARTS WITH 'J' RETURN a.nombre as  
datos UNION MATCH (b) WHERE b.noboleta IS NULL RETURN b.nombre as  
datos
```

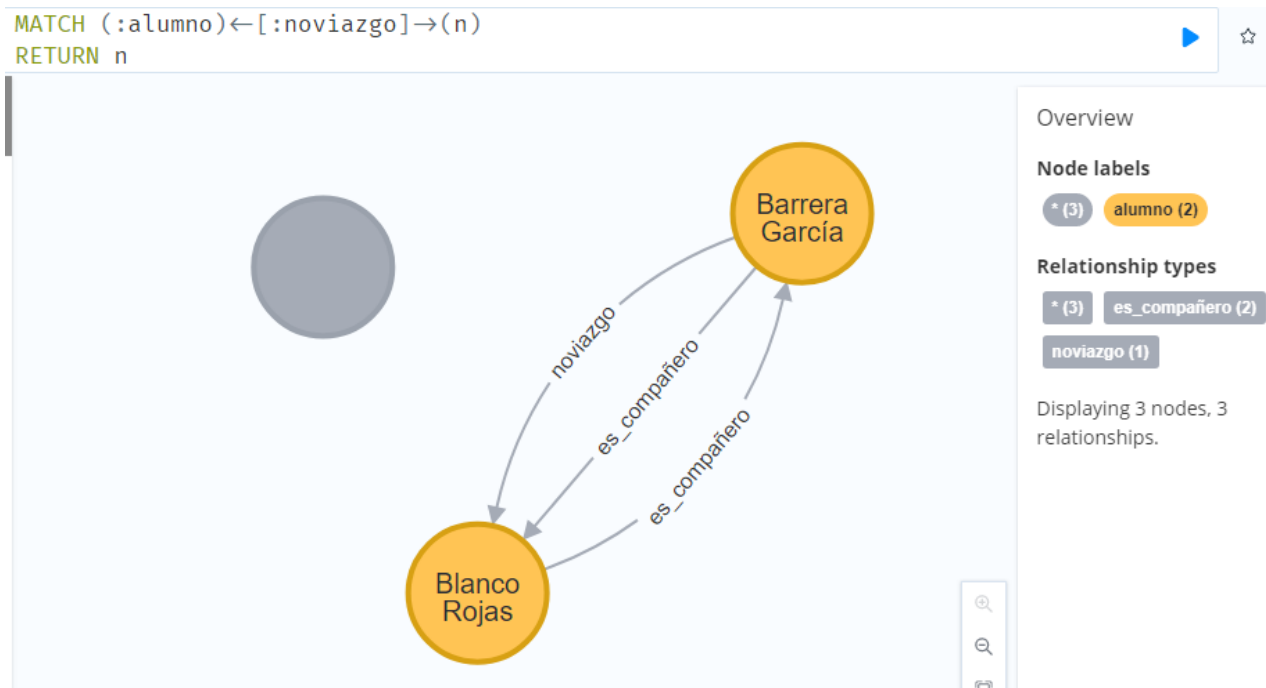
```
4j$ MATCH (a:alumno) WHERE a.nombre STARTS WITH 'J' RETURN a.nombre as datos  
UNION MATCH (b) WHERE b.noboleta IS NULL RETURN b.nombre as datos
```

	datos
1	"José"
2	null

```
4j$ MATCH (a:alumno) WHERE a.nombre STARTS WITH 'J' RETURN a.nombre as datos  
UNION ALL MATCH (b) WHERE b.noboleta IS NULL RETURN b.nombre as datos
```

	datos
1	"José"
2	"José"
3	null

# MATCH por relación



- Se pueden recuperar nodos basados en la relación mediante la cláusula MATCH.

```
MATCH (nodo:etiqueta)←[:relacion]-(n)
RETURN n
```

```
MATCH (nodo:etiqueta)-[:relacion]->(n)
RETURN n
```

```
MATCH (nodo:etiqueta)←[:relacion]->(n)
RETURN n
```

## Ejemplo

```
MATCH (:alumno)←[:noviazgo]-(n)
```

```
RETURN n
```

```
MATCH (:alumno)-[:noviazgo]->(n)
```

```
RETURN n
```

```
MATCH (:alumno)←[:noviazgo]->(n)
```

```
RETURN n
```

# cláusula RETURN

- Se emplea para regresar nodos como salida.

```
CREATE (nodo:etiqueta {propiedades})  
RETURN nodo [, nodo...]
```

- También puede devolver relaciones

```
CREATE (nodo)-[etiqueta:relacion]->(nodo)  
RETURN relacion
```

- También puede devolver propiedades

```
MATCH (nodo:etiqueta {propiedades . . . . . })  
RETURN nodo.propiedad
```

- Se pueden devolver todos los elementos de la base de datos

```
MATCH (nodo:etiqueta {propiedades . . . . . })  
RETURN *
```

- Se puede devolver una propiedad en particular con alias

```
MATCH (nodo:etiqueta {propiedades . . . . . })  
RETURN nodo.propiedad AS Alias
```

## Cláusula ORDER BY

- Puede organizar los datos de resultados en orden utilizando la cláusula ORDER BY.

```
MATCH (n)
RETURN n.propiedad1,
n.propiedad2 . . . . .
ORDER BY n.propiedad [,
n.propiedad...] [DESC]
```

- Ejemplo

```
MATCH (a:alumno)
RETURN a
ORDER BY a.apellidos
```

```
1 MATCH (a:alumno)
2 RETURN a
3 ORDER BY a.apellidos
```



"a"

{"apellidos":"Barrera García","fecha\_nac":"1997-03-28","genero":"F","noboleta":1999001,"nombre":"Guadalupe"}

{"apellidos":"Basurto Pérez","fecha\_nac":"2020-03-15","genero":"F","noboleta":2021678,"nombre":"María"}

{"apellidos":"Blanco Rojas","genero":"M","noboleta":2001657,"nombre":"José"}

{"apellidos":"Martínez López","fecha\_nac":"2020-09-19","genero":"F","noboleta":2020987,"nombre":"Isela"}

{"apellidos":"Martínez Rojas","genero":"M","noboleta":2000259,"nombre":"José"}

{"apellidos":"Olivera Martínez","genero":"M","noboleta":2000468,"nombre":"Ruben"}

{"apellidos":"Pérez Cañedo","genero":"F","noboleta":20198765,"nombre":"Martín"}

# Cláusula **LIMIT**

- La cláusula **LIMIT** se utiliza para limitar el número de nodos en la salida.

```
MATCH (n)
RETURN n
ORDER BY n.propiedad
LIMIT tamaño
```

- Ejemplo

```
MATCH (a:alumno)
RETURN a
ORDER BY a.apellidos
LIMIT 3
```

```
neo4j$ MATCH (a:alumno) RETURN a ORDER BY a.apellidos LIMIT 3
```



"a"

{"apellidos":"Barrera García","fecha\_nac":"1997-03-28","genero":"F","noboleta":1999001,"nombre":"Guadalupe"}

{"apellidos":"Basurto Pérez","fecha\_nac":"2020-03-15","genero":"F","noboleta":2021678,"nombre":"María"}

{"apellidos":"Blanco Rojas","genero":"M","noboleta":2001657,"nombre":"José"}

# Cláusula SKIP

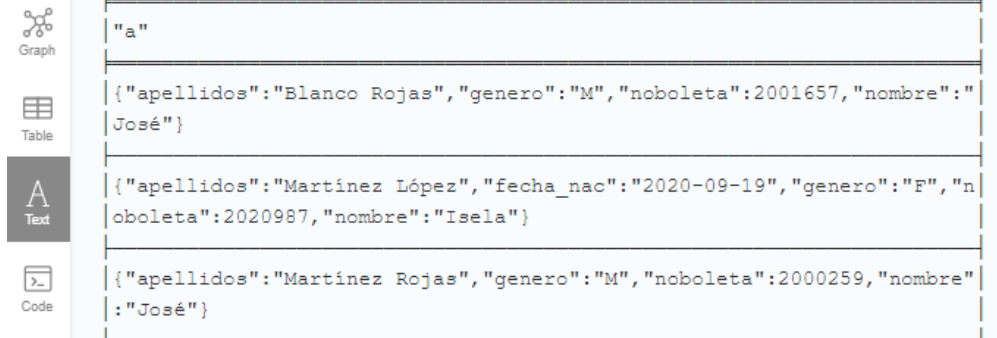
- La cláusula SKIP se utiliza para definir desde qué nodo comenzar, incluyendo los nodos en la salida.

```
MATCH (n)
RETURN n
ORDER BY n.propiedad
SKIP tamaño
```

- Ejemplo

```
MATCH (a:alumno)
RETURN a
ORDER BY a.apellidos
SKIP 2
LIMIT 3
```

```
1 MATCH (a:alumno)
2 RETURN a
3 ORDER BY a.apellidos
4 SKIP 2
5 LIMIT 3
```



"a"
{"apellidos":"Blanco Rojas","genero":"M","noboleta":2001657,"nombre":"José"}
{"apellidos":"Martínez López","fecha_nac":"2020-09-19","genero":"F","noboleta":2020987,"nombre":"Isela"}
{"apellidos":"Martínez Rojas","genero":"M","noboleta":2000259,"nombre":"José"}

# Funciones de agregación

- CQL proporciona algunas funciones de agregación para usar en la cláusula RETURN. Es similar a la cláusula GROUP BY en SQL.
- Se usa el comando RETURN + Aggregation Functions en MATCH para trabajar en un grupo de nodos y devolver algún valor agregado.

Función	Descripción
COUNT	Devuelve el número de nodos devueltos por el comando MATCH.
MAX	Devuelve el valor máximo de un conjunto de nodos devueltos por el comando MATCH.
MIN	Devuelve el valor mínimo de un conjunto de nodos devueltos por el comando MATCH.
SUM	Devuelve el valor de suma de todos los nodos devueltos por el comando MATCH.
AVG	Devuelve el valor medio de todos los nodos devueltos por el comando MATCH.



# COUNT

- La función **count()** se utiliza para contar el número de nodos (también dentro de etiquetas).

```
MATCH (node:label{props...})  
RETURN count(node | :label)
```

- Ejemplo

```
MATCH (a:alumno)  
RETURN COUNT(a)
```

- También se utiliza para contar relaciones


- Ejemplo

```
MATCH ()-[r:es_amigo]-()  
RETURN COUNT(r)
```


- Para contar las distintas relaciones que hay entre nodos

```
MATCH ()-[r]-() RETURN DISTINCT TYPE(r),  
COUNT(r)
```

```
1 MATCH ()-[r:es_amigo]-()  
2 RETURN COUNT(r)
```



Table




Text

"COUNT (r) "
6




Code

```
1 MATCH ()-[r:es_amigo]-()  
2 RETURN COUNT(r)
```



Table




Text

"COUNT (r) "
6




Code


```
neo4j$ MATCH()-[r]-() RETURN DISTINCT TYPE(r), COUNT(r)
```



Table



Text

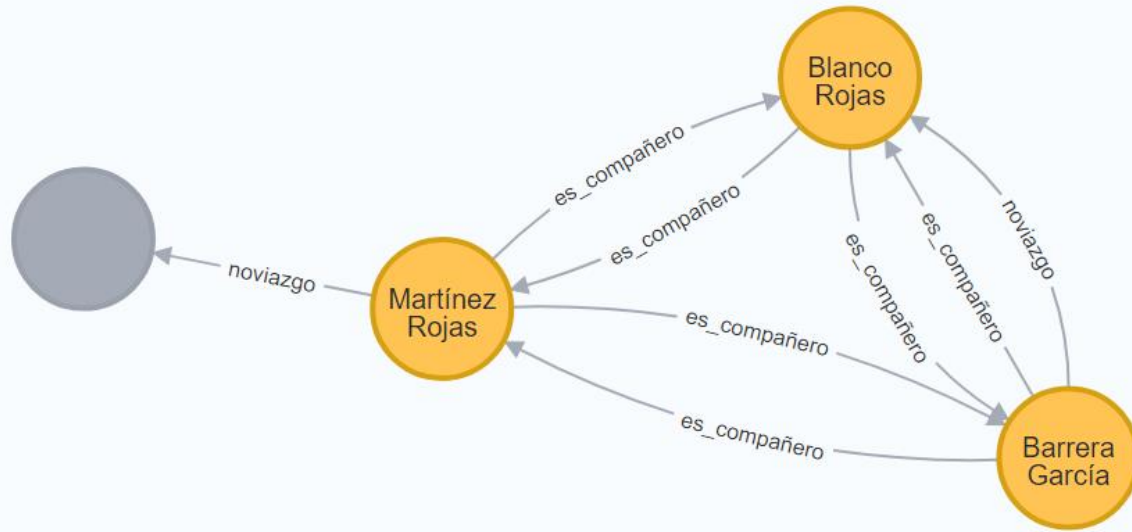


Code

"TYPE (r) "	"COUNT (r) "
"es_amigo"	6
"noviazgo"	4
"es_compañero"	112

# Funciones

```
MATCH p = (a:alumno)-[r:noviazgo]-(b) RETURN HEAD(COLLECT(a)), ENDNODE(r), LENGTH(p), TYPE(r)
```



Función	Descripción
<code>length()</code>	Retorna la longitud de una ruta
<code>endNode()</code>	Retorna el nodo final de una relación
<code>head()</code>	Retorna el primer elemento de una lista
<code>type()</code>	Retorna la representación en texto de un tipo de relación
<code>collect()</code>	Retorna una lista conteniendo los valores regresados por una expresión

"HEAD(COLLECT(a))"	"ENDNODE(r)"	"LENGTH(p)"	"TYPE(r)"
{ "apellidos": "Barrera García", "fecha_nac": "1997-03-28", "genero": "F", "noboleta": "1999001", "nombre": "Guadalupe" }	{ "apellidos": "Blanco Rojas", "genero": "M", "noboleta": "2001657", "nombre": "José" }	1	"noviazgo"
{ "apellidos": "Martínez Rojas", "genero": "M", "noboleta": "2000259", "nombre": "José" }	{ "apellidos": "Ramírez López", "genero": "F", "noboleta": "1999002", "nombre": "Martha" }	1	"noviazgo"

Función	Descripción
<code>toBoolean()</code>	Convierte un texto o entero a un valor booleano
<code>toBooleanOrNull()</code>	Idéntico a <code>toBoolean()</code> . Para cualquier otro valor de entrada, se retornará null.
<code>toFloat()</code>	Convierte un texto o entero a un valor de punto flotante
<code>toFloatOrNull()</code>	Idéntico a <code>toFloat()</code> . Para cualquier otro valor de entrada, se retornará null.
<code>toInteger()</code>	Convierte un texto, valor en punto flotante o booleano a un valor entero.
<code>toIntegerOrNull()</code>	Idéntico a <code>toInteger()</code> . Para cualquier otro valor de entrada, se retornará null.
<code>toString()</code>	Convierte un entero, valor flotante, booleano, <i>date</i> , <i>time</i> o <i>datetime</i> a un valor de texto
<code>toStringOrNull()</code>	Idéntico a <code>toString()</code> . Para cualquier otro valor de entrada, se retornará null.

## Funciones de conversión de tipos de datos

# Funciones en textos

Función	Descripción
<code>left()</code>	Retorna un texto conteniendo el número de caracteres indicado de la izquierda extrema del texto original
<code>ltrim()</code>	Retorna la cadena original con los caracteres de espacio iniciales removidos
<code>replace()</code>	Retorna un texto con todas las ocurrencias de un patrón de búsqueda en el texto original, reemplazando el texto regresado con otro patrón de texto
<code>reverse()</code>	Regresa el texto original en el orden inverso de sus caracteres que lo constituyen
<code>right()</code>	Retorna un texto conteniendo el número de caracteres indicado de la derecha extrema del texto original
<code>rtrim()</code>	Retorna la cadena original con los caracteres de espacio finales removidos
<code>split()</code>	Regresa una lista de textos resultantes de la división del texto original de acuerdo con el(los) patrón(es) delimitador(es) dado(s)
<code>substring()</code>	Regresa un subtexto (con determinada longitud) de un texto original, de acuerdo con la posición de indexado dada
<code>toLowerCase()</code>	Regresa el texto original en minúsculas
<code>toUpperCase()</code>	Regresa el texto original en mayúsculas
<code>trim()</code>	Retorna la cadena original con los caracteres de espacio iniciales y finales removidos.

# Cláusula WITH

- La cláusula WITH permite realizar consultas por partes, encadenando los resultados parciales entre ellas, en el orden en que se definieron
- También puede ser usada para ingresar variables conteniendo el resultado de las expresiones anteriores para ser usando en expresiones posteriores

```
MATCH (n)
WITH n
ORDER BY n.property
RETURN collect(n.property)
```

```
j$ MATCH (a:alumno)-[:noviazgo]-(b) WITH a ORDER BY a.apellidos RETURN COLLECT(a.apellidos)
```

```
"COLLECT(a.apellidos)"
```

```
["Barrera García", "Blanco Rojas", "Martínez Rojas"]
```



- Es posible ordenar los resultados antes de pasarlos a la función `collect()`, y así ordenar la lista resultante.

```
MATCH (n)
WITH n
ORDER BY n.prop DESC
LIMIT i
RETURN collect(n.prop)
```

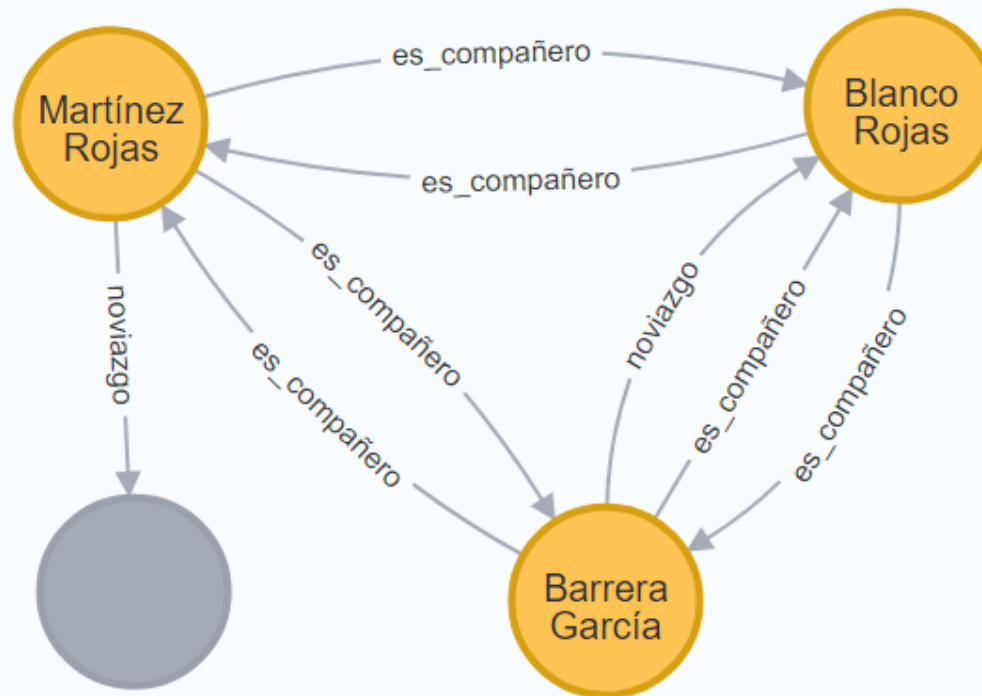
```
ij$ MATCH (a:alumno)-[r:es_amigo]-() WITH a ORDER BY a.apellidos RETURN COLLECT(a.apellidos) AS APELLIDOS
```

	APELLIDOS
1	["Barrera García", "Martínez López", "Martínez Rojas", "Olivera Martínez", "Olivera Martínez", "Pérez Cañedo"]

- Una subconsulta EXISTS puede ser usada si el patrón especificado existe en al menos un documento de datos. Permite incluir las sentencias MATCH y WHERE internamente. También puede aparecer en cualquier posición de la expresión.
- Si la subconsulta evalúa al menos un documento, la expresión completa será evaluada en verdadero

```
MATCH (node:label1)
WHERE EXISTS { (node)-[:labelR]->(:label2) }
RETURN node.prop AS out
```

```
j$ MATCH (a)-[r]-(b) WHERE EXISTS {(a:alumno)-[:noviazgo]->(b)} RETURN a,b,r
```



# Cláusula EXISTS



```

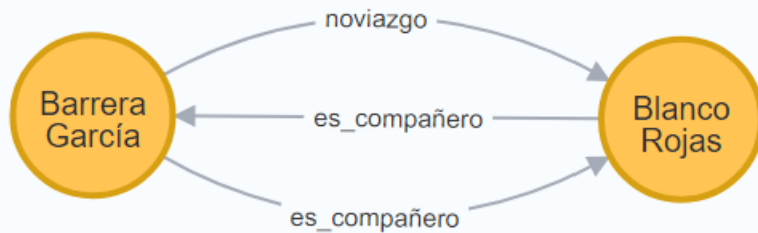
MATCH (node1:label1)
WHERE EXISTS { MATCH (node1)-[:labelR]->(node2:label2) WHERE node1.prop =
node2.prop }
RETURN node1.prop AS out

```

```

cypher$ MATCH (a)-[r]-(b) WHERE EXISTS {MATCH (a)-[r:noviazgo]->(b:alumno) WHERE a.genero<>b.genero} RETURN a,b,r

```



```

MATCH (node1:label1)
WHERE EXISTS { MATCH (node1)-[:labelR]->(node2:label2) WHERE EXISTS { MATCH
(node2)-[:labelS]->(node3:label3) WHERE node3.prop = value } }
RETURN node1.prop AS out

```

```

MATCH (a)-[r]-(b)-[s]-(c) WHERE EXISTS {MATCH (a:alumno)-[r:es_amigo]->(b:alumno) WHERE a.genero=b.genero AND
EXISTS{MATCH(b)-[:noviazgo]-(c) WHERE c.genero='F' }} RETURN a,b,r,c

```



Overview

Node labels

\* (3)

Relationships

\* (4)

noviazgo

# Combinar un nodo con etiqueta

- La cláusula MERGE permite combinar un nodo en función de la etiqueta. Si se intenta combinar un nodo basado en la etiqueta, se verifica si existe algún nodo con la etiqueta dada. De lo contrario, se creará el nodo actual.

```
MERGE (nodo:etiqueta{propiedades . . . })
```

```
RETURN nodo
```

- Ejemplo

```
MERGE (y:materia{id:1,nombre:'Ingeniería de Software',creditos:  
8})
```

```
RETURN y
```

```
MERGE (y:materia{id:2,nombre:'Inteligencia Artificial',creditos  
:10})
```

```
RETURN y
```

# Combinar un nodo con propiedades

- También se puede combinar un nodo con un conjunto de propiedades. Neo4j busca una coincidencia igual para el nodo especificado, incluidas las propiedades. Si no encuentra ninguno, lo crea.

```
MERGE (nodo:etiqueta {clave:valor, clave:valor,  
clave:valor . . .})
```

- Ejemplo:

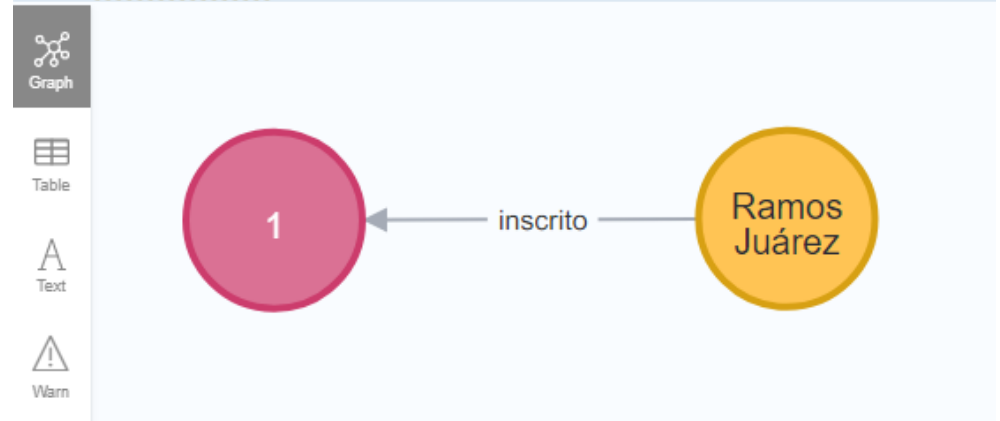
```
MERGE (x:alumno{genero:'F'})  
RETURN x
```

# Combinar una relación

- Al igual que con los nodos, es posible combinar las relaciones utilizando la cláusula MERGE

```
MATCH (a:alumno), (m:materia)
WHERE a.noboleta = 2023456
AND m.id = 1
MERGE (a)-[r:inscrito]->(m)
RETURN a, m
```

```
1 MATCH (a:alumno), (m:materia)
2 WHERE a.noboleta=2023456 AND m.id = 1
3 MERGE (a)-[r:inscrito]->(m)
4 RETURN a,m
```



# Cláusula SET

- Con la cláusula SET, se pueden agregar nuevas propiedades a un nodo o relación existente, y también agregar o actualizar valores de propiedades existentes.

MATCH

(nodo:etiqueta{propiedades...})

SET nodo.propiedad = valor

RETURN nodo

- Ejemplo

```
MATCH (a:alumno)
```

```
WHERE a.fecha_nac IS NULL AND  
a.genero='F'
```

```
SET a.fecha_nac=date('2000-6-  
14')
```

```
RETURN a
```

```
1 MATCH (a:alumno)  
2 WHERE a.fecha_nac IS NULL AND a.genero='F'  
3 SET a.fecha_nac=date('2000-6-14')  
4 RETURN a
```



# Establecer varias propiedades

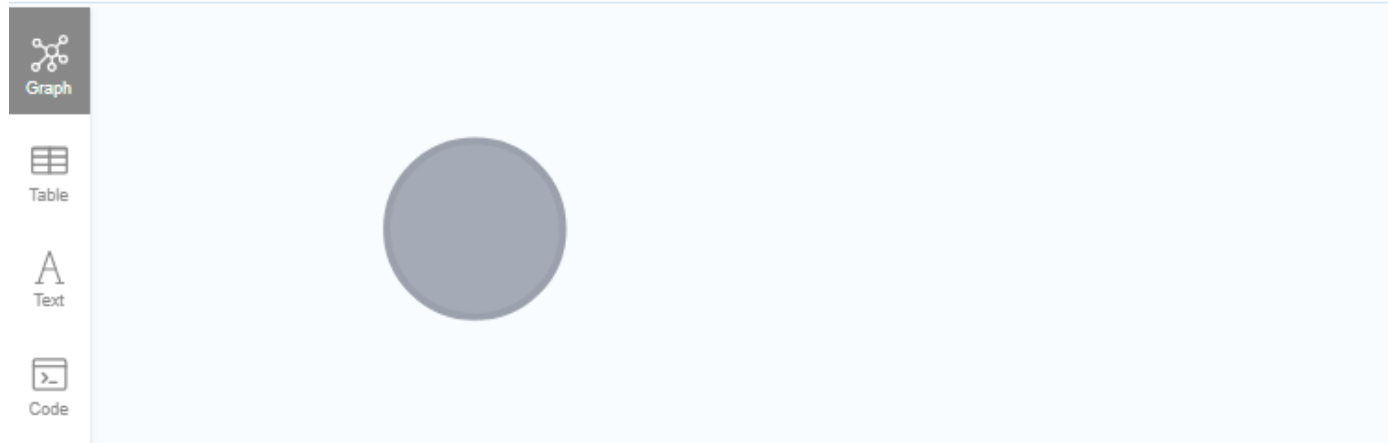
- De la misma manera, se pueden establecer varias propiedades en un nodo utilizando la cláusula SET. Para ello, debe especificar los pares clave-valor separados con comas.

```
MATCH (nodo:etiqueta {propiedades})  
SET nodo.propiedad1 = valor, nodo.propiedad2 = valor  
RETURN nodo
```

- Ejemplo

```
MATCH (a:alumno)  
WHERE a.noboleta IS NULL  
SET a.nombre='José Juan', a.apellidos='Rocha Segura', a.genero='M'  
RETURN a
```

```
1 MATCH (a)  
2 WHERE a.noboleta IS NULL  
3 SET a.nombre='José Juan', a.apellidos='Rocha Segura', a.genero='M'  
4 RETURN a
```



# Establecer una etiqueta en un nodo

- Puede establecer una etiqueta en un nodo existente mediante la cláusula SET.

```
MATCH (n {propiedades . . . })  
SET n:etiqueta  
RETURN n
```

- Ejemplo

```
MATCH (x:materia{id:1})  
SET x:asignatura  
RETURN x
```

```
MATCH (x:materia{id:1}) SET x:asignatura RETURN x
```



Overview

Node labels

\* (1) **asignatura (1)** **materia (1)**

Displaying 1 nodes, 0 relationships.

# Configuración de varias etiquetas en un nodo

- Puede establecer varias etiquetas en un nodo existente mediante la cláusula SET. Se deben especificar las etiquetas separándolas con dos puntos ":"

```
MATCH (n {propiedades . . . })  
SET n:etiqueta1:etiqueta2  
RETURN n
```



# Eliminación de una propiedad

- Se puede quitar una propiedad existente pasándole NULL como valor a la propiedad de un nodo mediante la cláusula SET.

```
MATCH (nodo:etiqueta {propiedades})  
SET nodo.propiedad = NULL  
RETURN nodo
```

## Ejemplo:

```
MATCH (n)  
WHERE n.noboleta IS NULL  
SET n.nombre = NULL, n.apellidos = NULL  
RETURN n
```

- Para la eliminación de las relaciones entre nodos, hay que realizar la búsqueda que coincida con el criterio y posteriormente eliminar el conjunto de resultado

- Ejemplo:

```
MATCH (x:alumno) - [y:noviazgo] - (:alumno)
```

```
DETACH DELETE y
```

```
RETURN x
```

- La opción DETACH permite "desligar" los nodos relacionados para proceder a eliminar la relación

# Eliminación de un nodo en particular

- Para eliminar un nodo en particular, se deben especificar los detalles del nodo para poder eliminarlo.

```
MATCH (nodo:etiqueta {propiedades . . . })  
DETACH DELETE nodo
```

Ejemplo:

```
MATCH (n)  
WHERE id(n) = 3  
DELETE n
```

Si el nodo tiene una relación, no será posible eliminarlo hasta eliminar la relación que mantiene, por eso el uso de DETACH

# Cláusula REMOVE

- La cláusula REMOVE se utiliza para eliminar propiedades y etiquetas de elementos de los grafos (Nodos o Relaciones).
  - La principal diferencia entre los comandos DELETE y REMOVE es que la operación DELETE se utiliza para eliminar nodos y relaciones asociadas, mientras que la operación REMOVE se utiliza para quitar etiquetas y propiedades.
- Eliminación de una propiedad
  - Se puede quitar una propiedad de un nodo mediante MATCH junto con la cláusula REMOVE.

```
MATCH (nodo:etiqueta{propiedades . . . })  
REMOVE nodo.propiedad  
RETURN nodo
```

- Ejemplo

```
MATCH (m:materia)  
WHERE m.id=3  
REMOVE m.creditos  
RETURN m
```

# Eliminación de una etiqueta de un nodo

- Al igual que la propiedad, también puede quitar una etiqueta de un nodo existente mediante la cláusula REMOVE.

```
MATCH (nodo:etiqueta {propiedades . . . })  
REMOVE nodo:etiqueta  
RETURN nodo
```

- Ejemplo

```
MATCH (x:materia{id:1})  
REMOVE x:asignatura  
RETURN x
```

# Eliminación de varias etiquetas

- También se pueden quitar varias etiquetas de un nodo existente.

```
MATCH (nodo:etiqueta1:etiqueta2 {propiedades . . . })  
REMOVE nodo:etiqueta1:etiqueta2  
RETURN nodo
```

# Eliminación de todos los nodos y relaciones

- Para eliminar todos los nodos y las relaciones de la base de datos se usa la cláusula DELETE.

`MATCH (n) DETACH DELETE n`

- Esto eliminará todos los nodos y relaciones de la base de datos y la vaciará.

# Índices

- Neo4j admite índices en propiedades de nodo o relación para mejorar el rendimiento de la aplicación. Podemos crear índices en propiedades para todos los nodos, que tienen el mismo nombre de etiqueta.
- Podemos usar estas columnas indexadas en el operador MATCH o WHERE o IN para mejorar la ejecución del comando CQL.
- Creación de un índice
  - CQL proporciona el comando "CREATE INDEX" para crear índices en las propiedades Node o Relationship.  
`CREATE INDEX ON :etiqueta (propiedad)`
  - Proporciona el comando "DROP INDEX" para eliminar índices en las propiedades Node o Relationship.  
`DROP INDEX ON :etiqueta (propiedad)`



# Restricción UNIQUE

- El comando CREATE siempre crea un nuevo nodo o relación, lo que significa que aunque use los mismos valores, inserta una nueva fila. Para evitar la duplicidad, se deben utilizar algunas restricciones de base de datos para crear una regla sobre una o más propiedades de un nodo o relación.
- Al igual que SQL, Neo4j también admite la restricción UNIQUE en las propiedades de nodo o relación. La restricción UNIQUE se utiliza para evitar registros duplicados y para hacer cumplir la regla de integridad de datos.
- Crear restricción UNIQUE
  - CQL proporciona el comando "CREATE CONSTRAINT" para crear restricciones únicas en las propiedades de nodo o relación.

```
CREATE CONSTRAINT ON (a:alumno) ASSERT a.noboleta IS UNIQUE
```

- Para eliminar la restricción:

```
DROP CONSTRAINT ON (a:alumno) ASSERT a.noboleta IS UNIQUE
```

# Sentencia LOAD CSV

- LOAD CSV es usada para importar datos de archivos en formato CSV
- La dirección URL del archivo CSV se especifica usando la cláusula FROM seguida de una expresión arbitraria que evalúa dicha URL
- Se requiere especificar una variable para los datos obtenidos usando la cláusula AS
- Si el valor de la configuración en *server.directories.import*
- Esta definido con el valor por omisión, la dirección URL tratará de leer el archivo de la ruta dada  
por <NEO4J\_HOME>/import/myfile.csv y  
<NEO4J\_HOME>/import/myproject/myfile.csv respectivamente

```
LOAD CSV FROM 'file:///artists.csv' AS line  
CREATE (:Artist {name: line[1], year:  
toInteger(line[2])})
```

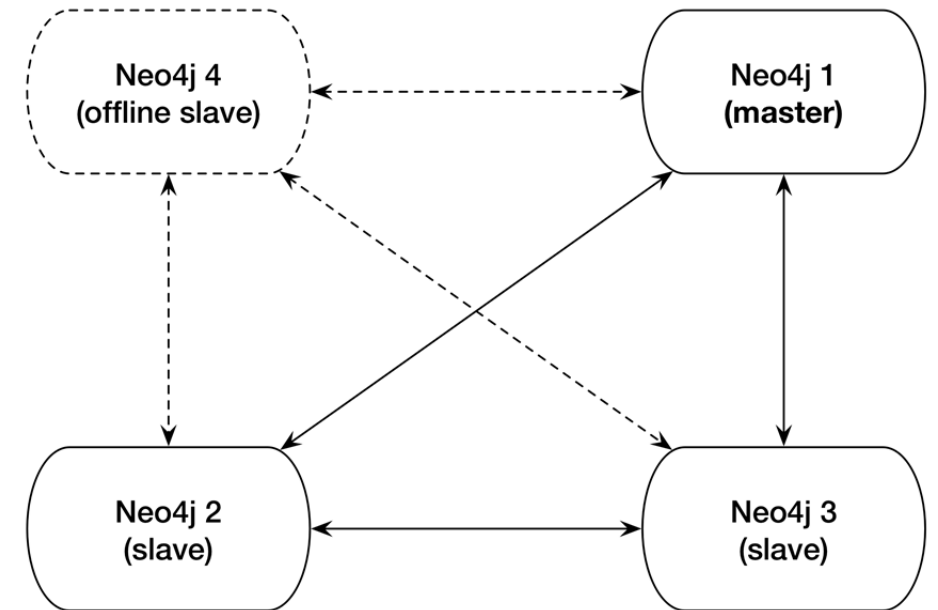
- La cláusula WITH HEADERS indica que el archivo origen debe ser importado considerando la primera línea de datos como la cabecera

```
LOAD CSV WITH HEADERS FROM 'file:///artists-  
with-headers.csv' AS line CREATE (:Artist {name:  
line.Name, year: toInteger(line.Year)})
```

# Cluster HA (High Availability)

---

- Para usar Neo4j con HA, primero se debe configurar un clúster.
- Anteriormente, Neo4j usaba clústeres en ZooKeeper como un mecanismo de coordinación externa, que funcionaba bien, pero requería mucha administración adicional, ya que ZooKeeper tendría que ejecutarse por separado. Eso ha cambiado en versiones más recientes.
- Ahora, los clústeres Neo4j son autogestionados y auto coordinados.
- Los clústeres pueden elegir su propia configuración maestro/esclavo y volver a coordinarse cuando los servidores se desconectan.



Los nodos 1, 2 y 3 están actualmente en línea y se replican entre sí correctamente, mientras que el nodo 4 está fuera de línea.

Cuando el nodo 4 vuelva a estar en línea, volverá a entrar como nodo esclavo.

Si el nodo 1, el nodo maestro actual, se desconecta, los otros nodos elegirían automáticamente un líder (sin la ayuda de un servicio de coordinación externo).

- Esta es una configuración de ALTA disponibilidad bastante estándar en la industria hoy en día, y los ingenieros detrás de Neo4j han hecho a los administradores un gran servicio al habilitar una configuración de alta disponibilidad sin ZooKeeper.

---

En los clústeres de HA Neo4j, la elección del maestro ocurre automáticamente. Si el servidor maestro se desconecta, otros servidores lo notarán y elegirán un líder de entre ellos.

---

Reiniciar el servidor maestro anterior lo agregará de nuevo al clúster, pero ahora el antiguo maestro será un esclavo (hasta que otro servidor se caiga).

---

La alta disponibilidad permite que los sistemas de lectura muy pesados se ocupen de replicar grafos en múltiples servidores y, por lo tanto, balancear la carga.

---

Aunque el clúster en su conjunto sólo es consistente eventualmente, hay trucos que puede aplicar para reducir la posibilidad de leer datos obsoletos en sus propias aplicaciones, como asignar una sesión a un servidor.

---

Con las herramientas adecuadas, la planificación y una buena configuración, puede crear una base de datos de grafos lo suficientemente grande como para manejar miles de millones de nodos y vértices, y casi cualquier número de solicitudes que se puedan necesitar.

---

Simplemente se agregan copias de seguridad periódicas y se tendrá la receta para un sistema de producción sólido.

# Copias de seguridad

Las copias de seguridad son un aspecto necesario de cualquier base de datos.

Aunque las copias de seguridad se integran de manera efectiva cuando se utiliza la replicación en un grupo de alta disponibilidad, las copias de seguridad periódicas (nocturnas, semanales, por hora, etc.) que se almacenan en sitio siempre son una buena idea para la recuperación ante desastres.

- Neo4j Enterprise ofrece una herramienta llamada **neo4j-admin** que realiza una amplia variedad de acciones, incluidas las copias de seguridad.

El método más eficaz al ejecutar un servidor de alta disponibilidad es crear un comando de copia de seguridad completa para copiar el archivo de base de datos del clúster a un archivo con estampa de tiempo en una unidad montada.

- Apuntar la copia a todos los servidores del clúster permitirá obtener los datos más recientes disponibles.
- El directorio de copia de seguridad creado es una copia totalmente utilizable.
- Si se necesita recuperarse de un fallo, se reemplaza el directorio de datos de cada instalación con el directorio de copia de seguridad, y estará listo para comenzar.
- Se debe comenzar con una copia de seguridad completa.



# Comentarios sobre Neo4j

Neo4j es una implementación de código abierto de clase superior (relativamente rara) de bases de datos de grafos.

Las bases de datos de grafos se centran en las relaciones entre los datos, en lugar de los puntos en común entre los valores.

Modelar datos basado en grafos es simple.

- Solo se crean nodos y las relaciones entre ellos y, opcionalmente, pares clave-valor para ellos.
- Consultar es tan fácil como declarar cómo recorrer el grafo desde un nodo inicial.



# Fortalezas

Las bases de datos de grafos son perfectas para datos no estructurados, en muchos sentidos, incluso más que las bases de datos de documentos.

- Neo4j no solo es sin tipos ni esquemas, sino que no impone restricciones sobre cómo se relacionan los datos.
- Actualmente, Neo4j puede soportar 34.4 mil millones de nodos y 34.4 mil millones de relaciones, lo cual es más que suficiente para la mayoría de los casos de uso.
- Las distribuciones Neo4j proporcionan varias herramientas para búsquedas rápidas con Lucene, el lenguaje de consulta Cypher y la interfaz REST.

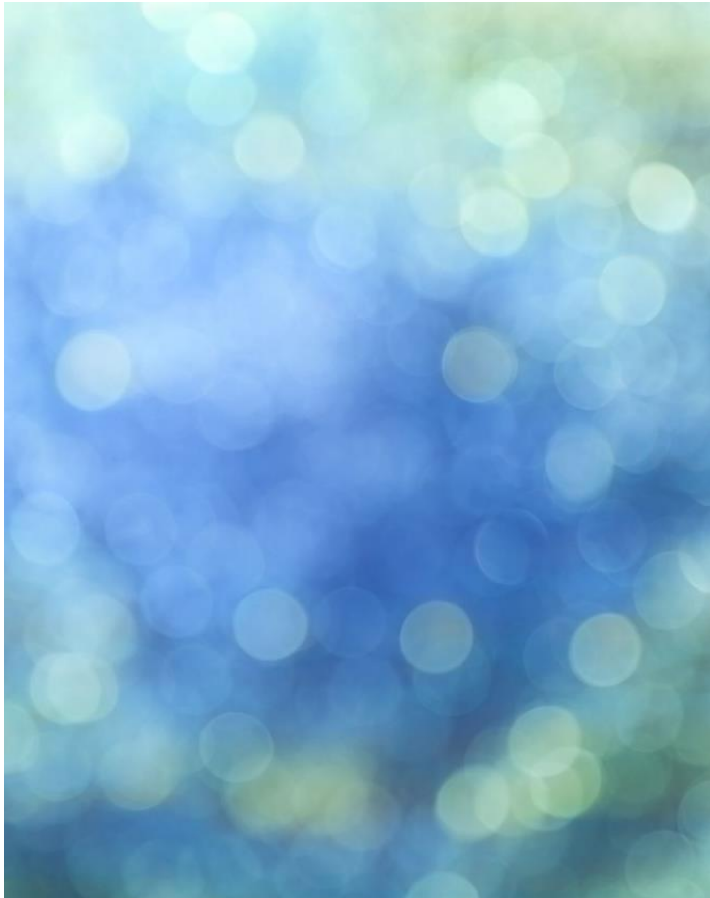
Más allá de la facilidad de uso, Neo4j es rápido.

- A diferencia de las operaciones de unión en bases de datos relacionales o las operaciones de map-reduce en otras bases de datos, los recorridos de grafos son en un tiempo constante.
- Los datos están a solo un paso entre nodos, en lugar de unir valores y filtrar los resultados deseados, que es cómo funcionan la mayoría de las bases de datos.
- No importa cuán grande sea el grafo; pasar del nodo A al nodo B siempre es a un paso si comparten una relación.

Por último, la edición Enterprise ofrece sitios de alta disponibilidad y alto tráfico de lectura a través de Neo4j HA.



# Debilidades de Neo4j



Aunque la Community Edition es GPL, probablemente se necesitará comprar una licencia si desea ejecutar un entorno de producción utilizando las herramientas Enterprise (que incluyen HA y copias de seguridad).

Actualmente no se puede fragmentar subgrafos, lo que todavía pone un límite en el tamaño del grafo

Aunque HA es excelente en la replicación, solo puede replicar un grafo completo a otros servidores.

Neo4j HA está disponible y es tolerante a particiones (AP).

- Cada esclavo regresará solo lo que tiene actualmente, lo que puede estar fuera de sincronía con el nodo maestro temporalmente.

La utilidad de Neo4j puede ser desagradable si no está acostumbrado a modelar datos de grafos.

Proporciona una poderosa API de código abierto con años de uso en producción y, sin embargo, no ha obtenido la misma tracción que otras bases de datos.

- Atribuimos esto a la falta de conocimiento porque las bases de datos de grafos se entrelazan de manera tan natural con la forma en que los humanos tienden a conceptualizar los datos.

Para ciertas clases de problemas, como las redes sociales, Neo4j es una opción obvia.