Bases de Datos de Grafos - Neo4j

Centro de Investigación en Computación / Instituto Politécnico Nacional

> ***Autor:*** *Alan Ignacio Delgado Alarcon*
> *Junio 2025*
> *Aspectos Avanzados de Bases de Datos*

Instalación

Para versiones para Windows y macOS visite el sitio y descargue el archivo de instalación:

Instalador - neo4j.com

Linux - Ubuntu 24.04

Para instalación en sistemas basados en debian, como el caso de Ubuntu version `24.04 LTS` para neo4j `2025.05.0`, siga los siguinetes pasos:

> NOTA
> las versiones de neo4j 2025 requieren de `java 21`. Para comprobar las versiones disponibles de java en su sistema use:
>
> ```
> update-java-alternatives --list
> ```
>
> La salida del comando podria ser similar a esta:
>
> ```
> java-1.21.0-openjdk-amd64 2111 /usr/lib/jvm/java-1.21.0-openjdk-amd64
> java-1.17.0-openjdk-amd64 1711 /usr/lib/jvm/java-1.17.0-openjdk-amd64
> ...
> ```
>
> Asigne la versón 21 de java por defecto con el siguinete comando:
>
> ```
> sudo update-java-alternatives --jre --set java-1.21.0-openjdk-amd64
> ```
>
> valide que la actualización se realizo correctamente con:
>
> ```
> java -version
> ```

1. Agregar los repositorios oficiales, las llaves de verificación y actualizamos la lista de paquetes disponibles para instralación.

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo gpg --dearmor -o
/etc/apt/keyrings/neotechnology.gpg
echo 'deb [signed-by=/etc/apt/keyrings/neotechnology.gpg] https://debian.neo4j.com
stable latest' | sudo tee -a /etc/apt/sources.list.d/neo4j.list
sudo apt-get update
```

2. Verificamos que el paquete y versiones de neo4j estan disponibles.

```
apt list -a neo4j
```

3. Instalamos neo4j en su version `Community` que nos permite hacer uso del gestor sin necesidad de licencia. Instalamos la ultima versión estable: `2025.05.0`.

```
sudo apt-get install neo4j=1:2025.05.0
```

4. Ejecutamos el siguiente comando para asegurarnos que neo4j se inicie automaticamente al iniciar el sistema.

```
sudo systemctl enable neo4j
```

Ubicación de archivos

Los directorios y sus ubicaciones por defecto para configuraciones adicionales se pueden consultar diractamente de la documentación oficial.

Default file locations - neo4j.com

Para fines de los ejercicios en este documento no es necesario realizar ajustes adicionales a la instalación previa.

Neo4j Desktop

Este software habilita una interfaz grafica para la gestion de las bases de datos locales, incluye una licencia de neo4j `Enterprice Edition Developer`, limitada para ser usada unicamente en una máquina.

Para los ejercicios permitira ver de forma visual los resultados de las consultas y comprender mejor el modelo de este tipo de baes de datos. Neo4j recomienda no ejecutar la aplicación Desktop en entornos de producción debido a los problemas de seguridad derivados por el tipo de ejecuciones.

Ejecución

1. En la pagina del centro de desarrollo de neo4j, en el apartado de neo4j Desktop deje seleccionada la ultima versión disponible y seleccione el sistema operativo `Linux (x86 AppImage)`.

Deployment Center - neo4j.com

2. Llene el formulario que muestra a continuación y el archivo comenzará su descarga automaticamente.
3. Abra una terminal en el sistema y vaya al directorio donde se ha descargado el archivo de neo4j desktop.
4. Asignamos permisos de ejecución con el usuario que inicio sesión en el sistema:

```
sudo chmod +x neo4j-desktop-2.0.1-x86_64.AppImage
```

5. Instalamos el paquete `fuse` necesario para el funcionamiento del aplicativo

```
sudo apt install fuse
```

6. Iniciamos la aplicación con el siguiente comando para ejecutarla sin el entorno seguro. Esta opción se utiliza únicamente con fines prácticos en este documento.

```
./neo4j-desktop-2.0.1-x86_64.AppImage --no-sandbox
```

> NOTA
> `.AppImage` es un formato portable de aplicaciones Linux que no requiere instalación, solo permisos de ejcución.

7. Dentro de la aplicación creamos una nueva instancia local, a la cual le asignamos un nombre así como un usuario y contraseña para poder acceder a el.
8. Iniciamos la instancia. Al arrancar se iniciaran 2 bases de datos por defecto: `system` y `neo4j`.
9. En un navegador ingresamos a la url: `http://localhost:7474` donde se iniciara `neo4j browser`, donde ejecutaremos las pruebas de código que nos permitira observar de forma visual el comportamiento de la base de datos.

CRUD

Create - Crear

Crear un nodo simple

Creamos un nodo con etiqueta Investigador y propiedades básicas.

> Lo importante: Los nodos se representan con paréntesis ( ), las etiquetas con :, y las propiedades dentro de { } como clave-valor.

```
CREATE (:Investigador {nombre: "Ana", area: "IA", edad: 35})
```

Crear dos nodos sin relación

Creamos un investigador y un laboratorio, pero aún no están conectados.

> Lo importante: Puedes crear múltiples nodos en una sola instrucción separándolos con comas.

```
CREATE (:Investigador {nombre: "Luis"}),
       (:Laboratorio {nombre: "Lab Computación"})
```

Crear un nodo y una relación en una sola instrucción

Creamos un investigador y lo conectamos a un proyecto nuevo.

> Lo importante: Las relaciones se representan con -[:RELACION]->, y también pueden llevar propiedades.

```
CREATE (:Investigador {nombre: "Clara"})-[:PARTICIPA_EN]->(:Proyecto {titulo: "Visión
por Computadora"})
```

Crear una relación entre nodos existentes

Conectamos a un investigador con un laboratorio que ya existen.

> Lo importante: Usamos MATCH para buscarlos y luego CREATE para establecer la relación.

```
MATCH (i:Investigador {nombre: "Luis"}), (l:Laboratorio {nombre: "Lab Computación"})
CREATE (i)-[:PERTENECE_A]->(l)
```

Crear una relación con propiedades

Indicamos información adicional sobre la relación, como desde cuándo existe.

> Lo importante: Las propiedades se agregan en {} después del nombre de la relación.

```
MATCH (i:Investigador {nombre: "Clara"}), (p:Proyecto {titulo: "Visión por
Computadora"})
CREATE (i)-[:PARTICIPA_EN {desde: 2022, rol: "colaboradora"}]->(p)
```

Crear nodo si no existe cypher Copiar Editar MERGE (:Investigador {nombre: "Ana", area: "IA"}) Lo importante:

Si ya existe un nodo con esas propiedades exactas, no hace nada.

Si no existe, lo crea.

Evita duplicados sin necesidad de MATCH + IF.

◇ Crear relación solo si no existe c Copiar Editar MATCH (a:Investigador {nombre: "Ana"}), (p:Proyecto {titulo: "Proyecto A"}) MERGE (a)-[:PARTICIPA_EN]->(p) Lo importante:

Si ya existe la relación entre esos nodos con ese tipo, no la duplica.

Ideal para relaciones que no deben repetirse.

◇ Controlar nodos + relaciones con propiedades cypher Copiar Editar MERGE (i:Investigador {nombre: "Luis"}) ON CREATE SET i.area = "Redes", i.edad = 40 ON MATCH SET i.accesos = coalesce(i.accesos, 0) + 1 ON CREATE: se ejecuta solo si el nodo se acaba de crear.

ON MATCH: se ejecuta solo si ya existía.

Read - Leer

Leer toda la base de datos (nodos y relaciones)

1. Mostramos todo el grafo: todos los nodos, relaciones y sus conexiones.

> Lo importante: Es útil para explorar la base al inicio o después de cargar datos.

```
MATCH (n)-[r]->(m)
RETURN n, r, m
```

2. Solo nodos (sin relaciones)

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

2. Todos los nodos conectados por cualquier relación (sin importar dirección)

```
MATCH (n)-[r]-(m)
RETURN n, r, m
```

3. Todo el grafo, incluidos nodos sin relaciones

```
MATCH (n)
OPTIONAL MATCH (n)-[r]->(m)
RETURN n, r, m
```

Leer todos los nodos de un tipo

Consultamos todos los nodos con la etiqueta Investigador.

> Lo importante: MATCH busca patrones en el grafo; RETURN muestra los resultados.

```
MATCH (i:Investigador)
RETURN i
```

Leer con filtro de propiedad

Buscamos un investigador por nombre.

> Lo importante: Se puede usar WHERE para aplicar condiciones sobre propiedades.

```
MATCH (i:Investigador)
WHERE i.nombre = "Ana"
RETURN i
```

Leer relaciones entre nodos

Recuperamos investigadores y los laboratorios a los que pertenecen.

> Lo importante: Se pueden leer relaciones con MATCH (a)-[:REL]->(b).

```
MATCH (i:Investigador)-[:PERTENECE_A]->(l:Laboratorio)
RETURN i.nombre, l.nombre
```

Leer relaciones con propiedades

Obtenemos investigadores y desde cuándo participan en su proyecto.

> Lo importante: Las relaciones también tienen propiedades, y puedes proyectarlas con un alias.

```
MATCH (i:Investigador)-[r:PARTICIPA_EN]->(p:Proyecto)
RETURN i.nombre, p.titulo, r.desde
```

Leer relaciones sin importar dirección

Buscamos cualquier conexión entre investigadores (sin importar hacia dónde va la flecha).

> Lo importante: Usa -- para ignorar dirección.

```
MATCH (a:Investigador)--(b:Investigador)
RETURN a.nombre, b.nombre
```

Leer con profundidad variable

Buscamos investigadores conectados indirectamente a través de colaboraciones de hasta 3 saltos.

> Lo importante: El operador *1..3 recorre relaciones múltiples.

```
MATCH (a:Investigador {nombre: "Ana"})-[:COLABORA_CON*1..3]-(otro)
RETURN DISTINCT otro.nombre
```

Encontrar el camino más corto (shortest path)

Buscamos el investigador más cercano a Ana por relaciones COLABORA_CON.

> Lo importante: shortestPath() devuelve solo el camino más corto, ideal para análisis de proximidad.

```
MATCH p = shortestPath(
  (a:Investigador {nombre: "Ana"})-[:COLABORA_CON*]-(otro:Investigador)
)
RETURN otro.nombre, length(p) AS saltos
```

Update - Actualizar

Agregar o modificar propiedades de un nodo

Cambiamos o añadimos propiedades a un nodo Investigador.

> Lo importante: Si la propiedad no existe, se crea. Si ya existe, se actualiza.

```
MATCH (i:Investigador {nombre: "Ana"})
SET i.edad = 36,
    i.es_jefa = true
RETURN i
```

Modificar propiedades de una relación

Actualizamos la propiedad rol en una relación PARTICIPA_EN.

> Lo importante: Usamos un alias para la relación con [r].

```
MATCH (:Investigador {nombre: "Luis"})-[r:PARTICIPA_EN]->(:Proyecto {titulo: "Visión
por Computadora"})
SET r.rol = "investigador principal"
RETURN r
```

Cambiar valor existente

Sobrescribimos el valor anterior de una propiedad.

> Ejemplo: Cambiar el área de un investigador.

```
MATCH (i:Investigador {nombre: "Clara"})
SET i.area = "Robótica"
RETURN i
```

Agregar nuevas propiedades con un map

Usamos un map para añadir varias propiedades a la vez.

> Lo importante: Similar a un SET múltiple.

```
MATCH (i:Investigador {nombre: "Luis"})
SET i += {grado: "Doctorado", publicaciones: 12}
RETURN i
```

Delete

Eliminar un nodo sin relaciones

Eliminamos un nodo que no tiene relaciones activas.

> Lo importante: DELETE solo funciona si el nodo está aislado; si tiene relaciones, lanza error.

```
MATCH (i:Investigador {nombre: "Carlos"})
DELETE i
```

Eliminar un nodo con relaciones activas

Eliminamos el nodo y todas sus relaciones.

> Lo importante: Usamos DETACH DELETE para borrar el nodo aunque esté conectado.

```
MATCH (i:Investigador {nombre: "Luis"})
DETACH DELETE i
```

Eliminar solo una relación

Borramos la relación entre un investigador y un proyecto, sin eliminar los nodos.

> Lo importante: Usamos alias para la relación y aplicamos DELETE.

```
MATCH (:Investigador {nombre: "Clara"})-[r:PARTICIPA_EN]->(:Proyecto {titulo: "Visión
por Computadora"})
DELETE r
```

Eliminar múltiples nodos o relaciones

```
MATCH (l:Laboratorio)
WHERE l.nombre STARTS WITH "Lab"
DETACH DELETE l
```

REMOVE – Eliminar propiedades (no el nodo)

1. Quitar una propiedad de un nodo

```
MATCH (i:Investigador {nombre: "Ana"})
REMOVE i.edad
RETURN i
```

2. Quitar varias propiedades

```
MATCH (i:Investigador {nombre: "Luis"})
REMOVE i.grado, i.publicaciones
RETURN i
```

Otras clausulas

La documentación provee de ejmplos completos para las clausulas de agrergación y complementarias para consultas más complejas.
neo4j - cypher cheat sheet

Carga de datos

Lenguajes de programación

Aunque neo4j fue diseñada como una base de datos para Java, actualmente ya estan disponibles diversos drivers para poder conectarse al gestor a travez de los lenguajes de programación con mas uso.
Estan divididos en 2 grupos:

Oficiales:

- .NET
- Go
- Java
- Javascript
- Python

Comunidad

- C
- Elixir
- Perl
- PHP
- Ruby
- Rust

Ejemplo con Python

Instalación del driver

```
pip install neo4j
```

Código de ejemplo

```
from neo4j import GraphDatabase, RoutingControl


URI = "neo4j://localhost:7687"
AUTH = ("neo4j", "password")


def add_friend(driver, name, friend_name):
    driver.execute_query(
        "MERGE (a:Person {name: $name}) "
        "MERGE (friend:Person {name: $friend_name}) "
        "MERGE (a)-[:KNOWS]->(friend)",
        name=name, friend_name=friend_name, database_="neo4j",
    )


def print_friends(driver, name):
    records, _, _ = driver.execute_query(
        "MATCH (a:Person)-[:KNOWS]->(friend) WHERE a.name = $name "
        "RETURN friend.name ORDER BY friend.name",
        name=name, database_="neo4j", routing_=RoutingControl.READ,
    )
    for record in records:
        print(record["friend.name"])


with GraphDatabase.driver(URI, auth=AUTH) as driver:
    add_friend(driver, "Arthur", "Guinevere")
    add_friend(driver, "Arthur", "Lancelot")
    add_friend(driver, "Arthur", "Merlin")
    print_friends(driver, "Arthur")
```