

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](https://neo4j.com/instalador)

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 siguientes 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 podría 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 versión 21 de java por defecto con el siguiente comando:

```
sudo update-java-alternatives --jre --set java-1.21.0-openjdk-amd64
```

valide que la actualización se realizó correctamente con:

```
java -version
```

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

```
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 directamente 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 **Enterprise 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 bases 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 automáticamente.
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 ejecución.

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 una base de datos

Desde neo4j desktop es posible crear una nueva base de datos sobre la instancia creada o bien desde la consola de cypher-shell

```
neo4j@neo4j> CREATE DATABASE name;  
neo4j@neo4j> :use name;
```

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

```
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

```
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

```
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
```

Eliminar todos los datos

```
MATCH (n) DETACH DELETE n;
```

Otras cláusulas

La documentación provee de ejemplos completos para las cláusulas de agregación y complementarias para consultas más complejas.

[neo4j - cypher cheat sheet](#)

Carga de datos

En neo4j, una forma eficiente de importar datos estructurados es utilizando archivos **.csv** a través de la herramienta de línea de comandos neo4j-admin.

💡 La forma recomendada es separar los archivos en:

Uno o más archivos para nodos.

Uno o más archivos para relaciones.

A continuación, veremos un ejemplo con tres archivos:

- movies.csv → nodos tipo Movie
- actors.csv → nodos tipo Actor
- roles.csv → relaciones :ACTED_IN entre actores y películas

⚠ La base de datos debe estar creada previamente.

neo4j-admin database import no puede crear la base, solo llenarla.

Si la base no existe, el comando fallará. Si alguna relación esta mal, el comando fallará y no cargara los datos.

```
bin/neo4j-admin database import full neo4j \
  --nodes=import/movies.csv \
  --nodes=import/actors.csv \
  --relationships=import/roles.csv
```

LOAD CSV

Cuando se trabaja con archivos **.csv** que no tienen formato específico o tiene un formato generico de tabla para neo4j-admin (es decir, no tienen columnas como :ID, :LABEL, :START_ID, etc.), usamos el comando **LOAD CSV** desde Cypher.

Esta opción se ejecuta dentro de una consulta Cypher.

```
LOAD CSV FROM 'https://data.neo4j.com/bands/artists.csv' (1)
AS row (2)
MERGE (:Artist {name: row[1], year: toInteger(row[2])}) (3)
```

(1) **FROM** toma un **STRING** que contiene la ruta donde se encuentra el archivo CSV.

(2) La cláusula analiza una fila cada vez, almacenando temporalmente la fila actual en la variable especificada con AS.

(3) La cláusula MERGE accede a la variable de fila para insertar los datos en la base de datos.

LOAD CSV admite URL locales y remotas. Las rutas locales se resuelven relativas a la carpeta de instalación de Neo4j.

Ejemplo con artist.csv

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

El siguiente ejemplo, contempla la carga de datos con la generación de relaciones dentro de la carga.

```
LOAD CSV WITH HEADERS FROM 'file:///envios.csv' AS row

MERGE (c1:Ciudad {nombre: trim(row.origen)})
MERGE (c2:Ciudad {nombre: trim(row.destino)})

MERGE (c1)-[r:ENVIA_A]->(c2)
SET r.distancia_km = toInteger(row.distancia_km),
    r.transporte    = trim(row.transporte),
    r.costo          = toFloat(row.costo)
```

WITH HEADERS permite ignorar la primer línea del archivo csv si este tiene las cabeceras de los datos.

Ejemplo con archivo de 3,000 filas

```
LOAD CSV WITH HEADERS FROM 'file:///envios_masivos.csv' AS row
MERGE (c1:Ciudad {nombre: trim(row.origen)})
MERGE (c2:Ciudad {nombre: trim(row.destino)})
MERGE (c1)-[r:ENVIA_A]->(c2)
SET r.distancia_km = toInteger(row.distancia_km),
    r.transporte    = trim(row.transporte),
    r.costo          = toFloat(row.costo)
```

Lenguajes de programación

Aunque neo4j fue diseñada como una base de datos para Java, actualmente ya están disponibles diversos drivers para poder conectarse al gestor a través de los lenguajes de programación con más uso.

Están 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") # Cambia la contraseña si es necesario

def agregar_investigador(driver, nombre, area, edad):
    driver.execute_query(
        "MERGE (i:Investigador {nombre: $nombre}) "
        "SET i.area = $area, i.edad = $edad",
        nombre=nombre, area=area, edad=edad,
        database_="neo4j",
    )

def agregar_colaboracion(driver, nombre1, nombre2, proyecto, año):
    driver.execute_query(
        "MATCH (a:Investigador {nombre: $nombre1}) "
        "MATCH (b:Investigador {nombre: $nombre2}) "
        "MERGE (a)-[r:COLABORA_CON]->(b) "
        "SET r.proyecto = $proyecto, r.año = $año",
        nombre1=nombre1, nombre2=nombre2, proyecto=proyecto, año=año,
        database_="neo4j",
    )

def ver_colaboradores(driver, nombre):
    records, _, _ = driver.execute_query(
        "MATCH (i:Investigador)-[:COLABORA_CON]->(colaborador) "
        "WHERE i.nombre = $nombre "
        "RETURN colaborador.nombre, colaborador.area",
        nombre=nombre, database_="neo4j", routing_=RoutingControl.READ,
    )
    for record in records:
        print(f"{record['colaborador.nombre']} ({record['colaborador.area']})")

# --- Ejemplo de uso ---
with GraphDatabase.driver(URI, auth=AUTH) as driver:
    agregar_investigador(driver, "Ana", "Inteligencia Artificial", 35)
    agregar_investigador(driver, "Luis", "Robótica", 42)
    agregar_investigador(driver, "Clara", "Visión por Computadora", 38)

    agregar_colaboracion(driver, "Ana", "Luis", "IA aplicada a salud", 2022)
    agregar_colaboracion(driver, "Ana", "Clara", "Proyecto de visión", 2023)

    print("Colaboradores de Ana:")
    ver_colaboradores(driver, "Ana")

```

Uso de la terminal

cypher cuenta con su propio interprete de comandos accesible desde terminal para poder ejecutar las consultas directamente, aunque sin la interpretación grafica del grafo.

cypher-shell

Una herramienta incluida con Neo4j que permite ejecutar comandos Cypher desde la terminal.

Ideal para ambientes de producción, automatización, o simplemente para demostrar que no necesitas Neo4j Browser.

Para acceder a ella:

```
bin/cypher-shell -u neo4j -p <tu_contraseña>
```

- `:exit` para salir de la terminal