



## 5. BASES DE DATOS ORIENTADAS A GRAFOS

<b>5. BASES DE DATOS ORIENTADAS A GRAFOS.....</b>	<b>1</b>
5.1. Conceptos básicos.....	1
5.2. Instalaciones previas.....	4
5.2.1. OpenJDK 17.....	4
5.3. Instalaciones de Neo4J.....	5
5.4. Iniciando Neo4J.....	6
5.5. Autenticando en Neo4J.....	7
Ejemplo.....	7
5.6. Creando nodos.....	7
Sintaxis.....	7
Ejemplo:.....	8
5.7. Creando relaciones.....	8
5.8. Agregando atributos.....	8
Sintaxis.....	8
Ejemplo.....	8
5.9. Utilizando match.....	9
Ejemplo.....	9
5.10. Actualizando atributos.....	11
Ejemplo.....	11
5.11. Realizando consultas con un rango de valores.....	11
5.12. Neo4j Desktop.....	12
5.12.1. Instalación básica.....	12
5.12.2. Proyectos en Neo4j Desktop.....	13
5.12.3. Interactuando con DBMS Movie.....	13
5.13. Importar datos en Neo4J.....	17
5.14. Otras características de las bases de datos orientadas a grafos.....	19
5.14.1. Consistencia.....	19
5.14.2. Transacciones.....	20
5.14.3. Disponibilidad.....	20
5.14.4. Funcionalidades para realizar consultas.....	20
5.14.5. Escalabilidad.....	21
5.14.6. Casos de uso.....	22

## 5.14.7. Máquinas generadoras de recomendaciones.....23

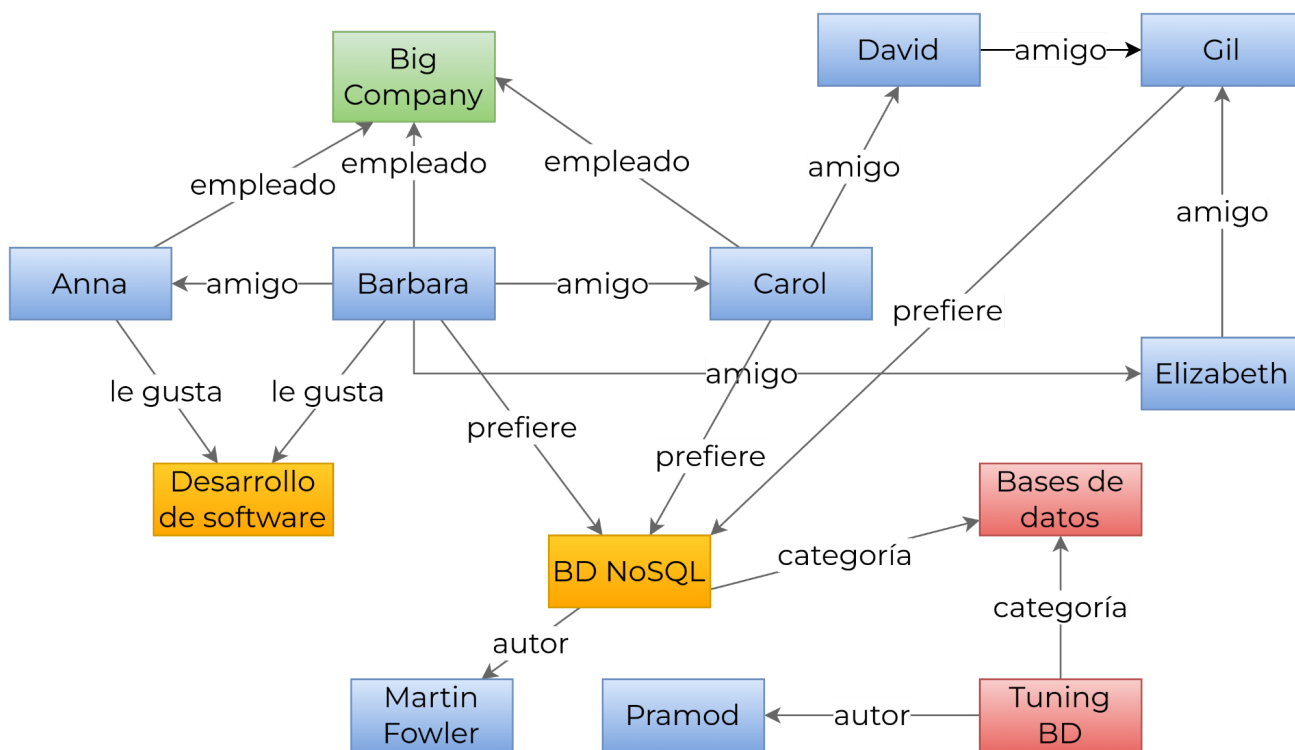
## 5.1. Conceptos básicos

Una base de datos orientada a grafos permite almacenar instancias de entidades y sus relaciones. A las entidades también se les conoce como **nodos** los cuales tienen o definen una serie de **atributos**. Un nodo puede ser visto como una instancia de un objeto en una aplicación. Las relaciones entre las entidades se representan a través de los **conectores** que unen a los nodos. Estos conectores también pueden definir **atributos**. El significado de estas relaciones puede ser **bidireccional**.

Las relaciones que se representan a través de los conectores pueden ser muy útiles para detectar patrones de comportamiento de la aplicación.

Al conjunto de nodos y sus conectores se le conoce como **grafo**. Por su naturaleza, los datos se almacenan y se representan una sola vez, pero pueden ser interpretados de formas **diferentes**.

Realizando una comparativa con una BD Key-Value en donde los 2 elementos básicos son key y value, en una BD orientada grafos, sus elementos son: nodos, relaciones y atributos. Los atributos son empleados para describir tanto a los nodos como a las relaciones.



- En el ejemplo anterior se puede observar varios nodos relacionados entre sí.

- Los nodos definen atributos, por ejemplo, el **nombre** de una persona.
- Los conectores de las entidades pueden definir también atributos que permiten organizar a los nodos, por ejemplo: **empleado**, **amigo**, **prefiere**. A cada uno de estos valores se les conoce como **tipos de relaciones**.
- Estos tipos de relaciones tienen un significado **direccional** que puede ser en un sentido (unidireccional) o en ambos sentidos (bidireccional). Por ejemplo, la relación **amigo** puede ser bidireccional, pero la relación **prefiere** es unidireccional
- Cada conector puede definir múltiples atributos.

Una vez que la gráfica de nodos y conectores ha sido definida, se pueden establecer consultas en diversas formas. Algunos ejemplos:

- Obtener todos los nodos que son empleados de la empresa Big Company que les gustan las bases de datos NoSQL.
- Obtener a todas las personas que les gusta NoSQL sin importar si son empleados de Big Company

Como se puede observar, las relaciones entre estos nodos pueden cambiar sin la necesidad. En el modelo relacional, si se desea agregar una nueva relación, se requieren realizar varios cambios, por ejemplo, cambios a nivel de esquema, mover datos, etc.

En este tipo de bases de datos, realizar recorridos transversales sobre el grafo son muy rápidos y eficientes respecto a una BD relacional. Realizar un recorrido transversal significa recorrer o hacer joins entre relaciones. Los nodos pueden tener múltiples relaciones y de diferentes tipos.

En una BD orientada a grafos las relaciones se persisten (almacenan) en lugar de ser calculadas durante el proceso de ejecución de la consulta como lo es en una BD relacional.

Ejemplos de BD orientadas a grafos

- Neo4J
- Infinite Graph
- OrientDB
- FlockDB

Este tipo de bases de datos son adecuadas cuando existen múltiples instancias que se relacionan entre ellas de diferentes formas, generalmente de forma compleja. Estas relaciones pueden definir ciertos atributos. Las consultas hacia estos grafos pueden resolver preguntas como:

- Encontrar a los nodos vecinos más cercanos

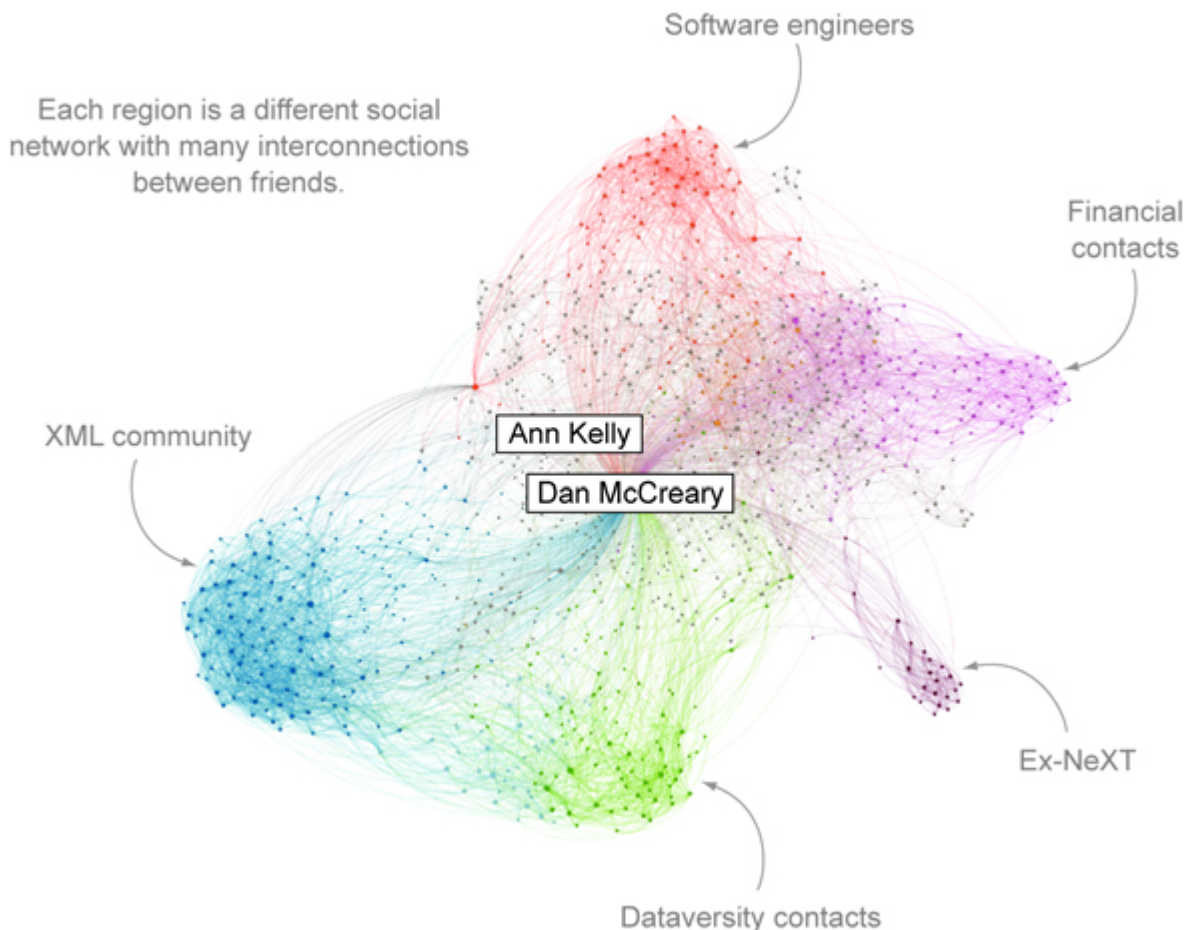
- Encontrar patrones de comportamiento al realizar un análisis profundo (recorrido largo de nodos), por ejemplo, encontrar una lista de amigos que son candidatos a comprar vino. La BD no solo indica la existencia de relaciones, también pueden mostrar detalles, características o propiedades de esas relaciones.

## Ejemplos con Neo4J

```
Node martin = graphDb.createNode();  
martin.setProperty("name", "Martin");  
  
Node pramod = graphDb.createNode();  
pramod.setProperty("name", "Pramod");
```

## Creación de relaciones

```
martin.createRelationshipTo(pramod, FRIEND);  
  
pramod.createRelationshipTo(martin, FRIEND);
```



## 5.2. Instalaciones previas

### 5.2.1. OpenJDK 17

- A. Acceder al sitio <https://jdk.java.net/archive/>
- B. Descargar la versión de Linux/x64.

<b>17 GA (build 17+35)</b>		
<b>Windows</b>	<b>64-bit</b>	zip (sha256) 178M
<b>Mac/AArch64</b>	<b>64-bit</b>	tar.gz (sha256) 174M
<b>Mac/x64</b>	<b>64-bit</b>	tar.gz (sha256) 176M
<b>Linux/AArch64</b>	<b>64-bit</b>	tar.gz (sha256) 177M
<b>Linux/x64</b>	<b>64-bit</b>	tar.gz (sha256) 179M
	<b>Source</b>	Tags are jdk-17+35, jdk-17-ga

- C. En una terminal, cambiarse al directorio de descarga y descomprimir el archivo.

```
tar -xvf openjdk-<version>_linux-x64_bin.tar.gz
```

- D. Mover la carpeta al directorio **/opt**. En esta carpeta es donde generalmente se realiza la instalación de software adicional.

```
sudo mv jdk-<version> /opt
```

- E. Configurar las variables de entorno **JAVA\_HOME** y **PATH**, agregar las siguientes líneas en el archivo **/etc/bash.bashrc**

```
#Entorno Java
export JAVA_HOME=/opt/jdk-<version>
export PATH=${JAVA_HOME}/bin:${PATH}
```

- F. Para que la configuración tome efecto de forma segura, reiniciar sesión. Ejecutar el siguiente comando para verificar resultados.

```
java -version
openjdk version "<version>" yyyy-mm-dd
OpenJDK Runtime Environment XX.X (build 11+28)
OpenJDK 64-Bit Server VM XX.X (build XX+XX, mixed mode)
```

### 5.3. Instalaciones de Neo4J

- A. Ejecutar los siguientes comandos para añadir el repositorio de las versiones disponibles de Neo4J.

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key | sudo apt-key add -
```

Se deberá mostrar un OK en la salida.

```
echo 'deb https://debian.neo4j.com stable latest' | sudo tee -a  
/etc/apt/sources.list.d/neo4j.list
```

```
sudo apt-get update
```

- B. El siguiente comando muestra las versiones disponibles de Neo4J.

```
apt list -a neo4j
```

- C. Habilitar el repositorio **universal**.

```
sudo add-apt-repository universe
```

- D. Instalando Neo4J Community Edition.

```
sudo apt-get install neo4j=1:5.6.0
```

### 5.4. Iniciando Neo4J.

- A. El siguiente comando inicia Neo4J.

```
sudo -E neo4j start
```

Se deberá mostrar el siguiente mensaje.

```
Starting Neo4j.  
Started neo4j (pid:8593). It is available at http://localhost:7474  
There may be a short delay until the server is ready.
```

B. El siguiente comando se utiliza para verificar el estatus de neo4j.

```
sudo -E neo4j status
```

Se deberá mostrar la siguiente salida.

```
Neo4j is running at pid 8593
```

## 5.5. Autenticando en Neo4J

Para crear un usuario se utilizará Cypher Shell.

```
cypher-shell -u <username> -p <password>
```

Ejemplo

```
cypher-shell -u neo4j
```

El password debe contar con más de 8 caracteres  
Se obtendrá una salida similar a la siguiente.

```
Connected to Neo4j using Bolt protocol version 5.0 at neo4j://localhost:7687 as  
user neo4j.  
Type :help for a list of available commands or :exit to exit the shell.  
Note that Cypher queries must end with a semicolon.  
neo4j@neo4j>
```

El **neo4j** de la izquierda del prompt es el nombre del usuario con el que nos conectamos y el **neo4j** de la derecha es el nombre de la base de datos creada por default.

Para salir de Cypher Shell.

```
:exit
```

## 5.6. Creando nodos

Para crear un nodo se utilizan `( )`, dado que se parece a un nodo. Para hacer referencia a un nodo se le puede asignar una variable, se recomienda ser específico con el nombre de

las variables ya que para realizar consultas a dicho nodo se necesitar el nombre de la variable.

También se le pueden asignar etiquetas a los nodos para agruparlos dado cierto patrón, haciendo referencia a una tabla en SQL. Por lo que se recomienda usar estas etiquetas.

### Sintaxis

```
([nombre_variable]:<nombre_etiqueta>)
```

### Ejemplo:

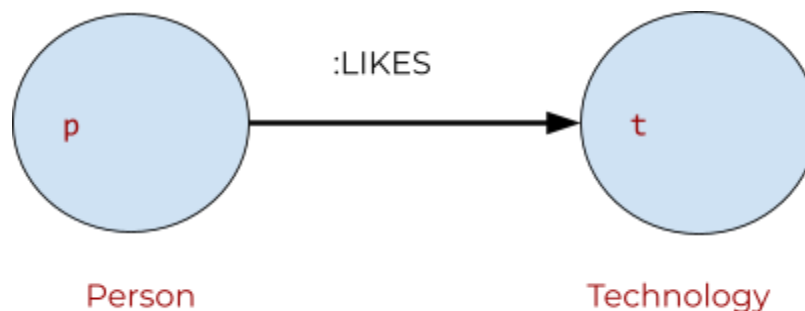
```
(p:Persona)  
(w:Company)  
(t:Technology)
```

## 5.7.Creando relaciones

La relación entre nodos se representa con flechas  $\rightarrow$   $\leftarrow$ , tener cuidado con la dirección, también se pueden representar relaciones bidireccionales con  $--$

### Ejemplo:

```
CREATE (p:Person)-[:LIKES]->(t:Technology)
```



Observar que lo que se coloca entre corchetes es la etiqueta que se le coloca a la relación.

## 5.8. Agregando atributos

Tanto los nodos como las relaciones pueden tener atributos. Por lo general estos atributos van entre `{}`.

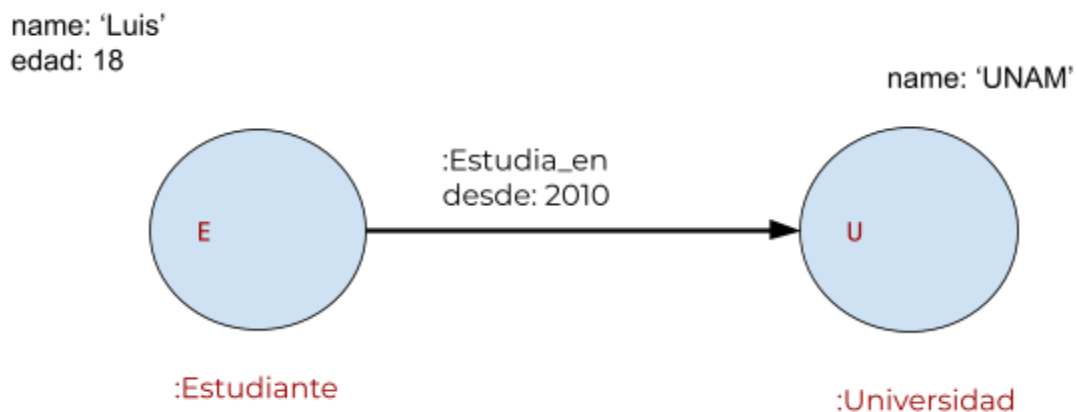


## Sintaxis

```
([nombre_variable]: <nombre_etiqueta> {<nombre_atributo>:<valor>, ...})
```

## Ejemplo

```
CREATE (E:Estudiante {name:'Luis', edad:20})-[r:Estudia_en {desde: 2010}]->(U:Universidad {name:'UNAM'});
```



## 5.9. Utilizando match

MATCH se utiliza para obtener datos de los nodos, las relaciones o patrones, para ello se utilizarán variables para hacer referencia al nodo o relación.

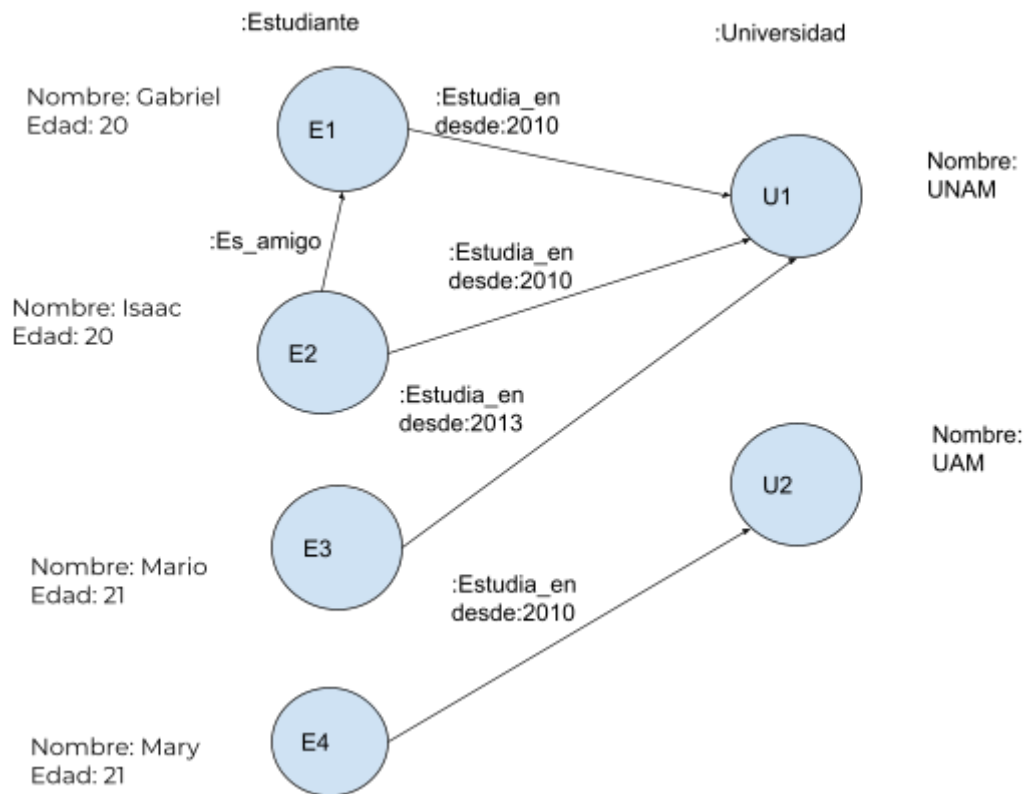
Para recuperar la información anterior.

```
MATCH (Q:Estudiante)-[rel:Estudia_en {desde: 2010}]->(U:Universidad {name:'UNAM'}) return Q;
```

La consulta anterior se puede interpretar como muestra a todos los estudiantes que estudian en la unam desde 2010.

Observar el return Q al final, la variable Q hace referencia a los nodos con la etiqueta :Estudiante que cumplan con la condición, si se quiere mostrar los datos de la universidad colocar la variable U en el return.

## Ejemplo



Se recomienda primero empezar a crear los nodos para no generar confusión, se puede crear los nodos junto con la relación en una sola instrucción pero puede llegar a ser algo confuso. No colocar punto y coma hasta el final del código.

```
CREATE(e1:Estudiante {Nombre:'Gabriel', Edad:20})
CREATE(e2:Estudiante {Nombre:'Isaac', Edad:20})
CREATE(e3:Estudiante {Nombre:'Mario', Edad:21})
CREATE(e4:Estudiante {Nombre:'Mary', Edad:21})
```

```
CREATE(u1:Universidad {Nombre:'UNAM'})
CREATE(u2:Universidad {Nombre:'UAM'})
```

```
CREATE(e1)-[:Estudia_en {desde:2010}]->(u1)
CREATE(e2)-[:Estudia_en {desde:2010}]->(u1)
CREATE(e3)-[:Estudia_en {desde:2013}]->(u1)
CREATE(e4)-[:Estudia_en {desde:2010}]->(u2)
CREATE(e2)-[:Es_amigo]->(e1);
```

Se pueden realizar consultas utilizando los atributos de los nodos o de las relaciones. Si se quiere mostrar los datos de todos los alumnos que estudian en la UNAM sin importar la fecha de inicio.

```
MATCH(Q:Estudiante)-->(U:Universidad {Nombre:'UNAM'}) RETURN Q;
```

Si se quiere mostrar solo la edad de los alumnos.

```
MATCH(Q:Estudiante)-->(U:Universidad {Nombre:'UNAM'}) RETURN Q.edad as edad;
```

Si se desea consultar a los amigos de Isaac.

```
MATCH(q:Estudiante {Nombre:'Isaac'})-[:Es_amigo]->(l:Estudiante) return l.Nombre as amigo;
```

Eliminando los nodos y sus relaciones.

```
Match(q:Estudiante) DETACH DELETE q;  
Match(q:Universidad) DETACH DELETE q;
```

## 5.10. Actualizando atributos

Si se quiere agregar o modificar algún atributo de un nodo se puede utilizar MATCH, MATCH busca a todos los nodos que cumplan con un predicado.

### Ejemplo

Creando el nodo.

```
CREATE (:Persona {Nombre:'Mario'});
```

Agregando un atributo.

```
MATCH (p:Persona {Nombre:'Mario'})  
SET p.birthdate=date('1980-01-01');
```

## 5.11. Realizando consultas con un rango de valores.

En cuanto a consultas a parte de utilizar la relación o poner los atributos a los nodos en la cláusula MATCH se puede utilizar la cláusula WHERE parecido a SQL.

```
MATCH(p:Estudiante)
WHERE p.Nombre STARTS WITH 'G'
RETURN p.Nombre;
```

```
MATCH(p:Estudiante)
WHERE p.Edad >=19 AND p.Edad<21
RETURN p.Nombre, p.Edad;
```

## 5.12. Neo4j Desktop

Es una aplicación gráfica que corre del lado del cliente empleada para trabajar con Neo4J. Permite realizar conexiones a servidores tanto de forma local como remota.

Neo4j Desktop es empleado para configuraciones de un solo servidor. Para configuraciones Multi máquina basados en servicios en la nube existe una herramienta llamada Neo4j Aura.

Neo4j se emplea para administrar bases de datos locales creadas dentro de una instancia DBMS. También se pueden realizar conexiones remotas, pero solo para efectos de visualización, no para administrar.

Neo4J proporciona diversas herramientas gráficas empleadas principalmente para construir aplicaciones que trabajan con bases de datos orientadas a grafos. Algunos ejemplos de estas herramientas: **Neo4j Browser**, **Neo4J Bloom**.

- Browser se emplea para ejecutar comandos, consultas sobre los datos empleando Cypher así como la visualización de los resultados.
- Bloom se emplea para visualizar grafos empleando otras fuentes de búsqueda

### 5.12.1. Instalación básica

Requisitos para realizar la instalación de Neo4J Desktop:

- Java 17+
- Descargar Neo4j Desktop del enlace <https://neo4j.com/download-center/#desktop>
- Ejecutar la aplicación

```
./neo4j-desktop-1.5.7-x86_64.AppImage
```

### 5.12.2. Proyectos en Neo4j Desktop

Un proyecto representa una carpeta de desarrollo en la máquina del cliente. En esta carpeta (dentro del proyecto) es posible crear bases de datos locales, agregar archivos, etc.

Cada DBMS dentro de un proyecto contiene o define una lista de bases de datos.

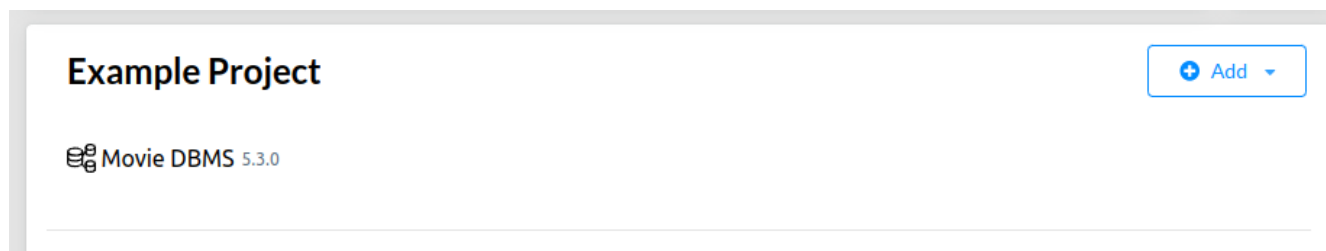
Un DBMS representa a una instancia de un Servidor Neo4j con al menos una base de datos llamada **system** y una base de datos por default llamada **neo4j**. La base de datos por default puede tener un nombre diferente.

Cada DBMS puede extender su funcionalidad a través de plugins. La siguiente lista puede ser habilitada por cada DBMS local:

- APOC: Librería empleada para definir procedimientos y funciones que permitan la automatización de tareas de distintos tipos, por ejemplo, integración de datos, algoritmos para grafos, conversión de datos.
- GDS Library ofrece una lista de algoritmos comunes para trabajar con Grafos.
- Neo4J Streams Permite integrar aplicaciones Kafka con Neo4j

### 5.12.3. Interactuando con DBMS Movie

Neo4j Desktop Incluye un DBMS de ejemplo llamado DBMs Movie.



Seleccionar “Movie DBMS” e iniciarla. Ejecutar los siguientes comandos , analizarlos y revisar los resultados tanto en modo texto como de forma gráfica.

Ejemplo 1:

```
MATCH (m:Movie)
RETURN m
```

Ejemplo 2:

```
MATCH (p:Person)
RETURN p
```

Ejemplo 3:

```
MATCH (p:Person {name: 'Keanu Reeves'})  
RETURN p
```

Ejemplo 4:

```
MATCH (p:Person {name: 'Tom Hanks'})-[r:ACTED_IN]->(m:Movie)  
RETURN m.title, r.roles
```

Ejemplo 5:

```
MATCH (p:Person)  
RETURN p  
LIMIT 1
```

Ejemplo 6:

```
MATCH (:Person {name: 'Tom Hanks'})-[:DIRECTED]->(movie:Movie)  
RETURN movie
```

Ejemplo 7

```
MATCH (p:Person {name: 'Tom Hanks'})-[:DIRECTED]->(m:Movie)  
RETURN p,m;
```

Ejemplo 8

```
MATCH (p:Person {name: 'Tom Hanks'})  
CREATE (m:Movie {title: 'Cloud Atlas', released: 2012})  
CREATE (p)-[r:ACTED_IN {roles: ['Zachry']}]>(m)  
RETURN p, r, m
```

Ejemplo 9

```
MATCH (p:Person)-[r:ACTED_IN]->(m:Movie)  
WHERE p.name =~ 'K.+' OR m.released > 2000 OR 'Neo' IN r.roles  
RETURN p, r, m;
```

Ejemplo 10

```
MATCH (p:Person)-[:ACTED_IN]->(m)  
WHERE NOT (p)-[:DIRECTED]->()  
RETURN p, m;
```

## Ejemplo 11


```



MATCH (actor:Person)-[r:ACTED_IN]->(movie:Movie)
RETURN actor.name AS name, type(r) AS type, movie.title AS title
UNION
MATCH (director:Person)-[r:DIRECTED]->(movie:Movie)
RETURN director.name AS name, type(r) AS type, movie.title AS title


```

Crear un nuevo DBMS llamado **ejemplos DBMS**, posteriormente crear una nueva BD llamada **ejemplos**. Ejecutar las siguientes instrucciones, mostrar y analizar resultados.

## Example Project

[+ Add](#)
 Movie DBMS 5.3.0

 ejemplos DBMS 5.3.0 ● ACTIVE
 system

 neo4j (default)

[+ Create database](#)
[Refresh](#)

## Creación de objetos

```

CREATE (diana:Person {name: "Diana"})-[:LIKES]->(query:Technology {type: "Query Languages"})
CREATE (melissa:Person {name: "Melissa", twitter: "@melissa"})-[:LIKES]->(query)
CREATE (dan:Person {name: "Dan", twitter: "@dan", yearsExperience: 6})-[:LIKES]->
  (etl:Technology {type: "Data ETL"})<-[:LIKES]-(melissa)
CREATE (xyz:Company {name: "XYZ"})<-[:WORKS_FOR]-
  (sally:Person {name: "Sally", yearsExperience: 4})-[:LIKES]->
  (integrations:Technology {type: "Integrations"})<-[:LIKES]-(dan)
CREATE (sally)<-[:IS_FRIENDS_WITH]-
  (john:Person {name: "John", yearsExperience: 5, birthdate: "1985-04-04"})-[:LIKES]->
  (java:Technology {type: "Java"})
CREATE (john)<-[:IS_FRIENDS_WITH]-
  (jennifer:Person {
    name: "Jennifer", twitter: "@jennifer", yearsExperience: 5, birthdate:
    "1988-01-01"})-[:LIKES]->(java)
CREATE (john)-[:WORKS_FOR]->(xyz)
CREATE (sally)<-[:IS_FRIENDS_WITH]-(jennifer)-[:IS_FRIENDS_WITH]->(melissa)

```

```

CREATE (joe:Person {name: "Joe", birthdate: "1988-08-08"})-[:LIKES]->(query)
CREATE (mark:Person {name: "Mark", twitter: "@mark"})
CREATE (ann:Person {name: "Ann"})
CREATE (x:Company {name: "Company X"})<-
  [:WORKS_FOR]-(diana)<-[:IS_FRIENDS_WITH]-(joe)-[:IS_FRIENDS_WITH]->
  (mark)-[:LIKES]->(graphs:Technology {type: "Graphs"})<-
  [:LIKES]-(jennifer)-[:WORKS_FOR]->(:Company {name: "Neo4j"})
CREATE (ann)<-[:IS_FRIENDS_WITH]-(jennifer)-[:IS_FRIENDS_WITH]->(mark)
CREATE (john)-[:LIKES]->
  (:Technology {type: "Application Development"})<-
  [:LIKES]-(ann)-[:IS_FRIENDS_WITH]->(dan)-[:WORKS_FOR]->(abc:Company {name: "ABC"})
CREATE (ann)-[:WORKS_FOR]->(abc)
CREATE (a:Company {name: "Company A"})<-
  [:WORKS_FOR]-(melissa)-[:LIKES]->(graphs)<-[:LIKES]-(diana)
CREATE (:Technology {type: "Python"})<-
  [:LIKES]-(:Person {name: "Ryan"})-[:WORKS_FOR]->(:Company {name: "Company Z"})

```

Ejecutar y analizar los siguientes ejemplos:

#### Ejemplo 1

```

MATCH (p:Person)-[r:IS_FRIENDS_WITH]->(friend:Person)
WHERE exists((p)-[:WORKS_FOR]->(:Company {name: 'Neo4j'}))
RETURN p, r, friend

```

#### Ejemplo 2

```

MATCH (p:Person)-[r:IS_FRIENDS_WITH]->(friend:Person)
WHERE exists((p)-[:WORKS_FOR]->(:Company {name: 'Neo4j'}))
RETURN p, r, friend;

```

#### Ejemplo 3

```

MATCH (p:Person)-[r:IS_FRIENDS_WITH]->(friend:Person)
WHERE EXISTS {
  MATCH (p)-[:WORKS_FOR]->(:Company {name: 'Neo4j'})
}
RETURN p, r, friend

```

#### Ejemplo 4

Obtener a las personas que trabajan para una compañía cuyo nombre inicie con 'Company' y que les guste al menos una tecnología que haya sido del gusto de al menos 3 personas. No interesa mostrar de qué tecnologías se tratan.



```

MATCH (person:Person)-[:WORKS_FOR]->(company)
WHERE company.name STARTS WITH "Company"
AND EXISTS {
  MATCH (person)-[:LIKES]->(t:Technology)
  WHERE COUNT { (t)-[:LIKES]-() } >= 3
}
RETURN person.name as person, company.name AS company;

```

## Ejemplo 5

### Índices y constraints

```

CREATE (forrestGump:Movie {title: 'Forrest Gump', released: 1994})
CREATE (robert:Person:Director {name: 'Robert Zemeckis', born: 1951})
CREATE (tom:Person:Actor {name: 'Tom Hanks', born: 1956})
CREATE (tom)-[:ACTED_IN {roles: ['Forrest']}]->(forrestGump)
CREATE (robert)-[:DIRECTED]->(forrestGump)

CREATE INDEX example_index_1 FOR (a:Actor) ON (a.name)

MATCH (actor:Actor {name: 'Tom Hanks'})
RETURN actor

SHOW INDEXES

CREATE CONSTRAINT constraint_example_1 FOR (movie:Movie) REQUIRE movie.title IS
UNIQUE
CREATE CONSTRAINT constraint_example_1 ON (movie:Movie) ASSERT movie.title IS
UNIQUE Deprecated

```

## 5.13. Importar datos en Neo4J

Existen varios mecanismos:

- LOAD CSV Cypher:
- Neo4j-admin database import
- Neo4j ETL tool
- Kettle import tool

Ejemplos:

```
LOAD CSV FROM "file:///data.csv"
LOAD CSV FROM "file:///northwind/customers.csv"
LOAD CSV FROM 'https://data.neo4j.com/northwind/customers.csv'
LOAD CSV WITH HEADERS FROM
'https://docs.google.com/spreadsheets/d/<yourFilePath>/export?format=csv'
```

Ejemplos con diversas opciones

```
Id,Name,Location,Email,BusinessType
1,Neo4j,San Mateo,contact@neo4j.com,P
2,AAA,,info@aaa.com,
3,BBB,Chicago,,G
```

```
//quitar nulos
LOAD CSV WITH HEADERS FROM 'file:///companies.csv' AS row
WITH row WHERE row.Id IS NOT NULL
MERGE (c:Company {companyId: row.Id});

// limpiar datos
MATCH (n:Company) DELETE n;

//valores por default
LOAD CSV WITH HEADERS FROM 'file:///companies.csv' AS row
MERGE (c:Company {companyId: row.Id, hqLocation: coalesce(row.Location,
"Unknown")})

//limpiar datos
MATCH (n:Company) DELETE n;

//uso de cadenas vacías
LOAD CSV WITH HEADERS FROM 'file:///companies.csv' AS row
MERGE (c:Company {companyId: row.Id})
SET c.emailAddress = CASE trim(row.Email) WHEN "" THEN null ELSE row.Email END
```

Ejemplo con procesamiento de relaciones

```
Id,Name,Skills,Email
1,Joe Smith,Cypher:Java:JavaScript,joe@neo4j.com
2,Mary Jones,Java,mary@neo4j.com
3,Trevor Scott,Java:JavaScript,trevor@neo4j.com
```

```
LOAD CSV WITH HEADERS FROM 'file:///employees.csv' AS row
MERGE (e:Employee {employeeId: row.Id, email: row.Email})
WITH e, row
UNWIND split(row.Skills, ':') AS skill
MERGE (s:Skill {name: skill})
MERGE (e)-[r:HAS_EXPERIENCE]->(s)
```

Ejemplo con compañías

```
MATCH (n:Company) DELETE n;

LOAD CSV WITH HEADERS FROM 'file:///companies.csv' AS row
WITH row WHERE row.Id IS NOT NULL
WITH row,
(CASE row.BusinessType
WHEN 'P' THEN 'Public'
WHEN 'R' THEN 'Private'
WHEN 'G' THEN 'Government'
ELSE 'Other' END) AS type
MERGE (c:Company {companyId: row.Id, hqLocation: coalesce(row.Location, "Unknown")})
SET c.emailAddress = CASE trim(row.Email) WHEN "" THEN null ELSE row.Email END
SET c.businessType = type
RETURN *
```

## 5.14. Otras características de las bases de datos orientadas a grafos

### 5.14.1. Consistencia

Debido a que las bases de datos orientadas a grafos operan sobre nodos interconectados, la mayoría de las implementaciones generalmente no soportan la distribución de sus nodos en múltiples máquinas. *Infinity Graph* es una implementación que sí soporta esta característica.

Dentro de un mismo nodo, los datos *siempre* son consistentes. Neo4J implementa las propiedades ACID de las transacciones.

Cuando Neo4J corre en un cluster, se realiza la escritura en un nodo llamado nodo master y posteriormente se realiza la sincronización con otros nodos llamados nodos esclavos.

La consistencia en bases de datos se implementa a través del concepto de una transacción. No permiten el registro de relaciones incompletas. El inicio y fin de una relación siempre deben existir. Un nodo puede ser eliminado únicamente si este no tiene relaciones con otros.

### 5.14.2.Transacciones

Antes de realizar un cambio a un nodo o antes de agregar nuevas relaciones se deberá iniciar una transacción. Ejemplo en Java:

```
Transaction tx = database.beginTx();
try {
    Node node = database.createNode();
    node.setProperty("nombre", "NoSQL");
    node.setProperty("añoPublicacion", 2012);
    tx.success();
} finally {
    transaction.finish();
}
```

- Para que la transacción sea confirmada, adicional de invocar a `success`, la transacción debe ser terminada al invocar al método `finish`.

### 5.14.3.Disponibilidad

Neo4J entre otras soportan alta disponibilidad a través del uso de replicación en nodos esclavos. Un nodo esclavo puede permitir escrituras. En este caso, la escritura se debe sincronizar con el nodo master. La transacción primero debe confirmarse en el nodo master y posteriormente en el nodo esclavo. Tiempo después, el cambio será reflejado en otros nodos esclavos.

Neo4J hace uso de Apache ZooKeeper para llevar el control del último transaction ID persistido en cada nodo esclavo y su correspondiente nodo master. Cuando Neo4J se inicia, se establece comunicación con Zookeeper para determinar quién es el nodo master. Si el nodo master se cae o no está disponible, se selecciona a otro nodo.

### 5.14.4. Funcionalidades para realizar consultas

Existen diversos lenguajes empleados en bases de datos orientadas a grafos:

- Gremlin
- Cypher para Neo4J

Los atributos de un nodo o de una relación pueden ser indexados haciendo uso de algún servicio de indexado. En Java la creación de un índice se muestra en el siguiente código:

```
Transaction tx = graphDB.beginTx();
try {
    Index<Node> nodeIndex= graphDB.index().forNodes("nodes");
    nodeIndex.add(martin, "name", martin.getProperty("name"));
    transaction.success();
}
finally {
    transaction.finish();
}
```

- Neo4J utiliza Lucene como servicio de indexado. Este mismo servicio se emplea para realizar búsquedas en texto.

#### 5.14.5. Escalabilidad

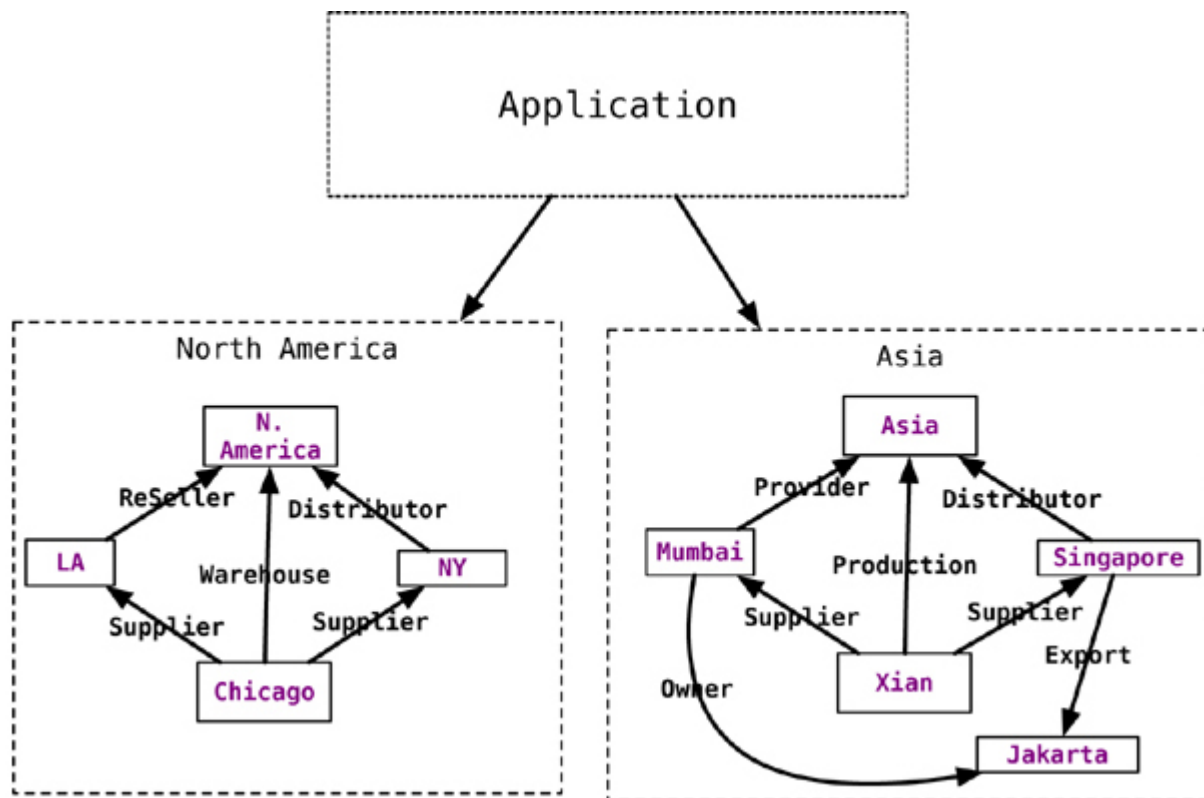
Como se ha revisado en otras bases de datos NoSQL, la estrategia principal para implementar escalabilidad es a través del concepto de sharding.

Sharding en una BD orientada a grafos es complicado para implementar. Una BD orientada a grafos se considera **orientada a relaciones**, no orientada a agregaciones como en el caso de otras soluciones.

Debido a que un nodo puede asociarse con cualquier otro, resulta mucho mejor almacenar esos nodos en el mismo servidor para simplificar su recorrido al momento de realizar una búsqueda y ofrecer mejor desempeño.

A pesar de esta restricción, es posible implementar escalabilidad empleando las siguientes técnicas:

- Almacenar todo el conjunto de nodos y relaciones completamente en RAM. Esto requiere una mayor cantidad de memoria RAM en los servidores.
- Otra técnica es agregar más nodos llamados **nodos esclavos** que permiten únicamente leer datos, todas las escrituras se realizan en el nodo principal, o **nodo master**. Esta técnica representa a un patrón *writing once and reading from many servers*.
  - Esta técnica se emplea para grandes cantidades de datos que pueden ser replicados en diferentes nodos.
- Si la técnica anterior no es adecuada, entonces se puede emplear *sharding*. La forma en la que se reparten los nodos es específica para cada aplicación. Por ejemplo, distribuir nodos con base al valor de su atributo *ubicación* (norte, sur, etc).



#### 5.14.6. Casos de uso

Redes sociales representando relaciones como:

- Amistad
- Empleados
- Clientes
- Conocimiento compartido
- En general relaciones llamadas *Link-rich domain*. Pueden existir múltiples dominios en una misma BD., por ejemplo, una red social, una red de comercio, etc.

Revisar casos de estudio y recomendaciones de uso:

- Recomendaciones de uso: <https://neo4j.com/use-cases/>
- Casos reales donde se ha empleado Neo4J: <https://neo4j.com/case-studies/>

Otro ejemplo: Routing , Dispatcher, servicios basados en ubicaciones.

- Cada ubicación que define una dirección de entrega representa un nodo.
- Todos los nodos en los cuales se realiza la entrega de un paquete se representan por un grafo.
- Las relaciones entre estos nodos pueden tener atributos como: distancia, de tal forma que permita identificar la mejor ruta para realizar entregas eficientes.

- Distancia y ubicación pueden ser empleadas como atributos para realizar recomendaciones de buenos restaurantes, opciones de entretenimiento.
- Nodos también pueden representar puntos de venta, para notificar a los usuarios cuando se encuentren cerca de cada uno de estos nodos.

#### 5.14.7. Máquinas generadoras de recomendaciones

Tanto nodos como relaciones pueden ser empleadas para realizar recomendaciones como: *“Tus amigos también compran este producto”, “Cuando un producto se incluye en una factura, estos otros productos también se han incluido”*.

Conforme los datos crecen en la BD, el número de nodos y las relaciones disponibles para hacer recomendaciones también se incrementan de manera considerable. Este comportamiento puede ser aplicado también para realizar minería de datos:

¿qué productos siempre se venden juntos? , ¿qué productos se facturan juntos?. En algunos casos, si estas condiciones no se cumplen, se pueden levantar alarmas. Estos patrones o comportamientos pueden ser empleados para detectar fraudes.