

INFORME TÉCNICO DEL PRIMER PROYECTO
GESTIÓN Y MODELACIÓN DE DATOS

INTEGRANTES

SEBASTIÁN TORO FRANCO
CRISTIAN STEVEN OSORIO PAZ
WILLIAM AGUIRRE ZAPATA

FECHA ENTREGA

29/09/2018

PRESENTADO A

MARÍA CONSTANZA PABÓN BURBANO

PONTIFICIA UNIVERSIDAD JAVERIANA

CALI - VALLE DEL CAUCA



BASE DE DATOS NoSQL NEO4J

Neo4j es un software libre de Base de datos NoSQL orientada a grafos, implementado en Java. Neo4j fue desarrollado por Neo Technology, una startup sueca con base en Malmö y San Francisco Bay Area en Estados Unidos. El Consejo de administración de Neo Technology consta de: Magnus Christerson (Vicepresidente de Intentional Software Corp.), Nikolaj Nyholm (CEO de Rosa polar), Sami Ahvenniemi (socio de Conor Venture Partners) y Johan Svensson (CTO de Neo Technology) [1].

Los desarrolladores describen a Neo4j como un motor de persistencia embebido, basado en disco, implementado en Java, completamente transaccional, que almacena datos estructurados en grafos en lugar de tablas. La versión 1.0 de Neo4j fue lanzada en febrero de 2010 [2]. La base de datos está licenciada en un modelo dual, tanto bajo Affero General Public License (AGPL) v3 como bajo licencia comercial. Se encuentra sólo en idioma Inglés, su última versión actualizada es “Neo4j 3.5.0-alpha09”, lanzado el 14 de septiembre de 2018 [3]; compatible con múltiples lenguajes de programación y actualmente disponible en multiplataformas como Linux, Windows y MacOS.

PASOS DE INSTALACIÓN Y USO DE LA HERRAMIENTA

Neo4j cuenta con varios tipos de versiones de las cuales se explicará brevemente en qué consiste cada una y se procederá a profundizar en la instalación de la más acorde para el proyecto.

Neo4j Desktop: Aplicación para administrar instancias locales de Neo4j. La descarga gratuita incluye la licencia de Neo4j Enterprise Edition. También se conectará a sus servidores de producción y eventualmente también instalará otros componentes como los algoritmos de grafos o las actualizaciones de Java [4].

Community Server: Ideal para aprender Neo4j y proyectos de bricolaje más pequeños que no requieren altos niveles de escalamiento o servicios profesionales y de soporte.

Enterprise Server: Se recomienda cuando las aplicaciones tengan grafos grandes o requieran altos niveles de rendimiento, escalabilidad, seguridad, integridad de datos o disponibilidad, brindando así a los desarrolladores el diseño, desarrollo, mantenimiento y potencia operacional que necesitan para aplicaciones de grafos de clase empresarial [5].

Neo4j Bloom (próximamente): Herramienta de visualización para usuarios empresariales que no requiere ningún código o habilidades de programación para ver y analizar datos.

REQUERIMIENTOS DEL SISTEMA

Esta descripción en general son los requisitos del sistema para ejecutar Neo4j en un entorno de producción.

	Mínimo	Recomendado (Sugerencias de Neo4j)
Procesador	Intel Core i3	Intel Core i7
Memoria	2GB	16- 32 GB o más
Disco	10GB SATA	SSD con SATA

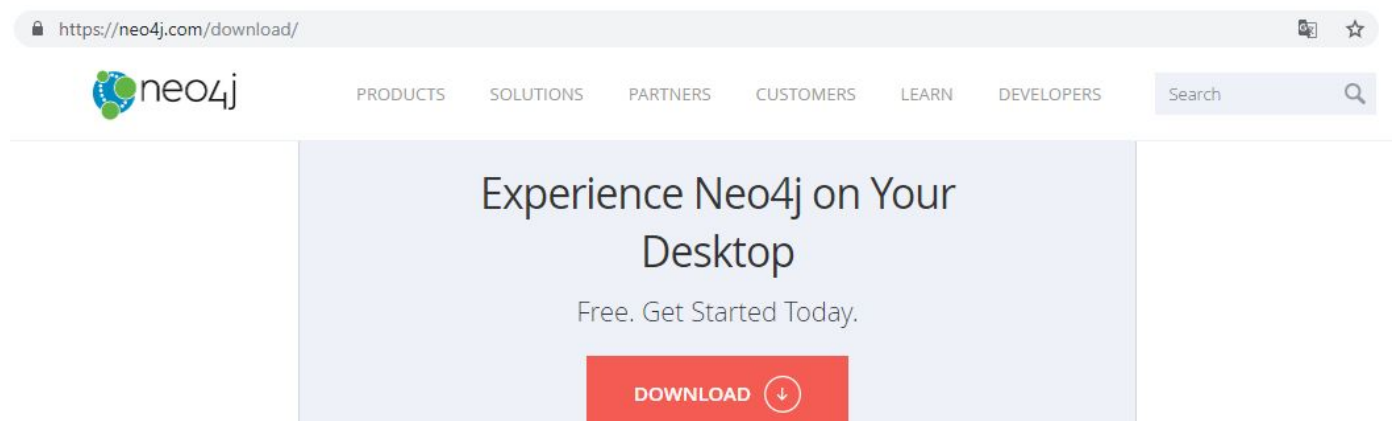
Software Neo4j requiere una Máquina Virtual Java, JVM , para operar. Los instaladores de Community Edition para Windows y Mac incluyen una JVM por conveniencia. Todas las demás distribuciones, incluidas todas las distribuciones de Neo4j Enterprise Edition, requieren una JVM preinstalada [6].

OpenJDK 8 Oracle Java 8 IBM Java 8

Sistemas Operativos Ubuntu 14.04, 16.04; Debian 8, 9; CentOS 6, 7; Fedora, Red Hat, Amazon Linux, Windows Server 2012, 2016.

INSTALACIÓN DE LA BASE DE DATOS LOCAL NEO4J EN WINDOWS

Vaya a <https://neo4j.com/download/> y siga las instrucciones para instalar Neo4j Desktop en su computadora por medio de un ejecutable .exe



INSTALACIÓN DE LA BASE DE DATOS NEO4J EN LINUX DEBIAN

Descargue la última versión de <https://neo4j.com/download-center/#releases>

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

Current Releases

Enterprise Server	Community Server	Neo4j Desktop
Neo4j Desktop 1.1.10		
OS	Download	
Mac	Neo4j Desktop (dmg)	
Linux	Neo4j Desktop (ApplImage)	
Windows	Neo4j Desktop (exe)	

Seleccione la pestaña *Neo4j Desktop*, después clic en *Neo4j Desktop (ApplImage)*.

Enseguida de descargado el archivo use `chmod x+ <filename>`

De esta manera tendrá una instalación portable y local de Neo4j en Linux.

INSTALACIÓN DE LA BASE DE DATOS EN SERVIDOR VPS LINUX DEBIAN

Usar los siguientes comandos por medio de una Terminal del repositorio de Neo4j

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

```
echo 'deb https://debian.neo4j.org/repo stable/' | sudo tee  
/etc/apt/sources.list.d/neo4j.list
```

```
sudo apt-get update
```

Para instalar Neo4j Community Edition:

```
sudo apt-get install neo4j
```

Para instalar Neo4j Enterprise Edition:

```
sudo apt-get install neo4j-enterprise
```

INSTALACIÓN DEL MÓDULO DE NEO4J PARA PYTHON EN LINUX DEBIAN

Requisitos tener instalado Python 3.6 (1) y Pip3 (2)(sistema de gestión de paquetes utilizado para instalar y administrar paquetes de software escritos en Python 3.6).

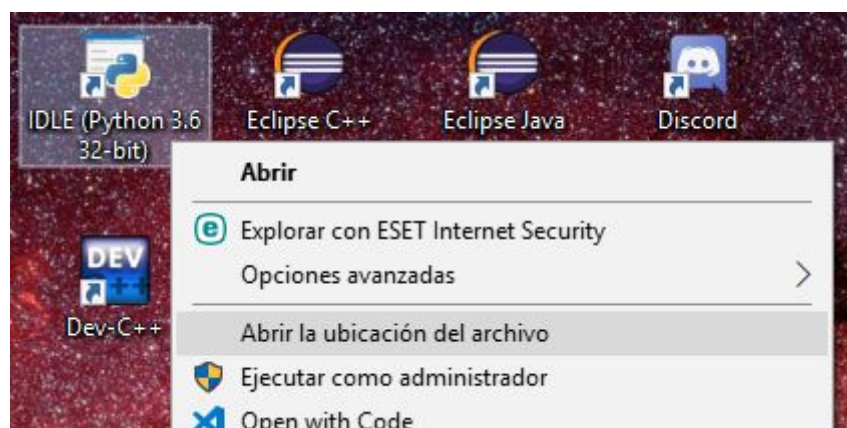
1	<code>sudo apt-get install python3.6</code>
2	<code>sudo apt-get install python3-pip</code>

Instalar el controlador de Neo4j Python (3) y Py2neo (4) (biblioteca cliente y un completo juego de herramientas para trabajar con Neo4j desde las aplicaciones de Python y desde la línea de comandos).

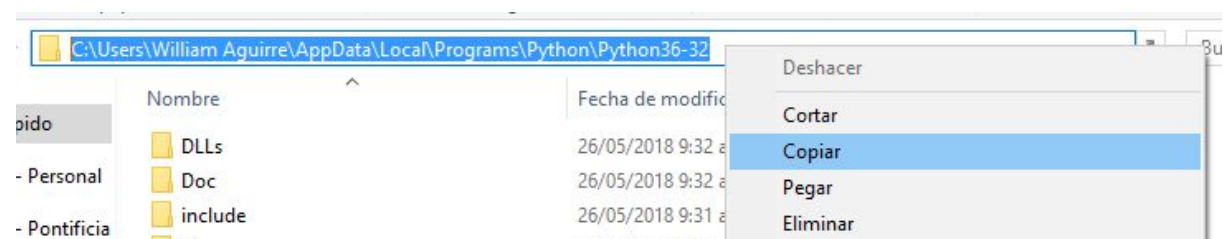
3	<code>pip3 install neo4j-driver</code>
4	<code>pip3 install py2neo</code>

INSTALACIÓN DEL MÓDULO DE NEO4J PARA PYTHON EN WINDOWS

Una vez instalado Python 3.6 tenemos que ubicarnos en la carpeta donde se instaló Python, dando clic derecho en el acceso directo y en “Abrir la ubicación del archivo”



Copiamos la dirección de la ruta de la carpeta donde se encuentra la instalación de Python 3.6.



Abrimos un intérprete de comandos de Windows como CMD o PowerShell y escribimos “*cd*” seguido de Ctrl+v para pegar la ruta anterior y damos ENTER.

```
cmd: Símbolo del sistema
Microsoft Windows [Versión 10.0.17134.286]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.
C:\Users\William Aguirre>cd C:\Users\William Aguirre\AppData\Local\Programs\Python\Python36-32_
```

Una vez dentro de la ubicación de la carpeta en el CMD, escribimos “*pip install neo4j-driver*” y damos ENTER.

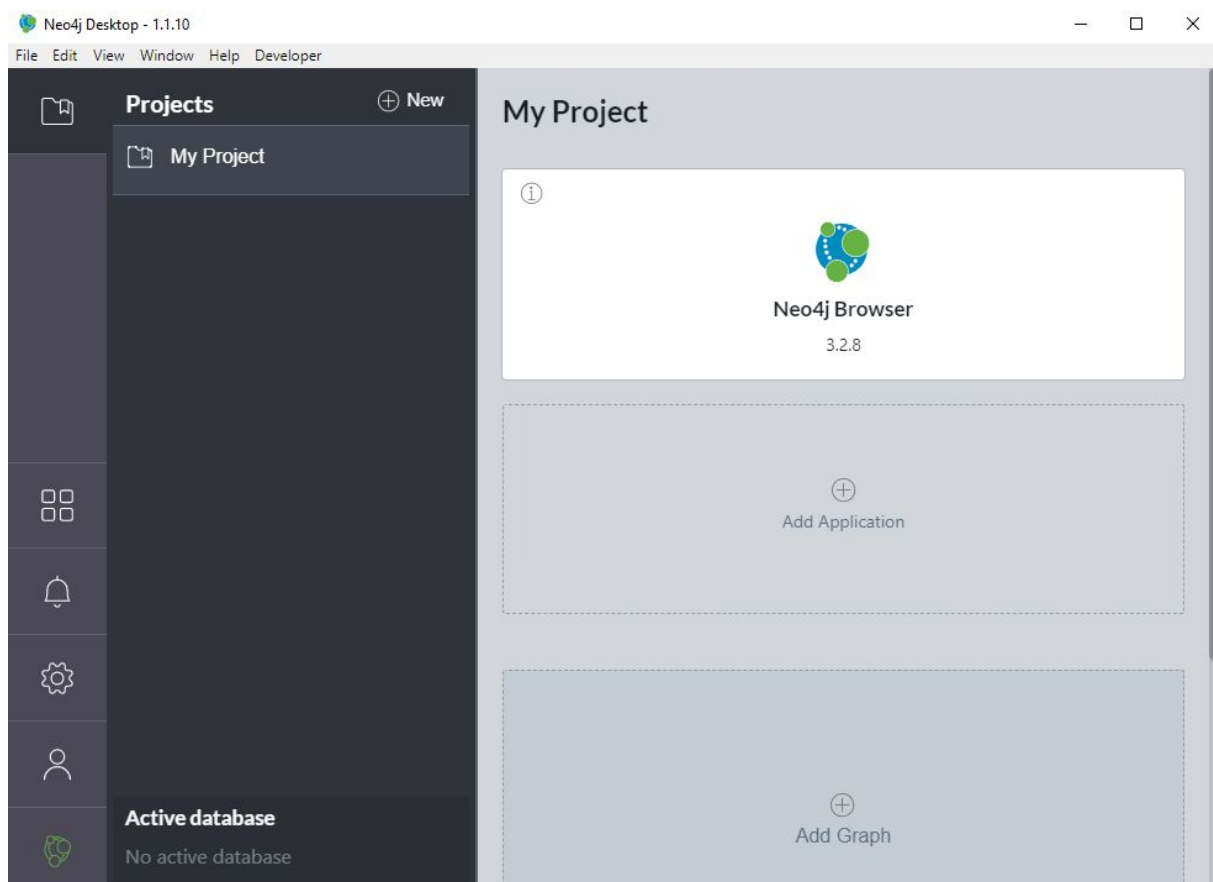
```
C:\Users\William Aguirre\AppData\Local\Programs\Python\Python36-32>pip install neo4j-driver
```

Terminada la instalación escribimos “*pip install pyneo*” y damos ENTER.

```
C:\Users\William Aguirre\AppData\Local\Programs\Python\Python36-32>pip install py2neo
```

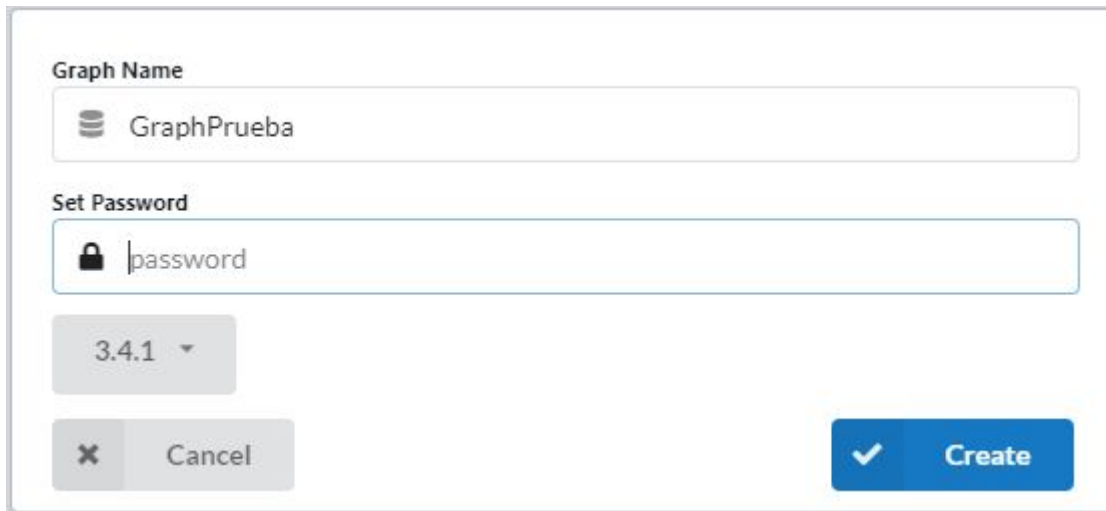
CREACIÓN Y CONFIGURACIÓN DE UNA BASE DE DATOS EN NEO4J

Una vez instalada exitosamente Neo4j, su base de datos y su módulo para Python, ejecutamos en Windows el programa “Neo4j Desktop” o en Linux el .ApplImage y esta es la primera pantalla del programa, damos clic en “Add Graph” para crear una base de datos nueva.

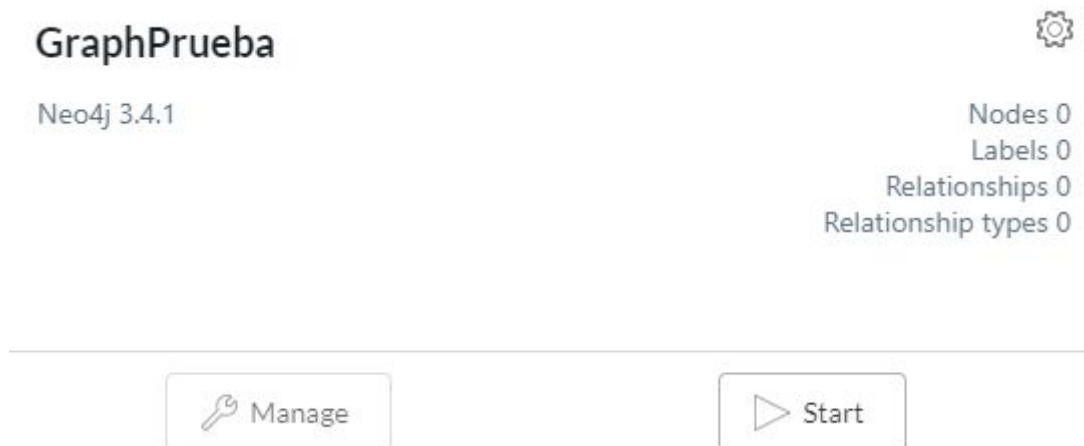


Seguido damos clic en “Create a Local Graph”

Se abrirá dos campos de texto donde digitamos el nombre de la base de datos y su contraseña, también podemos seleccionar que versión de Neo4j queramos, pero este campo lo dejamos por defecto y damos clic en “Create”.

A screenshot of the Neo4j 'Create' dialog box. It features a 'Graph Name' field with a database icon and the text 'GraphPrueba'. Below it is a 'Set Password' field with a lock icon and the text 'password'. A version dropdown menu shows '3.4.1'. At the bottom, there are three buttons: a grey 'Cancel' button with an 'X' icon, a blue 'Create' button with a checkmark icon, and a grey button with an 'X' icon.

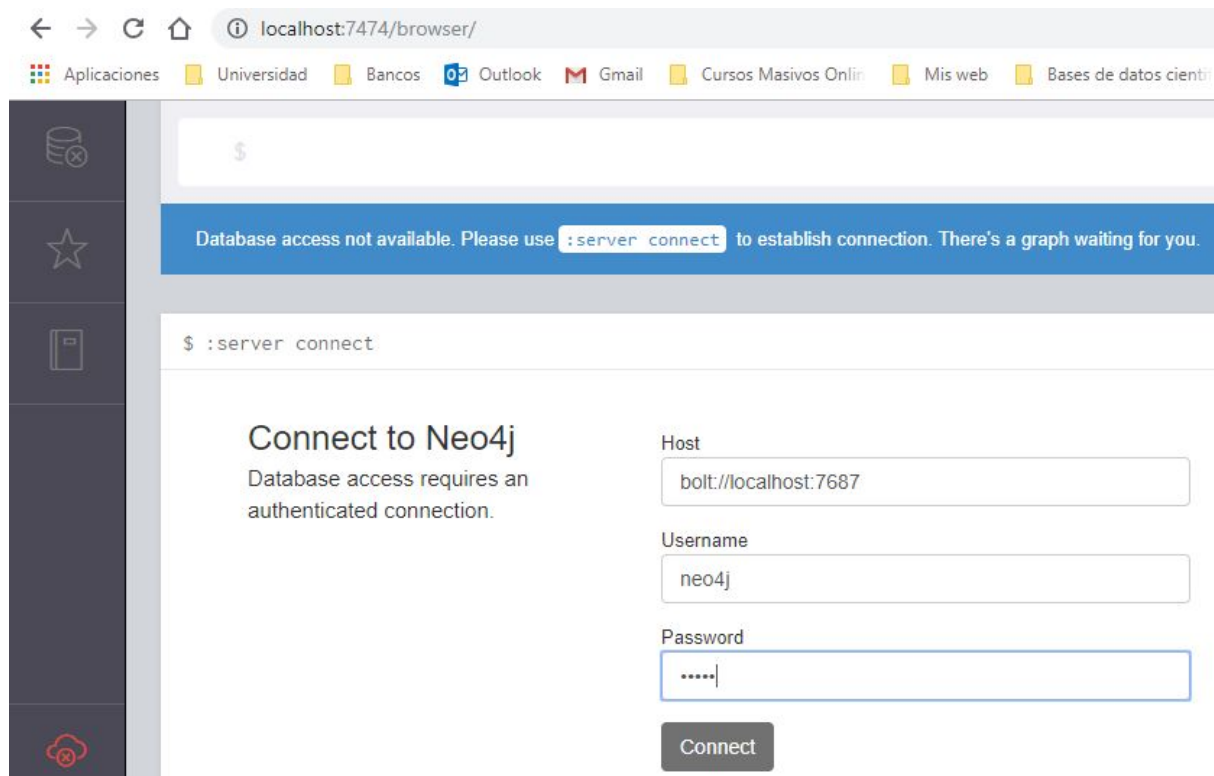
Una vez creada la base de datos sale este menú donde damos clic en “Start” para empezar a desarrollar y trabajar en nuestra base de datos.

A screenshot of the Neo4j dashboard for a database named 'GraphPrueba'. The title 'GraphPrueba' is at the top left, with a gear icon to its right. Below the title, it says 'Neo4j 3.4.1'. On the right side, there are statistics: 'Nodes 0', 'Labels 0', 'Relationships 0', and 'Relationship types 0'. At the bottom, there are two buttons: a grey 'Manage' button with a key icon and a grey 'Start' button with a play icon.

Para acceder a desarrollar e interactuar con la base de datos, debemos acceder a un navegador web, y acceder a la url: <http://localhost:7474/browser/> y establecer la conexión con los datos de autenticación, dados por el El host por defecto [bolt://localhost:7687](http://localhost:7687), Username: [neo4j](#), Password asignado a la hora de la creación de base de datos GraphPrueba y damos clic en “Connect”.

Cabe destacar de que si estamos trabajando en una red de una VPS o Servidor con asignación de IP fija, podremos acceder a la base de datos de neo4j reemplazando la palabra “localhost” por la dirección ip del equipo en que esté instalado el motor de base de datos, en cualquier parte del mundo.

Ejemplo: <http://194.182.87.136:7474/browser/>

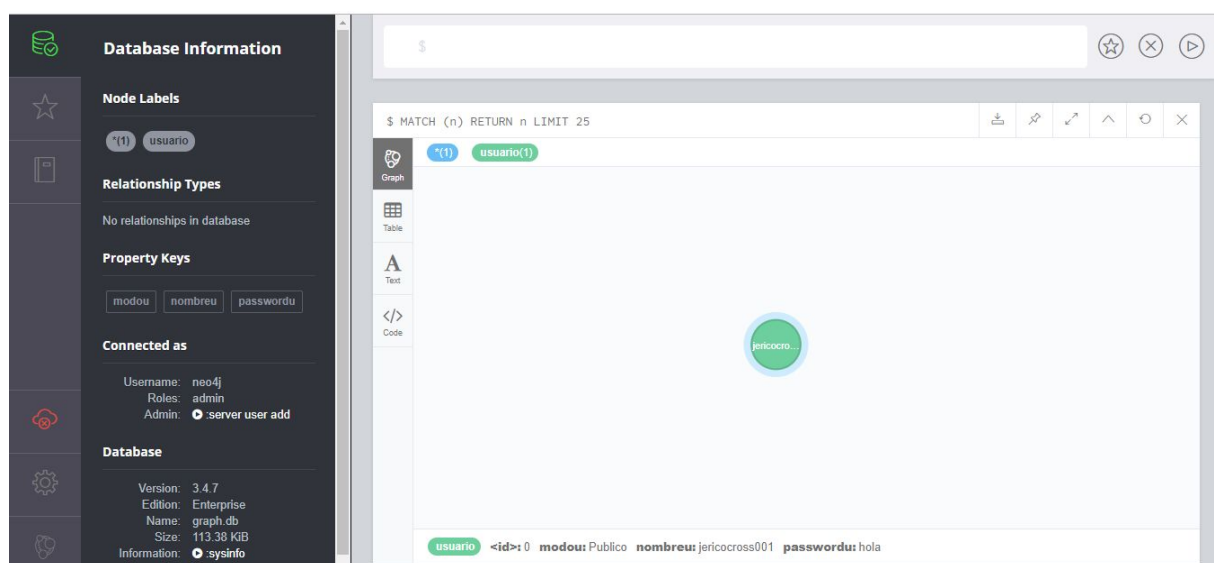


Después de establecer conexión y acceso exitoso a neo4j a través de la web, este será nuestro modo de codificar la base de datos. La barra superior donde aparece el signo pesos “\$” es el bash de neo4j que interpretará a Cypher (es un lenguaje propio de Neo4j para realizar consultas. Es la forma declarativa de realizar consultas).

NOTA: Cypher es a Neo4j lo que T-SQL es a Sql Server [7].



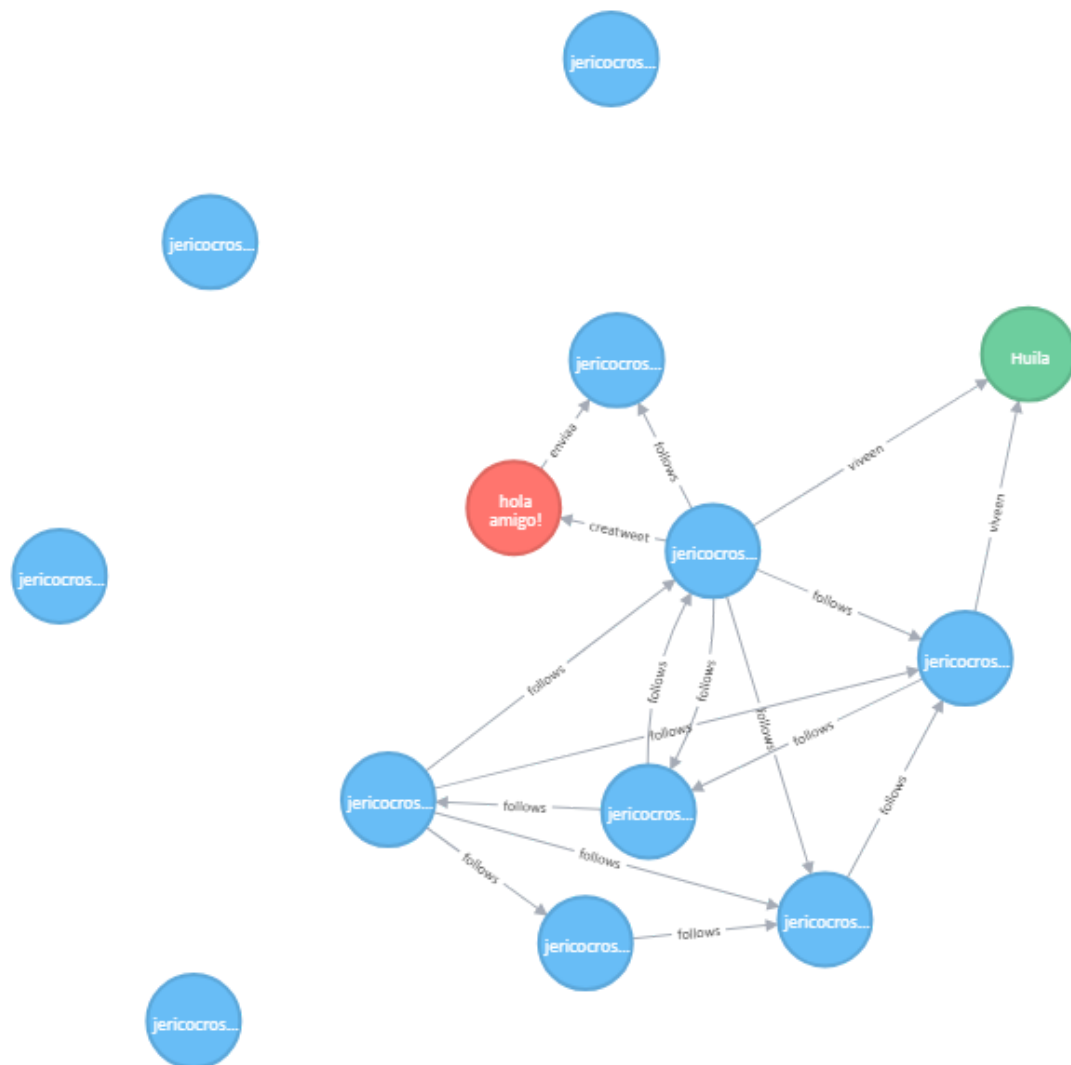
Introducimos un código de muestra para demostrar la creación de un nodo en Neo4j



Para visualizar todos los nodos realizamos la consulta "MATCH (n) RETURN n"

MODELO DE DATOS PROPUESTO

El modelo propuesto para la creación de la base de datos se encuentra disponible en la siguiente dirección: <http://194.182.87.136:7474/browser/> Username: neo4j Password: bases12345 Utiliza la siguiente consulta para visualizar toda la base de datos "MATCH (n) RETURN n"



CONEXIÓN DE NEO4J CON PYTHON

En la construcción del proyecto en python "proyecto.py" se debe agregar

```
from py2neo import Graph, Node, Relationship
```

Librería de py2neo, aquella que permite definición de cada componente de esta y conexión con la base de datos

```
m=Graph("http://194.182.87.136:7474",host="194.182.87.136",password="bases12345")
```

Se realiza la conexión a la base de datos local con atributos con contraseña, esto extrae un grafo al cual llamamos m.

FUNCIONES Y CONSULTAS DEL PROYECTO EN NEO4J A TRAVÉS DE PYTHON

En el proyecto se realizaron las siguiente funciones a manera especial:

- **Ver mis seguidores:** función que se encarga de mostrar los seguidores del usuario, imprimiendolos linea por linea.
- **Ver usuarios seguidos:** función que se encarga de mostrar los usuarios que sigue el usuario, imprimiendolos linea por linea.
- **Personas que quizás conozcas:** función que se encarga de mostrar los seguidores consecuentes que sean aceptados, es decir, el estado de la relación de seguimiento sea 1 en el 5 nivel.
- **Ver mis tweets:** función que tiene como parámetro un objeto usuario para a partir de su nombre obtener todos los tweets en los que aparece como creador.
- **Realizar tweet:** función que a partir del objeto usuario y nodo del usuario, se encarga de realizar las transacciones debidas en la base de datos para crear un tweet y definir concretamente su origen y destinatarios.
- **Ver tweets recibidos:** función que tiene como parámetro un objeto usuario para a partir de su nombre obtener todos los tweets en los que aparece como destinatario.
- **Explorar:** función que tiene como parámetro un objeto usuario para obtener aquellos a los que sigue y poder mostrar los últimos 5 tweets realizados por estos.
- **Buscar usuario:** función que tiene como parámetros un nombre a buscar(str) y el usuario como nodo y objeto; se dedica hacer la búsqueda de un usuario dado en el grafo y dar viabilidad a acciones tales como ver sus tweets en caso de que este sea público y agregarlo como amigo o mandarle una solicitud de amistad dependiendo también de sus ajustes de privacidad.
- **Cambiar ajustes de privacidad:** función que se encarga de cambiar los ajustes de privacidad del usuario(Privado ó Público), buscando su nodo y modificando su atributo de modou en alternancia al previo.
- **Cambiar o establecer ciudad:** función que tiene como parámetros el un objeto usuario y su respectivo nodo, y permite mostrar la ciudad registrada por el usuario, y da opcion de cambiarla, en caso de tener una previamente asignada elimina la relación entre estos dos; a la ciudad ingresada se le realiza una búsqueda de preexistencia para evitar redundancia de datos y en tal caso de no existencia se crea su nodo y relaciona con el usuario, de otra forma solo los relaciona.

- **Manejo de solicitudes de amistad:** función que se encarga de mostrar las solicitudes de amistad pendientes que el usuario posee, se muestra un listado de los usuarios que las realizan y opción de aceptarlas, al aceptarlas se realiza una búsqueda de su relación para cambiar sus propiedades y volverlo un seguidor mas.

A lo largo del desarrollo del proyecto se utiliza de manera recurrente el uso de la función `Graph.run().evaluate()` y `Graph.run().data()`, las cuales por medio de la librería `py2neo` realizan comandos de cypher en la base de datos designada, respectivamente estas funciones retornan el primer nodo con coincidencia de la consulta y una lista de diccionarios para referirse a todos los nodos que coinciden; a su vez se utiliza constantemente transacciones para la creación de nuevos datos en la base de datos.

Obtener el nodo al cual le pertenece un nombre igual al proporcionado por el usuario para iniciar sesión:

```
MATCH (a:usuario) WHERE a.nombreu="" + Nombre + "" RETURN a
```

Obtener el nombre de la ciudad en la que vive el usuario por medio de su nombre, retornado como Nombre:

```
MATCH (a:usuario)-[:viveen]->(b:ciudad) WHERE
a.nombreu="" + user.getUsername() + "" RETURN b.nombre AS Nombre
```

Obtener los nombres de aquellas cuentas que han enviado solicitud de amistad a el usuario, es decir, el estado de la relación seguir es igual a 0:

```
MATCH (b:usuario)-[:follows {estado:['0']}]>(a:usuario) where
a.nombreu="" + user.getUsername() + "" RETURN b.nombreu AS Nombre")
```

Obtener aquellas cuentas que son seguidos por el usuario, es decir, el estado de la relacion seguir es igual a 1:

```
MATCH (a:usuario)-[:follows {estado:['1']}]>(b:usuario) where
a.nombreu="" + user.getUsername() + "" RETURN b.nombreu AS Nombre
```

Obtener los nombres de aquellas cuentas que siguen a el usuario, es decir, el estado de la relación seguir es igual a 1:

```
MATCH (b:usuario)-[:follows {estado:['1']}]>(a:usuario) where
a.nombreu="" + user.getUsername() + "" RETURN b.nombreu AS Nombre
```

Obtener los nombres de aquellas cuentas que se encuentran en un quinto nivel despues del usuario en seguimiento aceptado:

```
MATCH (a:usuario)-[:follows {estado:['1']}]>(b:usuario)-[:follows
{estado:['1']}]>(c:usuario)-[:follows {estado:['1']}]>(d:usuario)-[:follows
```

```
{estado:['1']}->(e:usuario)-[:follows {estado:['1']}]>(f:usuario) WHERE  
a.nombreu="" + user.getUsername() + "" RETURN f.nombreu AS nombre
```

Transacción realizada para crear una relación de seguimiento donde la variable o depende de la privacidad de dic, es decir si se le manda solicitud o no:

```
tx=m.begin()  
ab = Relationship(node, "follows", dic)  
ab['estado']=str(o)  
tx.create(ab)  
tx.commit()
```

Retorna los tweets que han sido creados por una cuenta con nombre dado:

```
MATCH (a:usuario)-[:creatweet]->(b:tweet) where a.nombreu="" + nombre + ""  
RETURN b
```

Retorna la relación entre dos usuarios de nombre dado:

```
MATCH (b:usuario)-[r:follows {estado:['1']}]>(a:usuario) WHERE  
a.nombreu="" + nombre + "" and b.nombreu="" + node['nombreu'] + "" RETURN r
```

Retorna el nodo completo de un usuario con nombre dado:

```
MATCH (a:usuario) WHERE a.nombreu="" + nombre + "" RETURN a
```

Retorna el nodo completo de una ciudad con nombre dado:

```
MATCH (a:ciudad) WHERE a.nombre="" + str(ciudad) + "" RETURN a
```

Elimina la relación entre un usuario y ciudad de nombre dados:

```
MATCH (a:usuario)-[r:viveen]->(b:ciudad) WHERE  
a.nombreu="" + user.getUsername() + "" and b.nombre="" + user.getciudad() + "" DELETE  
r
```

Búsqueda del usuario con nombre proporcionado, al cual posteriormente se le hace cambio de la propiedad "modou"(privacidad):

```
MATCH (a:usuario) WHERE a.nombreu="" + user.getUsername() + "" SET a.modou  
="" + user.getusermode() + ""
```

Obtener el nodo completo del tweet creado por un usuario de nombre dado:

```
MATCH (a:usuario)-[:creatweet]->(t:tweet) WHERE a.nombreu = "" + x + "" RETURN t
```

Obtener los tweets que han sido remitidos al usuario:

```
MATCH (a)-[:enviaa]->(b) WHERE b.nombreu = '"+user.getUsername()+" RETURN  
a
```

Cambia la propiedad estado de la relación follows de 0 a 1(es seguidor):

```
MATCH (b:usuario)-[r:follows {estado:['0']}]>(a:usuario) WHERE  
a.nombreu='"+user.getUsername()+" and b.nombreu='"+user.getinvitations()[0-1]+"  
SET r.estado =['1']
```

DEFINICIÓN Y CARGA DE LA BASE DE DATOS

Para poder definir de manera detallada los diferentes datos que se recibían del grafo creado en Neo4j, se recurrió la creación de objetos que permitieran acceder a los diferentes datos de las entidades en el grafo de manera fácil y rápida.

CLASE USUARIO:

La clase del usuario que tiene atributos para los datos que serán más relevantes: nombre de usuario, modo de usuario, ciudad de residencia, seguidores y a quienes sigue. Además, como es de esperarse se tienen funciones propias de la clase que permiten recuperar los datos anteriormente mencionados.

```

class usuario:
    username=""
    usermode=""
    ciudad=""
    userfollows=[]
    followers=[]
    invitations=[]
    def __init__(self, n,p):
        self.username=n
        self.usermode=p
    def setusername(self,n):
        self.username=n
    def setusermode(self):
        if(self.usermode=="Publico"):
            self.usermode="Privado"
        else:
            self.usermode="Publico"
    def getusername(self):
        return self.username
    def getusermode(self):
        return self.usermode
    def setnewfollower(self,user):
        if(not(user in self.followers)):
            self.followers.append(user)
    def getfollowers(self):
        return self.followers
    def setnewfollowed(self,user):
        if(not(user in self.userfollows)):
            self.userfollows.append(user)
    def getfollowed(self):
        return self.userfollows
    def getinvitations(self):
        return self.invitations
    def setnewinvitation(self,user):
        if(not(user in self.invitations)):
            self.invitations.append(user)
    def setciudad(self,n):
        self.ciudad=n
    def getciudad(self):
        return self.ciudad

```

VER TWEETS REALIZADOS CÓDIGO:

```

def vertweetsRealizados(user, node): #retorna todos los tweets realizados por el usuario
    dic=m.run("MATCH (a)-[:creatweet]->(b) WHERE a.nombreu='"+user.getusername()+"' RETURN b").data()
    if(dic is None):
        print("Aun no has realizado ningun tweet.\n")
        time.sleep(5)
        os.system('cls')
    else:
        for x in dic:
            for j in x.keys():
                print("tweet realizado: "+str(x[j]['texto'])+"\n"+"fecha en que se realizo: "+str(x[j]['fecha']))
+"\\n")
    if(int(input("Cerrar? \n 1:Si. \n"))):
        os.system('cls')
    else:
        os.system('cls')

```

EJECUCIÓN EN CONSOLA:

```
Bienvenido jericocross001, que desea hacer
0:Ver mis seguidores          1:Ver usuarios seguidos
2:personas que quizás conozcas 3:Ver mis tweets realizados
4:Realizar tweet              5:Ver tweets recibidos
6:Explorar mis seguidos       7:Buscar usuario
8:Cambiar ajustes de privacidad 9:Cambiar o establecer ciudad
10:Ver soicitudes de amistad  11:Cerrar sesion

3
tweet realizado: mil bendiciones
fecha en que se realizo: 01:21:33,29/09/2018

tweet realizado: adios
fecha en que se realizo: 00:40:09,29/09/2018

tweet realizado: buen dia
fecha en que se realizo: 00:39:12,29/09/2018

tweet realizado: tengo hambre
fecha en que se realizo: 24:15,25/09/2018

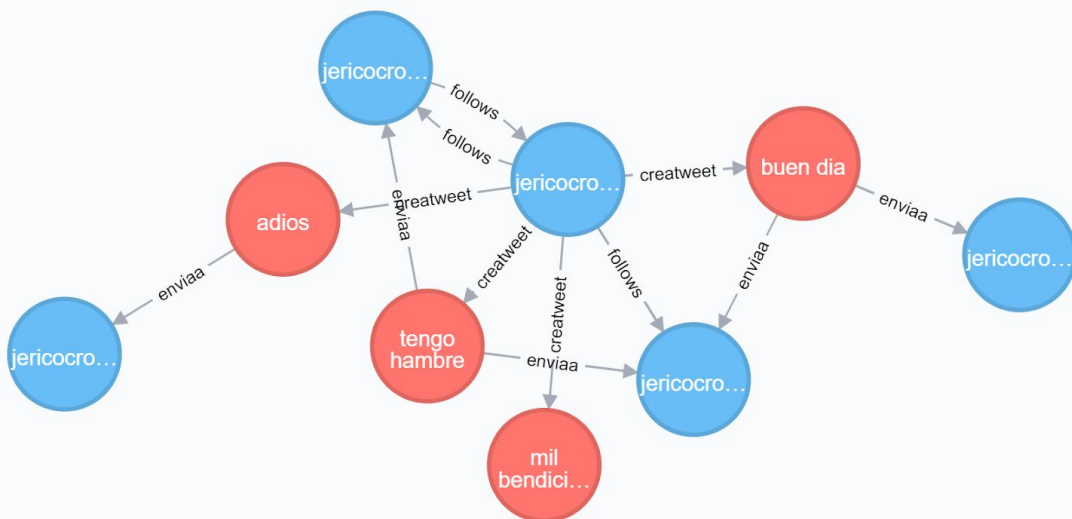
Cerrar?
1:Si.
```

GRAFO EN BASE DE DATOS:

*(9)

usuario(5)

tweet(4)



De esta forma se, logran crear las 10 instancias de las entidades que maneja la base de datos, en este caso:

- Usuario.
- Tweet.

Lo anterior a través de un SCRIPT que permite crear las diferentes entidades y relaciones que hay en la base de datos por defecto y que serán los sobre los que se trabajarán las consultas:

```
CREATE (jericocross001:usuario {nombre:'jericocross001',passwordu:'hola', modou:'Publico'})
CREATE (jericocross002:usuario {nombre:'jericocross002',passwordu:'hola', modou:'Privado'})
CREATE (jericocross003:usuario {nombre:'jericocross003',passwordu:'hola', modou:'Privado'})
CREATE (jericocross004:usuario {nombre:'jericocross004',passwordu:'hola', modou:'Publico'})
CREATE (jericocross005:usuario {nombre:'jericocross005',passwordu:'hola', modou:'Publico'})
CREATE (twet:tweet {fecha:'24:15,25/09/2018', texto:'tengo hambre'})
CREATE
  (jericocross001)-[:follows {estado:['1']}]>(jericocross005),
  (jericocross001)-[:follows {estado:['0']}]>(jericocross002),
  (jericocross001)-[:createtweet]>(twet),
  (jericocross005)-[:follows {estado:['1']}]>(jericocross001),
  (twet)-[:enviaa]>(jericocross005),
  (twet)-[:enviaa]>(jericocross002)
```

Una vez se han definido correctamente las diferentes partes que componen la bases de datos, será posible realizar consultas variadas así como modificar la bases de datos mediante la creación de nuevos usuarios, nuevos tweets o nuevas relaciones de seguimiento.

REPOSITORIO EN GITHUB DEL PROYECTO EN PYTHON

https://github.com/monowilliam/ProyectoI_BasesDeDatos_Neo4j.git

REFERENCIAS

[1]"Neo4j", es.wikipedia.org, 2018. [Online]. Available: <https://es.wikipedia.org/wiki/Neo4j> [Accessed: 25- Sep- 2018].

[2]N. Staff, "The top 10 ways to get to know Neo4j - Neo4j Graph Database Platform", Neo4j Graph Database Platform, 2018. [Online]. Available: <https://neo4j.com/blog/the-top-10-ways-to-get-to-know-neo4j/> [Accessed: 25- Sep- 2018].

[3]"Neo4j 3.5.0-alpha09 - Neo4j Graph Database Platform", Neo4j Graph Database Platform, 2018. [Online]. Available: <https://neo4j.com/release-notes/neo4j-3-5-0-alpha09/> [Accessed: 25- Sep- 2018].

[4]"Neo4j Desktop User Interface Guide - Neo4j Graph Database Platform", Neo4j Graph Database Platform, 2018. [Online]. Available:
<https://neo4j.com/developer/neo4j-desktop/>. [Accessed: 26- Sep- 2018].

[5]"What's in Neo4j Enterprise Edition? - Neo4j Graph Database Platform", Neo4j Graph Database Platform, 2018. [Online]. Available:
<https://neo4j.com/business-edge/whats-in-neo4j-enterprise-edition/>. [Accessed: 26-Sep- 2018].

[6]"2.1. System requirements - Chapter 2. Installation", Neo4j.com, 2018. [Online]. Available:
<https://neo4j.com/docs/operations-manual/current/installation/requirements/>. [Accessed: 27- Sep- 2018].

[7]J. Sánchez, "Cypher lenguaje de Neo4j - cómo crear datos", Xurxodeveloper.blogspot.com, 2018. [Online]. Available:
<http://xurxodeveloper.blogspot.com/2014/04/cypher-como-crear-datos.html>. [Accessed: 27- Sep- 2018].