PROYECTO FINAL

Universidad Veracruzana
Facultad de Estadística e Informática
Maestría en Sistemas Interactivos Centrados en el Usuario

Experiencia Educativa:
Gestion de datos para los sistemas interactivos

Docente:
M.C.C. Lorena Alonso Ramírez

Alumno:
Jácome Domínguez Jorge Luis

17 DE DICIEMBRE DE 2018

Índice

Introducción

Un sistema gestor de bases de datos (DBMS, por sus siglas en inglés) es conjunto de herramientas, a través de los cuales es posible la administración, manipulación, interrelación y consulta de colecciones datos, sirviendo tambien, como una interfaz entre el usuario y los datos (Universidad de Murcia, 2006).

En informática se conoce como dato a cualquier elemento o característica informativa con relevancia y significado para un usuario o sistema (Camps Paré, y otros, 2005). Por otro lado, una base de datos es un conjunto estructurado de datos representado por entidades y relaciones (Vélez de Guevara, 2018).

Comúnmente los DBMS implementan un lenguaje de definición de datos, a través del cual trabajan con los datos en distintos niveles, ya sea para alterarlos u obtener un determinado conjunto de ellos (Camps Paré, 2005). Estos sistemas representan a una herramienta invisible para los usuarios, que facilita la tarea de mantener una estructura y organización para los datos, asi como controlar las operaciones ejecutadas con ellos (Vélez de Guevara, 2018).

En razón de la amplia variedad de retos relacionados al almacén y consulta de datos, se han desarrollado diferentes modelos de bases de datos (Camps Paré, y otros, 2005), por ejemplo (Universidad de Murcia, 2006): el modelo Orientado a Objetos (OO) y el modelo hibrido (tambien llamado Objeto-Relacional, por su siglas, OR), por mencionar algunos. Entre todos estos modelos, se puede destacar al modelo relacional como el más utilizado en el mundo (Camps Paré, 2005).

Una base de datos relacional ordena los datos en una estructura de tablas, donde cada fila representa a un registro de datos ordenados, y las columnas representan a campos que corresponden al nombre de los datos de cada registro (Vélez de Guevara, 2018). En el modelo relacional para una base de datos, los registros representan a cada objeto de la tabla y los campos a cada atributo del objeto. En este modelo, las tablas suelen compartir al menos un campo entre ellas, lo cual sirve para establecer relaciones y apoya a la tarea de realizar consultas complejas. La idea base del modelo relacional es permitir la existencia de entidades (filas de la tabla) que mediante sus propios atributos (columnas de la tabla) puedan relacionarse con otras (Universidad de Murcia, 2006).

Los sistemas gestores para bases de datos que trabajan bajo un móldelo relacional (RDBMS, por sus siglas en inglés) al igual que otros gestores, permiten crear, actualizar y administrar una base de datos, en este caso una base de datos relacional. Para la mayoría de este tipo de sistemas, comúnmente se suele implementar un lenguaje de consulta estructurada (SQL) para acceder a los datos de la base de datos relacional (Vélez de Guevara, 2018). No obstante, tambien existen otros modelos y tipos te sistemas gestores que no implementa un lenguaje de consulta estructurada (Universidad de Murcia, 2006), a los cuales se les conoce como “NoSQL” (ACENS, 2014).

Los sistemas de bases de datos “NoSQL” o “No solo SQL”, pese a no existir una definición formal para ellos, son una amplia clase de sistemas de gestión de datos que difieren en aspectos importantes del modelo relacional (ORACLE, 2016). Este modelo de sistemas para bases de datos no es nuevo, sin embargo, carecía de la misma popularidad y variedad aplicativa que el modelo relacional. No fue hasta que algunas empresas enfrentaron problemas con la administración de sus datos por el enorme

crecimiento de la Web y la necesidad de dar respuesta al procesamiento de grandes volúmenes de datos por peticiones de un gran número de usuarios en tiempo real, que las empresas trabajaron en sistemas específicos para resolver estos problemas (Castro Romero, González Sanabria, & Callejas Cuervo, 2012), con lo cual se dio lugar a soluciones robustas que posteriormente se volvieron la cara del ecosistema de modelos y aplicaciones NoSQL (ACENS, 2014). Por tanto, los sistemas "NoSQL" se refieren a un conjunto de tecnologías que contribuyen al manejo en tiempo real de grandes volúmenes de datos no estructurados (Castro Romero, González Sanabria, & Callejas Cuervo, 2012).

Ante la amplia variedad de sistemas NoSQL es difícil determinas las propiedades y características específicas de este tipo de bases de datos. No obstante, a través de la literatura se ha logrado encasillar a este modelo bajo los siguientes atributos (Castro Romero, González Sanabria, & Callejas Cuervo, 2012):

- Escalabilidad horizontal: Refiere a la facilidad de añadir, eliminar y realizar operaciones en conjunto a elementos del sistema sin afectar el rendimiento del mismo.
- Habilidad de distribución: Indica la capacidad de soportar la replicación, distribución de los datos y propagación eventual de los cambios en los diferentes nodos del sistema (ORACLE, 2016).
- Uso eficiente de recursos de hardware: Ya que este tipo de modelos al no implementar SQL demandan menor poder de computo (ACENS, 2014).
- Libertad de esquema: En razón de no obligar al uso de un esquema rígido, se otorga la libertad para modelar, organizar y estructurar a los datos (ORACLE, 2016), lo cual facilita su integración con otros lenguajes.
- Modelo de concurrencia débil: No implementa las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) y por tanto no tiene la capacidad de realizar operaciones transaccionales.
- Consultas simples: Se dice que las consultas sobre este modelo demandan menor cantidad de operaciones y son más naturales.
- Tolerancia a fallos: Al ser sistemas pensados para la gestión de grandes volúmenes de datos, los sistemas NoSQL suelen ser altamente especializados (ORACLE, 2016), y evitan los problemas de modelos bajo SQL como los cuellos de botella por un gran número de complejas peticiones (ACENS, 2014).

Existen muchas variantes tecnológicas dentro del ecosistema NoSQL, pero a través de la literatura se ha categorizado a este modelo de acuerdo con su forma de almacenar los datos (ACENS, 2014), siendo estas categorías las siguientes vertientes de almacén (Castro Romero, González Sanabria, & Callejas Cuervo, 2012):

- Por clave-valor: Refieren a un tipo de base de datos en donde se asigna una clave única (llamada llave) a un valor, el cual comúnmente es una cadena o un objeto binario (ORACLE, 2016).
- Por documentos: Es un tipo de base de datos que guarda registros de datos estructurados jerárquicamente, y proporciona mecanismos para recuperar todo o una parte de ellos en función de su contenido.
- Por familia de columnas: Este tipo de base de datos almacena su contenido en forma de columnas, en lugar de por filas como en el modelo relacional, y a diferencia de él, en el modelo orientado a columnas no es necesario conocer todas las columnas de un registro.
- Por grafos: Se refiere a bases de datos donde la relación de sus datos puede ser representados en forma de grafo. En este tipo almacena el contenido de sus datos en una estructura de grafo, la cual

es representada con nodos (conocidos como entidades), propiedades (información sobre las entidades) y líneas (conexión entre entidades).

Entre la variedad de sistemas de bases de datos NoSQL, podemos encontrar los siguientes ejemplos (ACENS, 2014):

- Cassandra para el tipo de almacenamiento Clave-Valor.
- MongoDB y CouchBD para el tipo de almacenamiento por documentos.
- Hbase para el tipo de almacenamiento por familia de columnas (Castro Romero, González Sanabria, & Callejas Cuervo, 2012).
- InfoGrid y Neo4j para el tipo de almacenamiento por grafos.

Conforme al objetivo del proyecto, abordar el análisis del sistema Neo4j, podemos destacar el modelo de base de datos NoSQL orientado a orientado a grafos. Este tipo de bases permite almacenar la información como nodos de un grafo, junto a sus respectivas relaciones con otros nodos, haciendo valido aplicar la teoría de grafos para este tipo de sistemas (Pinilla, Bello, & Peña, 2017). Siendo su capacidad para escalar y cambiar de manera natural su tamaño y esquema de los datos, uno de los principales atributos de este tipo de bases de datos (Castro Romero, González Sanabria, & Callejas Cuervo, 2012). Además de ello, tambien se puede destacar su utilidad para guardar información con múltiples relaciones (Pinilla, Bello, & Peña, 2017).

El modelado de bases de datos orientadas a grafos implica trasladar la mayoría de los elementos claves contenidos en una realidad a un espacio concreto, es decir, describir la realidad de acuerdo a un ámbito especifico y limitado. Generalmente el modelamiento de este tipo de grafos se centra en un dominio en específico y se busca expresar la semántica de preguntas que se desea resolver (Pinilla, Bello, & Peña, 2017).

¿Qué es Neo4j?

Neo4j es una base de datos orientada a grafos (BDOG, por sus siglas en inglés) que buscar apoyar la tarea de encontrar relaciones entre los datos y extraer su verdadero valor. Esta herramienta es una de las más conocidas dentro de los sistemas orientados a grafos, se encuentra implementado en java (F. Cía, 2015) y almacena la información en una estructura de grafo formada por nodos y aristas (Zorrilla & García-Saiz, 2017). Suele ser implementada en tareas comerciales y aplicaciones prácticas (Ezamudio, 2010) como la logista de servicios de entrega, fuente de datos para la exploración de información, el análisis y la obtención de recomendaciones en tiempo real con base en datos, entre otras aplicaciones (Zorrilla & García-Saiz, 2017).

En Neo4j no se define algún tipo de esquema, con él cada nodo y relación puede tener una estructura de datos diferente, lo cual favorece la escalabilidad del mismo (Sánchez, 2014). La forma para definir datos con esta tecnología es mediante JSON (Sánchez, 2014), y permite el uso de transacciones con sus procesos, dando la posibilidad de solo si se inician y terminan las operaciones de forma satisfactoria, entonces, los cambios surtirán afecto en la base de datos (Ezamudio, 2010).

Características de Neo4j

Neo4j es un sistema NoSQL orientado a grafos, con beneficios como su flexibilidad, capacidad para realizar búsquedas entre datos masivos y su facilidad de indexación, que destacan su potencial aplicativo para problemas especializados (Lozano, 2018). Siendo las características de esta herramienta los siguientes puntos (Sánchez, 2014):

- Soporte de transacciones ACID (Lozano, 2018): Un aspecto que hace confiable a este modelo, es poder soportar que todas sus operaciones puedan ser procesadas y completadas de forma correcta (Velásquez Vargas, 2017).
- Flexibilidad de esquema (Lozano, 2018): Esta tecnología no obliga a implementar un esquema en específico, por lo contrario, permite utilizar estructuras personalizadas en los nodos, relaciones y propiedades de la base de datos (Sánchez, 2014).
- Alta disponibilidad y balanceo de lectura (Lozano, 2018): Este sistema que puede ser fácilmente distribuido en varias máquinas (Velásquez Vargas, 2017).
- Independencia del tamaño de la red: Implica que la velocidad de las operaciones para este tipo de sistemas, siempre mantendrán una velocidad constante en el intercambio de paquetes (Lozano, 2018). Tarea apoyada por un motor de almacenamiento nativo, el cual es durable y rápido (Velásquez Vargas, 2017).
- Es intuitivo: Ya que es un modelo de datos que puede ser fácilmente representado mediante un grafo (Velásquez Vargas, 2017).
- Altamente escalable: Es una tecnología pensada para el soporte de datos masivos, por lo tanto, puede tolerar el crecimiento de sus datos, con respecto miles de nodos, relaciones y propiedades que se generen en él (Velásquez Vargas, 2017).
- Expresivo: En razón, de poseer un potente lenguaje llamado Cypher (Sánchez, 2014), para la consulta de datos (Velásquez Vargas, 2017).

- Simplicidad: Ya que permite su acceso mediante una interfaz REST o una API escrita en un lenguaje orientada a objetos (Velásquez Vargas, 2017), por ejemplo, Java.

Ventajas y desventajas de Neo4j

Las capacidades aplicativas de Neo4j para ofrecer soluciones a problemas específicos y complejos, nos permite observar las ventajas y desventajas (Tabla 1) que este modelo tiene frente a diferentes retos, los cuales se encuentran relacionadas con sus propias características.

Tabla 1. Ventajas y desventajas de Neo4j (F. Cía, 2015)

Ventajas	Desventajas
• Rendimiento. Las bases de datos orientadas a grafos, como Neo4j tiene un mejor rendimiento frente a operaciones relaciones (SQL) y no relacionales (NoSQL). Ya que, a pesar de ejecutar grandes consultas, el rendimiento de Neo4j no decrece. • Agilidad. La agilidad de Neo4j para la gestión de datos es de las más grandes en sistemas de bases de datos. Si se deseara llevar al límite esta capacidad, sería necesario realizar una operación que implique superar los 34,000 millones de nodos (datos) 34,000 millones de relaciones entre dichos nodos, con 34,000 millones de variantes entre estas relaciones y 68,000 millones de propiedades. • Flexibilidad y escalabilidad. Por naturaleza las bases de datos orientadas a grafos son sistemas altamente flexibles y escalables, ya que no presentan problemas al añadir más nodos y relaciones. • Lenguaje sencillo (De la Rosa Fernández, 2015). Neo4j tiene la capacidad de trabajar con un lenguaje de consulta intuitivo y fácil de usar. • Persistencia (Sosa Olascoaga, 2012). Neo4j ofrece un sistema de almacenamiento persistente, sustentando por su capacidad de replicación y distribución de datos. • Fácil de leer (Zorrilla & García-Saiz, 2017).	• Grandes volúmenes de datos (De la Rosa Fernández, 2015). A pesar de que Neo4j es un sistema pensado para la gestión de grandes volúmenes de información, suele presentar problemas en su gestión. No obstante, esto es causado, principalmente por la mala planeación e implementación del grafo. • Alto consumo de memoria (Sosa Olascoaga, 2012). Los sistemas orientados a grafos suelen demandar un alto consumo de memoria, esto por la finalidad de ofrecer una herramienta estable y de alto rendimiento. • Herramientas sin estandarizar (Zorrilla & García-Saiz, 2017). Debido a la alta flexibilidad de los modelos orientados a grafos, las herramientas para este tipo de sistemas no se encuentran estandarizadas, por ejemplo, el lenguaje de consulta Cypher o la estructura de este tipo de sistemas. • Dependencia con un gestor (Zorrilla & García-Saiz, 2017). La lógica implementada en la construcción de los sistemas orientados a grafos se encuentra íntimamente ligada al gestor de base de datos utilizado, por lo que su migración a otro gestor no es una tarea sencilla.

De forma natural Neo4j representa los datos con una forma de grafo, lo cual facilita la tarea de leerlos. • Teoría de grafos (Zorrilla & García-Saiz, 2017). Por naturaleza los sistemas orientados a grafos aceptan la aplicación de técnicas y operaciones de la teoría de grafos.	

Descarga e Instalación de Neo4j

El proceso para descargar e instalar Neo4j es el siguiente (Neo4j, 2018):

1. Acceder a la página oficial de Neo4j e ir a la sección para descargas (Ilustración 1).

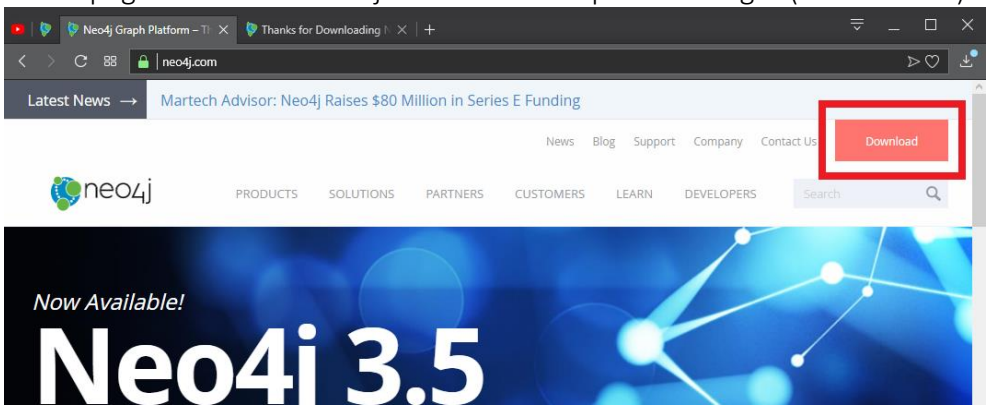


Ilustración 1. Página principal de Neo4j

2. En la sección para descargas, seleccionar la descarga del servidor Neo4j (Ilustración 2).

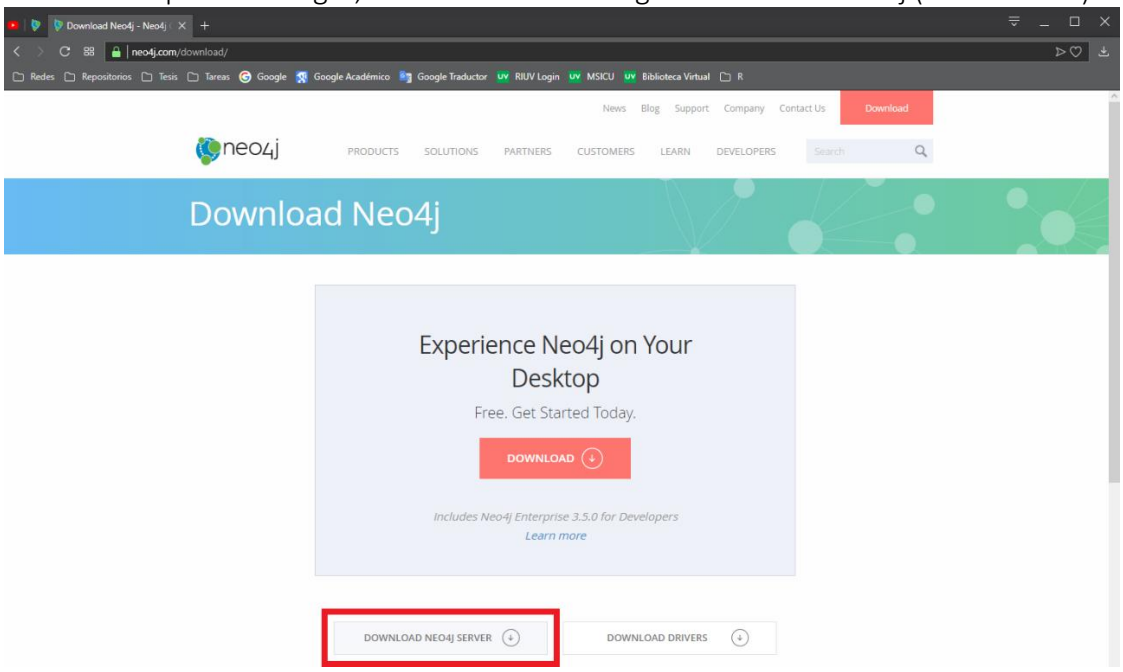


Ilustración 2. Ir a la descargar servidor Neo4j

3. Seleccionar la descarga deseada, en este caso la versión “Community Server” (Ilustración 3).

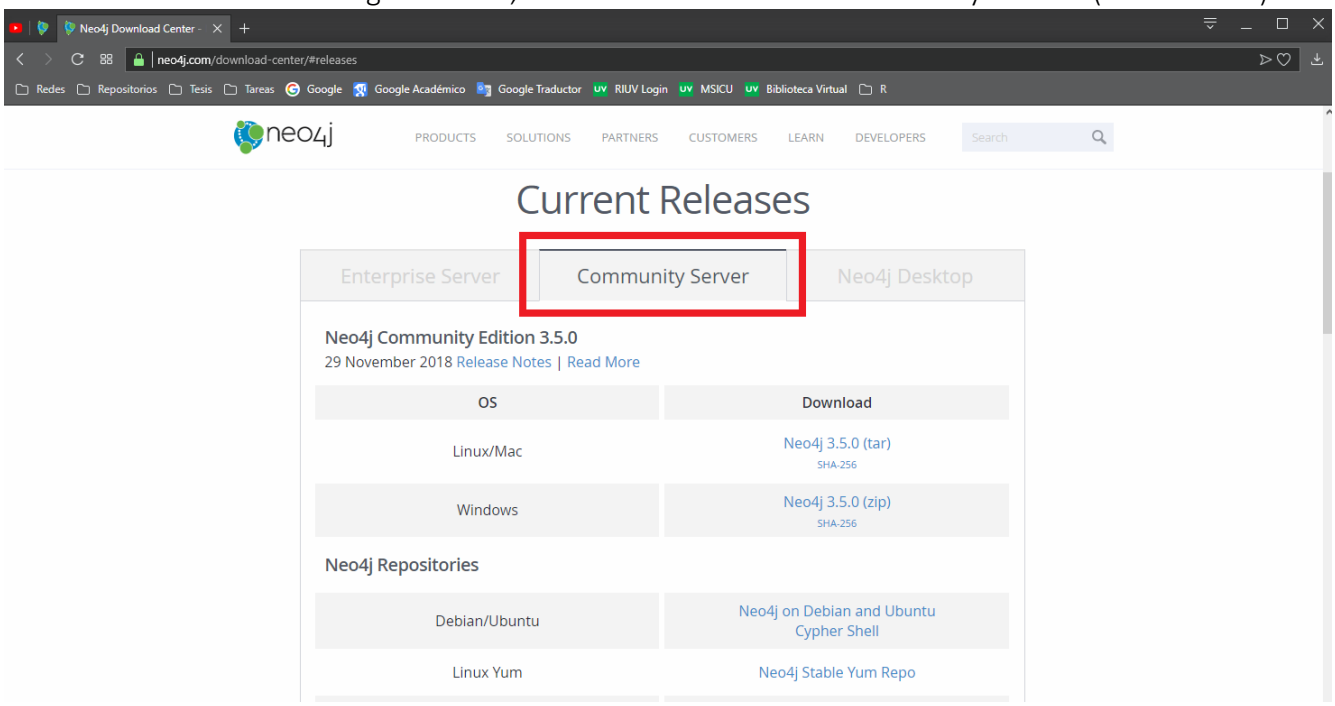


Ilustración 3. Selección del apartado “Community Server”

4. En mi caso, se llevó acabo la descarga e instalación de Java8, del cual depende Neo4j, en Ubuntu, bajo el siguiente proceso (Neo4j, 2018):
    a. Se ejecuto el siguiente comando para agregar el repositorio de Java8:
    sudo add-apt-repository ppa:webupd8team/java
    b. Se ejecuto el siguiente comando para actualizar la lista de repositorios:
    sudo apt-get update
    c. Se ejecuto el siguiente comando para instalar Java8:
    sudo apt-get install oracle-java8-installer

5. De forma similar, se llevó acabo la descarga e instalación de Neo4j en Ubuntu, de la siguiente manera (Neo4j, 2018):
    a. Se ejecuto el siguiente comando para agregar la llave del repositorio de Neo4j:
    wget -O - https://debian.neo4j.org/neotechnology.gpg.key | sudo apt-key add -
    b. Se ejecuto el siguiente comando para agregar el repositorio de Neo4j:
    echo 'deb https://debian.neo4j.org/repo stable/' | sudo tee /etc/apt/sources.list.d/neo4j.list
    c. Se ejecuto el siguiente comando para actualizar la lista de repositorios:
    sudo apt-get update
    d. Se ejecuto el siguiente comando para instalar Neo4j:
    sudo apt-get install neo4j

6. Inmediatamente después de instalar Neo4j, se realizó lo siguiente:
    a. Se ejecuto el siguiente comando para iniciar el servicio de Neo4j:
    sudo service neo4j start

b. Mediante el siguiente comando se habilito el servicio de Neo4j para iniciarse al inicio del sistema:
sudo systemctl enable neo4j

7. Se configuro Neo4j para que permitiera su acceso remoto de la siguiente forma (Neo4j, 2018):
   a. Se editó el archivo "/etc/neo4j/neo4j.conf" mediante la ejecución del comando:
   sudo nano /etc/neo4j/neo4j.conf
   b. Donde se des comentó la siguiente linea:
   dbms.connectors.default_listen_address=0.0.0.0

8. Se accedió al servicio web que ofrece Neo4j a través de la URL "http://0.0.0.0:7474/browser/" (Ilustración 4) donde "0.0.0.0" debe ser cambiado por la IP del servidor actual. Ante lo cual, se realizaron las siguientes acciones:

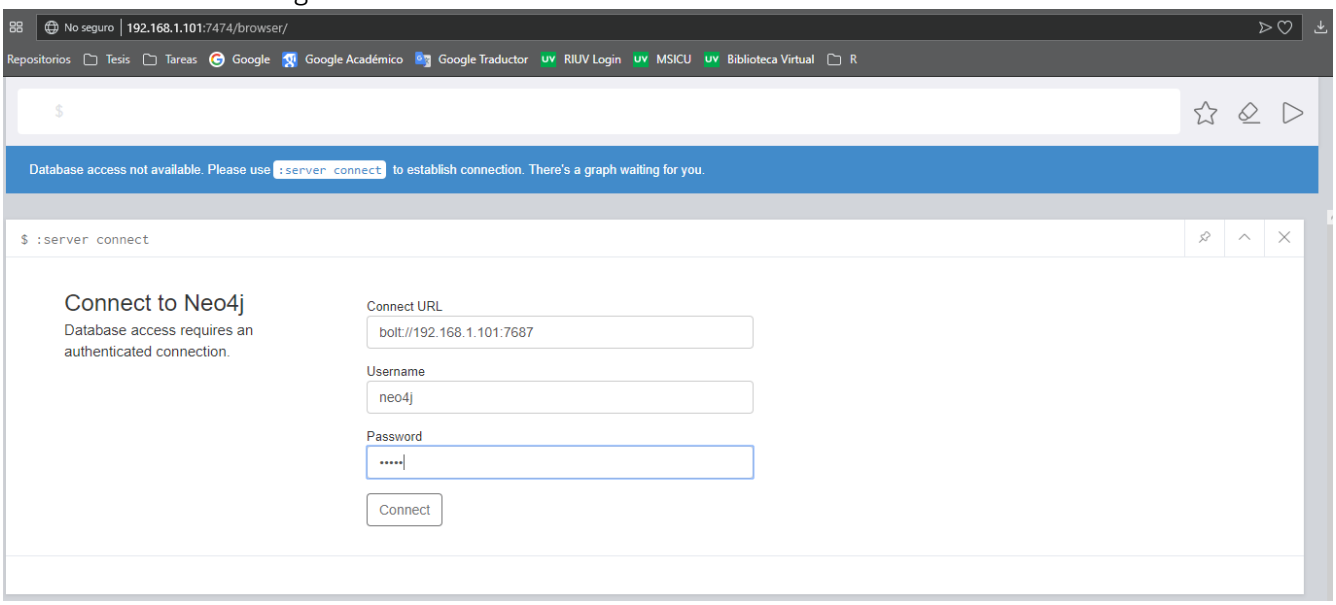


Ilustración 4. Página de inicio del servicio web que ofrece Neo4j.

   a. Cambiar la linea "bolt://localhost:7687" por "bolt://0.0.0.0:7687" donde "0.0.0.0" debe ser cambiado por la IP del servidor actual.
   b. Y fue cambiada la contraseña del administrador de Neo4j (Ilustración 5).

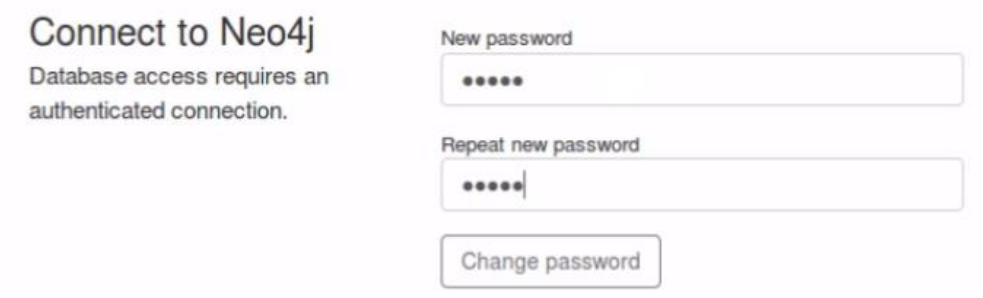


Ilustración 5. Cambio de contraseña en Neo4j

Definición de datos con Neo4j

En esta sección se describen algunas funciones para la manipulación de los nodos, relaciones y propiedades en la base de datos orientada a grafos “Neo4j”. Lo cual se responde con las siguientes preguntas.

¿Como crear nodos?

Para crear Nodos en Neo4j se debe ejecutar la sentencia “CRETATE”, cuya sintaxis es la siguiente:

CREATE (VariableN: EtiquetaN { propiedad: valor })

- VariableN: Variable que contendrá a la categoria del nodo.
- EtiquetaN: Categoria a la que pertenece el nodo.
- propiedad: Refiere a un atributo del nodo.
- valor: Contenido del atributo del nodo.

Consultas ejecutadas:

- Crear un nodo “persona” y tres nodos “nota”.

CREATE (psn: persona { nombre: 'Mariana', apellido_paterno: 'Mofíl', apellido_materno: 'Mendoza' })

```
$ CREATE (psn: persona { nombre: 'Mariana', apellido_paterno: 'Mofíl', apellido_materno: 'Mendoza' })
```

Table

Added 1 label, created 1 node, set 3 properties, completed after 3 ms.

CREATE (nta: nota { numero: '1', nota: 'Mensaje 1' })

```
$ CREATE (nta: nota { numero: '2', nota: 'Mensaje 2' })
```

Table

Added 1 label, created 1 node, set 2 properties, completed after 3 ms.

CREATE (nta: nota { numero: '2', nota: 'Mensaje 2' })

```
$ CREATE (nta: nota { numero: '1', nota: 'Mensaje 1' })
```

Table

Added 1 label, created 1 node, set 2 properties, completed after 94 ms.

CREATE (nta: nota { numero: '3', nota: 'Mensaje 3' })

```
$ CREATE (nta: nota { numero: '3', nota: 'Mensaje 3' })
```

Table

Added 1 label, created 1 node, set 2 properties, completed after 3 ms.

¿Como crear relaciones?

Para relacionar nodos en Neo4j se requiere ejecutar la siguiente sintaxis:

MATCH (x1: EtiquetaN1), (x2: EtiquetaN2) … CREATE (x1) - [VariableR: EtiquetaR { propiedad: valor }] -> (x2)

- x: Es una variable, se utiliza como si se renombrara la categoria del nodo.
- EtiquetaN: Categoria a la que pertenece el nodo.
- VariableR: Variable que contendrá al grupo de la relación.
- EtiquetaR: Grupo al cual pertenecerá la relación.
- propiedad: Refiere a un atributo de la relación.
- valor: Contenido del atributo de la relación.

Consultas ejecutadas:

- Relacionar el nodo “persona” con los nodos “nota”.

MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '1'
CREATE (a)- [rel1: escribe { fecha: '04/05/2018' }]-> (b)

```
1 MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '1'
2 CREATE (a) - [rel1:escribe { fecha: '04/05/2018' }] -> (b)
```

```
$ MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '1' CREATE (a) - [rel1:escribe {…
```

Table

Set 1 property, created 1 relationship, completed after 116 ms.

MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '2'
CREATE (a)- [rel2: escribe { fecha: '06/07/2018' }]-> (b)

```
1 MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '2'
2 CREATE (a) - [rel2:escribe { fecha: '06/07/2018' }] -> (b)
```

```
$ MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '2' CREATE (a) - [rel2:escribe {…
```

Table

Set 1 property, created 1 relationship, completed after 4 ms.

MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '3'
CREATE (a)- [rel3: escribe { fecha: '08/09/2018' }]-> (b)

```
1 MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '3'
2 CREATE (a) - [rel3:escribe { fecha: '08/09/2018' }] -> (b)
```

```
$ MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '3' CREATE (a) - [rel3:escribe {…
```

Table

Set 1 property, created 1 relationship, completed after 4 ms.

¿Como modificar nodos?

Para modificar un nodo en Neo4j se requiere hacer uso de la sentencia “SET” de acuerdo con la siguiente sintaxis:

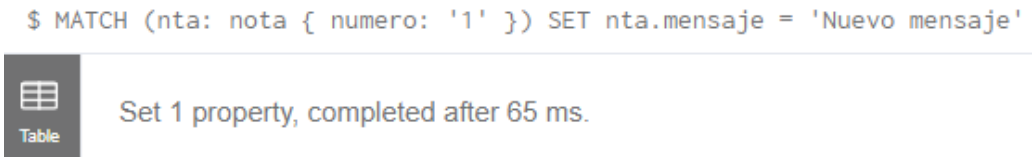
MATCH (VariableN: EtiquetaN { propiedad: valor }) SET VariableN.propiedad = 'NuevoValor'

- VariableN: Variable que contendrá a la categoria del nodo.
- EtiquetaN: Categoria a la que pertenece el nodo.
- propiedad: Refiere a un atributo del nodo.
- valor: Es el valor de un atributo del nodo.

Consultas ejecutadas:

- Se debe de agregar y editar la propiedad “mensaje”, del nodo “nota” con valor “1” en su propiedad “numero”.

MATCH (nta: nota { numero: '1' }) SET nta.mensaje = 'Nuevo mensaje'



MATCH (nta: nota { numero: '1' }) SET nta.mensaje = 'Mensaje editado'

¿Como borrar nodos?

Para eliminar un nodo en Neo4j es necesario que no existan dependencias hacia él, es decir, antes de eliminar un nodo, es necesario eliminar la relaciones del mismo. Para realizar este proceso se requiere utilizar la sentencia “DELETE” de la siguiente forma:

- Para eliminar relaciones entre nodos la sintaxis es:

MATCH (VariableN1: EtiquetaN1 { propiedad: valor }) - [VariableR: EtiquetaR { propiedad: valor }] -> (VariableN2: EtiquetaN2 { propiedad: valor }) DELETE VariableR

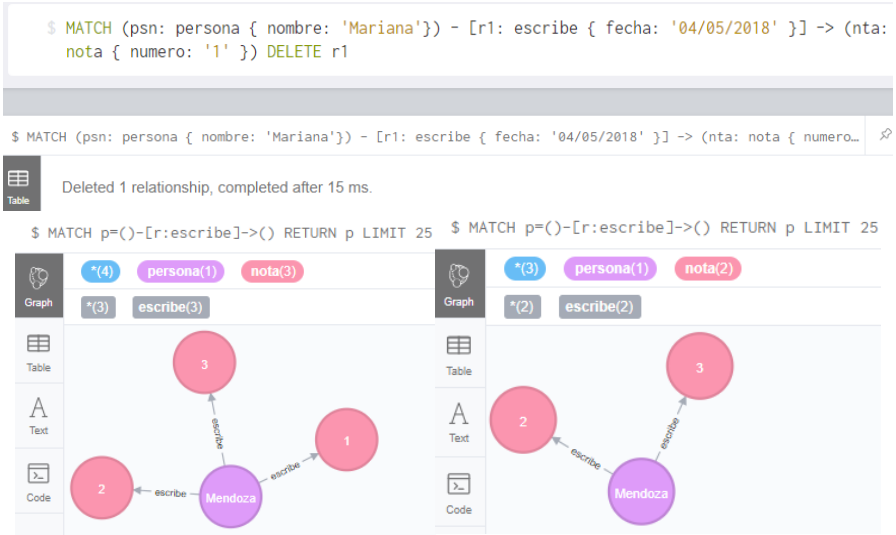
- Para eliminar el nodo la sintaxis es:

MATCH (VariableN: EtiquetaN { propiedad: valor }) DELETE VariableN

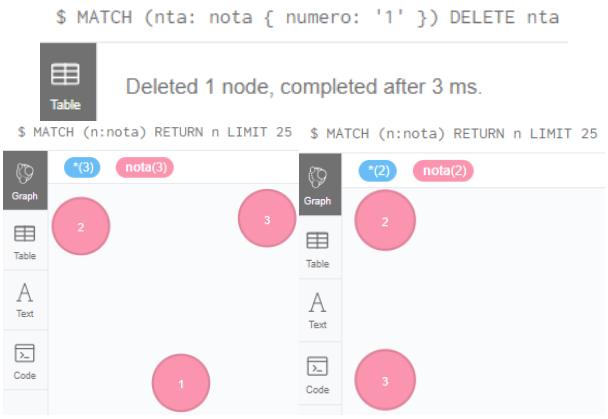
Consultas ejecutadas:

- Eliminar la relación entre el nodo “persona” y el nodo “nota” con valor “1” en su propiedad “numero”, para posteriormente eliminar el nodo “nota” ya mencionado.

MATCH (psn: persona { nombre: 'Mariana'})- [r1: escribe { fecha: '04/05/2018' }]-> (nta: nota { numero: '1' }) DELETE r1



MATCH (nta: nota { numero: '1' }) DELETE nta



¿Como modificar relaciones?

Para modificar relaciones en Neo4j, ya sea para remover, agregar o actualizar la propiedad de un nodo, se utilizan las sentencias “REMOVE” y “SET” de la siguiente forma:

- La sentencia “REMOVE” se emplea de forma similar a la sentencia “DELETE”, con la principal diferencia de que al final de ella, en lugar de solo indicar el nombre de la variable, con “REMOVE” se debe indicar el nombre de la variable y la propiedad que se desea eliminar (VariableR.propiedad):

MATCH (VariableN1: EtiquetaN1 { propiedad: valor }) - [VariableR: EtiquetaR { propiedad: valor }] -> (VariableN2: EtiquetaN2 { propiedad: valor }) REMOVE VariableR.propiedad

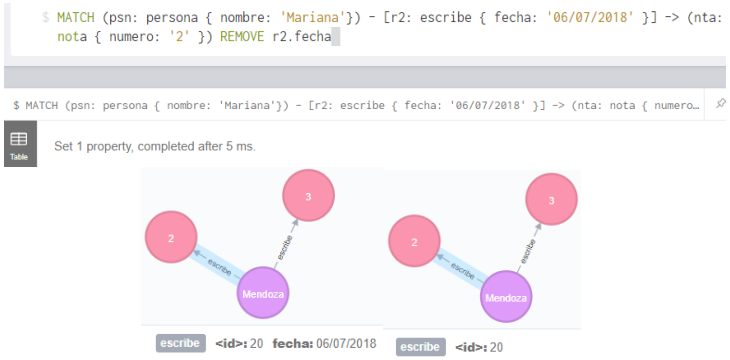
- La sentencia “SET” como se explicó en el apartado “¿Como modificar nodos?”, requiere que se indique el nombre de la Variable (en este caso el de la relación) y la propiedad que se va a ser agregada o actualizada (VariableR.propiedad):

MATCH (VariableN1: EtiquetaN1 { propiedad: valor }) - [VariableR: EtiquetaR { propiedad: valor }] -> (VariableN2: EtiquetaN2 { propiedad: valor }) SET VariableR.propiedad = 'NuevoValor'

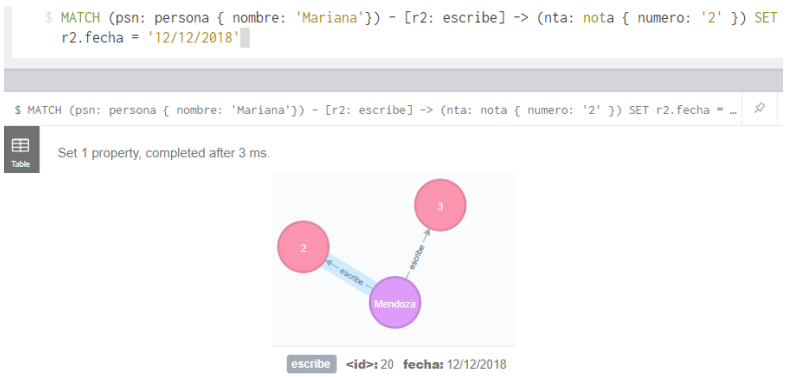
Consultas ejecutadas:

- Eliminar el campo “fecha” de la relación del nodo “persona” con el nodo “nota” de valor “2” en su propiedad “numero”, y posteriormente agregar el campo “fecha” con una fecha de este años al mismo nodo.

MATCH (psn: persona { nombre: 'Mariana'})- [r2: escribe { fecha: '06/07/2018' }]-> (nta: nota { numero: '2' }) REMOVE r2.fecha



MATCH (psn: persona { nombre: 'Mariana'})- [r2: escribe]-> (nta: nota { numero: '2' }) SET r2.fecha = '12/12/2018'



Consultas en Neo4j

En esta sección se describen algunas funciones para la consulta de datos en la base de datos orientada a grafos “Neo4j”.

Utilizando match

Para la consulta de Nodos en Neo4j se debe ejecutar la sentencia “MATCH” y “RETURN”, de acuerdo con siguiente la sintaxis:

MATCH (x1: EtiquetaN1 { propiedad: valor }), (x2: EtiquetaN2 { propiedad: valor }) RETURN x1, x2

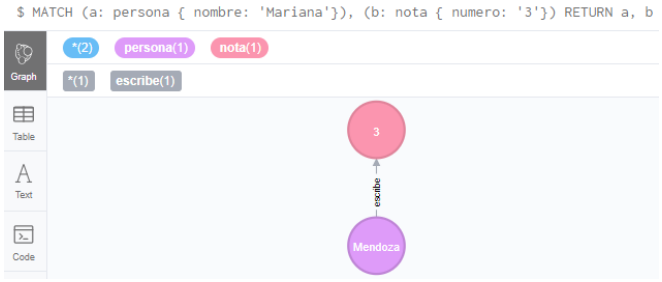
MATCH (x1: EtiquetaN1 { propiedad: valor }) --> (x2: EtiquetaN2 { propiedad: valor }) RETURN x1, x2

- x: Es una variable, se utiliza como si se renombrara la categoria del nodo.
- EtiquetaN: Categoria a la que pertenece el nodo.
- propiedad: Refiere a un atributo de la relación que será utilizado para filtrar los datos.
- valor: Contenido del atributo de la relación que será utilizado para filtrar los datos.

Consultas ejecutadas:

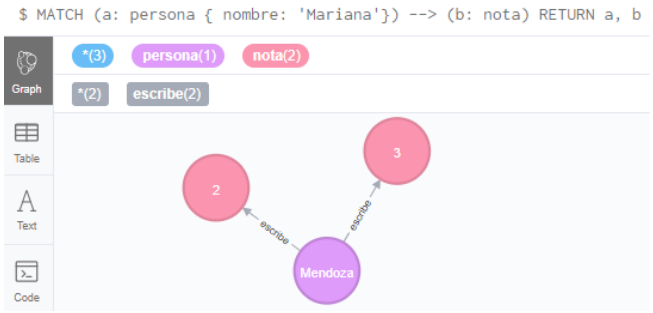
- Obtener el nodo “persona” cuyo valor de “nombre” sea “Mariana” y obtener el nodo “nota” con valor “3” en la propiedad “numero”.

MATCH (a: persona { nombre: 'Mariana'}), (b: nota { numero: '3'}) RETURN a, b



- Obtener el nodo “persona” cuyo valor de “nombre” sea “Mariana” y todos los nodos “nota” relacionados a él.

MATCH (a: persona { nombre: 'Mariana'}) --> (b: nota) RETURN a, b



Utilizando where

Una forma para realizar consultas complejas en los Nodos de Neo4j es implementando la sentencia "WHERE", de acuerdo con siguiente la sintaxis:

MATCH (x1: EtiquetaN1 { propiedad: valor }), (x2: EtiquetaN2 { propiedad: valor }) WHERE x1.propiedad = 'valor' AND x2.propiedad = 'valor' RETURN x1, x2

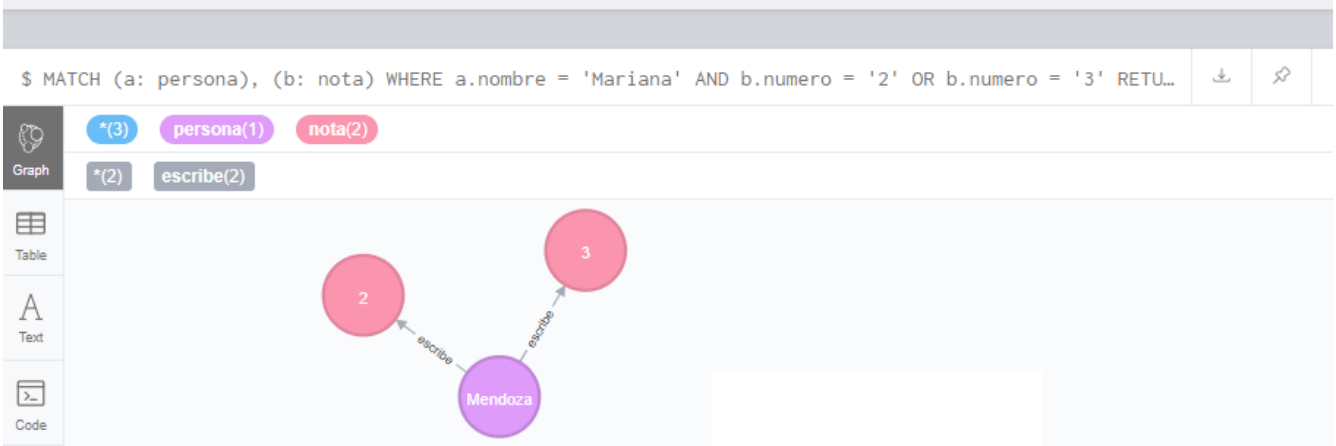
- x: Es una variable, se utiliza como si se renombrara la categoria del nodo.
- EtiquetaN: Categoria a la que pertenece el nodo.
- propiedad: Refiere a un atributo de la relación que será utilizado para filtrar los datos.
- valor: Contenido del atributo de la relación que será utilizado para filtrar los datos.

Consulta ejecutada:

- Obtener los nodos "persona" y "nota" donde el nodo "persona" en su propiedad "nombre" fuera igual a "Mariana" y los dos "nota" en su propiedad "numero" fueran igual a "2" o a "3". Ordenando el resultado de manera descendente.

MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '2' OR b.numero = '3' RETURN a, b ORDER BY b.numero DESC

```
$ MATCH (a: persona), (b: nota) WHERE a.nombre = 'Mariana' AND b.numero = '2' OR b.numero = '3'
  RETURN a, b ORDER BY b.numero DESC
```

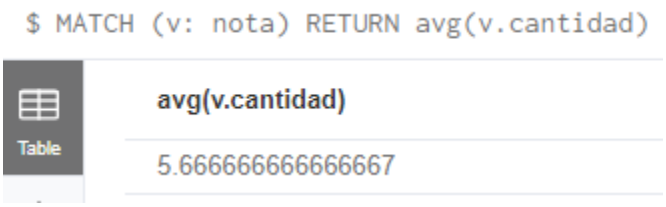


Función de agregación

En Neo4j las funciones de agregación son empleadas para realizar operaciones de especiales con los resultados de consultas convencionales (De Reserva, 2015). De acuerdo con la documentación de Neo4j, las funciones de agregación que soporta son las siguientes (Neo4j, 2018):

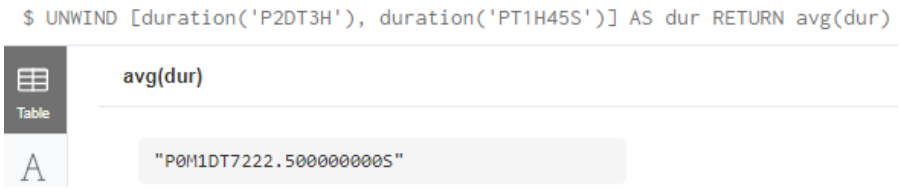
- avg() – Obtiene el valor promedio de un grupo de valores numéricos.
  - Por ejemplo:

  MATCH (v: nota) RETURN avg(v.cantidad)



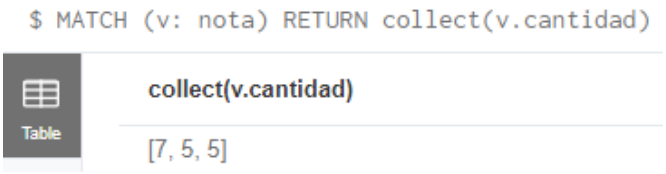
- avg() – Obtiene el promedio de duraciones.
  - Por ejemplo:

  UNWIND [duration('P2DT3H'), duration('PT1H45S')] AS dur RETURN avg(dur)



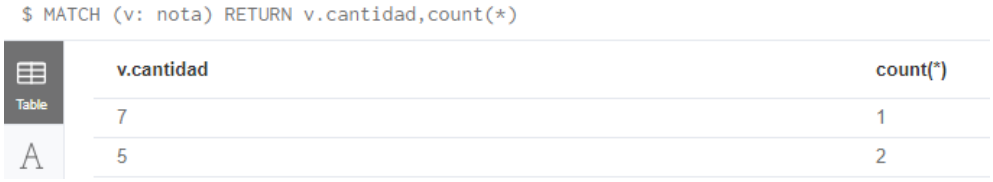
- collect() – Convierte un grupo en un arreglo de elementos.
  - Por ejemplo:

  MATCH (v: nota) RETURN collect(v.cantidad)



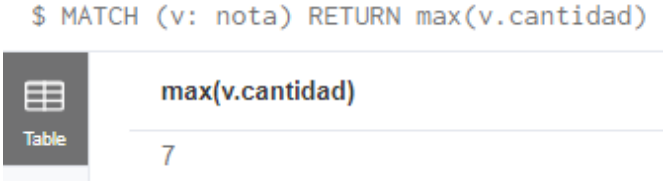
- count() – Cuenta el número de elementos de un grupo.
  - Por ejemplo:

  MATCH (v: nota) RETURN v.cantidad, count(*)



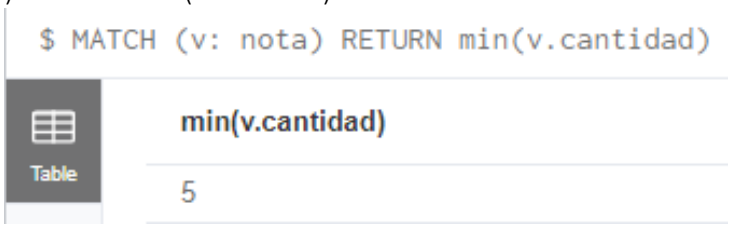
- max() – Obtiene el mayor valor numérico de un grupo.
  - Por ejemplo:

  MATCH (v: nota) RETURN max(v.cantidad)



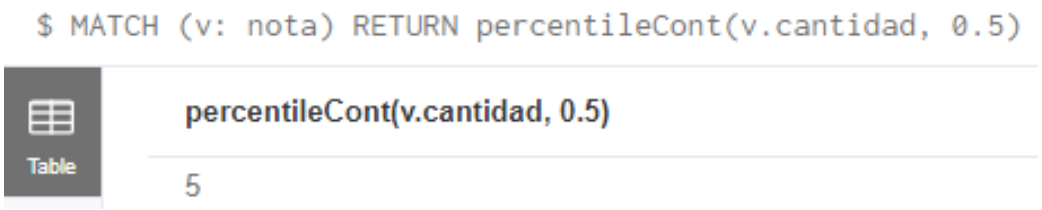
- min() – Obtiene el menor valor numérico de un grupo.
    - Por ejemplo:

    MATCH (v: nota) RETURN min(v.cantidad)



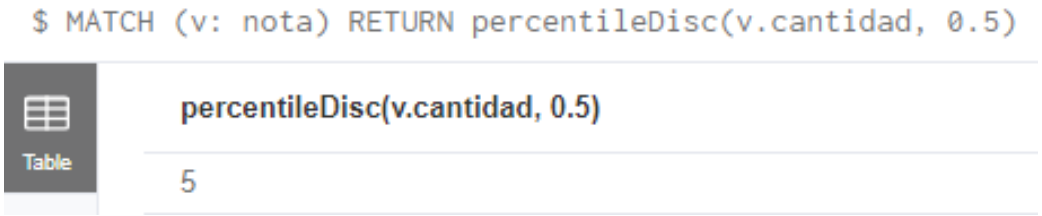
- percentileCont() – Obtiene el valor del percentil más cercano al segundo parámetro, cuyo valor debe encontrarse de 0 a 1.
    - Por ejemplo:

    MATCH (v: nota) RETURN percentileCont(v.cantidad, 0.5)



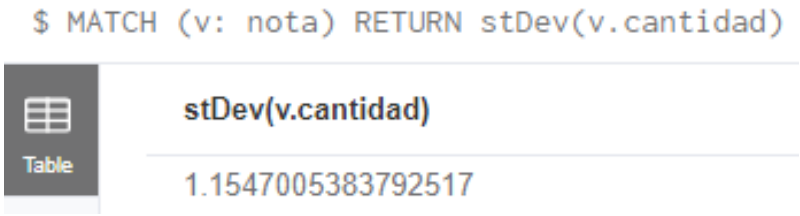
- percentileDisc() – Obtiene de un grupo, el percentil más cercano al segundo parámetro, cuyo valor debe ir de 0 a 1.
    - Por ejemplo:

    MATCH (v: nota) RETURN percentileDisc(v.cantidad, 0.5)



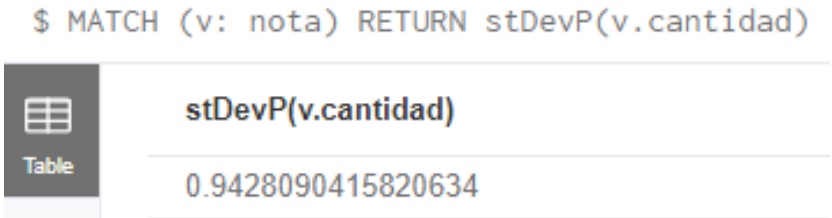
- stDev() – Obtiene la desviación estándar de una muestra de valores.
    - Por ejemplo:

    MATCH (v: nota) RETURN stDev(v.cantidad)



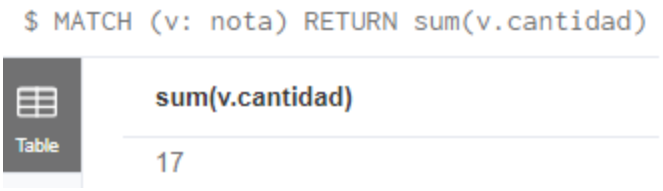
- stDevP() – Obtiene la desviación estándar de una población de valores.
    - Por ejemplo:

    MATCH (v: nota) RETURN stDevP(v.cantidad)



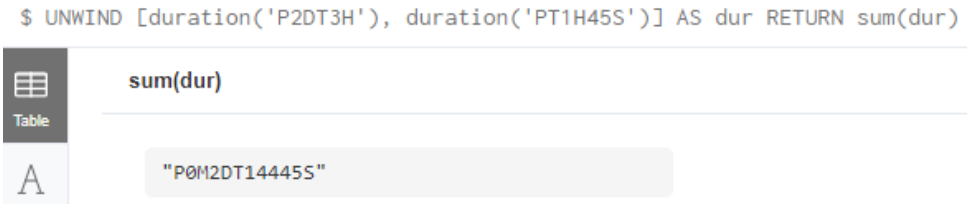
- sum() – Suma los valores numéricos.
    - Por ejemplo:

    MATCH (v: nota) RETURN sum(v.cantidad)



- sum() – Suma el valor de las duraciones.
    - Por ejemplo:

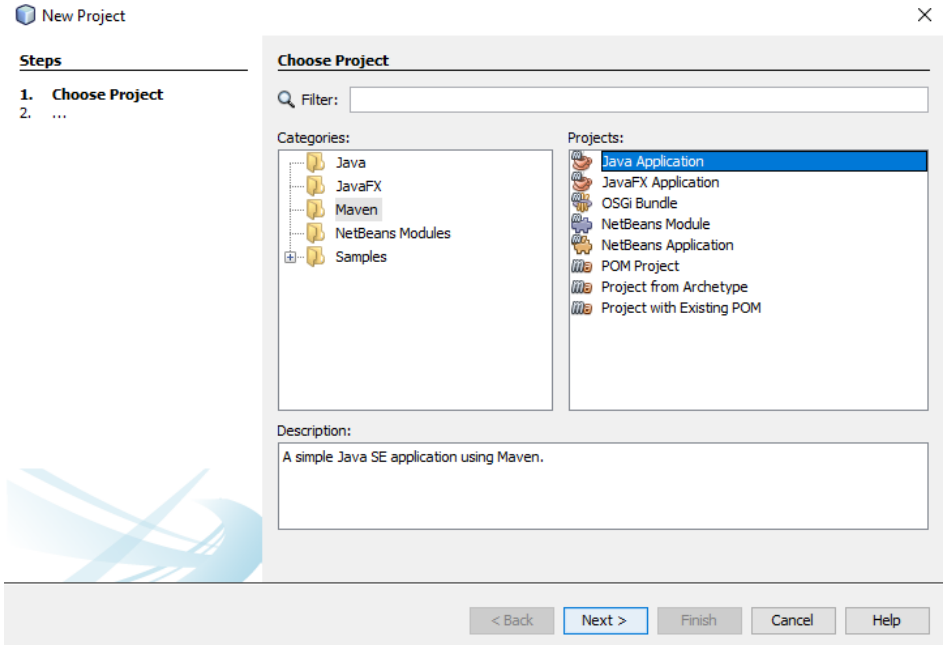
    UNWIND [duration('P2DT3H'), duration('PT1H45S')] AS dur RETURN sum(dur)



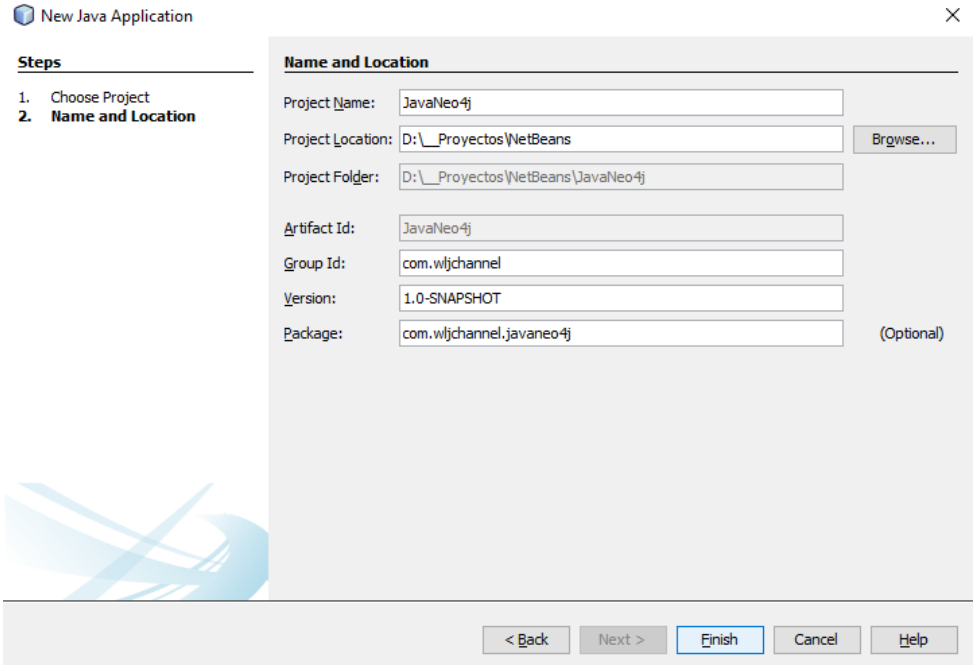
Conexión a una base de datos en Neo4j desde un lenguaje de programación

Se realizo la conexión de la base de datos orientada a grafos Neo4j con el lenguaje de programación Java, de acuerdo con la documentación que el sitio oficial de esta base de datos ofrece a sus usuarios, en este caso, para usuarios de Java (Neo4j, 2018). El proceso para conectar Neo4j con Java, fue el siguiente:

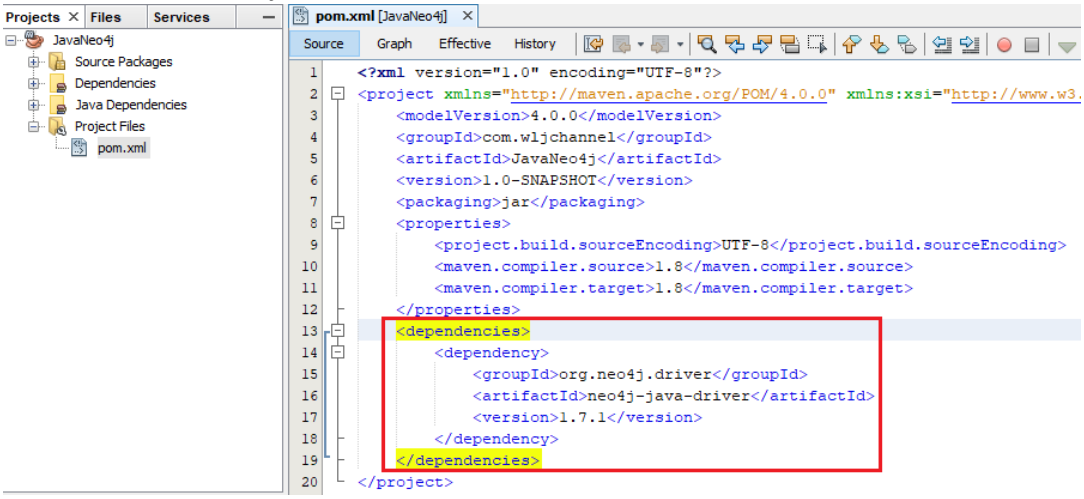
- Fue seleccionada una IDE que soportara el lenguaje de programación Java, con el cual se creó un proyecto de aplicación con categoria “Maven”.



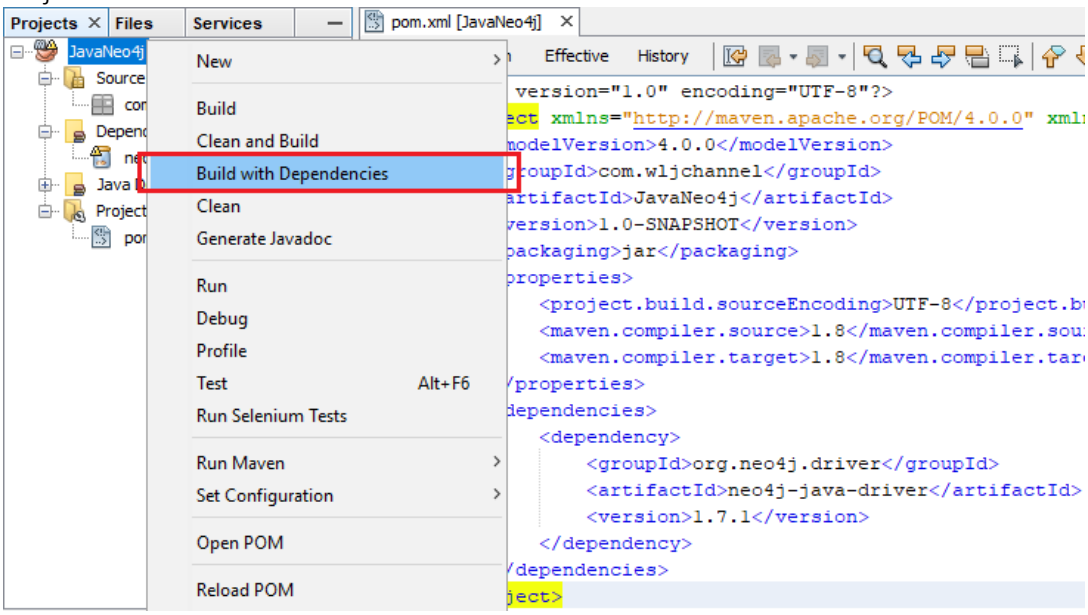
- La creación del proyecto de Java con categoria “Maven” no implico el cambio en alguna de sus directivas de configuración, salvo por su nombre.



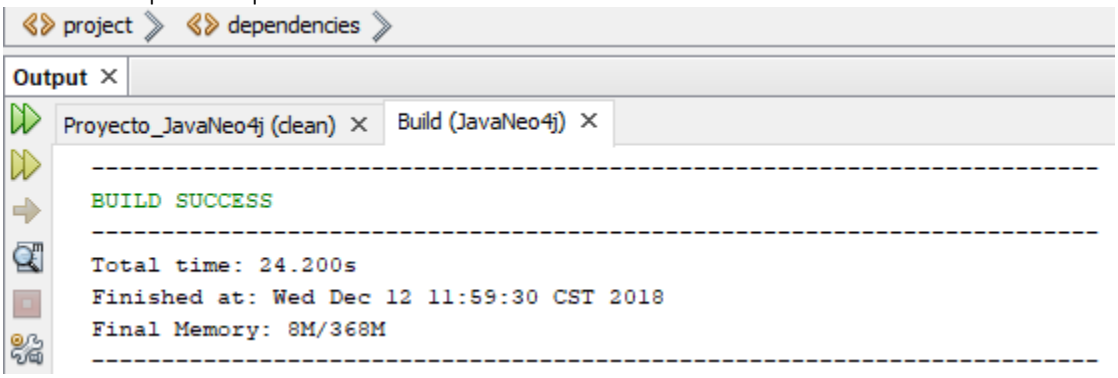
- Se agrego en el archivo “pom.xml” una serie de etiquetas para indicar que en el proyecto sería utilizado el driver de Neo4j.



- Posteriormente se compilo el proyecto junto a sus dependencias, para hacerlo consciente del driver de Neo4j.



- Esperando comprobar que el driver se instalara de manera correcta.



- Para conectar Neo4j con Java se codifico la siguiente función:

```
private void reconnect () {
    vDriver = GraphDatabase.driver(
            "bolt://" + vStrIP + ":" + vStrPort,
            AuthTokens.basic("neo4j", "Rider"));
}
```

- Para cerrar la conexión de Java con Neo4j se codifico la siguiente función:

```
public void close () {
    try {
        vDriver.close();
    } catch (Exception e) {
    }
}
```

- La siguiente función se codifico para obtener una lista con los nodos pertenecientes a la categoria "Nota":

```
public ListModel<String> getNodosList() {
    DefaultListModel listContNods = new DefaultListModel();
    vMapaUltNd = null;
    try {
        reconnect();
        Session sesion = vDriver.session();

        Transaction tx = sesion.beginTransaction();
        StatementResult result = tx.run("MATCH (v: nota) RETURN v.numero, v.mensaje ORDER BY v.numero");
        List<Record> listValsRecord = result.list();
        int numElements = listValsRecord.size();
        Map<String, Object> mapa = null;
        String strAx = "";

        for (int i = 0; i < numElements; i++) {
            mapa = listValsRecord.get(i).asMap();
            if (mapa.get("v.numero") != null) {
                strAx = (String) mapa.get("v.numero");
            }
            if (mapa.get("v.mensaje") != null) {
                strAx += ":" + (String) mapa.get("v.mensaje");
            } else {
                strAx += ":";
            }
            listContNods.addElement(strAx);
            vMapaUltNd = mapa;
        }
        close();
    } catch (Exception e) {
    }
    vListNods = listContNods;
    return listContNods;
}
```

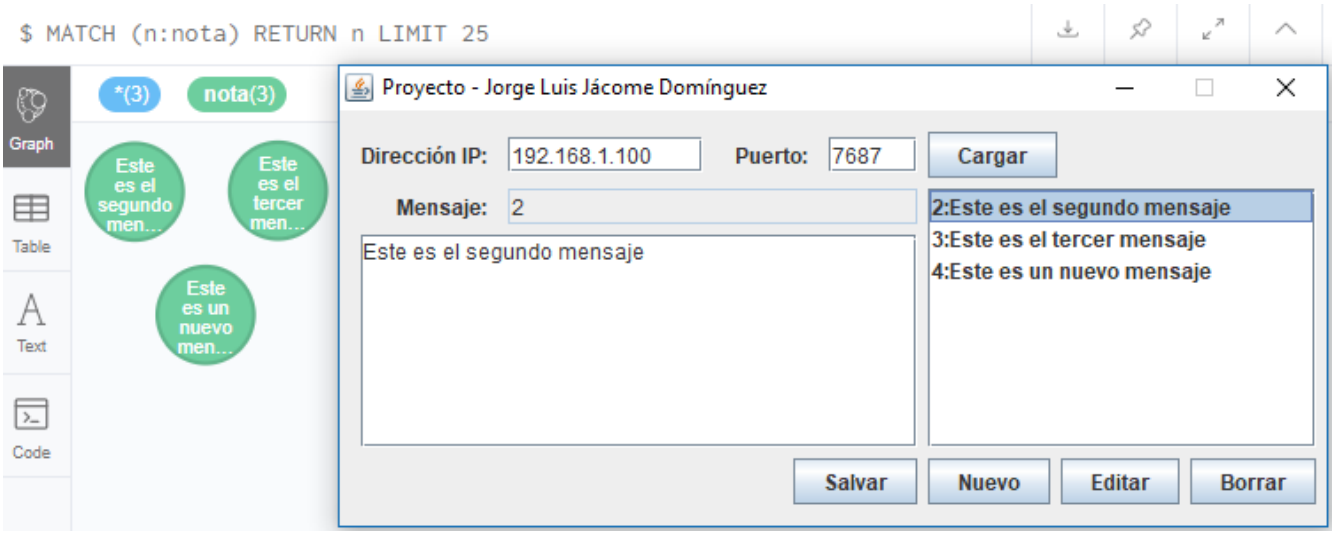
- La siguiente función fue codificada para eliminar un nodo de la categoria “nota”, con base en un valor que puede ser igualado al atributo “numero” de estas entidades.

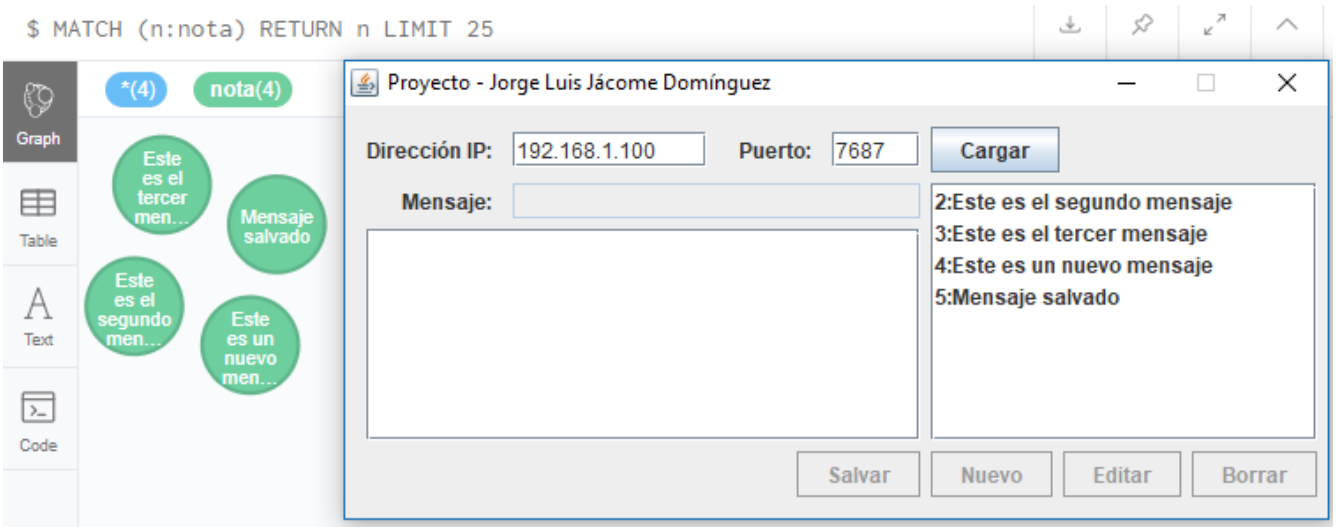
```
public void deleteNodo (final String numNodo) {
    try {
        reconnect();
        Session sesion = vDriver.session();
        String strTrans = sesion.writeTransaction(new TransactionWork<String>() {
            @Override
            public String execute(Transaction tx) {
                StatementResult result = tx.run("MATCH (nta: nota { numero: $numNodo }) DELETE nta", parameters("numNodo", numNodo));
                return "";
            }
        });
        close();
    } catch (Exception e) {
    }
}
```

- La siguiente función fue codificada para actualizar la propiedad “mensaje” de un nodo de la categoria “nota”, con base en el “numero” del nodo que será afectado. Tambien se codifico en esta función, el proceso para crear un nuevo nodo.

```
public void updateNodo (final String pNnumNodo, final String pMsjNodo) {
    try {
        reconnect();
        Session sesion = vDriver.session();
        final Map<String, Object> mapaUltNd = vMapaUltNd;
        String strTrans = sesion.writeTransaction(new TransactionWork<String>() {
            @Override
            public String execute(Transaction tx) {
                String msjNodo = pMsjNodo;
                String numNodo = pNnumNodo;
                if (msjNodo == null || msjNodo == "") {
                    msjNodo = "";
                }
                StatementResult result = null;
                if (numNodo != null && !numNodo.equals("")) {
                    result = tx.run("MATCH (nta: nota { numero: $numNodo }) SET nta.mensaje = $msjNodo", parameters("numNodo", numNodo, "msjNodo", msjNodo));
                } else {
                    if (mapaUltNd != null) {
                        if (mapaUltNd.get("v.numero") != null) {
                            numNodo = (String) mapaUltNd.get("v.numero");
                            int nmN = (int) Integer.parseInt(numNodo);
                            nmN += 1;
                            numNodo = nmN + "";
                        } else {
                            numNodo = "1";
                        }
                    } else {
                        numNodo = "1";
                    }
                    result = tx.run("CREATE (nta: nota { numero: $numNodo, mensaje: $msjNodo })", parameters("numNodo", numNodo, "msjNodo", msjNodo));
                }
                return "";
            }
        });
        String cad = "" + strTrans;
        close();
    } catch (Exception e) {
    }
}
```

- Como ejemplo de la aplicación desarrollada, con las siguientes imágenes se expone como es creado un nuevo nodo con el mensaje “Mensaje salvado”:





Referencias

ACENS. (2014). *Bases de datos NoSQL. Qué son y tipos que nos podemos encontrar.* Recuperado el 8 de Diciembre de 2018, de https://www.acens.com/wp-content/images/2014/02/bbdd-nosql-wp-acens.pdf

Camps Paré, R. (2005). *Introducción a las bases de datos.* Recuperado el 8 de Diciembre de 2018, de http://materials.cv.uoc.edu/cdocent/WO3ZK03VDJZB36VMNMRM.pdf

Camps Paré, R., Casillas Santillán, L. A., Costal Costa, D., Gibert Ginestà, M., Martín Escofet, C., & Pérez Mora, O. (2005). *Bases de datos.* Recuperado el 8 de Diciembre de 2018, de https://www.uoc.edu/masters/oficiales/img/913.pdf

Castro Romero, A., González Sanabria, J. S., & Callejas Cuervo, M. (2012). Utilidad y funcionamiento de las bases de datos NoSQL. *Facultad de Ingeniería, 21*(33), 21-32. Recuperado el 8 de Diciembre de 2018, de https://www.redalyc.org/pdf/4139/413940772003.pdf

De la Rosa Fernández, J. (2015). *Neo4j - A Graph Database.* Recuperado el 9 de Diciembre de 2018, de SlideShare: https://es.slideshare.net/JavierdelaRosaFernan/neo4j-47124820

De Reserva, B. C. (2015). *Tutorial neo4j en español.* Recuperado el 11 de Diciembre de 2018, de SlideShare: https://es.slideshare.net/zeley/tutorial-neo4j-en-espaol

Ezamudio. (2010). *Neo4J: Base de datos orientada a grafos*. Recuperado el 9 de Diciembre de 2018, de JavaMéxico: http://www.javamexico.org/blogs/ezamudio/neo4j_base_de_datos_orientada_grafos

F. Cía, J. (2015). *Neo4j: qué es y para qué sirve una base de datos orientada a grafos.* Recuperado el 9 de Diciembre de 2018, de BBVA: https://bbvaopen4u.com/es/actualidad/neo4j-que-es-y-para-que-sirve-una-base-de-datos-orientada-grafos

Lozano, E. (2018). *Neo4J: trabajando con grafos*. Recuperado el 9 de Diciembre de 2018, de Blog Técnologia para desarrollo: https://www.paradigmadigital.com/dev/neo4j-trabajando-grafos/

Neo4j. (2018). *2.3.1. Debian*. Recuperado el 10 de Diciembre de 2018, de https://neo4j.com/docs/operations-manual/current/installation/linux/debian/

Neo4j. (2018). *3.2. Configuration*. Recuperado el 10 de Diciembre de 2018, de https://neo4j.com/docs/operations-manual/current/docker/configuration/

Neo4j. (2018). *4.3. Aggregating functions.* Recuperado el 11 de Diciembre de 2018, de Neo4j: https://neo4j.com/docs/cypher-manual/current/functions/aggregating/

Neo4j. (2018). *Neo4j Debian Packages*. Recuperado el 10 de Diciembre de 2018, de http://debian.neo4j.org/?_ga=2.201793944.806623785.1544307324-1908646431.1544130088

Neo4j. (2018). *Using Neo4j from Java.* Recuperado el 12 de Diciembre de 2018, de https://neo4j.com/developer/java/

ORACLE. (2016). *¿Qué es una Base de Datos NoSQL?* Recuperado el 8 de Diciembre de 2018, de https://blogs.oracle.com/spain/qu-es-una-base-de-datos-nosql

Pinilla, C., Bello, M., & Peña, C. (2017). Bases de datos orientadas a grafos. *TIA, 5*(2), 153-160. Recuperado el 9 de Diciembre de 2018, de http://revistas.udistrital.edu.co/ojs/index.php/tia/article/download/8769/pdf

Sánchez, J. (2014). *Neo4j una base de datos NoSQL orientada a grafos*. Recuperado el 9 de Diciembre de 2018, de Xurxo Developer: http://xurxodeveloper.blogspot.com/2014/03/neo4j-una-base-de-datos-nosql-orientada.html

Sosa Olascoaga, J. M. (2012). *Uso de Neo4j como un sistema de base de datos orientado a grafos*. Recuperado el 9 de Diciembre de 2018, de Teofilismo.com: http://www.teofilismo.com/2012/02/uso-de-neo4j-como-un-sistema-de-base-de.html

Universidad de Murcia. (2006). *Sistemas de Gestión de Bases de datos y SIG*. Recuperado el 8 de Diciembre de 2018, de https://www.um.es/geograf/sigmur/sigpdf/temario_9.pdf

Velásquez Vargas, W. A. (2017). Algoritmo de recomendación sensible a contexto de elementos educativos reutilizables con almacenamiento Orientado a Grafos. *Revista Tecnológica ESPOL – RTE, 30*(1), 100-113. Recuperado el 9 de Diciembre de 2018, de https://www.researchgate.net/publication/316760386

Vélez de Guevara, L. (2018). *Gestión de Bases de Datos*. Recuperado el 8 de Diciembre de 2018, de https://media.readthedocs.org/pdf/gestionbasesdatos/latest/gestionbasesdatos.pdf

Zorrilla, M., & García-Saiz, D. (2017). *Gestores NoSQL – Neo4j.* Recuperado el 9 de Diciembre de 2018, de OpenCourseWare: https://ocw.unican.es/pluginfile.php/2396/course/section/2473/NoSQL_Tema2_Neo4j.pdf