UNIVERSIDAD VERACRUZANA
FACULTAD DE ESTADÍSTICA E INFORMÁTICA
XALAPA, VERACRUZ

PROGRAMA EDUCATIVO

Maestría en Sistemas Interactivos Centrados en el Usuario

EXPERIENCIA EDUCATIVA

Gestión de datos para los sistemas interactivos

DOCENTE

M.C.C. Lorena Alonso Ramírez

PROYECTO FINAL

Base de datos Neo4j

ALUMNO

Luis David Panamá Miranda

Xalapa, Ver. 17 de diciembre de 2018

Tabla de contenido

1. Introducción

SGBD

Definimos un Sistema Gestor de Bases de Datos o SGBD, también llamado DBMS (Data Base Managment System) como una colección de datos relacionas entre sí, estructurados y organizados, y un conjunto de programas que acceden y gestionan esos datos. La colección de esos datos se denomina Base de Datos o BD (Melgarejo & Moya, 2018).

Existen distintos objetivos que deben cumplir los SGBD (EcuRed, 2009):

- Abstracción de la información. Los SGBD ahorran a los usuarios detalles acerca del almacenamiento físico de los datos. Da lo mismo si una base de datos ocupa uno o cientos de archivos, este hecho se hace transparente al usuario. Así, se definen varios niveles de abstracción.
- Independencia. La independencia de los datos consiste en la capacidad de modificar el esquema (físico o lógico) de una base de datos sin tener que realizar cambios en las aplicaciones que se sirven de ella.
- Consistencia. En aquellos casos en los que no se ha logrado eliminar la redundancia, será necesario vigilar que aquella información que aparece repetida se actualice de forma coherente, es decir, que todos los datos repetidos se actualicen de forma simultánea. Por otra parte, la base de datos representa una realidad determinada que tiene determinadas condiciones, por ejemplo, que los menores de edad no pueden tener licencia de conducir. El sistema no debería aceptar datos de un conductor menor de edad. En los SGBD existen herramientas que facilitan la programación de este tipo de condiciones.
- Seguridad. La información almacenada en una base de datos puede llegar a tener un gran valor. Los SGBD deben garantizar que esta información se encuentra segura de permisos a usuarios y grupos de usuarios, que permiten otorgar diversas categorías de permisos.
- Manejo de transacciones. Una transacción es un programa que se ejecuta como una sola operación. Esto quiere decir que luego de una ejecución en la que se produce una falla es el mismo que se obtendría si el programa no se hubiera ejecutado. Los SGBD proveen mecanismos para programar las modificaciones de los datos de una forma mucho más simple que si no se dispusiera de ellos.
- Tiempo de respuesta. Lógicamente, es deseable minimizar el tiempo que el SGBD demora en proporcionar la información solicitada y en almacenar los cambios realizados.

Es entonces deber del SGBD ofrecer los servicios típicos (Melgarejo & Moya, 2018):

- Creación y definición de la base de datos.
- Manipulación de los datos.
- Acceso a los datos.
- Mantener la integridad y consistencia de los datos.
- Mecanismos de copias de respaldo y de recuperación.

Algunas de sus principales características son:

- Permite una vista muy centralizada y clara de los datos para que sean accedidos de la mejor manera posible.
- Se encargan de gestionar adecuadamente los datos, evitando a los usuarios o programas que les requieren, tener que entender dónde se encuentran físicamente los datos.
- Estos sistemas disponen de un lenguaje de programación llamado SQL (Structured Query Language) para poder proteger y acceder a los datos.
- La necesidad de requerir de un lenguaje para su acceso y su autonomía como sistema, proporcionan integridad y seguridad a los datos.
- Suelen disponer de un sistema de bloqueo para el acceso simultáneo, lo que le da un plus de seguridad a la integridad de los datos.
- Estos sistemas de base de datos disponen de API's (Application Programming Interface) muy visuales e intuitivas para poder gestionar los datos.
- Un correcto SGBD proporcionará economías de escala en el procesamiento de grandes cantidades de datos ya que está hecho para ese tipo de operaciones.
- Los SGBD proporcionan un nivel de abstracción entre la estructura lógica de la base de datos y el esquema físico que describe el contenido físico usado por la base de datos.
- El programa de gestión de almacenamiento y su gestión de datos (servidor) es totalmente independiente del programa con el cuál se realizan las consultas (cliente).
- Los SGBD realizan eficientes almacenamientos de los datos, pero estos se hacen de forma oculta para el usuario y nada tiene que ver con lo que finalmente se le presenta.
- Son capaces de gestionar distintos tipos de bases de datos, por ejemplo: bases de datos relacionales (suelen ser las estándar) y bases de datos orientadas a objetos.
- Multiplicidad de acceso a los datos.

El funcionamiento resumido y principal de un SGBD es el siguiente: un programa servidor de base de datos accede a esta, la lee y organiza, y posteriormente los programas del lado del cliente acceden a esta para gestionarla a su gusto. Por tanto, en un sistema de gestión de base de datos interactúan 3 actores: bases de datos, programa cliente y servidor (Kyocera, 2017).

Algunas de las ventajas que tienen son (EcuRed, 2009):

- Proveen facilidades para la manipulación de grandes volúmenes de datos. Entre éstas:
    - Simplifican la programación de equipos de consistencia.
    - Manejando las políticas de respaldo adecuadas, garantizan que los cambios de la base serán siempre consistentes sin importar si hay errores correctamente, etc.

    - Organizan los datos con un impacto mínimo en el código de los programas.
    - Disminuyen drásticamente los tiempos de desarrollo y aumentan la calidad del sistema desarrollado si son bien explotados por los desarrolladores.
- Usualmente, proveen interfaces y lenguajes de consulta que simplifican la recuperación de los datos.

Algunos de los inconvenientes son (EcuRed, 2009):

- Típicamente, es necesario disponer de una o más personas que administren la base de datos, de la misma forma en que suele ser necesario en instalaciones de cierto porte disponer de una o más personas que administren los sistemas operativos. Esto puede llegar a incrementar los costos de operación en una empresa. Sin embargo, hay que balancear este aspecto con la calidad y confiabilidad del sistema que se obtiene.
- Si se tienen muy pocos datos que son usados por un único usuario por vez y no hay que realizar consultas complejas sobre los datos, entonces es posible que sea mejor usar una hoja de cálculo.
- Complejidad: los softwares muy complejos y las personas que vayan a usarlo deben tener conocimiento de las funcionalidades del mismo para poder aprovecharlo al máximo.
- Tamaño: la complejidad y la gran cantidad de funciones que tienen hacen que sea un software de gran tamaño, que requiere de gran cantidad de memoria para poder correr.
- Coste del hardware adicional: los requisitos de hardware para correr un SGBD por lo general son relativamente altos, por lo que estos equipos pueden llegar a costar gran cantidad de dinero.

Como se puede observar, utilizar un SGBD aporta multitud de ventajas y beneficios para la gestión de bases de datos, debido a su agilidad, optimización, gran número de servicios, etc… y todo esto se refleja en un uso de la memoria y del CPU mucho mayor que un sistema de almacenamiento de archivos normal. Es quizás la “desventaja” más notoria, pero obvia debido al gran trabajo que realizan estos sistemas (Kyocera, 2017).

Es decir, dichos sistemas permiten la creación, gestión y administración de bases de datos, así como la elección y manejo de las estructuras necesarios para el almacenamiento y búsqueda de la información del modo más eficiente posible. En la actualidad, existen multitud de SGBD en la mayoría relacionales (Iruela, 2016).

Modelo relacional

Edgar Frank Codd definió las bases del modelo relacional a finales de los 60. En 1970 publica el documento “A Relational Model of data for Large Shared Data Banks” (“Un modelo relacional de datos para grandes bancos de datos compartidos”). Actualmente se considera que ese es uno de los documentos más influyentes de toda la historia de la informática. Lo es porque en él se definieron las bases del llamado Modelo Relacional de Bases de Datos. Anteriormente el único modelo teórico estandarizado era el modelo Codasyl que se utilizó masivamente en los años 70 como paradigma del modelo en red de bases de datos.

Codd se apoya en los trabajos de los matemáticos Cantor y Childs (cuya teoría de conjuntos es la verdadera base del modelo relacional). Según Codd, los datos se agrupan en relaciones (actualmente llamadas tablas), las cuales son una estructura que aglutina datos referidos a una misma entidad de forma organizada. Las relaciones, además, estructuran los datos de forma independiente respecto a su almacenamiento real en la computadora. Es decir, es un elemento conceptual, no físico.

Lo que Codd intentaba fundamentalmente es evitar que las usuarias y usuarios de la base de datos tuvieran que verse obligadas a aprender los entresijos internos del sistema. Esto es lo que ocurría con el modelo en red, dominante cuando Codd diseñó el modelo relacional, que era bastante físico. Su enfoque fue revolucionario al evitar conceptos del mundo de la computación en su modelo.

Aunque trabajaba para IBM, esta empresa no recibió de buen grado sus teorías. De hecho, IBM continuó trabajando en su sistema gestor de bases de datos en red IMS. Fueron otras empresas (en especial Oracle) las que implementaron sus teorías.

Pocos años después, el modelo se empezó a utilizar cada vez más hasta, finalmente, ser el modelo de bases de datos más popular. Hoy en día casi todas las bases de datos siguen este modelo, aunque en estos años han aparecido rivales cada vez más fuertes en las llamadas bases de datos NoSQL, que han demostrado una gran eficacia en bases de datos que necesitan una enorme cantidad de instrucciones de modificación por minuto (Sánchez, 2018).

Codd perseguía estos objetivos con su modelo relacional:

- Independencia física. La forma de almacenar los datos, debe ser absolutamente independiente del modelo conceptual de los mismos. Si la forma de almacenar los datos cambia (si cambia el esquema físico) , no es necesario cambiar los esquemas lógicos, funcionan perfectamente. Esto permite que los usuarios y usuarias se concentren en qué resultados desean obtener de la base de datos independientemente de cómo estén realmente almacenados los datos.
- Independencia lógica. Se refiere a que la lógica de la base de datos debe de ser independiente de la forma externa de acceso a la base de datos (los

esquemas externos). Las aplicaciones que utilizan la base de datos no deben ser modificadas porque se modifique el esquema lógico de la misma. De una manera más precisa: gracias a esta independencia el esquema externo de la base de datos es realmente independiente del modelo lógico. En la práctica, esta independencia es difícil de conseguir.

- Flexibilidad. La base de datos ofrece fácilmente distintas vistas en función de los usuarios y aplicaciones. La visión de los datos se adapta al usuario que los requiere.
- Uniformidad. Las estructuras lógicas siempre tienen una única forma lógica (las tablas). Es decir, manejar el modelo relacional es manejar las tablas.
- Sencillez. Facilidad de manejo (algo cuestionable, pero ciertamente verdadero si comparamos con los sistemas gestores de bases de datos anteriores a este modelo).

Según el modelo relacional (desde que Codd lo enunció) el elemento fundamental del modelo es lo que se conoce como relación, aunque más habitualmente se le llama tabla (o también array o matriz). Codd definió el significado de las relaciones utilizando un lenguaje matemático. Para comprender visualmente este concepto siempre se han utilizado las tablas, ya que permiten representar la información de las relaciones en forma de filas y columnas (Sánchez, 2018).

Las relaciones constan de:

- Atributos. Es cada una de las propiedades de los datos de la relación (nombre, dni...). Las relaciones representan conjuntos de objetos o elementos reales, cada atributo es propiedad o característica de dicho elemento.
- Tuplas. Referido a cada ejemplar, objeto o elemento de la relación. Por ejemplo, si una relación almacena personas, una tupla representaría a una persona en concreto.

A continuación se muestra el modelo de una relación (tabla):

	atributo 1	atributo 2	atributo 3	atributo 4	
valor 1,1	valor 1,2	valor 1,3	....	valor 1,n	**<- tupla 1**
valor 2,1	valor 2,2	valor 2,3	....	valor 2,n	**<- tupla 2**
.....	**.....**	.....	**....**	.....	....
valor m,1	valor m,2	valor m,3	....	valor m,n	**<- tupla m**

Tabla 1 Modelo de bases de datos relacional

Ventajas de este modelo de base de datos (Digital Guide, 2018):

- Sencillez: el modelo de datos que subyace a la base de datos relacional se implementa y gestiona más fácilmente que otros modelos. Las ingentes cantidades de información (datos de clientes, listas de pedido, movimientos de las cuentas) que las empresas quieren almacenar a largo plazo se organizan sin problemas en la estructura de tablas en que se basa el modelo relacional de base de datos.

- Escasa redundancia de datos: las formas normales del modelo relacional fijan una normativa que tiene el fin de evitar duplicaciones. Si las reglas de normalización se aplican de forma consecuente, los sistemas relacionales facilitan un almacenamiento de datos libre de redundancias, puesto que solo es necesario editar los datos una única vez, lo que simplifica sobre todo el mantenimiento interno y técnico del banco de datos.

- Alta consistencia de datos: las bases de datos relacionales normalizadas permiten almacenar datos sin contradicciones, contribuyendo así a la consistencia de los datos. Asimismo, los sistemas relacionales presentan funciones con las cuales se definen y se controlan automáticamente las condiciones de integridad. Aquellas transacciones que ponen en peligro la consistencia de los datos se bloquean.

- Procesamiento de datos orientado a conjuntos: el sistema de base de datos relacional se apoya en un procesamiento orientado a conjuntos que subdivide cada entidad en valores mínimos. Esto permite conectar entidades diferentes por medio del contenido, así como realizar consultas complejas como JOIN.

- Lenguaje de consultas homogéneo: para la realización de consultas a bases de datos relacionales se ha consolidado el lenguaje SQL, que ha sido estandarizado por la ISO y la IEC. El propósito de tal estandarización es que las aplicaciones puedan desarrollarse y ejecutarse con independencia del SGBD en que se utilicen. Con todo, el soporte de SQL varía mucho en función del SGBD.

Inconvenientes de las bases de datos relacionales (Digital Guide, 2018):

- Según el escenario en que se emplean los sistemas de bases de datos relacionales, ciertas ventajas, como el estar basados en tablas, así como el reparto de los datos en tablas interconectadas, pueden interpretarse también como desventajas. Además, algunas de sus propiedades más destacadas son difícilmente reconciliables con los modernos requisitos de la programación de aplicaciones (orientación a objetos, multimedia y big data).

- Presentación de los datos en tablas: no siempre es posible integrar cualquier tipo de dato en el formato fijo de las tablas bidimensionales aun cuando estén

interconectadas (impedance mismatch). Los datos abstractos o no estructurados que surgen en relación con las aplicaciones multimedia y las soluciones de big data no pueden representarse en el modelo relacional.

- Sistema no jerárquico: las bases de datos relacionales no ofrecen la posibilidad, a diferencia de las orientadas a objetos, de desarrollar tablas con clases organizadas de forma jerárquica, impidiendo, así, poner en práctica conceptos como las entidades subordinadas, que heredan propiedades de entidades superiores. De este modo no es posible, por ejemplo, crear subtuplas: todas las tuplas de una base de datos relacional están al mismo nivel jerárquico.

- Segmentación de los datos: el principio de base de los sistemas relacionales que consiste en almacenar la información en tablas separadas (normalización) conduce inevitablemente a su segmentación. Este diseño deriva en complejas consultas que abarcan varias tablas, de modo que el elevado número de segmentos resultantes acostumbra a reflejarse negativamente en el rendimiento.

- Peor rendimiento frente a las bases de datos NoSQL: el modelo relacional plantea elevados requisitos en cuanto a la consistencia de datos que van en detrimento de la velocidad de escritura en las transacciones.

Bases de datos NoSql

Las bases de datos relacionales han sido el modelo más popular desde finales de los años 70 por su solidez y gran facilidad para diseñar sistemas complejos. Sin embargo en estos últimos años empiezan a estar desbordadas ante el uso de bases de datos que tienen que dar servicio veloz y concurrente a miles de usuarios, los cuales son capaces de generar enormes cantidades de información en poco tiempo (Sánchez, 2018).

Por ello se han diseñado bases de datos que se saltan el modelo relacional y que ya no utilizan el lenguaje SQL. De ahí el nombre de sistemas NoSQL (Not Only SQL).

Las bases NoSQL utilizan un modelo diferente en el que los datos se almacenan de forma menos estricta, en especial no siguen estas reglas (Sánchez, 2018):

- Transacciones ACID, como sí hacen los SGBD potentes relacionales (como Oracle, DB2, SQLServer o PostgreSQL). ACID hace referencia a las propiedades:
    - Atomicidad (A). Que implica que ninguna instrucción se pueda quedar a medias. Es decir o se ha ejecutado completamente o no, aunque ocurra un error grave en el servidor.
    - Consistencia (C). Asegura que una transacción siempre mantiene la integridad de los datos, se anule o se lleve finalmente a cabo la transacción. Incluso en cualquier momento intermedio de la transacción.
    - Aislamiento (I). Asegura que las transacciones simultáneas no se afecten entre sí. Es decir que una transacción será independiente de la otra.
    - Durabilidad (D). Asegura que cuando se confirme la transacción, los efectos de sus instrucciones serán definitivos, independientemente de que el sistema se apague o cierre por un error grave.

  No obstante, hay bases de datos NoSQL que son capaces de gestionar transacciones ACID, al menos en los nodos centrales (los que compactan la información definitiva).
- Datos no relacionales. Los datos no se almacenan en tablas relacionales que se combinan mediante operaciones de tipo JOIN. El modelo lógico es distinto y variable dependiendo del sistema.
- SQL como lenguaje de consulta. En su lugar utilizan lenguajes de programación como Java, JavaScript o C++ para acceder a los datos y otros como XML o JSON para definir los datos y metadatos.

Las bases de datos NoSql se utilizan habitualmente en (Sánchez, 2018):

- Datos de registros que se modifican cada poco tiempo (logs de servidores web, listado de peticiones http, etc.)
- Datos que se producen de forma paralela (se graban a la vez)

- Datos con relaciones complejas, difícilmente consultables desde SQL o representables de forma relacional
- Datos desestructurados o combinaciones de datos estructurados y desestructurados.
- Datos que se producen a gran velocidad

Las bases de datos de NoSQL presentan muchas ventajas en comparación con las bases de datos tradicionales (Sara, 2017):

- A diferencia de las bases de datos relacionales, las bases de datos NoSQL están basadas en key-value pairs
- Algunos tipos de almacén de bases de datos NoSQL incluyen diferentes tipos de almacenes como por ejemplo el almacén de columnas, de documentos, de key value store, de gráficos, de objetos, de XML y otros modos de almacén de datos.
- Algunos tipos de almacén de bases de datos NoSQL incluyen almacenes de columnas, de documentos, de valores de claves, de gráficos, de objetos, de XML y otros modos de almacén de datos.
- Podría decirse que las bases de datos NoSQL de código abierto tienen una implementación rentable. Ya que no requieren las tarifas de licencia y pueden ejecutarse en hardware de precio bajo.
- Cuando trabajamos con bases de datos NoSQL, ya sean de código abierto o tengan un propietario, la expansión es más fácil y más barata que cuando se trabaja con bases de datos relacionales. Esto se debe a que se realiza un escalado horizontal y se distribuye la carga por todos los nodos. En lugar de realizarse una escala vertical, más típica en los sistemas de bases de datos relacionales.

Desventajas de las bases de datos NoSQL (Sara, 2017):

- La mayoría de las bases de datos NoSQL no admiten funciones de fiabilidad, que son soportadas por sistemas de bases de datos relacionales. Estas características de fiabilidad pueden resumirse en: “atomicidad, consistencia, aislamiento y durabilidad.” Esto también significa que las bases de datos NoSQL, que no soportan esas características, ofrecen consistencia para el rendimiento y la escalabilidad.
- Con el fin de apoyar las características de fiabilidad y coherencia, los desarrolladores deben implementar su propio código, lo que agrega más complejidad al sistema.
- Esto podría limitar el número de aplicaciones en las que podemos confiar para realizar transacciones seguras y confiables, como por ejemplo los sistemas bancarios.
- Otras formas de complejidad encontradas en la mayoría de las bases de datos NoSQL, incluyen la incompatibilidad con consultas SQL. Esto significa que se necesita un lenguaje de consulta manual, haciendo los procesos mucho más lentos y complejos.

Bases de datos orientadas a grafos

Alejándonos del modelo relacional de bases de datos, nos encontramos con las bases de datos NoSQL orientadas a grafos. Y es que, aunque la teoría de grafos no es nueva (Euler en el siglo XVIII resolvió el problema de los siete puentes de Könisberg con grafos), no han tenido mayor auge hasta el boom de la era de los datos y el Big Data (Abalos, 2014).

Este tipo de bases de datos utiliza la topología de un grafo con nodos como vértices y relaciones como aristas y propiedades, utilizada para almacenar y representar datos conectados sin necesidad de utilizar un índice (que es el método tradicional de simular una relación en una base de datos relacional). De esta forma, los vecinos de un nodo son accesibles directamente mediante una referencia directa, sin pasar por estructuras intermedias. Nuestros datos pueden seguir distintos tipos de grafo, tener distinto tipo de relaciones (dirigidas o no), distinta multiplicidad de relaciones entre nodos (unirelacional o multirelacional) y distinta aridad de relaciones (grafo o hipergrafo).

Una Base de Datos en Grafo (BDG) es una base de datos que tiene como propósito almacenar estructuras de datos que tienen topología de grafo, es decir, que la información que se almacena se puede representar por medio de nodos y aristas entre ellos. Por definición, una BDG agruparía a cualquier solución de almacenamiento en la que los elementos que están conectados se enlazan sin hacer uso de una referencia por medio de índices (que sería el método habitual de "simular" una relación en una Base de Datos relacional), de esta forma, los vecinos de una entidad son accesibles directamente por ella por medio de una referencia directa, sin pasar por estructuras intermedias que hagan el proceso de de referenciado. En esta definición no tenemos en cuenta el tipo de grafo (en su sentido más amplio) que nuestros datos seguirán, ni en el tipo de aristas (dirigidas o no), ni en la multiplicidad de las mismas entre dos nodos (unirelacional o multirelacional), ni en la aridad que reflejen las aristas (grafo o hipergrafo) (Sancho, 2014).

Por tanto, una BDG debe cumplir los siguientes criterios:

- El almacenamiento está optimizado para representar datos como un grafo, proporcionando los nodos y las aristas que lo forman.
- El almacenamiento está optimizado para realizar los traversals a través del grafo, sin uso de índices que "representen" las aristas. Por ello, una BDG está optimizada para realizar consultas en los datos en las que intervengan relaciones de proximidad (conexiones) entre datos, y no para la realización de consultas globales.
- El modelo de datos es flexible... lo que en algunas ocasiones nos permite no declarar tipos de nodos o de aristas (el tipo de nodo o arista sería equivalente a las distintas tablas que se definen en un modelo relacional).
- Integra funciones para aplicar los algoritmos clásicos de la Teoría de Grafos (caminos más cortos, A*, medidas de centralidad, etc.

Sin duda, una BDG facilita en general la exploración de datos que tienen una estructura de grafo, especialmente cuando las relaciones entre esos datos son tan significativas como los datos mismos. El caso ideal de una consulta sería comenzar por uno o varios nodos y ejecutar un traversal en el grafo, y aunque es posible realizar consultas como "todos los nodos de un tipo", éstas se basan en el uso de índice para que los resultados sean óptimos. Sin embargo, incluso este tipo de consultas se podrían convertir en consultas locales en el grafo añadiendo al grafo un supernodo que reflejara cada uno de los tipos involucrados y conectándolo con todos los nodos de ese tipo (Sancho, 2014).

Las bases de datos de grafos cumplen los siguientes requisitos que las hacen diferentes con respecto a otro tipo de bases de datos como las relacionales u otro tipo de bases de datos NoSQL (Abalos, 2014):

- El almacenamiento está optimizado para datos representados en un grafo, guardando nodos y relaciones a otros nodos.
- El almacenamiento está optimizado para hacer recorridos a través del grafo (‘traversal’) sin utilizar un índice para las relaciones entre nodos. Es decir, está optimizado para hacer consultas sobre datos próximos partiendo de uno o más nodos, más que para consultas globales.
- Modelo de datos flexible para muchas soluciones: no es necesario declarar el tipo de datos para los nodos o las relaciones, al contrario que para las bases de datos relacionales.
- Funciones integradas para aplicar los algoritmos clásicos de la teoría de grafos: el camino más corto, Dijkstra, A*, medidas de centralidad y otros.

Algunos ejemplos de bases de datos de grafos los encontramos en Neo4j, TitanDB, FlockDB (la base de datos de Twitter), AllegroGraph, OrientDB e InfiniteGraph (Abalos, 2014).

2. Características de Neo4j

Neo4j es una base de datos orientada a grafos escrita en Java, es decir la información se almacena de forma relacionada formando un grafo dirigido entre los nodos y las relaciones entre ellos. Se integra perfectamente con múltiples lenguajes como Java, PHP, Ruby, .Net, Python, Node, Scala, etc. La base de datos está embebida en un servidor Jetty. Está especialmente indicada para modelar redes sociales y sistemas de recomendación (Ramos, 2014).

Aunque hay varios tipos de grafos, Neo4j sigue el modelo property graph o grafo de propiedades en los que tenemos los datos en formato clave/valor, almacenando datos en los nodos y relaciones direccionales (Abalos, 2014).

Hay tres formas de utilizar Neo4j:

1. Con Cypher como lenguaje declarativo de consulta.
2. API REST con algo más de funcionalidad que Cypher (aunque esta tendencia va disminuyendo en las últimas versiones).
3. En casos especiales podemos utilizar plugins vía API.

Es una herramienta muy útil para prototipado rápido y para hacer pruebas de concepto, ya que es muy fácil aprender y es rápida de configurar (Abalos, 2014).

Ventajas

- Tiene gran soporte y comunidad: hay muchas aplicaciones y plugins en github, podemos utilizar Neo4j con cualquier lenguaje. Además, añaden mejoras a la herramienta en función de los problemas que surgen en la comunidad de usuarios.
- Muy rápida para consultas de pocos grados de profundidad y miles de nodos.
- Fácil importación de formatos propios .csv (con plugin batch-import).
- Han mejorado el lenguaje Cypher con respecto a versiones anteriores.
- Muy práctica para manejar grafos, ya que no es tan académica como TitanDB, la cual no tiene muy buen rendimiento.
- Aportan en la interfaz web una forma de visualización de las respuestas de las consultas (en forma de grafo) aunque no está enfocada a datos masivos. Hay otras herramientas para datos masivos como GraphChi, Linkurius o GraphAlchemist.
- Permiten usar Neo4j de forma embedded a través de la API de Java para crear endpoints específicos.
- Además, Neo4j es una herramienta de comunidad, con mucho movimiento y en continua evolución. La versión Neo4j Community es muy útil para prototipado y la versión Personal es perfecta para startups (licencia GPL). Si vamos a tratar un proyecto es más conveniente utilizar Neo4j Enterprise (licencia GPLv3 y comercial), que aunque con un coste mayor, da mejor rendimiento, tiene herramientas de monitorización y clustering (clúster con tres nodos) y backups en caliente.

Desventajas

- Neo4j Community no es escalable para cantidades de datos muy grandes, hay que cambiar la configuración de la memoria en Neo4j acorde con las necesidades del grafo.
- Lo que más ocupa en memoria es el almacenamiento de los nodos, se hacen ineficientes las consultas si no se ha modelado el problema bien y si se almacena demasiada información en los nodos.
- Neo4j quiere que tengas todo el grafo en memoria, si puedes (igual que MongoDB, etc), pero al final habrá cosas del grafo que nunca tocas y es ineficiente tenerlo cacheado. Lo más recomendable es tener un conjunto de datos mínimo en memoria (por ejemplo, lo que más vas a consultar).
- No hace sharding.
- Neo4j no tiene gestión de usuarios, nada de autenticación: hay una extensión para autenticar, o como alternativa, se puede poner un proxy delante de la BD para autenticar.

Tomando en cuenta las ventajas y desventajas, el uso de Neo4j está recomendado para (Rodriguez, 2016):

- Búsquedas en base a grafos
- Sistemas de recomendación
- Redes sociales
- Gestión de identidades y accesos
- Detección de fraude
- Gestión de datos maestros

3. Descarga e instalación

Descarga

Para instalar Neo4j se necesita tener instalado Java y JDK. A continuación se presentan los links de descarga

Java:
https://www.java.com/es/download/

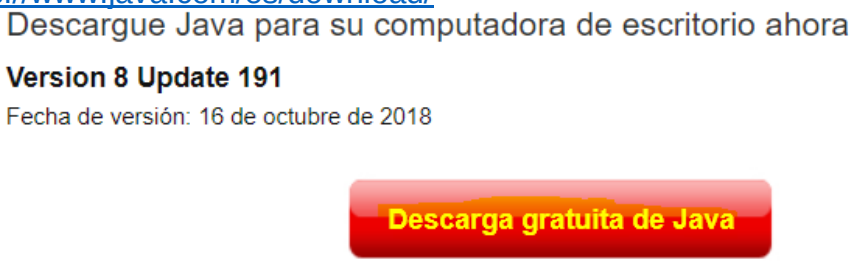

Ilustración 1 Descarga de Java

JDK:
https://www.oracle.com/technetwork/es/java/javase/downloads/index.html

Ilustración 2 Descarga de JDK

Neo4j:
https://neo4j.com/download-center/

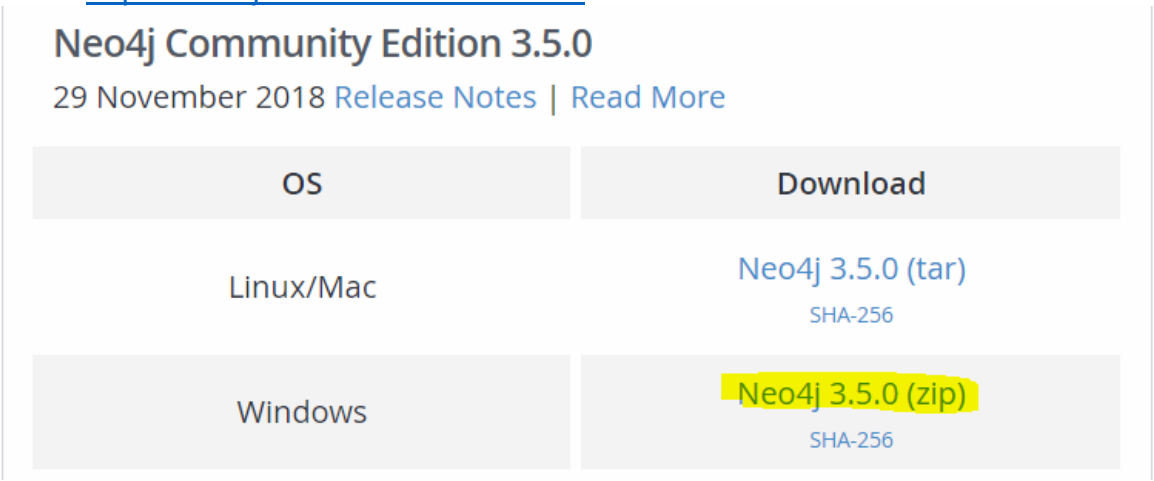

Ilustración 3 Descarga de Neo4j

Instalación

Una vez descargados, se tendrán los siguientes archivos y se procede a instalarlos:

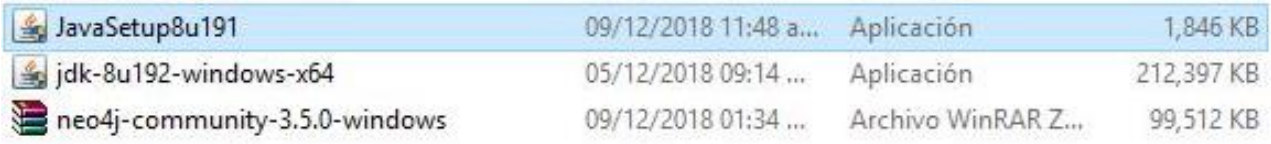

Ilustración 4 Archivos a instalar

Java

Doble clic al archivo "JavaSetup8u191" de la ilustración 4. Después clic en "Instalar":

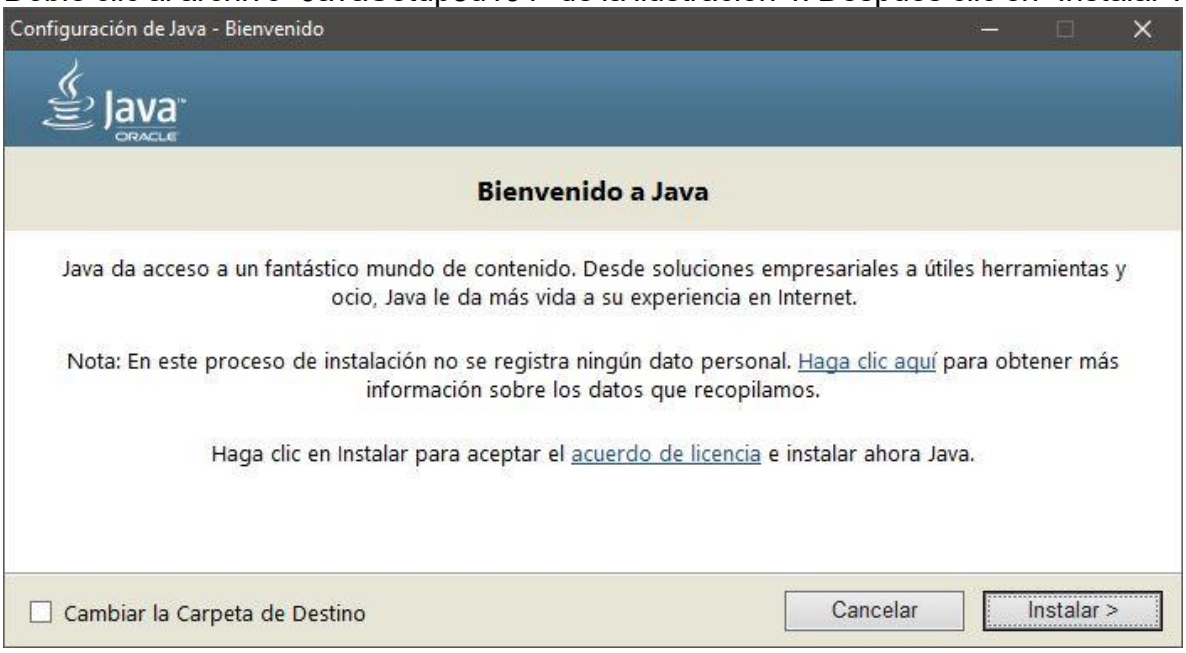

Ilustración 5 Instalación Java

Se espera a que termine la descarga:

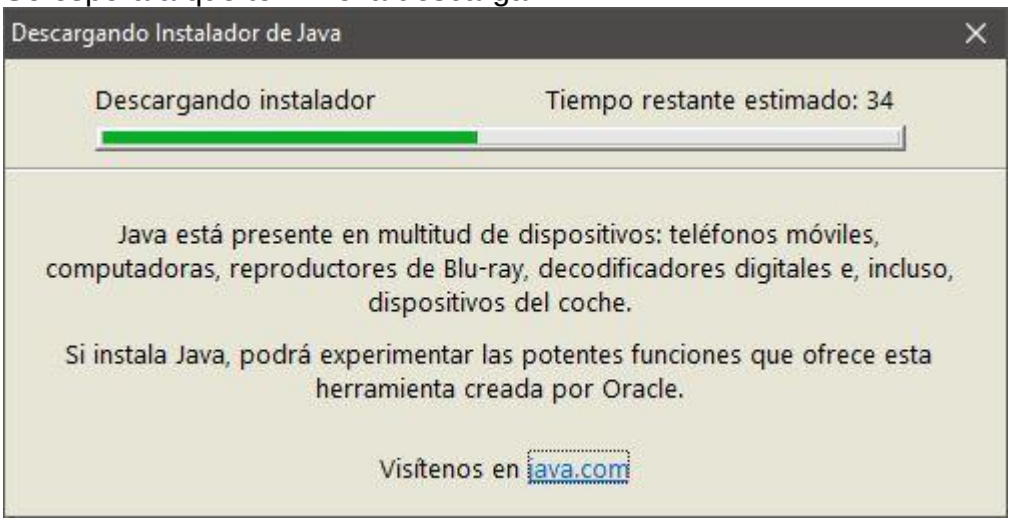

Ilustración 6 Instalación Java

Se espera que termine la instalación:

Ilustración 7 Instalación Java

Clic en cerrar:

Ilustración 8 Instalación Java

JDK

Ejecutar el archivo “jdk-8u192-windows-x64” de la ilustración 4 y dar clic en “Next”:

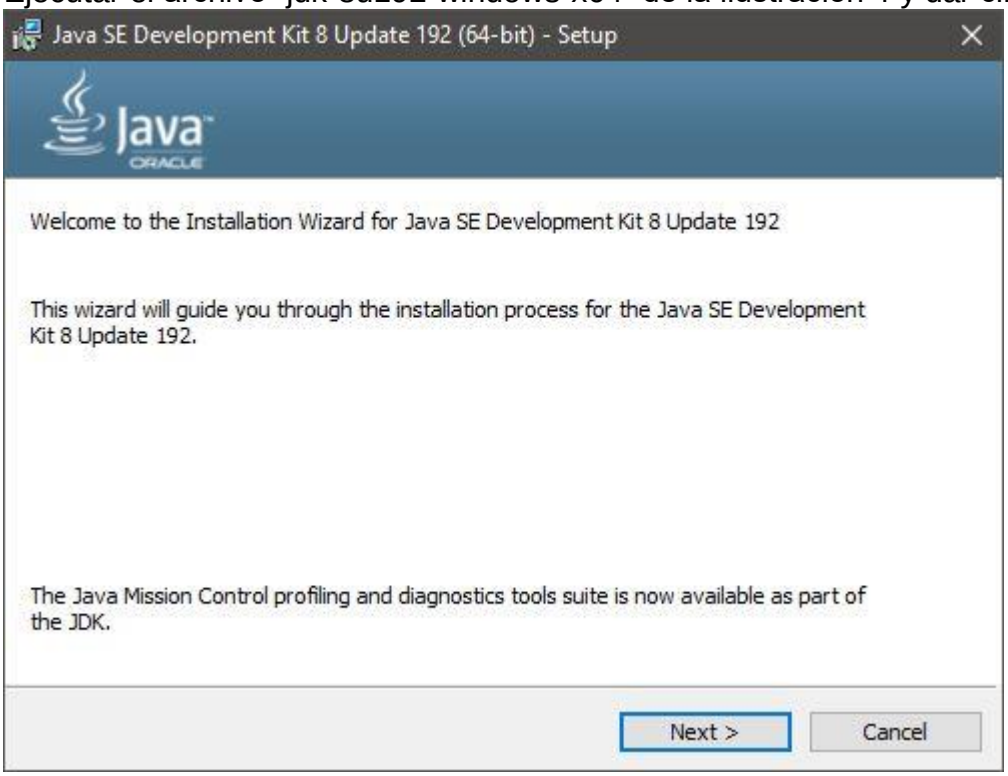

Ilustración 9 Instalación JDK

Click en “Next”

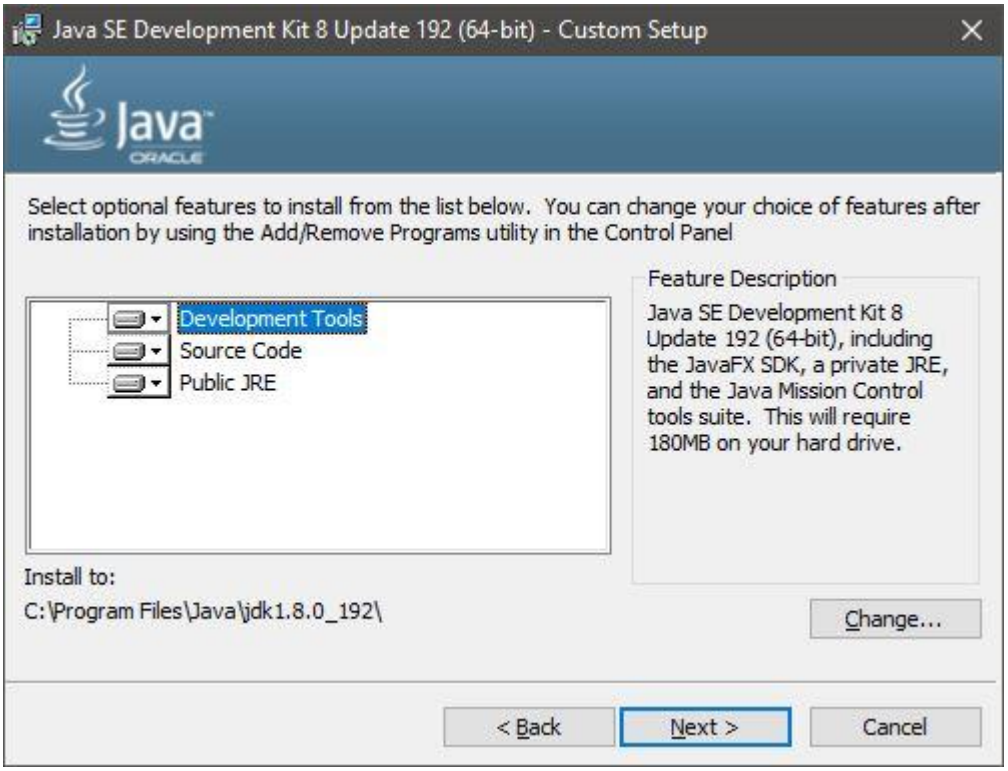

Ilustración 10 Instalación JDK

Esperar a que termine de cargar

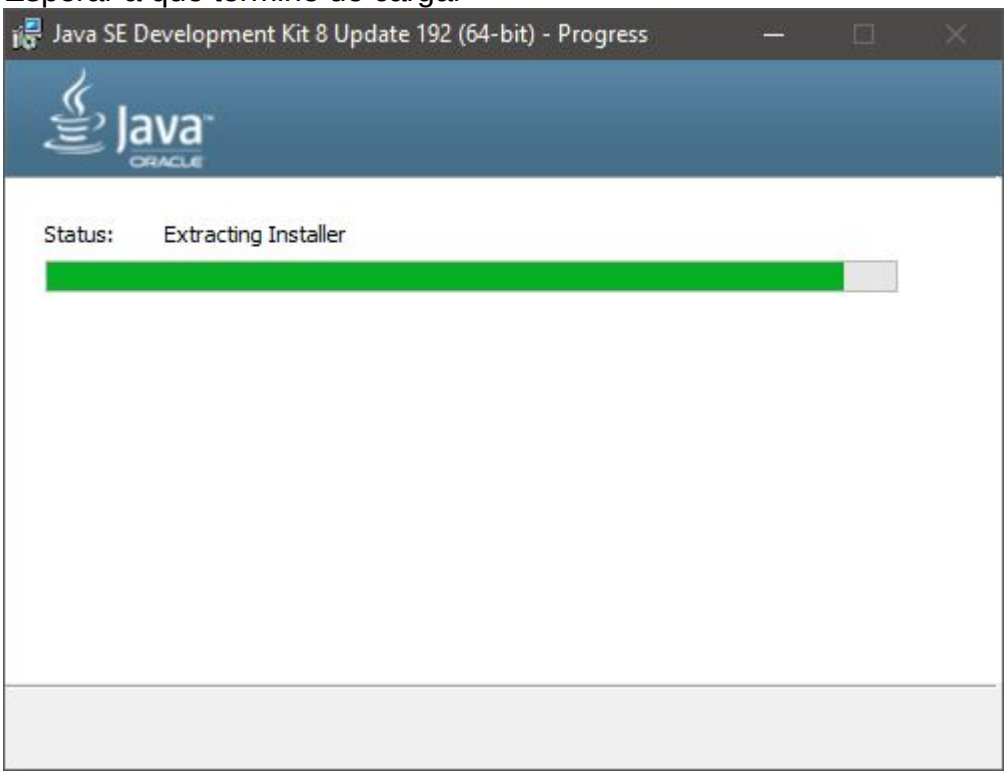

Ilustración 11 Instalación JDK

Clic en “Siguiente”

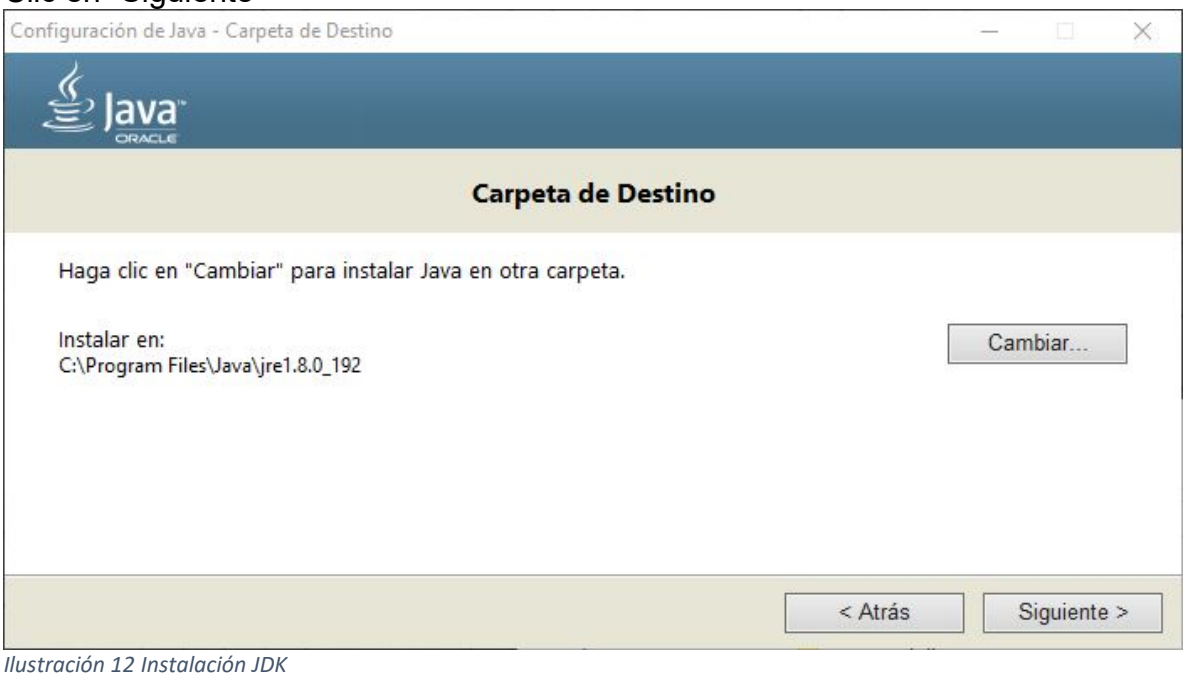

Ilustración 12 Instalación JDK

Esperar a que termine la instalación:

Ilustración 13 Instalación JDK

Clic en “Close”

Ilustración 14 Instalación JDK

Reiniciar el equipo:

Ilustración 15 Reinicio de sistema

Neo4j

Descomprimir el archivo "neo4j-community-3.5.0-windows" de la ilustración 4. Posteriormente y copiar los archivos contenidos en la carpeta creada:

Ilustración 16 Instalación Neo4j

Pegar los archivos en la carpeta de instalación:

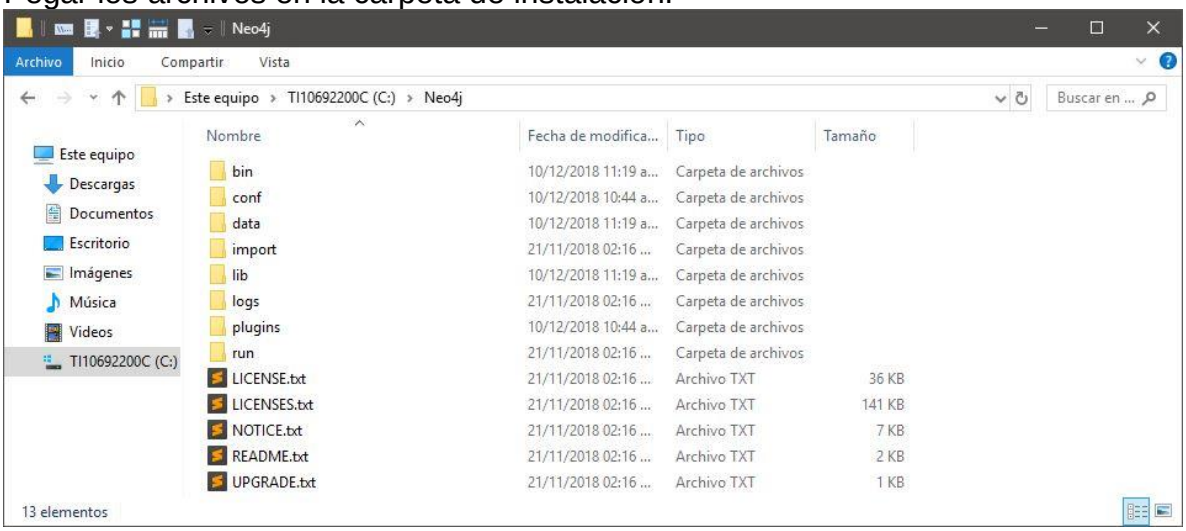

Ilustración 17 Instalación Neo4j

Posteriormente, en la línea de comandos (como administrador) se ejecutan los siguientes comandos:

cd C:\Neo4j\bin
neo4j install-service
neo4j start

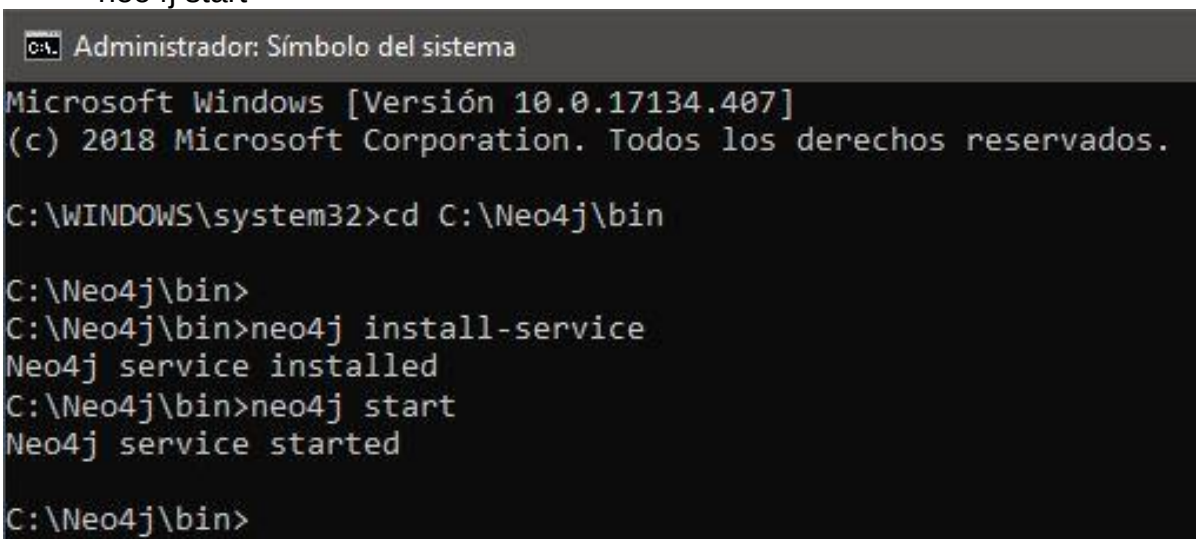

Ilustración 18 Instalación Neo4j

Luego se abre el navegador con la siguiente URL http://localhost:7474/browser/, y llenar con los siguientes datos:

Connect URL: `bolt://localhost:7687`
Username: neo4j
Password (por defecto): neo4j

Y dar clic en “Connect”:

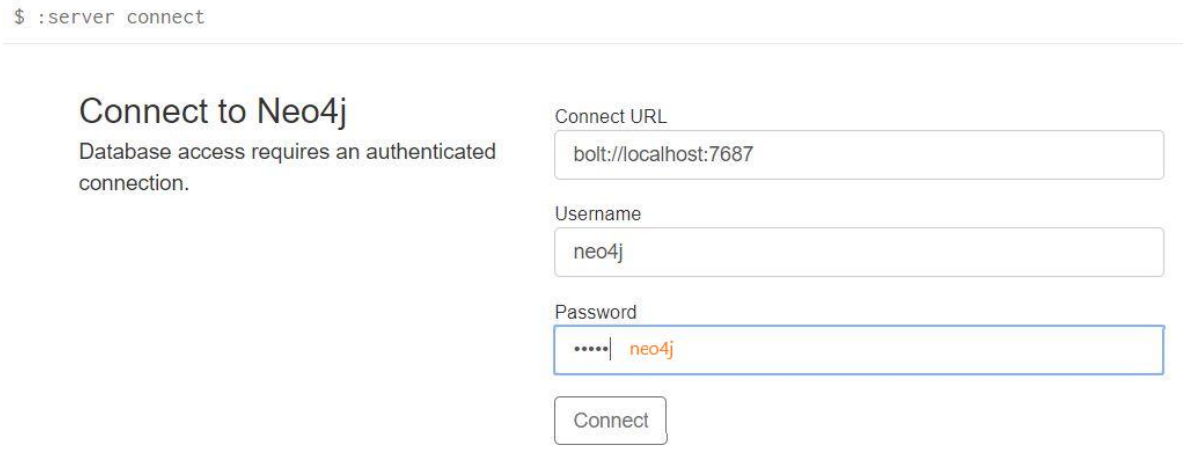

Ilustración 19 Instalación Neo4j

Posteriormente se ingresa la nueva contraseña que se desee, en este caso se ingresó “1234”, en ambos campos:

```
$ :server connect
```

Ilustración 20 Instalación Neo4j

Finalmente se puede observar la página de inicio de Neo4j:

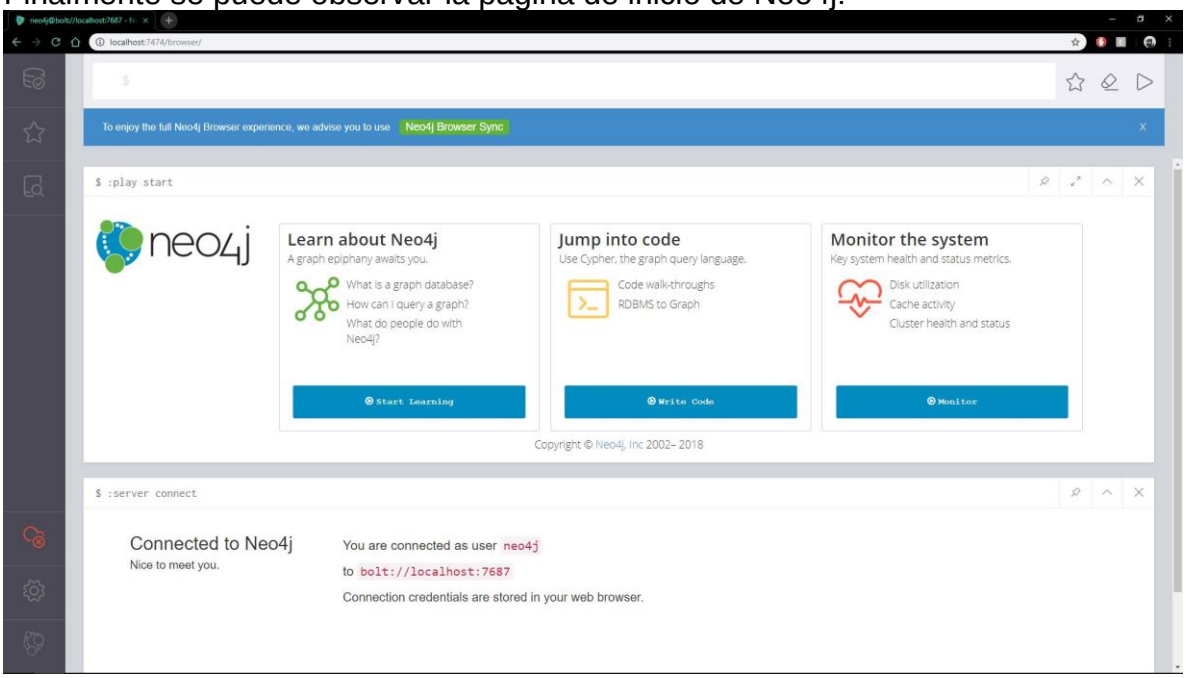

Ilustración 21 Instalación Neo4j

4. Definición de datos

A continuación, se presenta cómo se pueden crear y modificar a los nodos y a las relaciones, así como eliminar nodos en Neo4j.

a. Crear nodos

Se utiliza la sintaxis:

CREATE (_variable: _etiqueta { _propiedad: _valor })

Ejemplos:

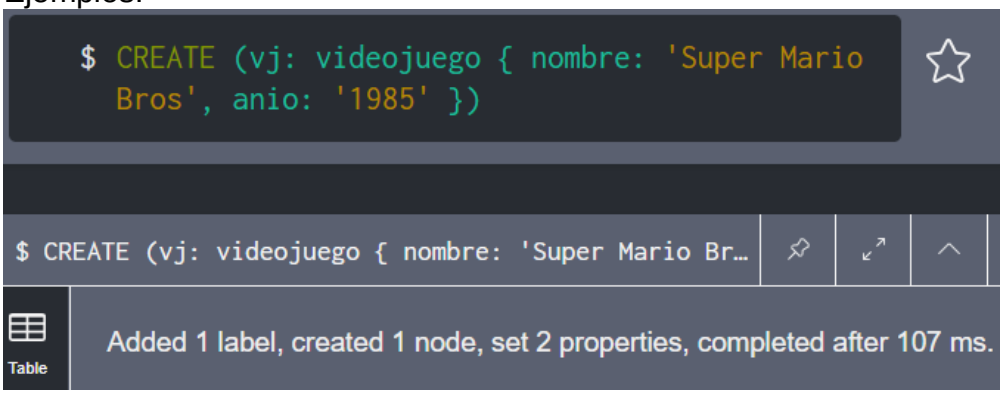

Ilustración 22 Crear nodos ej 1

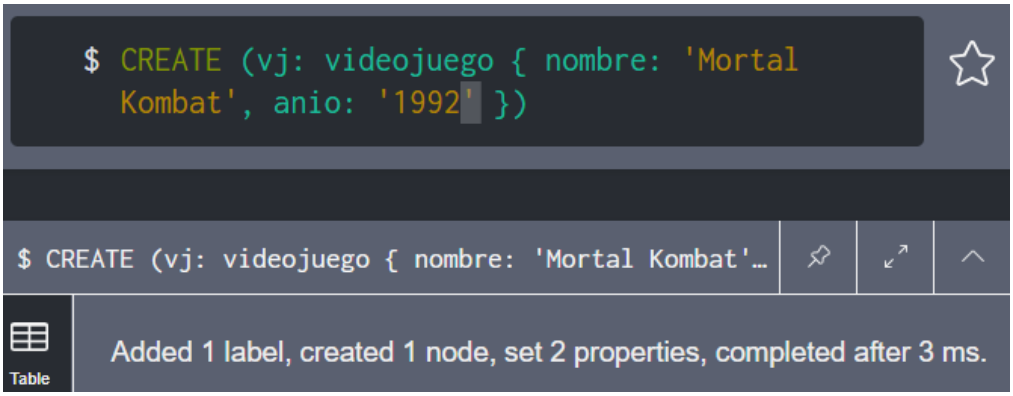

Ilustración 23 Crear nodos ej 2

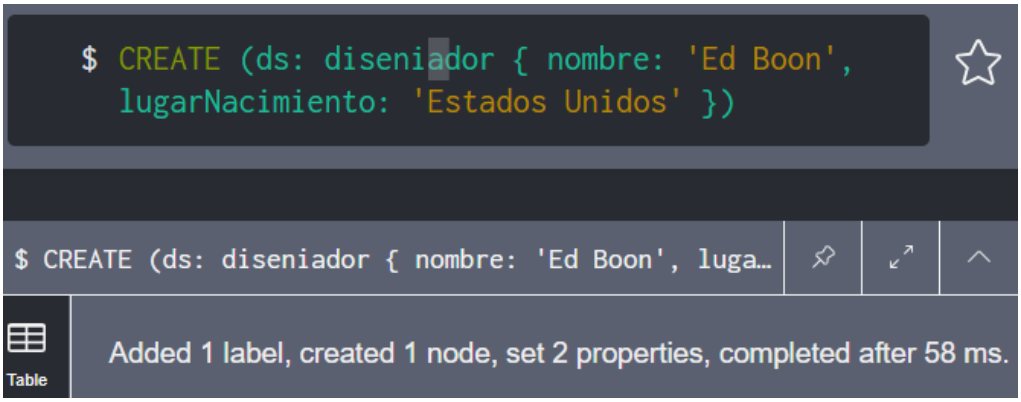

Ilustración 24 Crear nodos ej 3

b. Crear relaciones

Se utiliza la sintaxis:
MATCH (_variableN1: _etiquetaN1), (_variableN2: etiquetaN2) … CREATE (_variableN1)
- [_variableR1: _etiquetaR1 { _propiedadR1: _valorR1 }] -> (_variableN2)

Donde:

- _variableN: Es una variable, se utiliza como si se renombrara la categoría del nodo.
- _etiquetaN: Categoría a la que pertenece el nodo.
- _variableR: Variable que contendrá al grupo de la relación.
- _etiquetaR: Grupo al cual pertenecerá la relación.
- _propiedadR: Refiere a un atributo de la relación.
- _valorR: Contenido del atributo de la relación.

Ejemplo:
Creación de relación entre el videojuego Mortal Kombat y su diseñador Ed Boon:

MATCH (a: videojuego), (b: diseniador) WHERE a.nombre = 'Mortal Kombat' AND b.nombre = 'Ed Boon' CREATE (a) - [rel1: creadoPor { fecha: a.anio }]-> (b)

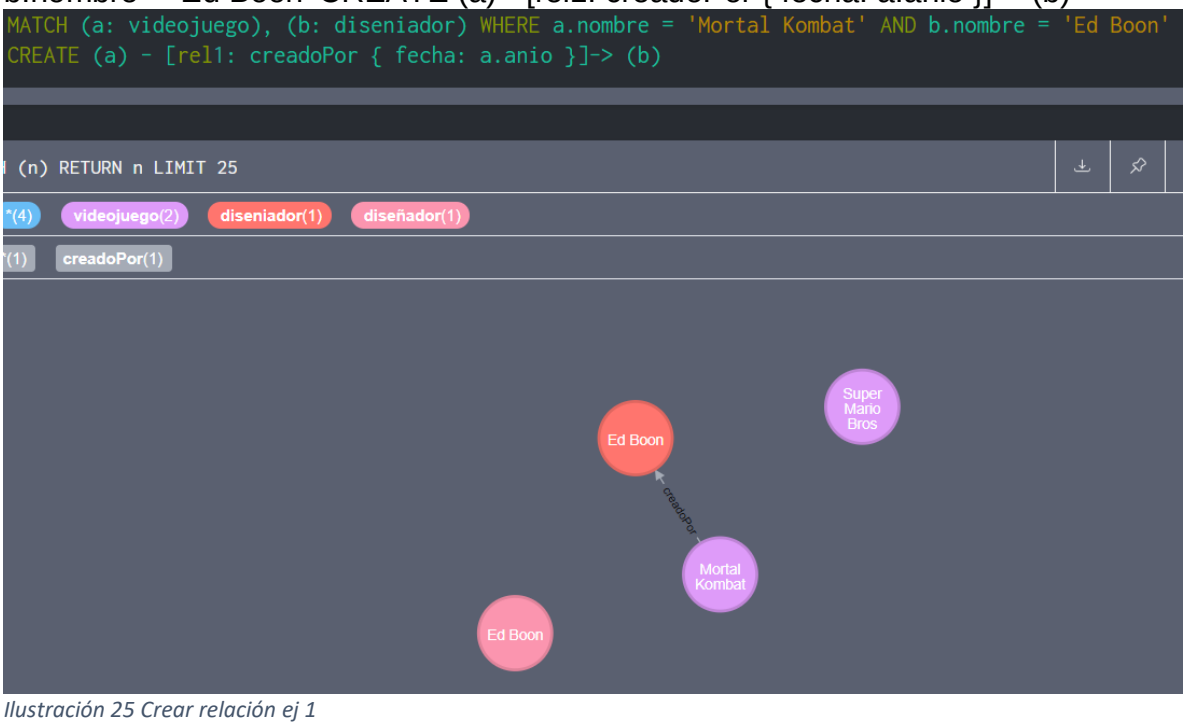

Ilustración 25 Crear relación ej 1

c. Modificar nodos

Se utiliza la sintaxis:

MATCH (_variableN1: _etiquetaN1 { _propiedadN1: _valorN1 }) SET
_variableN1._propiedadN1 = _nuevoValorN1

Donde:

- _variableN: Variable que contendrá a la categoría del nodo.
- _etiquetaN: Categoría a la que pertenece el nodo.
- _propiedadN: Refiere a un atributo del nodo.
- _valorN: Es el valor de un atributo del nodo.
- _nuevovalorN: Es el valor modificado de la propiedad (de nodo) establecida.

Ejemplo:
Modificar el título de “Super Mario Bros” a “Super Mario Brothers”:

MATCH (vj: videojuego { nombre: ' Super Mario Bros' }) SET vj.nombre = 'Super Mario Brothers'

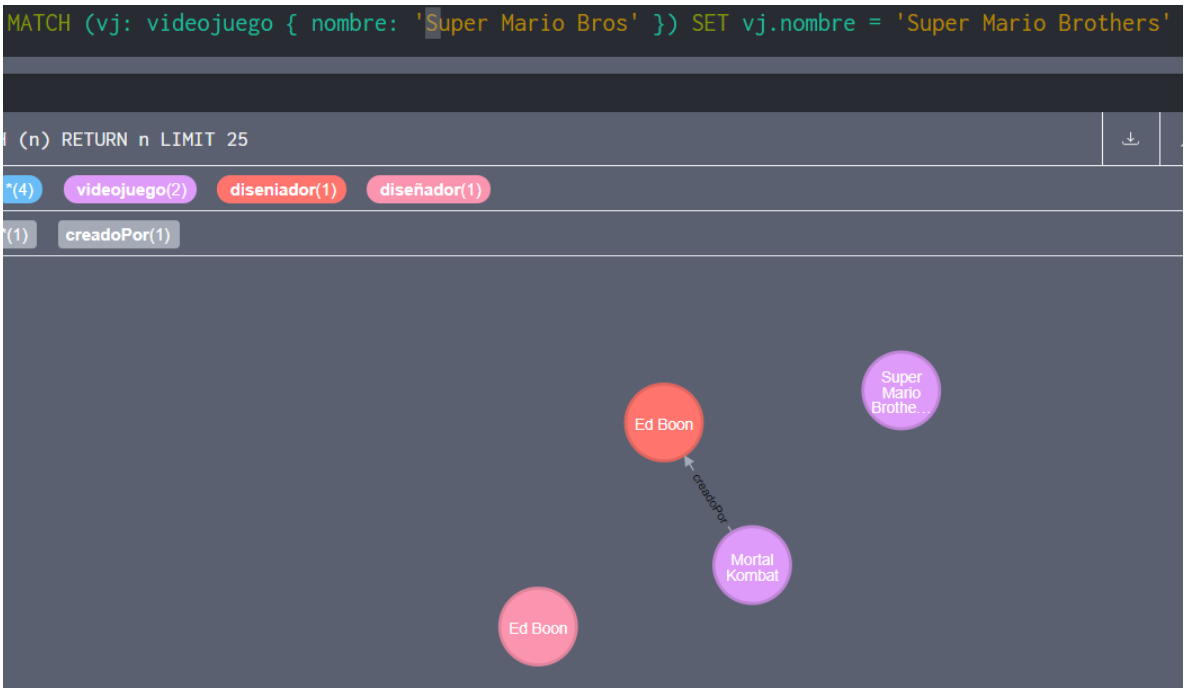

Ilustración 26 Modificar nodos ej 1

d. Borrar nodos

Se utiliza la sintaxis:

```
MATCH (_variableN1: _etiquetaN1 { _propiedadN1: _valorN1 }) - [ _variableR1:
                    _etiquetaR1 { _propiedadR1: _valorR1 }] -> (
  _variableN2: _etiquetaN2 { _propiedadN2: _valorN2 }) DELETE _variableR1
```

Donde:

- _variableN: Es una variable, se utiliza como si se renombrara la categoría del nodo.
- _etiquetaN: Categoría a la que pertenece el nodo.
- _variableR: Variable que contendrá al grupo de la relación.
- _etiquetaR: Grupo al cual pertenecerá la relación.
- _propiedadN: Refiere a un atributo del nodo.
- _valorN: Contenido del atributo del nodo.
- _propiedadR: Refiere a un atributo de la relación.
- _valorR: Contenido del atributo de la relación.

Nota: Primero se deben eliminar dependencias del nodo, por ejemplo, las relaciones que tenga asociadas.

Ejemplo:
Eliminar Ed Boon (Diseñador, color rosa). Para ello también se eliminará su relación:

```
MATCH (vj: videojuego { nombre: 'Mortal Kombat'}) - [r1: creadoPor { fecha: '1992' }]-> (ds: diseñador { nombre: 'Ed Boon' }) DELETE r1

MATCH (ds: diseñador { nombre: 'Ed Boon' }) DELETE ds
```

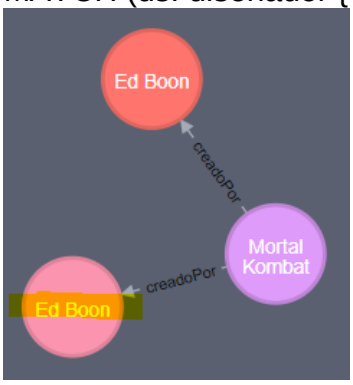

Ilustración 27 Borrar nodo antes

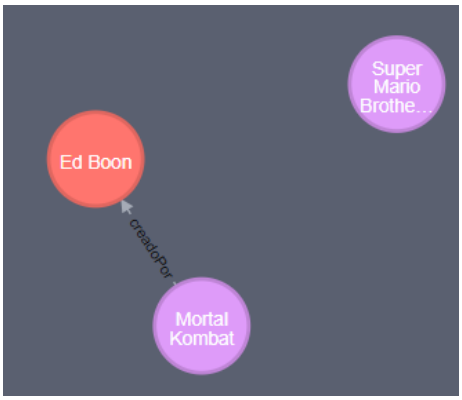

Ilustración 28 Borrar nodo después

e. Modificar relaciones

Para modificar se utiliza la sintaxis:

```
MATCH (_variableN1: _etiquetaN1 { _propiedadN1: _valorN1 }) - [ _variableR1:
_etiquetaR1 { _propiedadR1: _valorR1 }] -> (_variableN2: _etiquetaN2 { _propiedadN2:
_valorN2 }) SET _variableR1._propiedadR1 = _nuevoValorR1
```

Para remover se utiliza la sintaxis:

```
MATCH (_variableN1: _etiquetaN1 { _propiedadN1: _valorN1 }) - [ _variableR1:
_etiquetaR1 { _propiedadR1: _valorR1 }] -> (_variableN2: _etiquetaN2 { _propiedadN2:
_valorN2 }) REMOVE _variableR1._propiedadR1
```

Donde:

- _variableN: Es una variable, se utiliza como si se renombrara la categoría del nodo.
- _etiquetaN: Categoría a la que pertenece el nodo.
- _variableR: Variable que contendrá al grupo de la relación.
- _etiquetaR: Grupo al cual pertenecerá la relación.
- _propiedadN: Refiere a un atributo del nodo.
- _valorN: Contenido del atributo del nodo.
- _propiedadR: Refiere a un atributo de la relación.
- _valorR: Contenido del atributo de la relación.
- _nuevoValorR1: Es el valor modificado de la propiedad (de relación) establecida.

Ejemplo:

Modificar fecha de creación, en la relación “CreadoPor”

```
MATCH (vj: videojuego { nombre: 'Mortal Kombat'}) - [r1: creadoPor] -> (ds: diseniador { nombre:
'Ed Boon' }) SET r1.fecha = '22/08/1992'
```

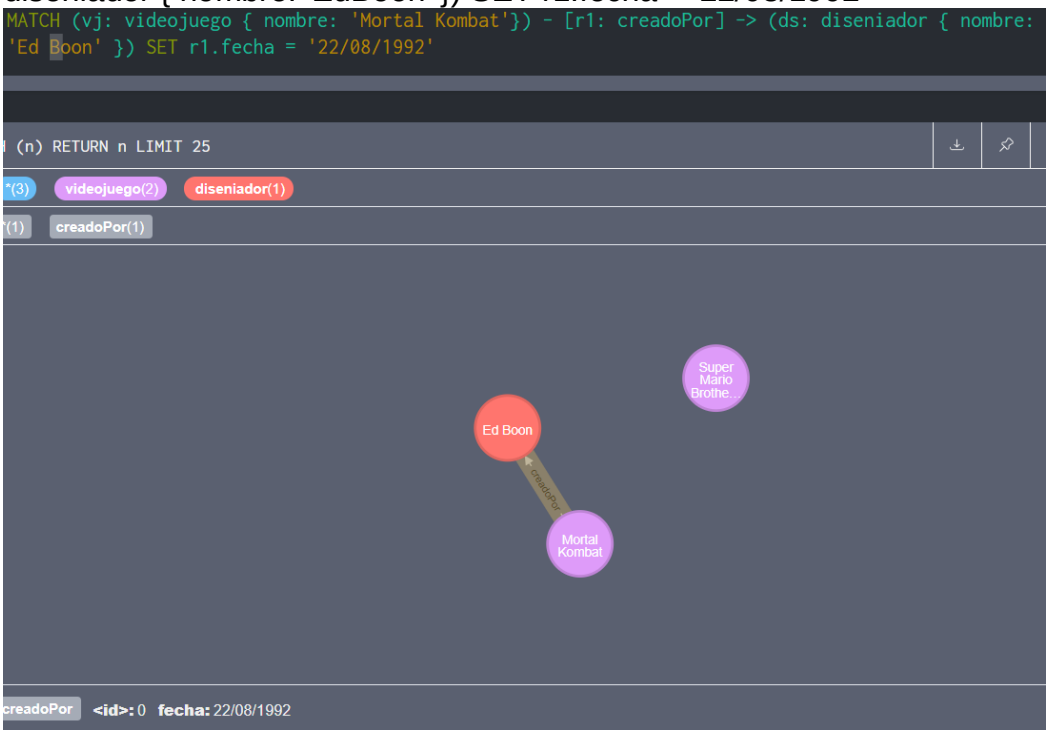

Ilustración 29 Modificar Relación

5. Consultas

A continuación se muestra como realizar las consultas con match, where y a utilizar las funciones de agregación.

a. Consultas con match, con where

Primero se mostrará cómo realizar consultas con match y con where:

Match

Se utiliza la sintaxis:

```
MATCH (_variableN1: _etiquetaN1 { _propiedadN1: _valorN1 }), (_variableN2:
  _etiquetaN2 { _propiedadN2: _valorN2 }) RETURN _variable1, _variable 2
```

Donde:

- _variableN: Es una variable, se utiliza como si se renombrara la categoría del nodo.
- _etiquetaN: Categoría a la que pertenece el nodo.
- _propiedadN: Refiere a un atributo del nodo.
- _valorN: Contenido del atributo del nodo.

Nota: Match es la búsqueda por medio de parámetros, mientras que las variables de Return son aquellas que devolverá la consulta.

Ejemplo:
Consultar el videojuego Mortal Kombat y a su diseñador(es) asociados:

MATCH (vj: videojuego { nombre: 'Mortal Kombat'}) --> (ds: diseniador) RETURN vj, ds

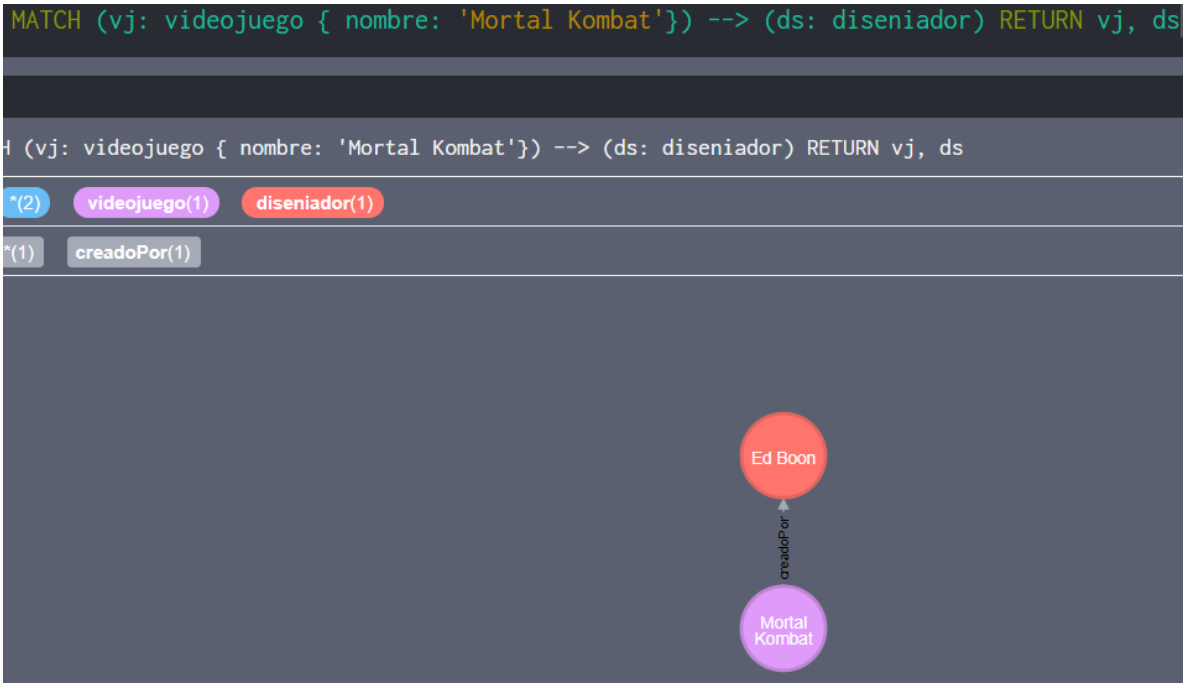

Ilustración 30 Consulta Match

Where

Se utiliza la sintaxis:

```
MATCH (_variableN1: _etiquetaN1 { _propiedadN1: _valorN1 }), (_variable2: _etiquetaN2
    { _propiedadN2: _valorN2 }) WHERE _variableN1._propiedadN1 = _valorFiltro1 AND
        _variableN2._propiedadN2 = _valorFiltro2 RETURN _variableN1, _variableN2
```

Donde:

- _variableN: Es una variable, se utiliza como si se renombrara la categoría del nodo.
- _etiquetaN: Categoría a la que pertenece el nodo.
- _propiedadN: Refiere a un atributo del nodo.
- _valorN: Contenido del atributo del nodo.
- _valorFiltro: es la variable que se utiliza para filtrar.

Nota: Match es la búsqueda por medio de parámetros, Where es utilizado para realizar consultas complejas, mientras que las variables de Return son aquellas que devolverá la consulta.

Ejemplo:
Consultar el videojuego que se llame Mortal Kombat y que haya salido en 1992:

```
MATCH (vj: videojuego) WHERE vj.nombre = 'Mortal Kombat' AND vj.anio = '1992'
RETURN vj
```

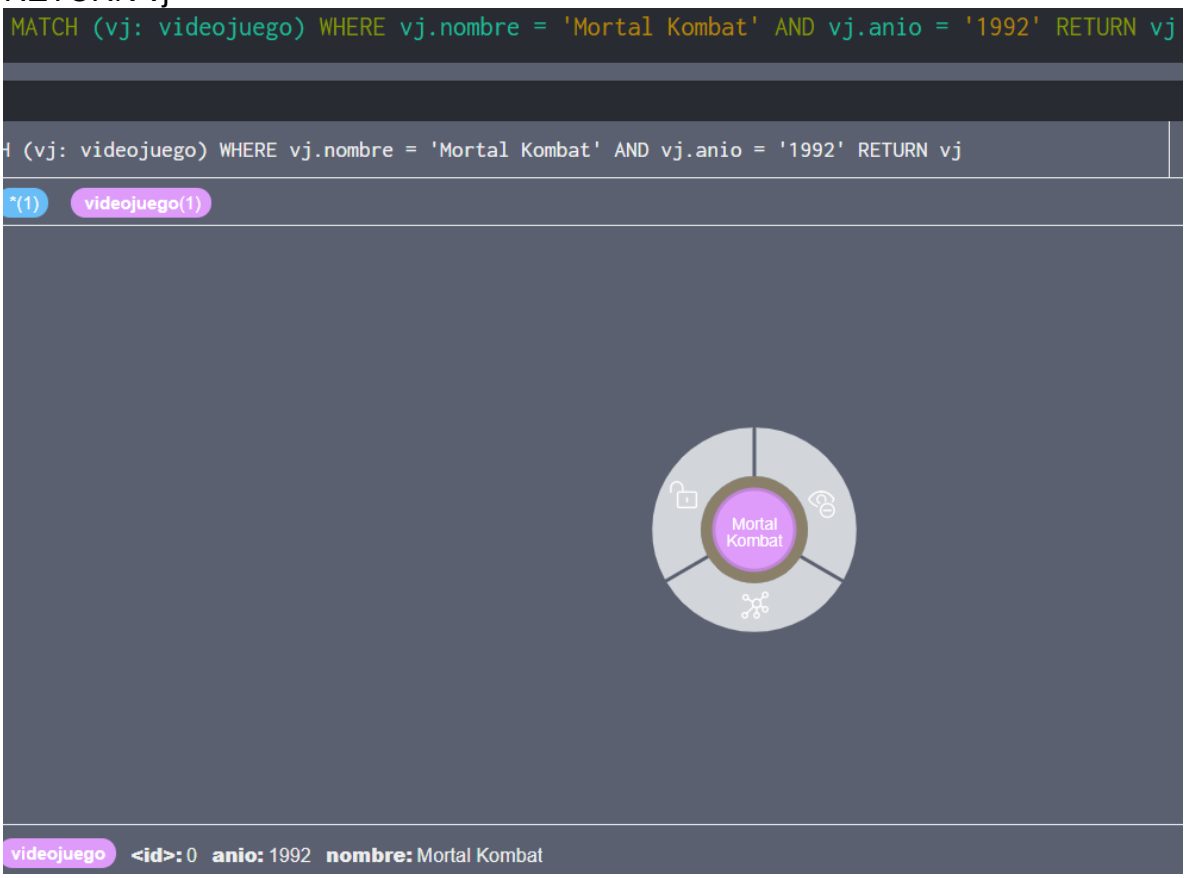

Ilustración 31 Consulta con Where

b. Funciones de agregación

Para calcular datos agregados, Cypher ofrece agregación, análoga a la de SQL `GROUP BY` (neo4j.com, 2018). Las funciones de agregación (y ejemplos de sintaxis) que se pueden realizar en Neo4j son:

avg() - Numeric values

```
MATCH (n:Person)
RETURN avg(n.age)
```

avg() - Durations

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

collect()

```
MATCH (n:Person)
RETURN collect(n.age)
```

count()

```
MATCH (n { name: 'A' })-->(x)
RETURN labels(n), n.age, count(*)
```

max()

```
UNWIND [1, 'a', NULL , 0.2, 'b', '1', '99'] AS val
RETURN max(val)
```

min()

```
UNWIND [1, 'a', NULL , 0.2, 'b', '1', '99'] AS val
RETURN min(val)
```

percentileCont()

```
MATCH (n:Person)
RETURN percentileCont(n.age, 0.4)
```

percentileDisc()

```
MATCH (n:Person)
RETURN percentileDisc(n.age, 0.5)
```

stDev()

```
MATCH (n)
WHERE n.name IN ['A', 'B', 'C']
RETURN stDev(n.age)
```

stDevP()

```
MATCH (n)
WHERE n.name IN ['A', 'B', 'C']
RETURN stDevP(n.age)
```

sum() - Numeric values

```
MATCH (n:Person)
RETURN sum(n.age)
```

sum() - Durations

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

Las funciones de agregación toman un conjunto de valores y calculan un valor agregado sobre ellos. Algunos ejemplos son los `avg()` que calculan el promedio de varios valores numéricos o `min()` que encuentran el valor numérico o de cadena más pequeño en un conjunto de valores. Cuando decimos a continuación que una función de agregación opera en un *conjunto de valores* , queremos decir que estos son el resultado de la aplicación de la expresión interna (como `n.age`) a todos los registros dentro del mismo grupo de agregación (neo4j.com, 2018).

La agregación se puede calcular sobre todos los subgrafos coincidentes, o se puede dividir aún más introduciendo claves de agrupación. Estas son expresiones no agregadas, que se utilizan para agrupar los valores que entran en las funciones agregadas.

Supongamos que tenemos la siguiente declaración de retorno: RETURN n, count(*).

Tenemos dos expresiones de retorno:, `n` y `count(*)`. La primera, `n` no es una función agregada, y por lo tanto será la clave de agrupación. El último, `count(*)` es una expresión agregada. Los subgrafos coincidentes se dividirán en diferentes grupos, dependiendo de la clave de agrupación. La función agregada se ejecutará en estos grupos, calculando un valor agregado por grupo. (neo4j.com, 2018)
Para usar agregaciones para ordenar el conjunto de resultados, la agregación debe incluirse en la `RETURN` que se usará en el archivo `ORDER BY`.
El `DISTINCT` operador trabaja en conjunto con la agregación. Se utiliza para hacer que todos los valores sean únicos antes de ejecutarlos a través de una función agregada.

Ejemplo:
Consultar el número de videojuegos que salieron en 1992.

MATCH (vj:videojuego) WHERE vj.anio = '1992' RETURN count(*)

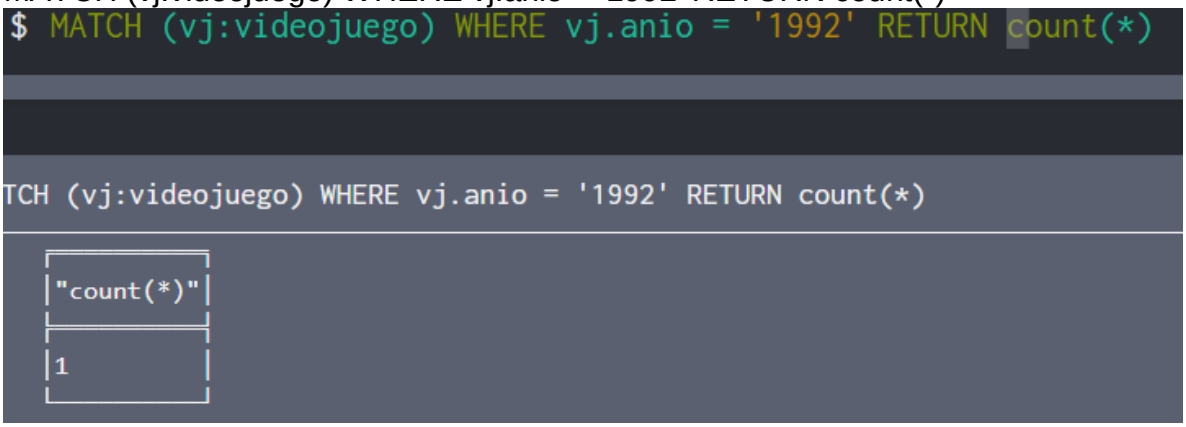

Ilustración 32 Funcion agregación Count

6. Conexión a la base de datos desde un lenguaje de programación

Se realizó la aplicación en JAVA, para ello se utilizó el driver de neo4j en las bibliotecas:

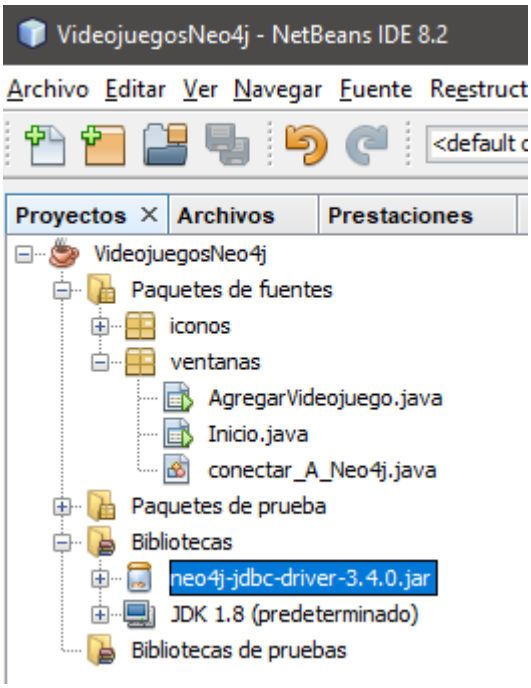

Ilustración 33 Conexión

Para crear la conexión se realizó la clase “conectar_A_Neo4j”, en la cuál se ingresan los parámetros tales como el driver, la url, el usuario, contraseña y esquema. Así como mensajes para saber si se conectó o no a la bd.

```
public class conectar_A_Neo4j {
    Connection con=null;
    public Connection conexion(){
        Connection con=null;
        try {
            //cargar driver
            Class.forName("org.neo4j.jdbc.Driver");
            con = (Connection) DriverManager.getConnection("jdbc:neo4j:bolt://localhost/?user=neo4j,password=1234,scheme=basic");
            System.out.println("conexion establecida");
            //JOptionPane.showMessageDialog(null, "conexion establecida");
        } catch (ClassNotFoundException | SQLException e) {
            System.out.println("conexion no establecida");
            JOptionPane.showMessageDialog(null, "conexion error"+e);
        }
        return con;

    }
}
```

Ilustración 34 Conexión

Posteriormente en la ventana de inicio se creó la interfaz, la cual hace uso de las imágenes en el paquete “íconos” de la ilustración 33:

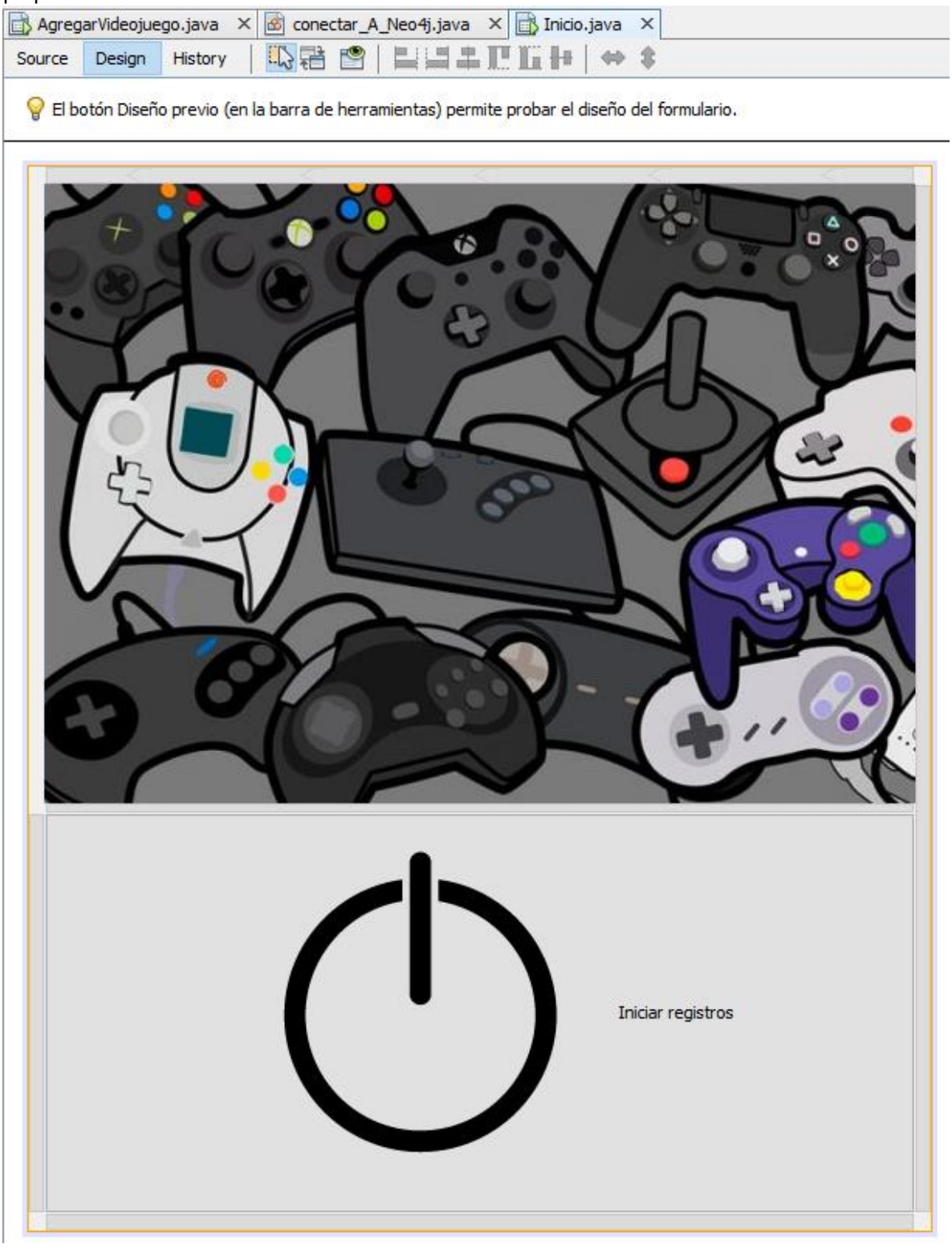

Ilustración 35 Conexión

Al hacer clic en Iniciar registros, se abre la ventana AgregarVideojuego:

Ilustración 36 Conexión

Dentro de esta ventana se crea la conexión:

```
public class AgregarVideojuego extends javax.swing.JDialog {
conectar_A_Neo4j Conexion = new conectar_A_Neo4j();
Connection BD= Conexion.conexion();


    public AgregarVideojuego(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        initComponents();
    }
```

Ilustración 37 Conexión

Así mismo el botón de guardar verifica si los datos ingresados son válidos y que no estén duplicados en la base de datos:

```
private void acepActionPerformed(java.awt.event.ActionEvent evt) {
String Query;
Query = "MATCH (videojuego) WHERE videojuego.nombre = {1} RETURN videojuego";
try {
    PreparedStatement pst= BD.prepareStatement(Query);
    pst.setString(1, bNombre.getText());
    ResultSet rs = pst.executeQuery();

    int cont = 0;
    while(rs.next()){
        cont++;
    }

    System.out.println("se enconntraron en la base de datos"+cont);
    if (cont > 0) {
        JOptionPane.showMessageDialog(null, "El nombre del videojuego duplicado");
   bNombre.setText("");
   bAnio.setText("");
    }else{
        String nombrev = bNombre.getText();
        int nomtam = nombrev.length();
        if(nomtam > 0){
           nuevoNodo();
        }else{
            JOptionPane.showMessageDialog(null, "Debe ingresar un nombre");
        }

    }

    } catch (SQLException e) {
        JOptionPane.showMessageDialog(null, e.getMessage());
    }
}
```

Ilustración 38 Conexión

Finalmente crea el nuevo nodo en la BD:

```
public void nuevoNodo(){
    String nombre,anio;
    String newNodo;
    nombre = bNombre.getText();
    anio = bAnio.getText();

    newNodo = "CREATE(vj:videojuego{nombre:'"+nombre+"', anio:'"+anio+"'})";

    try {
         PreparedStatement pst=BD.prepareStatement(newNodo);
         int n=pst.executeUpdate();
         if (n>0){
             JOptionPane.showMessageDialog(null, "Agregado correctamente");
             bNombre.setText("");
             bAnio.setText("");
         }
     } catch (SQLException ex) {
        JOptionPane.showMessageDialog(null, ex.getMessage());
     }
}
```

Ilustración 39 Conexión

Cuando se ingresan los datos y se presiona “Guardar” se muestra el siguiente mensaje:

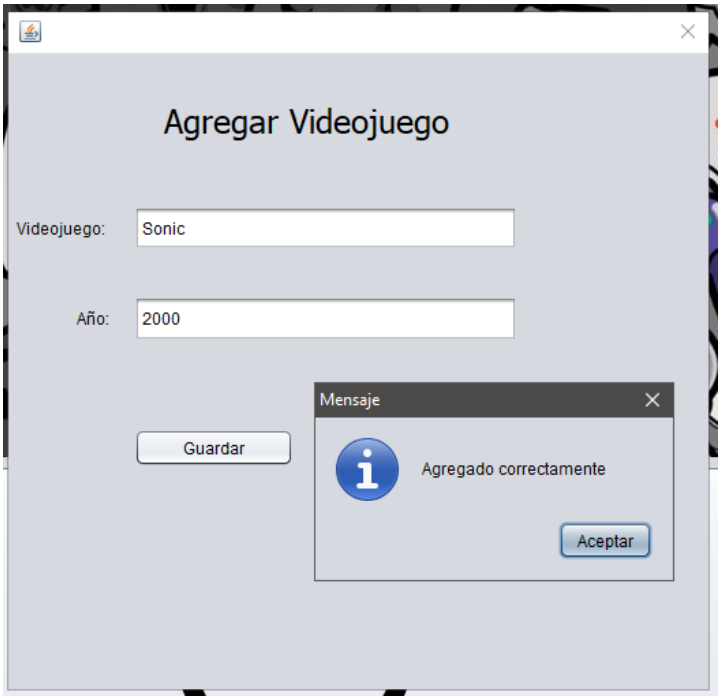

Ilustración 40 App

Entonces se puede observar que se guardó correctamente:

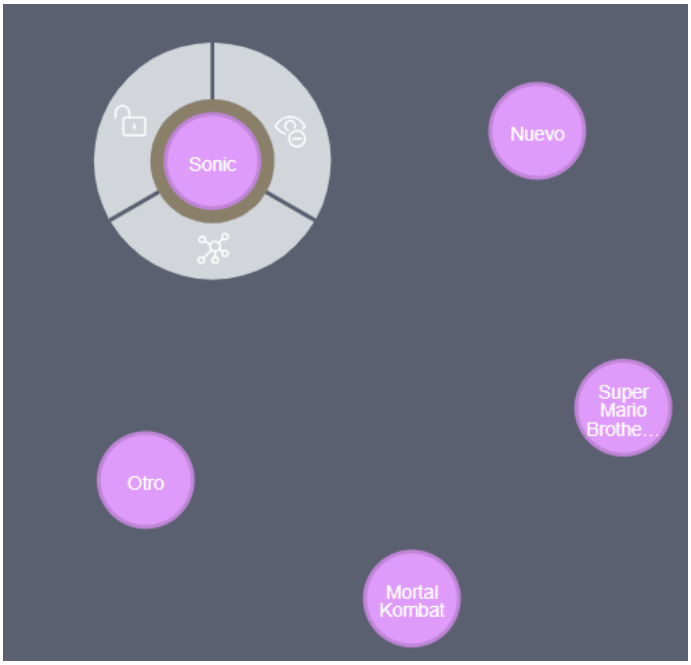

Ilustración 41 BD

7. Referencias

Abalos, N. (14 de abril de 2014). *Bases de datos orientadas a grafos*. Obtenido de BEEVA: https://www.beeva.com/beeva-view/tecnologia/bases-de-datos-orientadas-a-grafos/

Digital Guide. (agosto de 29 de 2018). *Bases de datos relacionales: el modelo de datos en detalle*. Obtenido de 1&1 IONOS: https://www.ionos.mx/digitalguide/hosting/cuestiones-tecnicas/bases-de-datos-relacionales/

EcuRed. (2009). *SGBD*. Obtenido de EcuRed: https://www.ecured.cu/SGBD

Iruela, J. (19 de enero de 2016). *Los gestores de bases de datos más usados*. Obtenido de Revista digital INESEM: https://revistadigital.inesem.es/informatica-y-tics/los-gestores-de-bases-de-datos-mas-usados/

Kyocera. (7 de abril de 2017). *Características avanzadas de un SGBD*. Obtenido de Kyocera Document Solutions: https://smarterworkspaces.kyocera.es/blog/caracteristicas-avanzadas-sgbd/

Melgarejo, Á. I., & Moya, Á. O. (2018). *Sistemas Gestores de Bases de Datos*. Obtenido de GPLSI: http://gplsi.dlsi.ua.es/bbdd/bd1/lib/exe/fetch.php?media=bd1:0910:trabajos:aimsgbd.pdf

Ramos, J. A. (7 de julio de 2014). *Primeros pasos con Neo4j*. Obtenido de Adictos al trabajo: https://www.adictosaltrabajo.com/2014/07/07/neo4j-first-steps/

Rodriguez, J. L. (7 de julio de 2016). *Uso práctico de Java y Neo4j: Sistema de recomendación*. Obtenido de Adictos al trabajo: https://www.adictosaltrabajo.com/2016/07/07/uso-practico-de-java-y-neo4j-sistema-de-recomendacion/

Sánchez, J. (2018). *Manual de Gestión de Bases de Datos*. Obtenido de JorgeSanchez.net: https://jorgesanchez.net/manuales/gbd/modelo-relacional.html

Sancho, F. (28 de enero de 2014). *Bases de Datos en Grafo*. Obtenido de Fernando Sancho Caparrini: http://www.cs.us.es/~fsancho/?e=79

Sara, M. (20 de abril de 2017). *Bases de datos NoSQL : Guía definitiva*. Obtenido de PandoraFMS: https://blog.pandorafms.org/es/bases-de-datos-nosql/