



CUADERNO DE TRABAJO SIBI

Adrián Diez Valbuena

CUADERNO DE TRABAJO

Este es mi cuaderno de trabajo de la asignatura Sistemas de Información de Gestión y Bussines Intelligence, en este documento iré reflejando mis avances a medida que pase el tiempo y además contendrá información acerca de las tecnologías y programas que iré utilizando.

En este curso conoceremos una de las nuevas tecnologías existentes, Neo4j.

Neo4j es un software de bases de datos. La gran diferencia respecto a los demás programas que implementan bases de datos, es que neo4j trabaja con bases de datos orientas a grafos, es decir, sustituyen las tablas de las bases de datos tradicionales por grafos de conocimiento que almacenan la información.

Una base de datos orientada a objetos es un sistema de gestión de bases de datos en línea que trabaja con las operaciones CRUD (Create, Read, Update, Delete) que nos permiten crear información en la base de datos, leer datos de la misma, actualizar nodos o relaciones y borrar datos ya existentes.

Este tipo de bases de datos son generalmente diseñadas para usar con sistemas de procesamiento de transacciones en línea (OLTP). En consecuencia, normalmente están optimizadas para el rendimiento transaccional y diseñadas teniendo en cuenta la integridad transaccional y la disponibilidad operativa.

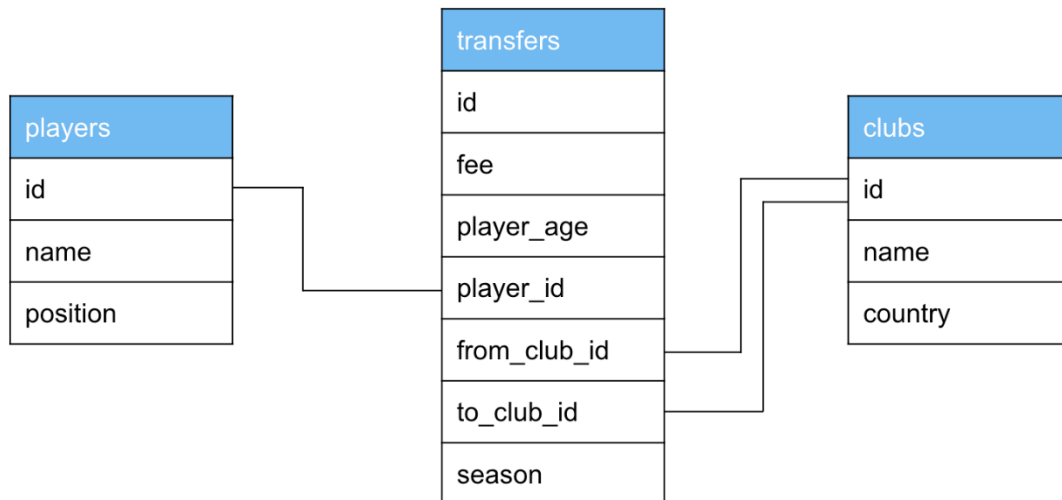
Las bases de datos orientadas a grafos representan las entidades como los nodos de un grafo y las relaciones entre ellos con las aristas del mismo, permitiendo que se pueda usar teoría de grafos para acceder a toda la información. Este tipo de bases de datos ofrecen servicios nuevos y mejorados de los que ofrecían las bases de datos tradicionales:

- Consultas mucho más amplias y sin necesidad de demarcarlas por tablas
- No es necesario definir un número determinado de atributo, es decir, cada nodo puede estar relacionado con un número diferente de nodos, y esto, sin ocupar espacio adicional
- Los registros no poseen una longitud limitada, por lo que no hay que definir un tamaño para cada atributo.

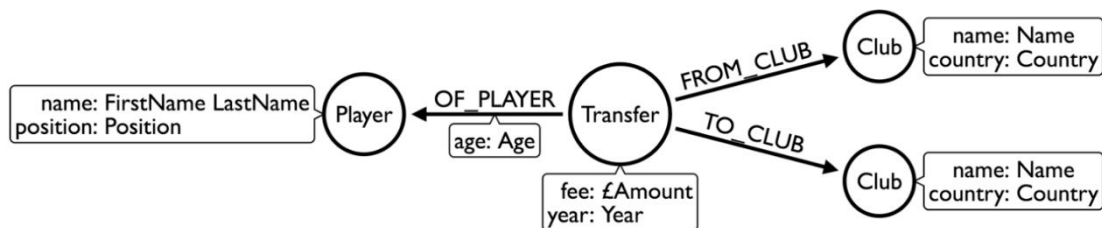
- Se puede recorrer la base de datos de una forma jerárquica, por ejemplo, obtener un nodo y su padre o abuelo si lo tuviese.

Aquí muestro la diferencia del esquema de una base de datos relacional con el de una base de datos orientada a grafos:

Esquema base de datos relacional:



Esquema base de datos orientada a grafos:



Como se puede observar, en las bases de datos de grafos las entidades se sustituyen por nodos, las relaciones por aristas, y los atributos por etiquetas de los nodos.

En cuanto a Neo4j, aprenderemos a utilizarlo gracias a su página web, donde ofrecen la posibilidad de realizar diferentes cursos de aprendizaje para poder utilizar su software.

Una vez explicado esto, comienzo con el curso de introducción a neo4j.

Este primer curso de introducción tratará sobre unas lecciones básicas del manejo de la aplicación Neo4j Desktop, que utiliza el lenguaje de bases de datos “Cypher”.

En primer lugar, nos encontramos con unas cuestiones básicas sobre las bases de datos orientadas a grafos y que simplemente son conceptos teóricos que nos ayudarán a entender el funcionamiento del software de Neo4j.

A continuación, nos explicarán ciertos aspectos de la plataforma de Neo4j, así como describir sus componentes y beneficios.

Una de sus características, es que tiene adyacencia sin límite (Index-free Adjacency), esto significa que cuando se añade un nodo o relación a la base de datos, se almacena en la misma y cualquier acceso posterior a los datos se realiza mediante navegación con punteros, que es bastante rápida. Por ello, admite grafos muy grandes donde los datos conectados pueden atravesarse en tiempo constante sin la necesidad de un índice.

La plataforma de Neo4j implementa ACID (Atomicity, Consistency, Isolation and Durability), es decir:

- **Atomicidad:** Si cuando una operación consiste en una serie de pasos, bien todos ellos se ejecutan o bien ninguno, es decir, las transacciones son completas.
- **Consistencia:** Asegura que sólo se empieza aquello que se puede acabar.
- **Aislamiento:** Asegura que una operación no puede afectar a otras.
- **Durabilidad:** Esta propiedad asegura que, una vez realizada la operación, esta no se podrá deshacer, aunque falle el sistema y que de esta forma los datos sobrevivan de alguna manera.

Neo4j permite clusters que proporcionan alta disponibilidad, escalabilidad para el acceso de lectura de datos y “failover” que es importante para muchas empresas.

El motor de gráficos Neo4j se usa para interpretar las consultas de Cypher y también ejecuta código a nivel de kernel para almacenar y recuperar datos, ya sea en el disco o en la memoria caché.

Por último, Neo4j soporta varios lenguajes de programación para la realización de aplicaciones que utilicen este tipo de bases de datos, estos son: Java, JavaScript, Python y C#. En este caso, el lenguaje que utilizaré posteriormente será Python.

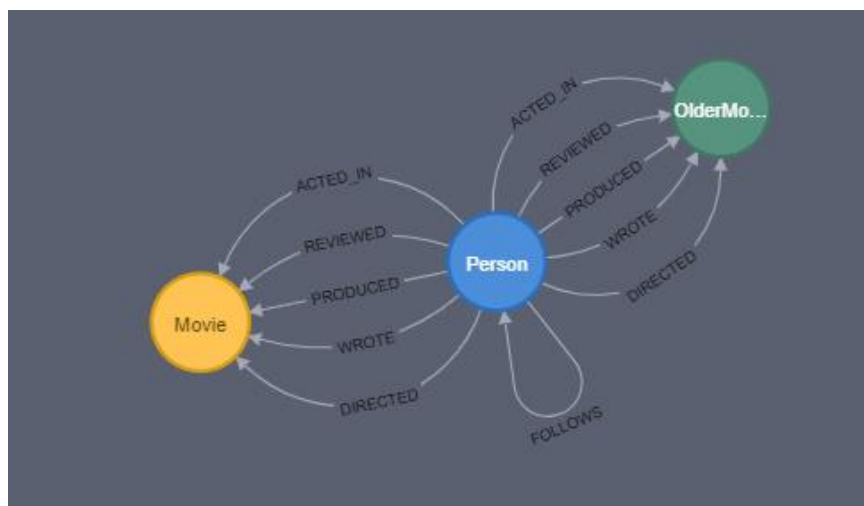
Una vez conocidas las funciones y propiedades de la plataforma de Neo4j, comenzaré a aprender su lenguaje de base de datos, Cypher.

Esta parte es la más extensa e importante del curso por lo que probablemente me lleve más tiempo que todo lo demás a la hora de entenderlo en su totalidad.

Para realizar esto, Neo4j nos ofrece una base de datos de películas realizada por ellos, en la que realizaré los diferentes ejercicios que vayan proponiendo.

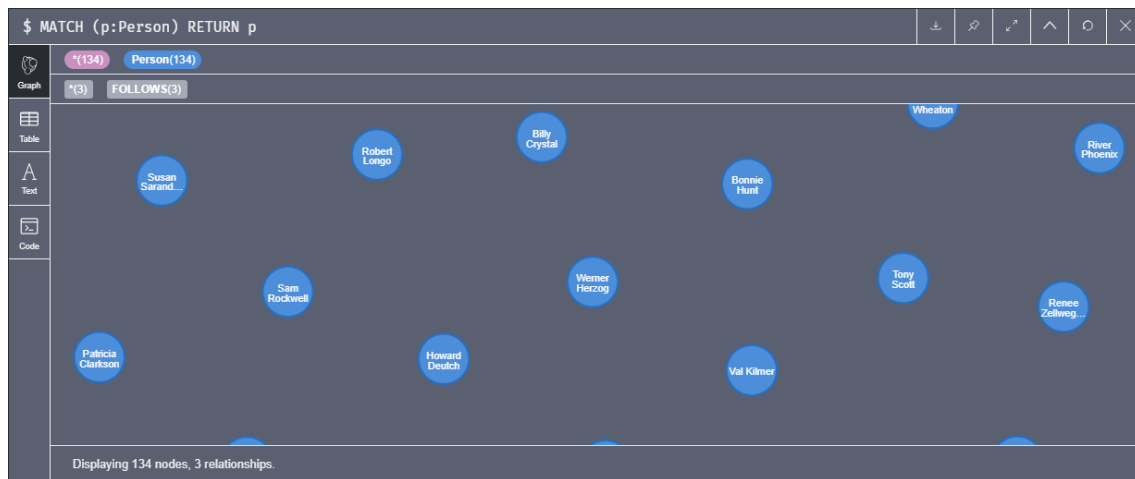
Iré mostrando algunas capturas de consultas que iré realizando.

El esquema de esta base de datos es el siguiente:

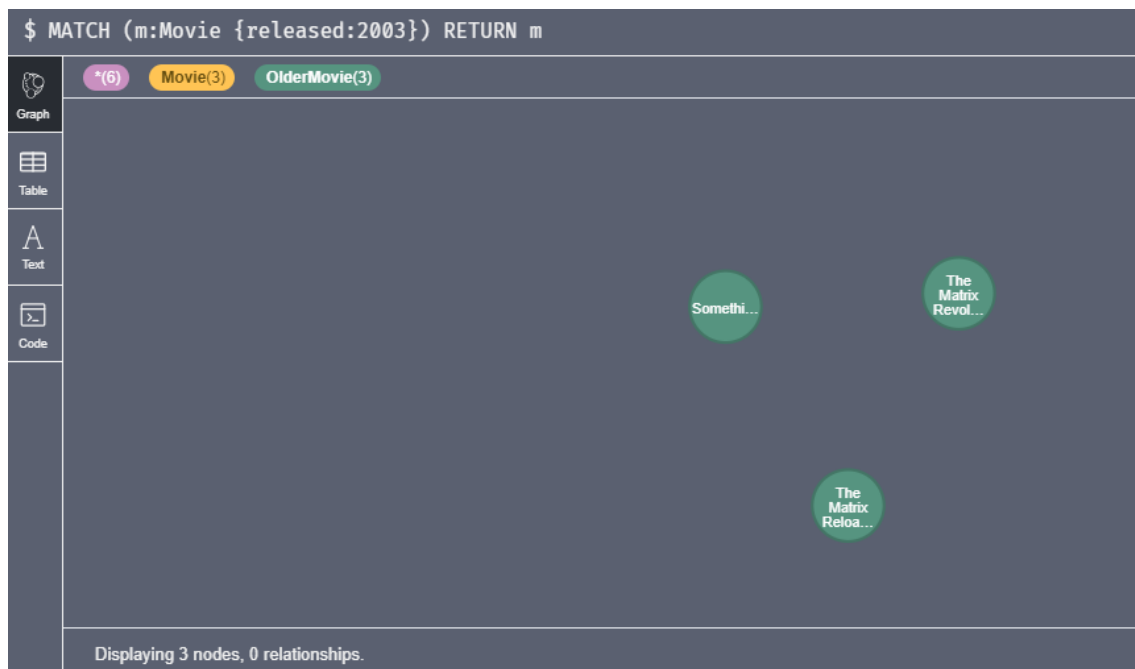


Como se puede observar, la base de datos cuenta con 3 tipos diferentes de nodos, películas, películas antiguas y personas. Las personas se relacionan de diferentes maneras con los otros dos tipos de nodos, además, de relacionarse con otros nodos de su mismo tipo.

Consulta que devuelve todos los nodos de personas en la base de datos.



Consulta que devuelve todas las películas estrenadas en 2003.



Consulta Cypher que nos muestra los nombres de todas las personas que han escrito la película "Speed Racer"

\$ MATCH (p:Person)-[:WROTE]->(m:Movie {title: 'Speed Racer'}) RETURN p.name	
Table	p.name
Text	"Lana Wachowski"
Code	"Lilly Wachowski"

Como se puede observar, estas consultas son bastante sencillas y simplemente nos sirven para empezar a tomar contacto con el lenguaje Cypher. A medida que voy realizando más ejercicios, las consultas son más complejas y concretas.

Consulta que busca todos los actores que no han actuado en más de 3 películas y devuelve su nombre y la lista de películas que han realizado.

```
1 MATCH (a:Person)-[:ACTED_IN]→(m:Movie)
2 WITH a, count(a) AS numMovies, collect(m.title) AS movies
3 WHERE numMovies ≤ 3
4 RETURN a.name, movies
```

\$ MATCH (a:Person)-[:ACTED_IN]→(m:Movie) WITH a, count(a) AS numMovies, collect(m.title) AS movies W...

a.name	movies
"Emil Eifrem"	["The Matrix"]
"Laurence Fishburne"	["The Matrix", "The Matrix Reloaded", "The Matrix Revolutions"]
"Carrie-Anne Moss"	["The Matrix", "The Matrix Reloaded", "The Matrix Revolutions"]
"Al Pacino"	["The Devil's Advocate"]
"Charlize Theron"	["The Devil's Advocate", "That Thing You Do"]
"James Marshall"	["A Few Good Men"]
"Kevin Pollak"	["A Few Good Men"]
"J.T. Walsh"	["A Few Good Men", "Hoffa"]
"Aaron Sorkin"	["A Few Good Men"]

Todas estas consultas aprendidas hasta ahora, solo sirven para realizar consultas a la base de datos, es decir, realizar el proceso de lectura de los registros contenidos en la base de datos. A continuación, veré como se pueden crear, modificar y eliminar registros de la misma.

Comando que crea una película con el título “Forrest Gump”

```
$ CREATE (:Movie {title: 'Forrest Gump'})
```

Es posible crear varios nodos con una misma consulta simplemente añadiéndolos a continuación del anterior.

Consulta de Cypher necesaria para crear una relación entre actores y una película.

```
1 MATCH (m:Movie)
2 WHERE m.title = 'Forrest Gump'
3 MATCH (p:Person)
4 WHERE p.name = 'Tom Hanks' OR p.name = 'Robin Wright' OR p.name = 'Gary Sinise'
5 CREATE (p)-[:ACTED_IN]→(m)
```

Código necesario para borrar un nodo y todas las relaciones que posee.

```
1 MATCH (m:Movie)
2 WHERE m.title = 'Forrest Gump'
3 DETACH DELETE m
```

Cypher nos permite añadir restricciones a los diferentes parámetros contenidos en la base de datos.

Con esta consulta, añadimos una restricción a la etiqueta de nombre de los nodos Persona de manera que este sea único obligatoriamente.

```
$ CREATE CONSTRAINT ON (p:Person) ASSERT p.name IS UNIQUE
```

Otra opción bastante interesante que nos ofrece Cypher, es importar datos desde un archivo csv, esta función es bastante atractiva ya que nos permite añadir a la base de datos una gran cantidad de información de una manera muy rápida y sencilla.

Mediante esta consulta, cargaríamos el archivo en la plataforma de Neo4j para poder sacar información del mismo.

```
LOAD CSV WITH HEADERS
FROM 'http://data.neo4j.com/intro-neo4j/actors.csv'
AS line
RETURN line.id, line.name, line.birthYear
```


Para que esto funcione correctamente, es necesario que el archivo csv este de la siguiente forma:

```
Id,name,birthyear
1,Charlie Sheen, 1965
2,Oliver Stone, 1946
3,Michael Douglas, 1944
4,Martin Sheen, 1940
5,Morgan Freeman, 1937
```

Se observa que Cypher es un lenguaje bastante sencillo e intuitivo que nos permite realizar todas las funciones que son necesarias a la hora de interactuar con una base de datos de grafos, lo que hace que la plataforma de Neo4j sea bastante completa y útil.

En mi caso, que anteriormente solo había utilizado SQL como lenguaje de gestión de bases de datos, creo que Cypher es mucho más sencillo de aprender y utilizar, además de más rápido y eficaz.

Con esto, he finalizado el curso de introducción de Neo4j y ya creo que podría realizar mi propia aplicación basada en una base de datos orientada a grafos así que me pondré a ello.

Mi idea de aplicación es un sistema de predicción de resultados de ciclismo, la idea será que la aplicación tenga dos diferentes apartados, el primero sería pronosticar la posición final de un corredor para una carrera específica basándome en sus resultados obtenidos anteriormente en esa misma carrera; la segunda parte será una predicción de un top 10 final para una carrera determinada, teniendo en cuenta todos los corredores que tendré almacenados en la base de datos. Por ello, lo primero que haré será comenzar con la construcción de la misma, para lo que me pondré a buscar información acerca de los corredores que pienso introducir.

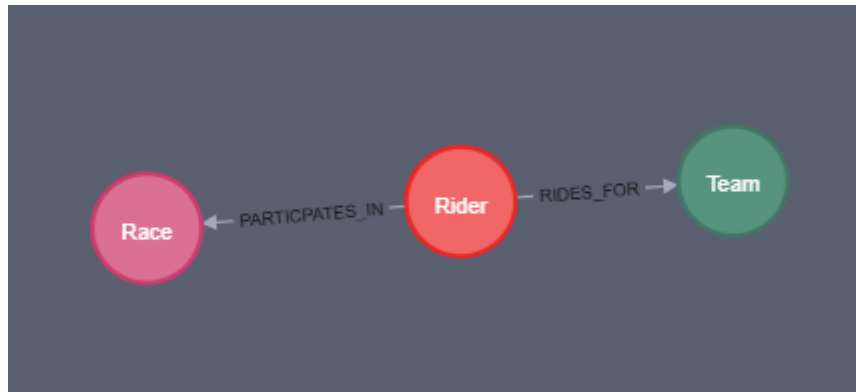
La idea inicial era que la base de datos constará de unas 25 carreras y unos 150 corredores, sin embargo, a la hora de ir construyéndola, me he dado cuenta de que sería algo que me llevaría mucho tiempo, y no es algo que sobre en este caso, por lo que me limitaré a trabajar solamente con 8 carreras, las 3 grandes vueltas y los 5 monumentos, y un número de unos 50 corredores.

Excel con los datos recopilados para tres corredores solamente.

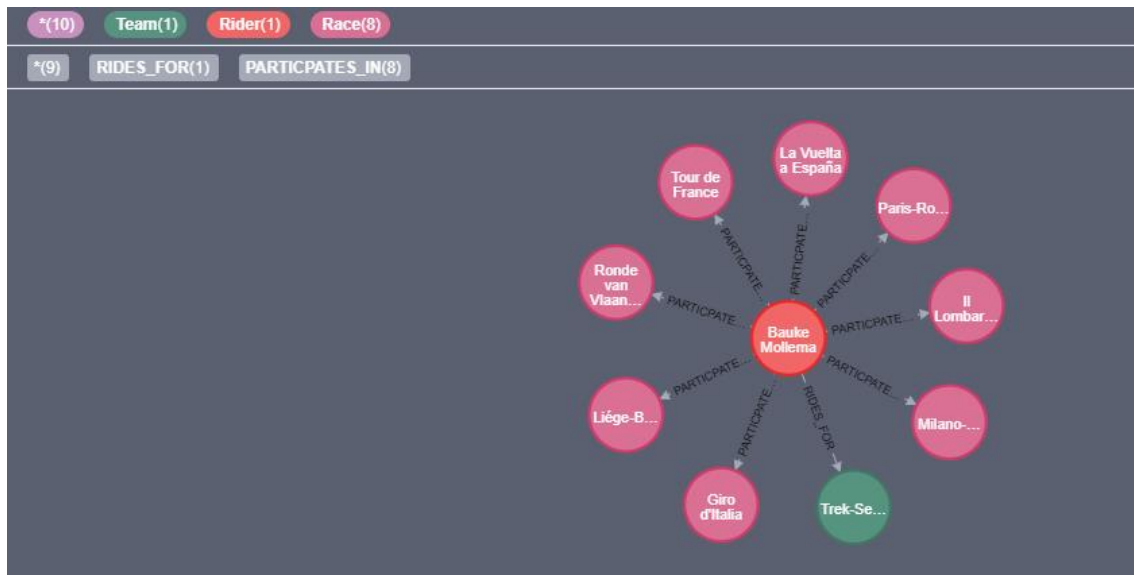
		San Remo	Ronde	Roubaix	Lieja	Giro	Tour	Vuelta	Lombardía
Roglic									
	2015	-	-	-	-	-	-	-	-
	2016	-	-	-	-	58	-	-	-
	2017	67	-	-	-	-	38	-	40
	2018	-	-	-	-	-	4	-	17
	2019	-	-	-	-	3	-	1	7
	Etapas	-	-	-	-	3	2	1	-
Fuglsang									
	2015	-	-	-	9	-	23	-	-
	2016	-	-	-	68	12	52	-	20
	2017	-	-	-	15	-	-	-	-
	2018	-	-	-	10	-	12	-	20
	2019	-	-	-	1	-	-	13	4
	Etapas	-	-	-	-	0	0	1	-
Valverde									
	2015	-	-	-	1	-	3	7	4
	2016	15	-	-	16	3	6	12	6
	2017	-	-	-	1	-	-	-	-
	2018	-	-	-	13	-	14	5	11
	2019	7	-	-	-	-	9	2	2
	Etapas	-	-	-	-	1	4	12	-

Finalizada la base de datos, me queda tal que así:

Esquema de la base de datos con corredores, carreras y equipos.



Nodo de un corredor con las carreras y el equipo al que pertenece.



Los resultados obtenidos de un corredor en una determinada carrera los he introducido en la relación entre ambos nodos, de manera que queden recogidos en un vector.

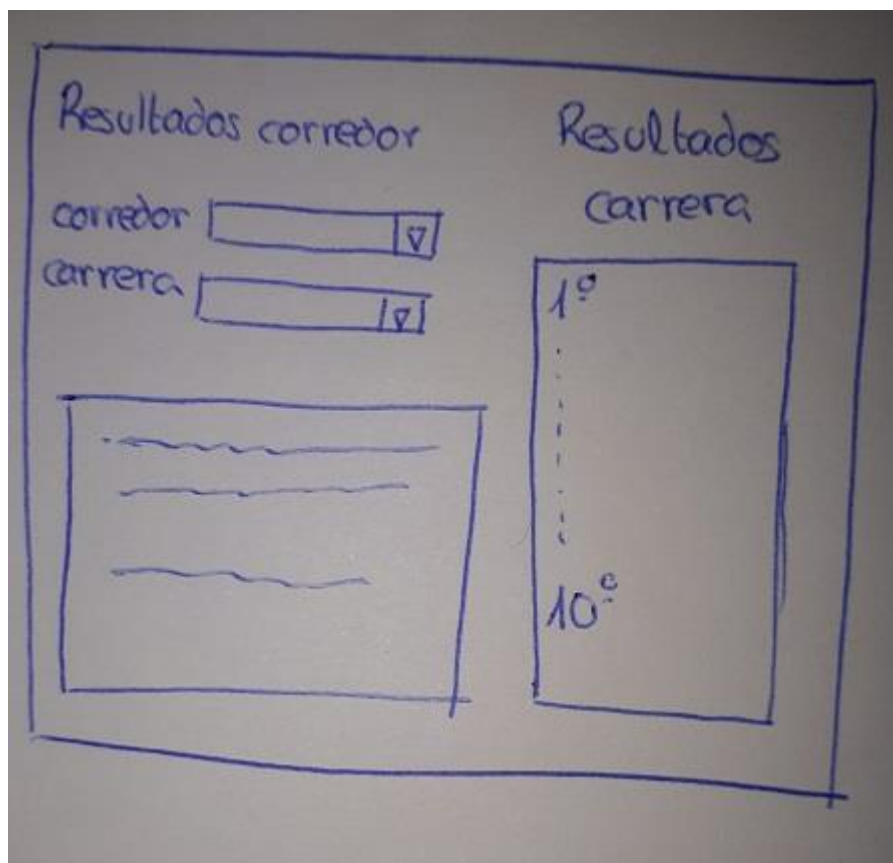
Una vez que ya he finalizado la base de datos, me dispongo a construir un código de Python, en primer lugar, debo de buscar información acerca de como realizar la conexión de mi código con la base de datos de Neo4j para disponer de los datos que contiene.

Tras probar varias formas para realizar la conexión, por fin he encontrado una que funcione.

```
#Conectar a la base de datos
from neo4j import GraphDatabase
uri = "bolt://localhost:11002"
driver = GraphDatabase.driver(uri, auth=("neo4j", "adrikun8"))
```

Donde uri es el puerto donde estamos utilizando la base de datos de neo4j (comprobar puerto en el Neo4j Desktop) y driver es lo que realizar la conexión mediante las credenciales de usuario y contraseña de la base de datos en cuestión.

A partir de aquí comenzaré con la realización de las ventanas y funcionalidades de la interfaz gráfica. He realizado un prototipo en papel que es más o menos como quiero que quede la aplicación al finalarla:



A partir de aquí iré realizando código de Python sin ninguna dificultad por lo que no lo considero algo relevante, hasta que llegue la hora de ver como estableceré los coeficientes para pronosticar los resultados.

He finalizado la interfaz y queda de la siguiente forma:



The screenshot shows a web application window titled "Cycling Results". It contains two main sections for predictions:

- Resultado pronosticado de corredores**: This section has a dropdown menu for "Corredor:" with "Romain Bardet" selected, and another dropdown for "Carrera:". Below these is an "Aceptar" button.
- Top 10 pronosticado de carrera**: This section has a dropdown menu for "Carrera:" and an "Aceptar" button below it.

Below these two sections is a large, solid black rectangular area, which is intended to display the prediction results.

En la parte izquierda podremos seleccionar el corredor del que queremos obtener su resultado y la carrera en la que lo queremos pronosticar, y, en la parte derecha, seleccionaremos la carrera de la que queremos saber cuál sería el top 10 pronosticado. La idea es que, una vez establecidos los coeficientes y realizados los cálculos necesarios, el resultado se muestre en el cuadro de texto negro de abajo

Como se puede observar es bastante simple de momento, pero lo que ahora me importa es que me dé resultados por lo que me dispongo a implementar el código que dará los coeficientes necesarios.

Teniendo en cuenta que el ciclismo es un deporte muy difícil de pronosticar, es decir, no es tan sencillo como victoria, empate, o derrota en el caso de un deporte como el fútbol o el baloncesto, no he encontrado un algoritmo ya establecido que pueda resultarme

atractivo para realizar la idea que tengo en mente, por lo que he decidido que los cálculos los haré yo mismo, añadiendo diferentes coeficientes a los resultados anteriores de un corredor en función de cuanto hace de ese resultado y de si ha participado más o menos veces en esa carrera. Aún con esto el sistema de predicción quedaría muy simple, ya que, siempre me daría los mismos resultados.

Como ya he mencionado anteriormente, el ciclismo es un deporte muy difícil de predecir ya que hay muchas circunstancias que se podrían tener en cuenta, o no, como podría ser el caso de una caída, lesión enfermedad, o un simple pinchazo que te desenganche de la carrera. Si bien estas circunstancias no son al 100% aleatorias, si que tienen en su mayor parte un factor de suerte, ya que muchas veces no es algo que dependa de ti, si no que te puede tocar y ya está. Por ello he decidido añadir al coeficiente final de cada corredor, un grado de aleatoriedad a sumar que le pueda llevar a variar sus resultados en diferentes ejecuciones del programa.

Aquí muestro como he calculado el coeficiente, explico (quede claro que a menor coeficiente, mejor resultado):

- Si el corredor posee 5 resultados en la carrera en cuestión, se hará una media ponderada de todos estos, dando mayor importancia a los más recientes y menor a los más lejanos.
- En caso de solo haber participado una vez en la carrera, al resultado obtenido en esa ocasión se multiplicará por 1.4 ya que un año no es suficiente por si solo para pronosticar un nuevo resultado

```
coeficiente = 0
if(contador == 5):
    coeficiente = 0.3 * float(res[0]) + 0.25 * float(res[1]) + 0.2 * float(res[2]) + 0.15 * float(res[3]) + 0.1 * float(res[4])
elif(contador == 4):
    coeficiente = 0.3 * float(res[0]) + 0.27 * float(res[1]) + 0.23 * float(res[2]) + 0.2 * float(res[3])
elif(contador == 3):
    coeficiente = (float(res[0]) + float(res[1]) + float(res[2]))/3
elif(contador == 2):
    coeficiente = (float(res[0]) + float(res[1]))/2
elif(contador == 1):
    coeficiente = float(res[0])
    if(float(res[0]) < 4):
        coeficiente = float(res[0])*1.4
```

Ahora continuo con el factor aleatorio:

Pues eso, un número random que en función del que sea obtenido sumará una cantidad mayor o menor al coeficiente calculado anteriormente en base a los resultados.

Además en el caso de las carreras de un solo día, los monumentos, es mucho más probable que ocurra una sorpresa que en las vueltas de 21 días ya que estas al final, acaban poniendo a cada uno en su sitio, por ello, he añadido que haya un 10% de probabilidad de que un corredor divida su coeficiente a la mitad en este tipo de carreras.

```
#Un factor aleatorio
if(coeficiente > 0):
    factor = random.randrange(6)
    if(factor == 1):
        coeficiente = coeficiente + 2
    elif(factor == 2):
        coeficiente = coeficiente + 3.5
    elif(factor == 3):
        coeficiente = coeficiente + 4.5

    if(coeficiente < 35 and random.randrange(10) == 3 and tipo == "Monument"):
        coeficiente = coeficiente/2
```

Con esto ya puedo comenzar a pronosticar resultados, una vez obtenidos los coeficientes, para sacar las opciones de un corredor en cierta carrera, simplemente realizo una estimación del coeficiente, es decir, si éste es menor que una cantidad, en concreto 3.5 para las vueltas y 5 para los monumentos, el sistema determinará al usuario que el corredor en cuestión tendrá altas opciones de quedar en primer lugar. En caso de estar entre 3.5 o 5 y 10 o 15 respectivamente, determinará que tendrá altas opciones de podio y así sucesivamente.

En cuanto a la función que pronostica el top10, esta calcula un coeficiente para cada corredor, como ya he mencionado anteriormente, y los ordena de manera que el que menor coeficiente obtenga, será el primero, y así hasta 10. Muestro alguna captura a continuación.

Resultado de un corredor:

Cycling Results

Resultado pronosticado de corredores

Corredor:

Romain Bardet

Carrera:

Tour de France


Aceptar

Top 10 pronosticado de carrera

Carrera:

Aceptar

Este corredor luchará por entrar en el podio.



Top 10 de una carrera:

Cycling Results

Resultado pronosticado de corredores

Corredor:

Carrera:

Aceptar

Top 10 pronosticado de carrera

Carrera:

Giro d'Italia

Aceptar

1-Chris Froome
2-Tom Dumoulin
3-Vincenzo Nibali
4-Nairo Quintana
5-Primoz Roglic
6-Alejandro Valverde
7-Miguel Ángel López
8-Sтивен Kruijswijk
9-Rafal Majka
10-Thibaut Pinot

Una vez hecho todo esto, la primera versión de la aplicación ya estaría terminada, como aún queda mucho para la fecha de la entrega, intentaré realizar alguno de los cursos de Neo4j.

A continuación, comenzaré con el curso de algoritmos aplicados a grafos de conocimiento. En este curso se aprenderá a implementar algoritmos en una base de datos de Neo4j. Para ello utilizaremos una base de datos dada por su propia página web, la llamada “Yelp public dataset”. Esta es una base de datos que ayuda a la gente a encontrar negocios locales basados en valoraciones, preferencias y recomendaciones. Sobre 163 millones de valoraciones han sido escritas en la plataforma.

Disponemos de un código de JavaScript que implementa las funcionalidades de búsqueda de un determinado negocio, permitiendo elegir si queremos todas las valoraciones de una misma persona, o incluso buscar por palabras, por ejemplo, negocios que ofrezcan hamburguesas. Este programa será el que editaremos aplicando algunos de los algoritmos de búsqueda ya existentes.

En primer lugar, aplicaremos una jerarquía de categoría. Estas jerarquías son utilizadas para clasificar productos o transacciones para reportar y analizar.

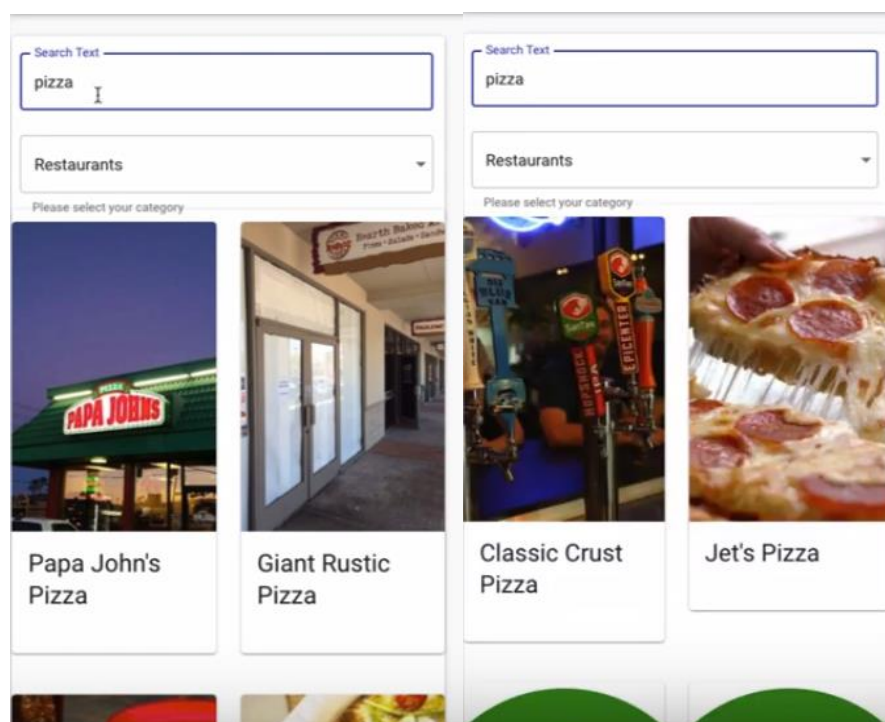
En la base de datos yelp podemos buscar negocios por nombre y categoría, pero actualmente hay 100 categorías entre las que elegir. Una forma de facilitar las cosas para los usuarios sería permitir navegar por una taxonomía de categorías, empezando por el nivel superior. Para realizarlo, usaremos el algoritmo de similitud de superposición (Overlap Similarity algorithm) para identificar una jerarquía de categoría en los datos. Necesitaremos aplicar el algoritmo y realizar algunas actualizaciones en nuestra base de datos con Neo4j. De esta manera, al realizar una búsqueda por categorías, a la hora de seleccionar la que deseamos, solo se mostrarán aquellas categorías pertenecientes a los niveles superiores.

Búsqueda de categorías antes de la modificación y después:

& Probates	Active Life
ATV Rentals/Tours	Arts & Entertainment
Acai Bowls	Automotive
Accessories	Beauty & Spas
Accountants	Education
Acne Treatment	Event Planning & Services
Active Life	Financial Services
Acupuncture	Food
Addiction Medicine	Health & Medical
Adoption Services	Home Services
Adult	Hotels & Travel

A continuación, mejoraremos la visualización de los resultados de búsqueda, en este caso, modificaremos el sistema para que resultados se muestren de más relevantes a menos. Para ello utilizaremos la métrica de similitud de Pearson. Esta métrica es la medida de la correlación lineal entre dos variables X e Y, tiene un valor entre 1 y -1.

Resultados de la búsqueda antes y después de aplicar el algoritmo:

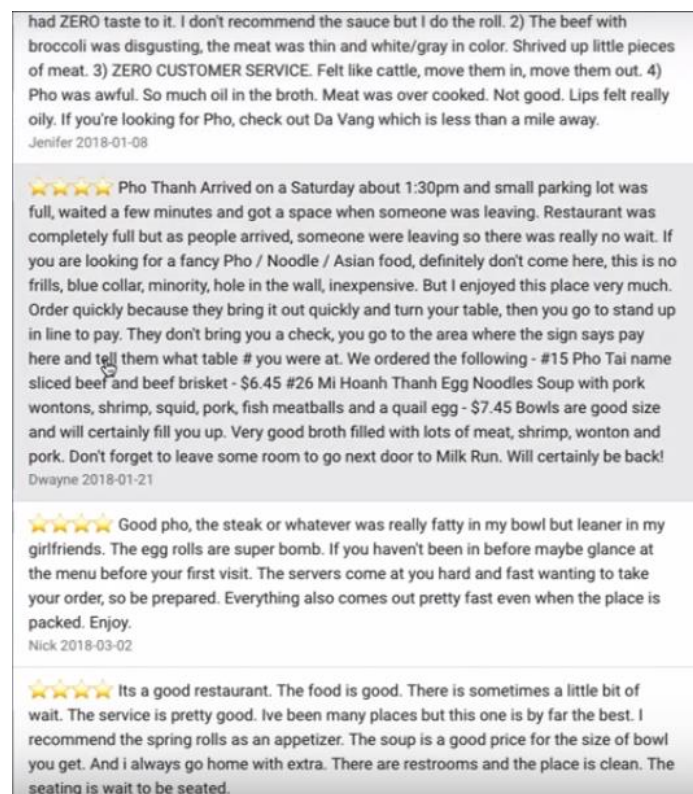


En la siguiente parte utilizaremos el algoritmo personalizado de “PageRank” para identificar a usuarios de confianza y mostrar sus reseñas cuando buscamos un negocio en concreto.

PageRank es una marca registrada por Google y ampara los algoritmos utilizados para asignar de forma numérica la relevancia de los documentos o páginas web indexados por un motor de búsqueda.

Una vez implementado el algoritmo ya podemos realizar una búsqueda y, en los resultados, podremos observar en primer lugar las valoraciones de los usuarios más confiables que están registrados en la base de datos.

Resultados después de la modificación:



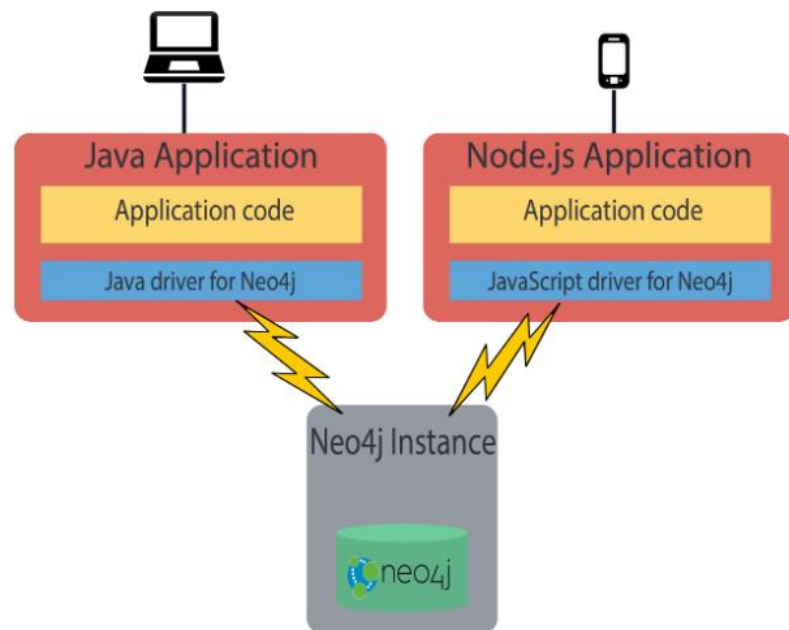
Por último y para finalizar este curso, modificaremos la aplicación para recomendar fotos a los usuarios que estén realizando búsquedas, para ello utilizaremos el algoritmo de propagación de etiquetas (Label Propagation). Label Propagation es un algoritmo de aprendizaje automático semi supervisado que asigna etiquetas a puntos de datos previamente no etiquetados. Una vez implementada esta nueva funcionalidad, podremos seleccionar 5 fotos de las que hay introducidas en la base de datos y, entonces, la aplicación nos mostrará negocios basados en las fotos seleccionadas.

Con esto ya he finalizado el curso y empezaré otro. Este curso que empezaré es de administración de Neo4j, aquí se aprenderá algunas de las tareas de administración más importantes para la creación de aplicaciones de Neo4j, para ello utilizaremos la versión gratuita de Neo4j, Neo4j Enterprise Edition 3.5. El curso está diseñado para administradores.

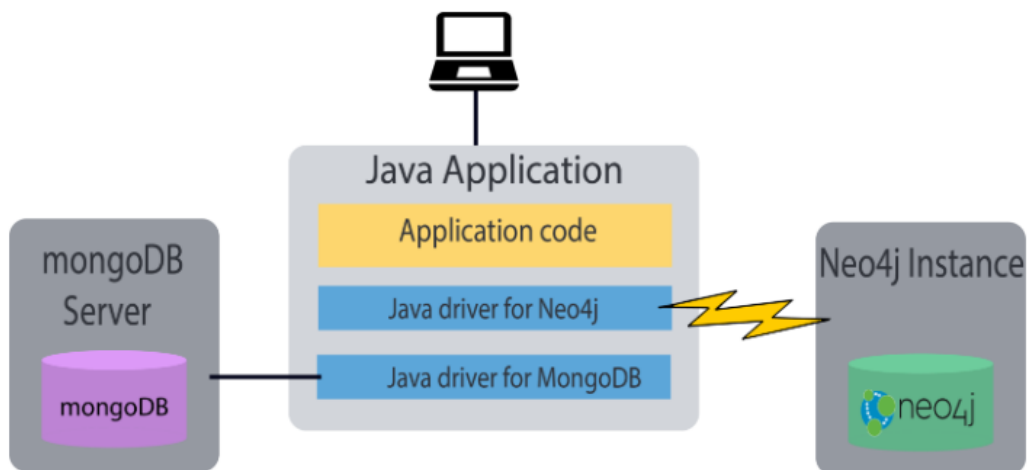
Un administrador de una base de datos de Neo4j es responsable de asegurar que la aplicación implementada se ejecute para satisfacer las necesidades de los usuarios que interaccionen con ella. Esto incluye que el software de Neo4j está actualizado a su más reciente versión y configurado correctamente.

En primer lugar, nos enseñan unas cuestiones teóricas básicas sobre la administración de este tipo de aplicaciones. Aquí muestro cinco ejemplos de la posible arquitectura de una aplicación realizada con Neo4j.

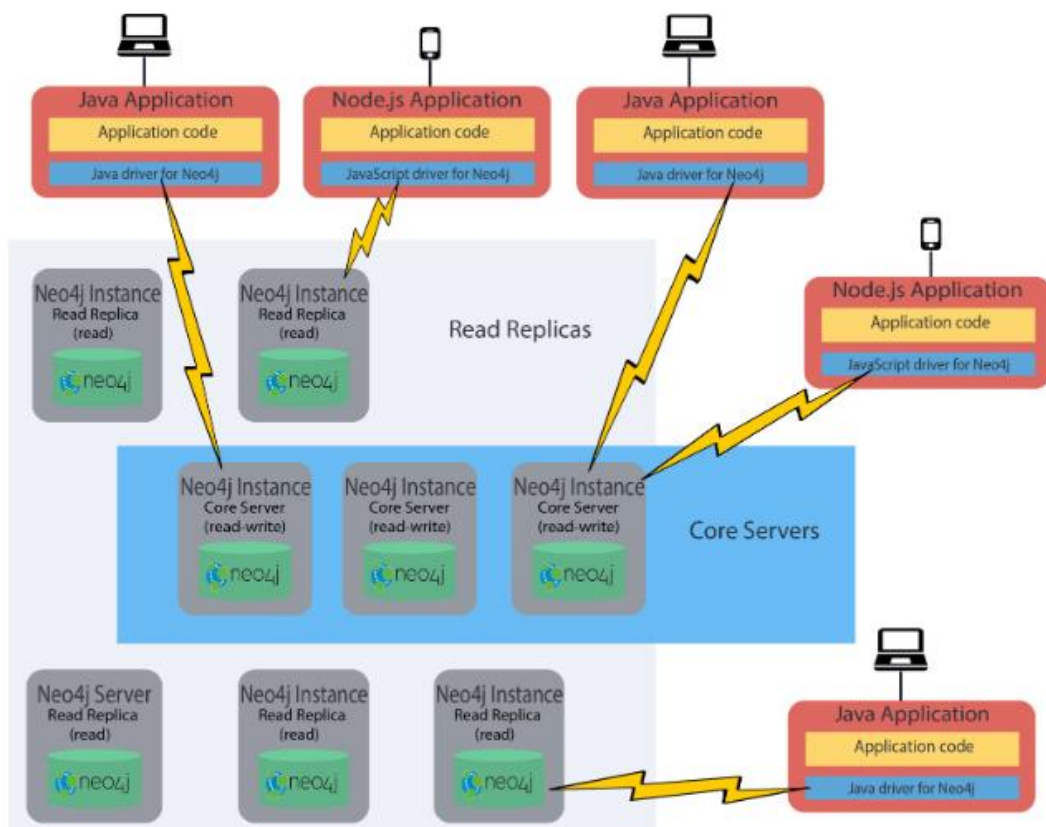
Uso de Neo4j como la base de datos primaria:



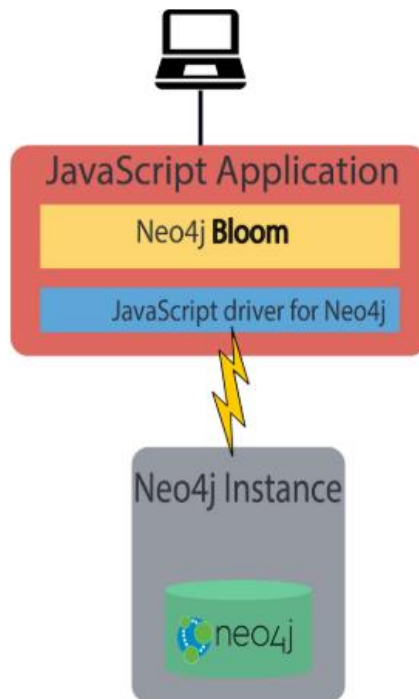
Integración de la base de datos Neo4j con otras bases de datos:



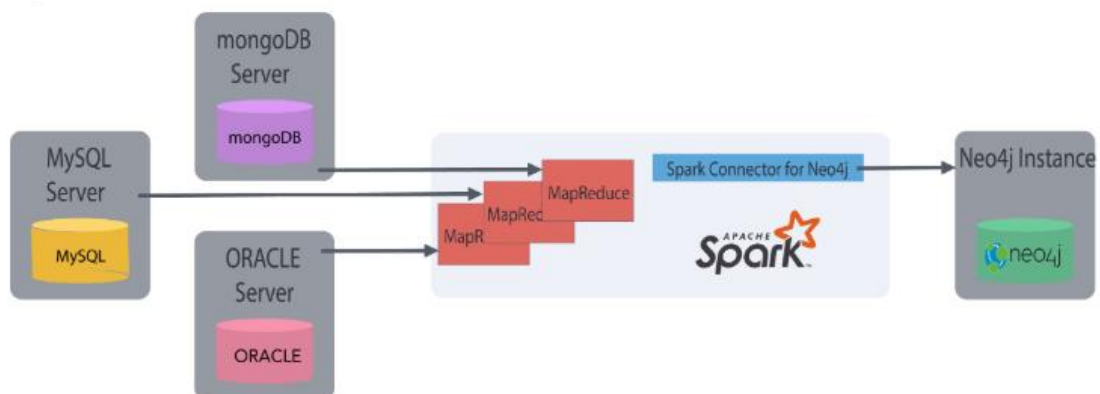
Utilizando el “casual clustering” de Neo4j:



Usando Neo4j únicamente para el análisis de grafos:



Utilizando Neo4j integrado con soluciones informáticas pesadas:



Lo primero que se hará en el curso de una manera práctica es la gestión de instancias de Neo4j. Esto se hará prácticamente todo desde la terminal, es decir, por línea de comandos, donde podremos arrancar, parar, reiniciar o chequear el estatus de una instancia de Neo4j creada anteriormente.

Podremos utilizar la terminal como si estuviéramos utilizando la aplicación de escritorio de Neo4j, esto es llamado Cypher Shell y desde la misma podremos utilizar comandos en Cypher a nuestro gusto de

manera que podremos gestionar una base de datos sin ningún problema. A mayores también se aprende a utilizar crear “backups” para luego poder acceder a estas bases de datos sin necesitar conexión a internet.

También aprenderemos a modificar el archivo de configuración del Neo4j Enterprise Edition, lo que nos permitirá seleccionar una base de datos por defecto, crear instancias en nuevas localizaciones y muchísimas acciones más.

Todo esto que he realizado esta comprendido en el apartado de gestión de bases de datos. A continuación, voy al siguiente apartado, que trata sobre “casual clustering”.

“Clustering” proporciona una solución de alta disponibilidad mediante la cual, si una instancia de Neo4j falla, otra instancia puede hacerse cargo automáticamente. Además, también suministra una database altamente escalable por lo cual, algunas partes de la aplicación pueden actualizar los datos.

Arquitectura de clúster:



En este curso he entendido el funcionamiento de los clusters, así como, aprender a crear uno, modificarlo, copiarlo, en definitiva, a gestionarlo. También a crear servidores de réplica de lectura.

Ahora me adentraré el modulo de seguridad, en esta parte aprenderemos a configurar una instancia de Neo4j utilizando LDAP como proveedor de autenticación y configurar y utilizar auditorías de seguridad.

También se aprenderán unos conceptos teóricos básicos como la seguridad de los datos, el control de acceso, la privacidad de usuario y lo que más nos interesa, la seguridad en las aplicaciones de Neo4j.

Lo principal será configurar una instancia de Neo4j para establecer una autenticación, para ello, lo primero es activar la autenticación editando el archivo de configuración de Neo4j, introduciendo la siguiente línea: `dbms.security.auth_enabled=true`

A continuación, se configurará LDAP en 3 pasos:

1. Añadir información de rol a LDAP
2. Configurar Neo4j para la autenticación LDAP
3. Testearlo

A partir de aquí ya podremos utilizar el sistema de autenticación, pudiendo acceder a los datos públicos de cada usuario, siendo el administrador.

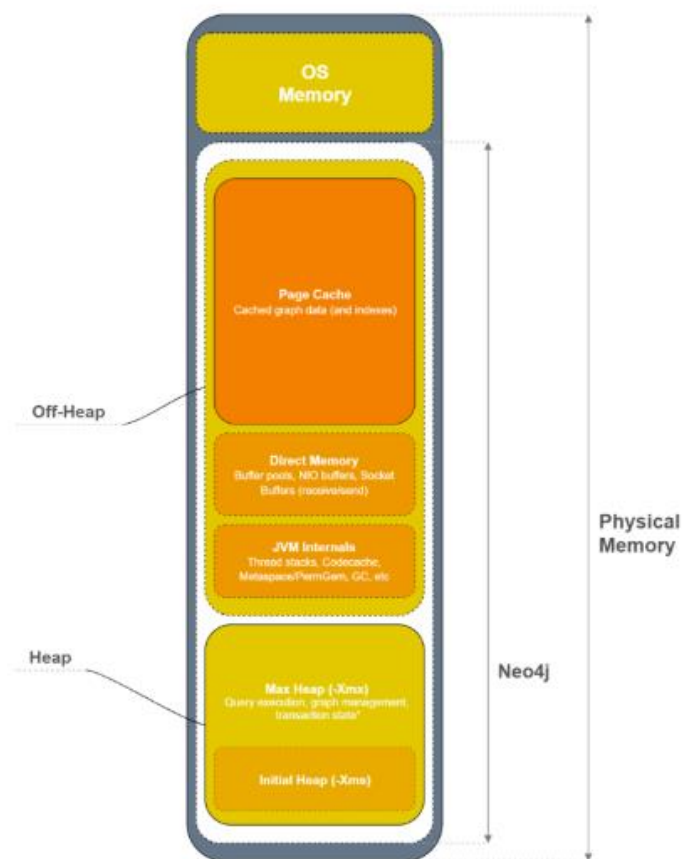
La última parte de este curso es la monitorización de Neo4j. Como administrador, se debería de configurar la aplicación desplegada para poder realizar una monitorización de la actividad, además de estar preparado para monitorear más profundamente y posiblemente reconfigurar Neo4j.

De esta manera podremos observar todas las consultas que han sido realizadas en la base de datos, así como terminar algunas consultas que aún están en proceso y llevan un largo periodo de tiempo. Esto también puede ser realizado automáticamente por la aplicación de Neo4j si se configurase de esta forma.

También se pueden monitorear y terminar las diferentes conexiones que se realizan de una forma externa a nuestra base de datos.

Otra función que podemos realizar es la monitorización de las solicitudes HTTP que son recibidas mediante el navegador.

Una función bastante importante y que puede ser muy útil es el control del uso de la memoria abarcando la memoria del sistema operativo, el caché, y la memoria propia de Neo4j, aquí enseño el esquema de las memorias utilizadas:



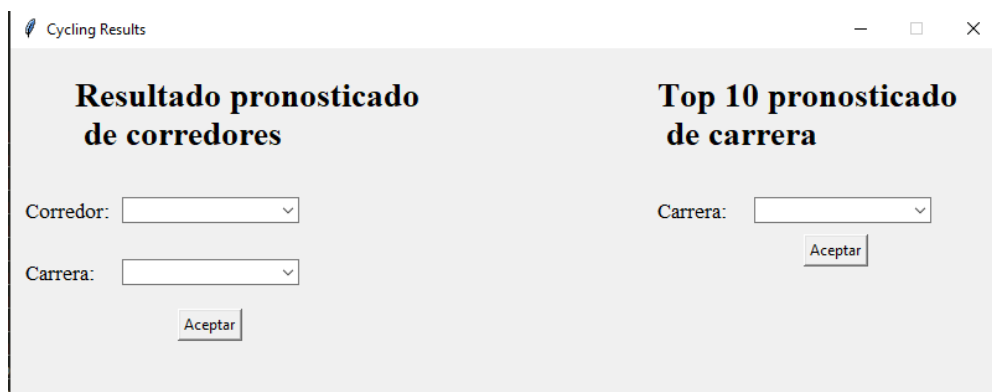
Esto también nos permite arreglar problemas que surjan en las diferentes memorias que he mencionado anteriormente, esto se realiza gracias a un log donde se almacenan todos los problemas que surgen a la hora de realizar alguna gestión en la base de datos.

Con esto finalizo este curso de Neo4j y ahora me dedicaré en todo el tiempo que me queda a la mejora de la practica final.

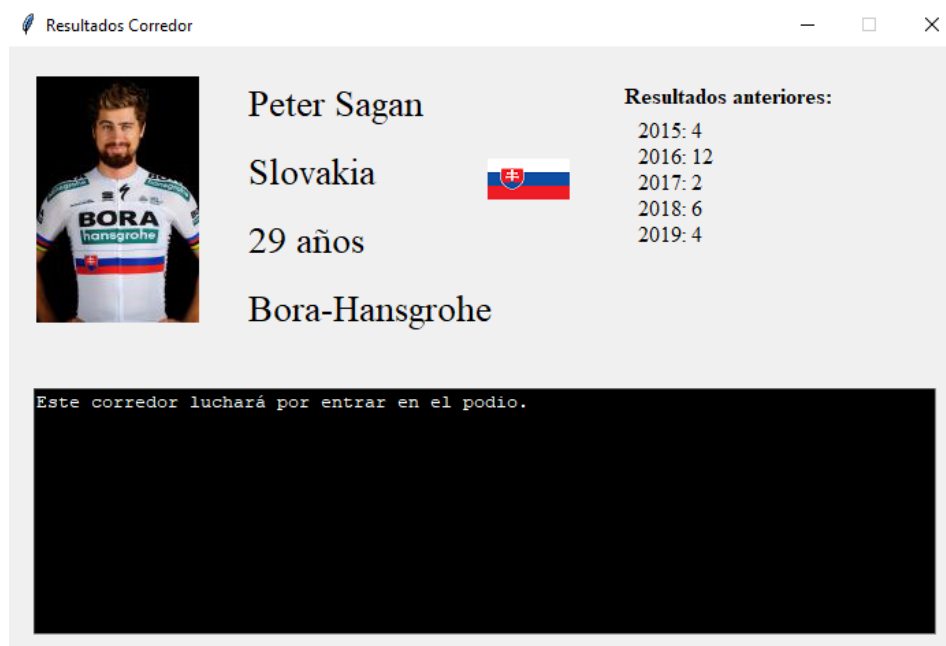
Vuelta a la aplicación de predicción de resultados de ciclistas, voy a intentar mejorar la interfaz de manera que sea más bonita y completa. La idea que tengo en mente es que en vez de mostrar los resultados en la misma ventana que se seleccionan los corredores y carreras, que salga una nueva ventana con datos sobre el corredor o carrera en cuestión junto con su resultado predicho, además de introducir algunas fotos para que el sistema quede más agradable a la vista. A mayores, la aplicación también pronosticará, en el caso de las vueltas por etapas, al ganador de la clasificación de la montaña y al ganador de la de la regularidad.

La nueva interfaz queda de la siguiente forma:

Ventana principal:




Pantalla de resultados de corredor:



Resultados de top 10 de carrera ejecutado dos veces para que se pueda ver la diferencia en la que influyen los factores aleatorios:

Resultados Carrera



Top 10 general:

- 1-Chris Froome
- 2-Primoz Roglic
- 3-Nairo Quintana
- 4-Steven Kruijswijk
- 5-Bauke Mollema
- 6-Tom Dumoulin
- 7-Vincenzo Nibali
- 8-Thibaut Pinot
- 9-Miguel Ángel López
- 10-Alejandro Valverde


Ganador regularidad:

Elia Viviani

Ganador montaña:

Thomas de Gendt

Resultados Carrera



Top 10 general:

- 1-Tom Dumoulin
- 2-Chris Froome
- 3-Thibaut Pinot
- 4-Primoz Roglic
- 5-Miguel Ángel López
- 6-Alejandro Valverde
- 7-Vincenzo Nibali
- 8-Nairo Quintana
- 9-Mikel Landa
- 10-Steven Kruijswijk

Ganador regularidad:

Fernando Gaviria

Ganador montaña:

Romain Bardet

Y con esto prácticamente finalizo mi cuaderno de trabajo y me dispondré a hacer la memoria y la presentación final.