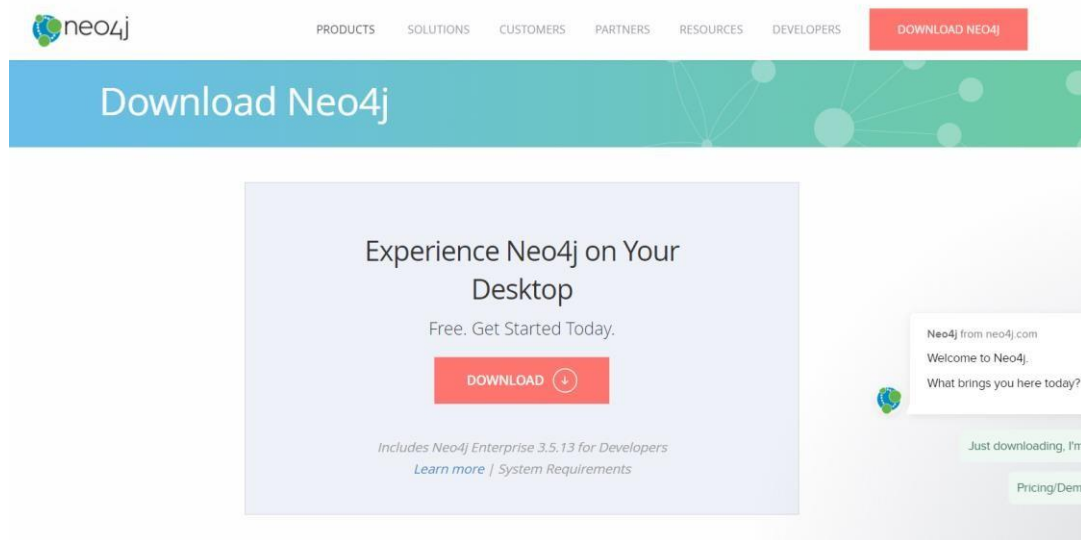


## Cuaderno de Trabajo

En este anexo presentaremos el cuaderno de trabajo de la asignatura, una especie de cuaderno de bitácora en el cual he ido reflejando de forma cronológica todo el proceso de aprendizaje primero y posterior estructuración del trabajo que he llevado a cabo.

En primer lugar, fue necesario entrar en contacto con el entorno de Neo4J, la aplicación que iba a usar, así como ponerme al día de las bases de datos con grafos de conocimiento. Para ello, comencé por instalar el entorno Neo4J



Desde la siguiente pantalla accedí a la solicitud de registro de Neo, donde tuve que rellenar la siguiente información:

## Get Started Now

Please fill out this form to begin your download

\*

\*

\*

\*

\*

\*

By downloading you agree to the [Neo4j License Agreement for Neo4j Desktop Software.](#)

Download Desktop

The information you provide will be used in accordance with the terms of our [privacy policy.](#)

Tras esto descargué un ejecutable con el cual pude iniciar la instalación de Neo. El proceso fue sencillo y bastante intuitivo, con el cual pude rápidamente empezar a usar Neo.

Como el sistema de grafos de conocimiento mediante nodos, así como el entorno visual y las propiedades de Neo4J eran totalmente ajenas a mi conocimiento, comencé a buscar información antes de meterme de lleno con ello. El primer lugar donde investigué fue en YouTube. Allí encontré muchos videos y entre otros canales, el oficial de Neo4J, con incesante información acerca de su propia plataforma. Sin embargo, ahora me encontraba en la disyuntiva inversa a la anterior. Si bien antes no tenía excesiva información y todo era nuevo para mí, tras realizar esta búsqueda, encontré demasiada información. Horas y horas de videos dentro del canal oficial, así que me dispuse a filtrar toda esa información para encontrar sólo la realmente útil en el punto en el cual me encontraba.

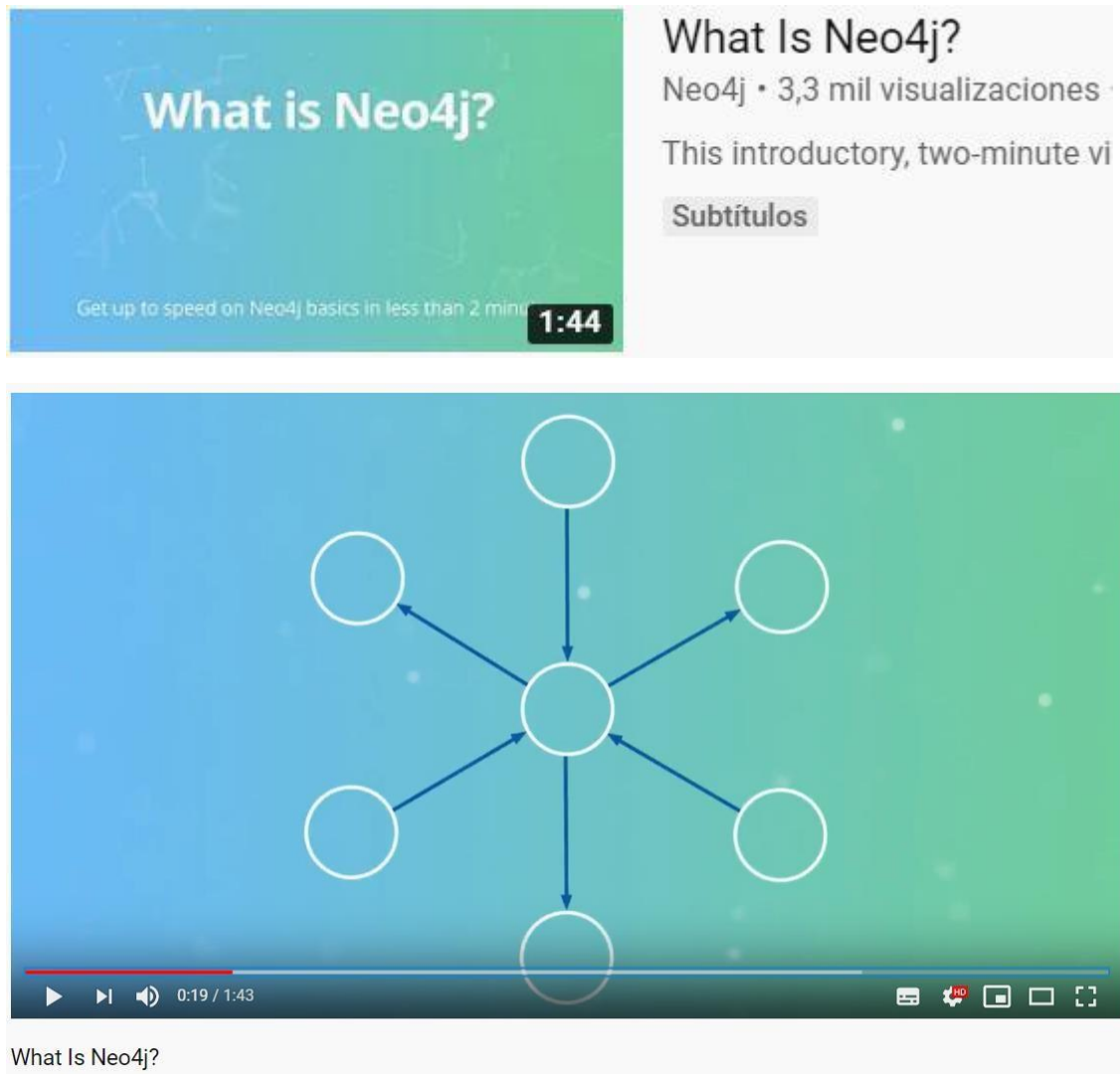
Primero necesitaba saber qué era eso de Neo4J, y después comenzar a usarlo. Para ello, uno de los videos más aclaratorios que encontré fue el siguiente:



Bases de datos de grafos - Neo4j

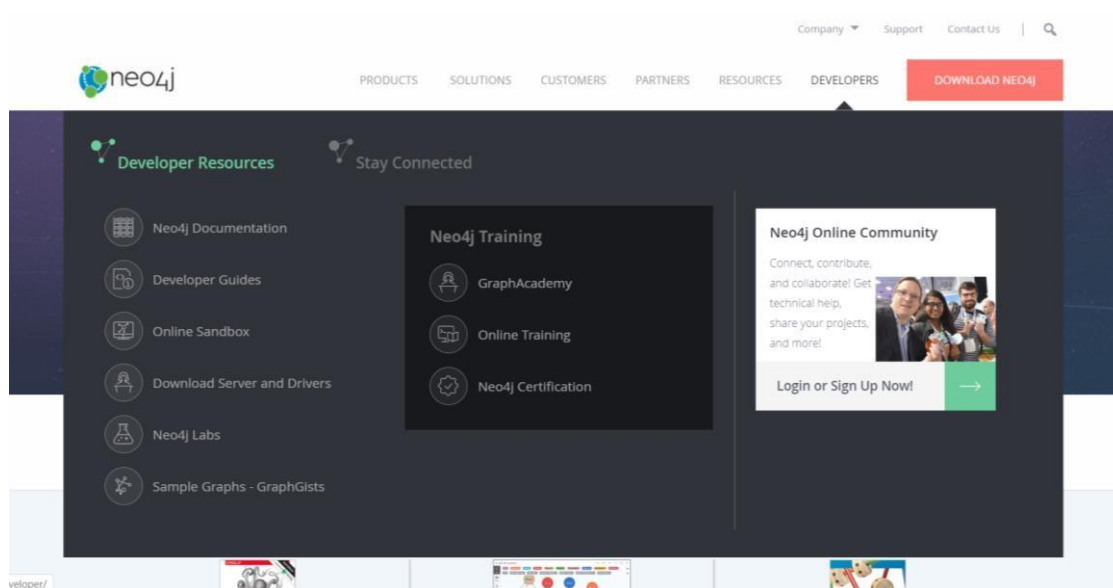
En él, se hacía una pequeña introducción rápida a las bases de datos de grafos de conocimiento con nodos, y como bien indica, de Neo4J. Esto me sirvió de ejemplo inicial para entender de qué se trataba Neo4J y cómo iba a poder trabajar con él. Así mismo, pude construir una idea inicial en mi cabeza de cuáles eran las posibilidades de que se ofrecían ante mí de cara al

trabajo final. Además, para terminar de tener claros todos los aspectos, visualicé el video introductorio del propio canal de Neo4J. Un video muy cortito de 2 minutos en el cual se establecen los principios más básicos explicado por los propios creadores de Neo.



Una vez que el concepto de bases de datos de grafos de conocimiento basado en nodos estaba conciso en mi cabeza, comenzó el proceso de hacer que el programa fuese familiar para mí, comprender cómo funcionaba y cómo debía utilizarlo.

Para ello, dentro de la propia página de Neo4J, se encuentran una serie de tutoriales, posteriormente certificados con los cuales podía poner en marcha esa parte.



En ese menú desplegable, opté por la opción GraphAcademy, donde se encuentran los tutoriales previamente explicados.

## Free Online Training & Certification

Learn All About Neo4j and how to Launch Neo4j Into Production when you take these Free online courses. After completing the Introduction to Neo4j course, take the Free 1-hour Neo4j Certification exam.

### Introduction to Neo4j

Get started quickly – learn about Graph Databases, Neo4j, and Cypher – the Graph Query Language.

START TRAINING

### Neo4j Administration

Learn deployment best practices and how to successfully launch Neo4j in a production environment.

START TRAINING

### Data Science with Neo4j

Learn how to use Neo4j as part of your Data Science and Machine Learning workflows using an aminer.org dataset.

START TRAINING

### Applied Graph Algorithms

Learn how to use graph algorithms in Neo4j, and also how to apply them to enhance web application functionality.

START TRAINING

Una vez llegados a este punto, decidí comenzar lógicamente por la introducción. Con ella aprendí rápidamente los conceptos más importantes de Neo4J. Aunque con los videos introductorios ya preparé el camino para comprender mejor estos cursillos rápidos, llevar a cabo la parte introductoria de Neo, fue vital para saber manejar todo lo necesario.

En los dos primeros apartados del tutorial se introducen las bases de datos de grafos y Neo4J, los dos aspectos en los que me había documentado antes en otras plataformas.

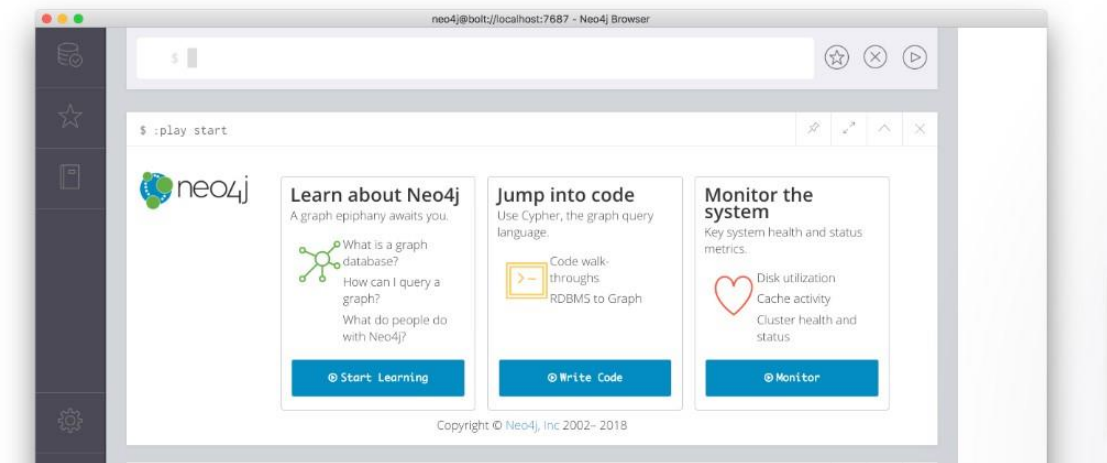
Posteriormente, se me explicaba el entorno gráfico de Neo4J Desktop, la aplicación de escritorio de Neo.

## Using Neo4j Browser

Neo4j Browser is a tool that enables you to access a Neo4j Database by executing Cypher statements to create a graph to return data. The data returned is typically visualized as nodes and relationships in a graph, but can also be used for other purposes. In addition to executing Cypher statements, you can execute a number of system calls that are related to the database being accessed, such as retrieving the list of queries that are currently running in the server.

There are two ways that you can use Neo4j Browser functionality:

- Open the Neo4j Browser application from Neo4j Desktop (database is local)
- Use the Neo4j Browser Web interface by specifying a URL in a Web browser using port 7474 (database is remote)



Una vez entendidos todos los aspectos que son necesarios para comprender cómo trabaja la interfaz gráfica de Neo, llegamos a la parte más importante para mí, Cypher. Este fue el apartado más relevante en cuanto al aprendizaje de cómo usar Neo. Se divide en tres partes:

### 1. Introducción a Cypher: En ella podremos aprender a:

- Encontrar nodos del grafo
- Filtrar nodos encontrados usando etiquetas y propiedades de los nodos
- Encontrar atributos de los nodos dentro del grafo
- Filtrar nodos encontrados utilizando relaciones

## Introduction to Cypher

### About this module

Cypher is the query language you use to retrieve data from the Neo4j Database, as well as create and update the data.

At the end of this module, you should be able to write Cypher statements to:

- Retrieve nodes from the graph.
- Filter nodes retrieved using labels and property values of nodes.
- Retrieve property values from nodes in the graph.
- Filter nodes retrieved using relationships.

Throughout this training, you should refer to:

- [Neo4j Cypher Manual](#)
- [Cypher Reference card](#)



## 2. Consultas:

- Filtrar consultas utilizando WHERE
- Conocer el procesamiento de consultas
- Conocer y comprobar los resultados obtenidos
- Trabajar con datos, fechas y listas de Cypher

# Getting More Out of Queries

## About this module

You have learned how to query nodes and relationships in a graph using simple patterns. You learned how to use node labels, relationship types, and properties to filter your queries. Cypher provides a rich set of `MATCH` clauses and keywords you can use to get more out of your queries.

At the end of this module, you should be able to write Cypher statements to:

- Filter queries using the `WHERE` clause
- Control query processing
- Control what results are returned
- Work with Cypher lists and dates

## 3. Creación de datos:

- **Crear un nodo:** ◦ Añadir y eliminar etiquetas de los nodos ◦ Añadir y eliminar propiedades de los nodos ◦ Actualizar propiedades de los nodos
- **Crear una relación:**
  - Añadir y eliminar propiedades de una relación
- **Eliminar un nodo**
- **Eliminar una relación**
- **Unir datos de un grafo** ◦ Crear nodos ◦ Crear relaciones

## About this module

You have learned how to query a graph using a number of Cypher clauses and keywords that start with the `MATCH` clause. You learned how to retrieve data based upon label values, property key values, and relationship types, and how to perform some useful intermediate processing during a query to control what data is returned.

At the end of this module, you should be able to write Cypher statements to:

- Create a node
  - Add and remove node labels
  - Add and remove node properties
  - Update properties
- Create a relationship
  - Add and remove properties for a relationship
- Delete a node
- Delete a relationship
- Merge data in a graph
  - Creating nodes
  - Creating relationships

Finalmente, tenemos una última pieza, que nos enseña:

- Usar parámetros en las declaraciones de Cypher
- Analizar cómo se ejecuta Cypher

- Consultas al monitor
- Manejar y gestionar restricciones y claves de los nodos para el gráfico
- Importar datos de un grafo en archivos CSV
- Manejar y gestionar índices de un grafo
- Acceder a los recursos adicionales de Neo4J

## Getting More Out of Neo4j

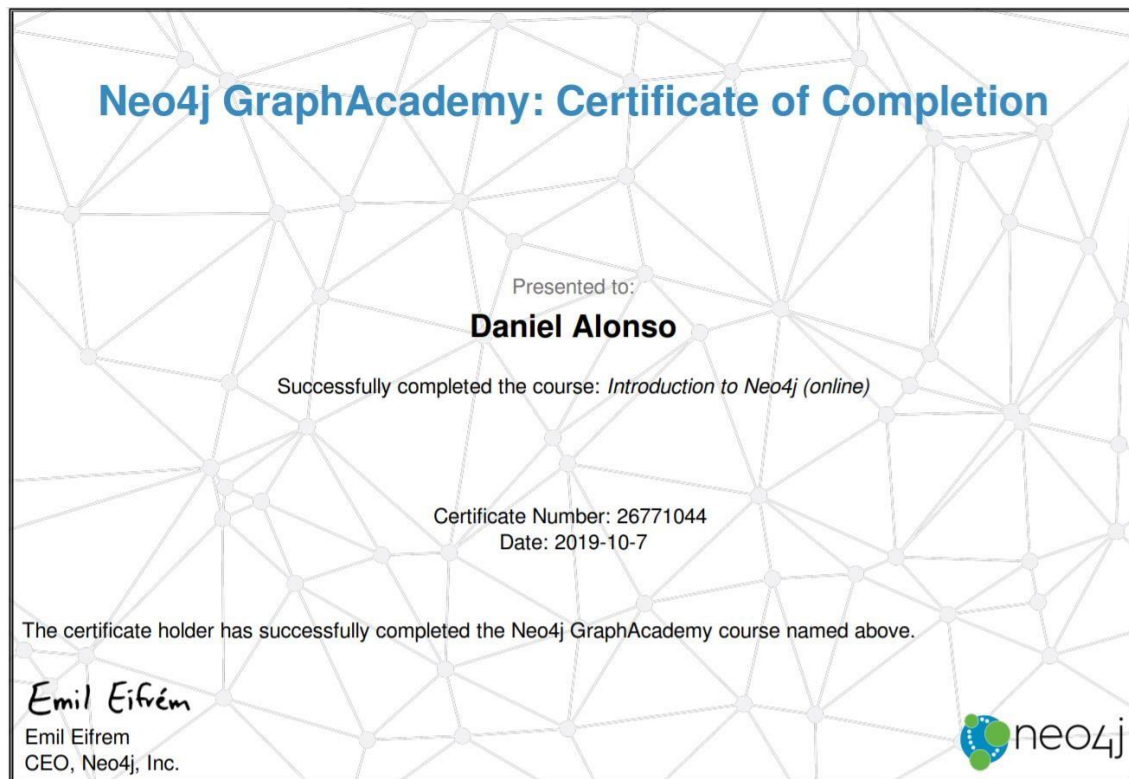
### About this module

You have learned how to set up your development environment for accessing a Neo4j graph and how to write basic Cypher statements for querying the graph modifying the graph.

At the end of this module, you should be able to:

- Use parameters in your Cypher statements.
- Analyze Cypher execution.
- Monitor queries.
- Manage constraints and node keys for the graph.
- Import data into a graph from CSV files.
- Manage indexes for the graph.
- Access Neo4j resources.

Al final de cada una de las partes, hay una serie de preguntas, las cuales tenemos que responder correctamente para continuar adelante y para obtener el certificado final que acredita que hemos completado la introducción de Cypher. Después de solicitarlo, podemos descargar el diploma que certifica que hemos consumado esta parte de la GraphAcademy.



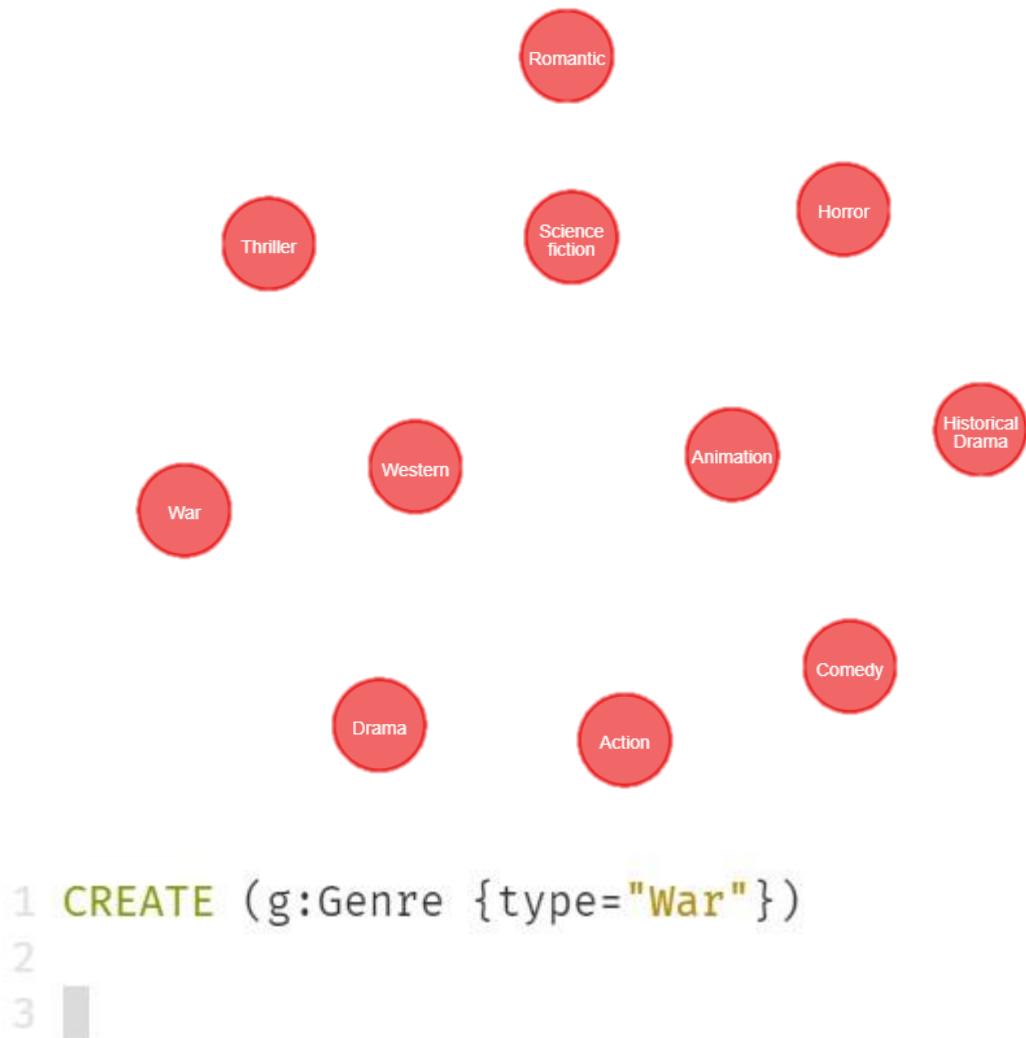
Ahora que ya conocemos los aspectos necesarios para llevar a cabo una base de datos de grafos de conocimiento, así que llegados a este punto era necesario elegir un tema en el cual centrar mi trabajo final. Después de valorar numerosas opciones, decidí decantarme por realizar una base de datos de grafos orientada a películas y libros, sobretodo películas basadas

en libros. Mi finalidad era construir un sistema de recomendación de películas y libros a partir de datos que introdujese el usuario.

Para comenzar, empecé organizando los datos que me interesaba que estuviesen en la base de datos. Llegué a la conclusión de las propiedades que consideraba ineludibles eran:

- **Películas:** Género, Título, Director, Actores, Guionista, Valoración y País □  
**Libros:** Título, Escritor y País.

Una vez establecidos estos puntos, me di cuenta que era muy lógico agrupar las películas por géneros, que me parecía que era la mejor manera de juntarlas, por tanto, creé nodos por cada género que iba a recoger.



De esta manera fui creando todos los nodos que se ven en la primera imagen. Así ya tengo la primera forma de agrupar las películas que posteriormente introduje en la base de datos. Decidí que cada género fuese un nodo y no una propiedad de las películas, ya que era un criterio muy general, y varias películas podían tener géneros distintos.



Luego de esto, fui creando nodos de películas y añadiéndoles sus respectivas propiedades. Esta fue la parte más larga y tediosa del trabajo ya que se dividía en varias partes. La primera de ellas supuso buscar información de películas que estuviesen basadas en un libro. Para ello realicé numerosas búsquedas en la web, y rápidamente encontré una serie de páginas web que recogían esta información. Debido a que había empezado por establecer los distintos géneros que había, fui seleccionando las películas en función de esto último, para que hubiese una distribución más o menos equitativa. Una vez elegidas las 100, 110 películas que iban a conformar la base de datos, pasé a buscar la información representativa de cada una de ellas. Fui metiendo todo en un Excel, en el cual iba almacenando todos los aspectos que antes he citado.

Película	Director	Actor Principal	Valoración	Guionista	País
El Padrino	Francis Ford Coppola	Marlon Brando, Al Pacino	10, 9'8, 9	Mario Puzo, Francis Ford Coppola	EE.UU.
El silencio de los corderos	Jonathan Demme	Anthony Hopkins, Jodie Foster	8'6, 9'6, 8'2	Ted Tally	EE.UU.
La Colmena	Mario Camus	Victoria Abril, Ana Belén	7'2, 7'1, 7	José Luis Dibildor	España
La Carretera	John Hillcoat	Viggo Mortensen, Chloë Grace Moretz	7'2, 8'1, 6'6	Joe Penhall	EE.UU.
Nazarín	Luis Buñuel	Francisco Rabal, María Félix	7'9, 7'8, 7'7	Luis Buñuel, Julio Aronson	México
El Gatopardo	Luchino Visconti	Burt Lancaster, Alain Delon	8'0, 8'7, 7'8	Suso Cecchi d'Amico	Italia
Carrie	Brian de Palma	Sissy Spacek, Piper Laurie	7'4, 7'9, 7	Lawrence D. Cohen	EE.UU.
La lengua de las mariposas	José Luis Cuerda	Fernando Fernán Gómez, José Luis Cuerda	7'6, 7'6, 7'5	José Luis Cuerda	España
Alguien voló sobre el nido del cuco	Milos Forman	Jack Nicholson, Louise Latham	8'7, 8'3, 8'4	Bo Goldman, Lavie Dizon	EE.UU.
To Kill a Mockingbird	Robert Mulligan	Gregory Peck, Mary Badham	8'3, 7, 8'3	Horton Foote	EE.UU.
Parque Jurásico	Steven Spielberg	Sam Neill, Samuel L. Jackson	8'1, 8, 7	Michael Crichton	EE.UU.
American Psycho	Mary Harron	Christian Bale, Willem Dafoe	7'6, 7'4, 6'6	Mary Harron, Guinevere Turner	EE.UU.
El Halcón Maltés	John Huston	Humphrey Bogart, Michael Caine	8, 8'8, 8	John Huston	EE.UU.
Cadena Perpetua	Frank Darabont	Tim Robbins, Morgan Freeman	9'3, 8'6, 8'6	Frank Darabont	EE.UU.
Los hombres que no amaban a las mujeres	David Fincher	Daniel Craig, Rooney Mara	7'8, 7'8, 7	Steven Zaillian	EE.UU.
Doctor Zhivago	David Lean	Omar Sharif, Alec Guinness	8, 7'4, 7'9	Robert Bolt	EE.UU.
El diario de Bridget Jones	Sharon Maguire	Renée Zellweger, Hugh Grant	6'7, 6'6, 6'3	Helen Fielding, Amanda Montiel	UK
Valor de ley	Joel Coen, Ethan Coen	Hailee Steinfeld, Matt Damon	7'6, 6'4, 7'1	Joel Coen, Ethan Coen	EE.UU.
La princesa prometida	Rob Reiner	Peter Falk, Robin Wright	8'1, 6'6, 7'4	William Goldmaster	EE.UU.
Sin novedad en el frente	Lewis Milestone	Louis Wolheim, Lewis Mumford	8, 7'4, 7'8	Maxwell Anderson	EE.UU.
Tenemos que hablar de Kevin	Lynne Ramsay	Tilda Swinton, John D'Ercole	7'5, 7'2, 7'1	Lynne Ramsay	UK
Brokeback Mountain	Ang Lee	Anne Hathaway, Jake Gyllenhaal	7'7, 8, 7	Dianna Ossana, Lisa Klein	EE.UU.
Hijos de los hombres	Alfonso Cuarón	Clive Owen, Julianne Moore	7'9, 7, 7'1	Alfonso Cuarón, Peter Jackson	UK
Grandes esperanzas	Alfonso Cuarón	Ethan Hawke, Gwyneth Paltrow	6'9, 8, 6'4	Mitch Glazer	EE.UU.
Blade Runner	Ridley Scott	Harrison Ford, Rutger Hauer	8'1, 9, 8'1	Hampton Fancher	EE.UU.
El Señor de los Anillos	Peter Jackson	Viggo Mortensen, Elijah Wood	8'9, 8'8, 8'2	Philippa Boyens, Peter Jackson	Nueva Zelanda
Drácula de Bram Stoker	Francis Ford Coppola	Gary Oldman, Winona Ryder	7'4, 7, 7'6	James V. Hart	EE.UU.
La lista de Schindler	Steven Spielberg	Liam Neeson, Ralph Fiennes	8'9, 9, 8'7	Steven Zaillian	EE.UU.
El almuerzo desnudo	David Cronenberg	Peter Weller, Judd Hirsch	7'1, 8, 6'6	David Cronenberg	Canada

Después de acabar de recopilar todos los datos necesarios era hora de ir metiéndolos en la base de datos de Neo4J de la siguiente manera:

```

1 MATCH (m:Movie)
2 WHERE m.title="El Padrino"
3 SET m.actores="", m.director="", m.guion="", m.pais="", m.critica=""
4

```

En la línea del “SET” actores representa a los actores principales, director al director, guion al guionista de la película, y el país al país del que pertenece la película. También tenemos la propiedad critica, en la cual aparecen tres notas distintas de IMBD, SensaCine y FilmAffinity, respectivamente. Las tres webs de cine más importantes y con más tráfico de usuarios. Poco a poco, durante varios días, fui completando las etiquetas de cada película. Este proceso duró unos tres o cuatro días.

Una vez finalizada esta tarea, cree los nodos pertenecientes a los libros, y los completé, de la misma manera que las películas, con las propiedades título, escritor y país. Para esto, volví a buscar información de todos los libros en los cuales se basaban las películas que había metido anteriormente y los reuní en un Excel antes de pasarlos a Cypher. Este proceso duró más o menos otros dos días de trabajo.

Libro	Escritor	País
El Padrino	Mario Puzo	Italia
El silencio de los c	Thomas Harris	EE.UU.
La Colmena	Camilo José Cela	España
La Carretera	Cormac McCarthy	EE.UU.
Nazarín	Benito Pérez Gald	España
El gatopardo	Giuseppe Tomasi	Italia
Carrie	Stephen King	EE.UU.
¿Qué me quieres,	Manuel Rivas	España
Alguien voló sobr	Ken Kessey	EE.UU.
Matar a un ruiseñ	Harper Lee	EE.UU.
Parque Jurásico	Michael Crichton	EE.UU.
American Psycho	Bret Easton Ellis	EE.UU.
El halcón maltés	Dashiell Hammet	EE.UU.
Rita Hayworth y l	Stephen King	EE.UU.
Millenium: Los ho	Stieg Larsson	Suecia
Doctor Zhivago	Boris Pasternak	URSS
El diario de Bridge	Helen Fielding	UK
True Grit	Charles Portis	EE.UU.
La princesa prom	William Goldman	EE.UU.
Sin novedad en e	Erich Maria Rema	Alemania
Tenemos que hak	Lionel Shriver	EE.UU.
Brokeback Moun	Annie Proulx	EE.UU.
The Children of M	P.D. James	UK
Grandes esperanz	Charles Dickens	UK
¿Sueñan los andr	Philip K. Dick	EE.UU.
El Señor de los Ar	J. R. R. Tolkien	UK
Drácula	Bram Stoker	UK
El arca de Schindl	Thomas Keneally	Australia
El almuerzo desn	William S. Burrou	Francia

Con todos los nodos ya listos, creados y con sus propiedades, era hora de juntarlos y crear las relaciones entre ellos. Evidente y lógicamente los géneros deben estar relacionados con las películas:

```
$ CREATE (m:Movie)-[b:Type_Of_Film]→(g:Genre )
```

y luego con los libros y las películas:

\$ CREATE (m:Movie)-[:Basado\_En]→(l:Libro)

De esta manera y tras varios días realizando esta tarea, finalmente tenía completado el grafo de conocimiento que formaba mi base de datos y con la cual ya podía ponerme a realizar el algoritmo de recomendación.



El nodo rojo corresponde a los géneros, los verdes a las películas y los morados a los libros. Aquí podemos ver el grafo entero después de todo este trabajo.



Finalizado el trabajo con Cypher, tocaba comenzar a montar el algoritmo mediante el cual se pudiese realizar el recomendador. Por recomendación del profesor, decidí usar Python y como era un lenguaje que no conocía y con el cual no había trabajado nunca, tuve que estudiar las diferentes posibilidades que este lenguaje me podía proporcionar. Como ya tenía claro que quería investigué directamente a tiro fijo. Mi intención era crear un entorno gráfico donde se pudiese usar el algoritmo que fuese a implementar, a modo de aplicación, que el usuario eligiese los criterios de búsqueda y que la aplicación buscase en la base de datos, hiciese una consulta y devolviera una película acorde a los criterios pedidos. Más o menos ya tenía una idea fija de lo que me apetecía hacer y de cómo quería que fuese la aplicación, pero necesitaba conocer qué posibilidades tenía Python. Después de buscar un tiempo, vi que podía realizar un entorno gráfico mediante una librería de Python llamada TKinter.

Lo primero del código debía ser conectar la base de datos de Cypher de Neo4J.

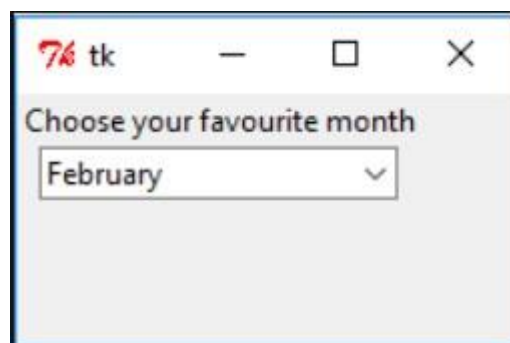
```
#Consultar a la base de datos
from neo4j import GraphDatabase
uri = "bolt://localhost:7687"
driver = GraphDatabase.driver(uri, auth=("neo4j", "dani"))
```

Tras comprobar que el programa estaba bien conectado con la base de datos, empezaba la fase de comenzar el algoritmo.

```
raiz = Tk()

raiz.title("Películas")
raiz.resizable(width=False, height=False)
raiz.geometry("850x800")
raiz.config(bg="grey")
```

De esta manera creaba la ventana principal de la aplicación y los aspectos principales de la misma. Lo siguiente fue crear los ComboBox. Un ComboBox es una caja en la cual se introducen distintas opciones seleccionables que el desarrollador proporciona. Adjunto una imagen de un ComboBox.



Como iban a haber 5 factores a tener en cuenta a la hora de elegir película, he creado 5 ComboBox distintos.



```
##### COMBOBOX #####
raiz.combo.place(x=115, y=25)
raiz.combo["values"]=["Género", "Director", "Actor", "País", "Guionista"]

raiz.combo5.place(x=115, y=65)
raiz.combo5["values"]=["Género", "Director", "Actor", "País", "Guionista"]

raiz.combo7.place(x=115, y=105)
raiz.combo7["values"]=["Género", "Director", "Actor", "País", "Guionista"]

raiz.combo9.place(x=115, y=145)
raiz.combo9["values"]=["Género", "Director", "Actor", "País", "Guionista"]

raiz.combo11.place(x=115, y=185)
raiz.combo11["values"]=["Género", "Director", "Actor", "País", "Guionista"]
```

En cada uno de ellos tendremos la posibilidad de seleccionar uno de los elementos que nos ofrece la lista entre Género, Director, Actor, País y Guionista. Además, por cada ComboBox, tendremos un botón “Aceptar”, es decir, otros 5 botones.

```
##### BOTONES #####
raiz.Button = ttk.Button(raiz, command=aceptar, text = "Aceptar")
raiz.Button.place(x=295, y=25)

raiz.Button2 = ttk.Button(raiz, command=aceptar2, text = "Aceptar")
raiz.Button2.place(x=295, y=65)

raiz.Button3 = ttk.Button(raiz, command=aceptar3, text = "Aceptar")
raiz.Button3.place(x=295, y=105)

raiz.Button4 = ttk.Button(raiz, command=aceptar4, text = "Aceptar")
raiz.Button4.place(x=295, y=145)

raiz.Button5 = ttk.Button(raiz, command=aceptar5, text = "Aceptar")
raiz.Button5.place(x=295, y=185)
```

Como podemos observar, en la propiedad `command` del `Button` de Tkinter, cada botón aceptar nos lleva a una función distinta del código. Esto se debe a que por cada ComboBox llamamos a un método distinto ya que, si no, se producía un error. Sin embargo, es curioso el hecho de que las funciones son casi exactamente iguales así que sólo pondré un ejemplo y lo explicaré. Cogeré para ejemplificar el funcionamiento de las 5 funciones sólo la primera: `aceptar`. Lo primero y más importante es declarar los vectores en los cuales almacenaremos la información elegida por el usuario.

```
32 vecGen = []
33 vecDir = []
34 vecAct = []
35 vecPais = []
36 vecGuion = []
```

En cada uno de ellos almacenaremos los datos referentes al nombre que llevan, y los utilizaremos en todas las funciones del tipo “`aceptar`”.

Lo primero que observamos es que valoramos la opción de que no se haya seleccionado nada en el ComboBox. Más adelante veremos por qué queremos controlar esta situación. Luego, dependiendo de cuál sea el criterio establecido en el ComboBox, vamos valorando opciones. Por ejemplo, si el usuario introduce director, introducimos en el vector de directores una llamada a la función director. Aquí sucede una de los momentos más importantes ya que en esta segunda función es donde realizamos la consulta a la base de datos, de la siguiente manera:

```
50 def director():
51     vector2 = []
52     vector = []
53     with driver.session() as session:
54         i = 0
55         for record in session.run("MATCH (m:Movie)
56                                   RETURN m.director"):
57
58             vector.append(record["m.director"])
59             vector2.append(str(vector[i]).split(", "))
60             i=i+1
61
62     return vector2
```

De este modo se realiza la consulta, pidiendo los datos de todos los directores que tenemos en la base de datos y los almacena en "vector", previamente creado al inicio de la función. Como en algunos casos, hay películas que tiene varios directores, partimos los directores donde encontramos una coma, que es como se había introducido en la base de datos y acumula los directores, esta vez ya sí, separados del todo, en "vector2".

Una vez los directores devueltos al vector general "vecDir", creamos el ComboBox, de la misma manera que se ha hecho el de antes, entonces aparece al apretar el botón de "aceptar" que hemos explicado anteriormente. Dentro del ComboBox, introducimos todo el contenido del vector "directores" que es donde introducimos todos los directores previamente controlando que no estén repetidos. Para ello vamos recorriendo el vector principal y si ya está introducido el mismo dato, se obvia y se pasa al siguiente. Este procedimiento se repite para los géneros, los actores, los guionistas y el país, con una función de consulta para cada una de las posibles elecciones del usuario. Asimismo, esta función aceptar es exactamente igual copiada para cada botón "aceptar". Esto tiene una fácil explicación: cada vez que apretamos al botón "aceptar", quería que apareciese el segundo ComboBox a la derecha del primero, y esta fue la única manera que encontré para realizarlo.



```

12 def aceptar():
13
14     if raiz.combo.get()!="":
15         if raiz.combo.get()=="Género":
16             vecGen = generos()
17             raiz.combo2.place(x=410, y=25)
18             raiz.combo2["values"]=vecGen
19
20         elif raiz.combo.get()=="Director":
21             vecDir=director()
22             raiz.combo2.place(x=410, y=25)
23             directores=[]
24
25             for i in range(len(vecDir)):
26                 for j in range (len(vecDir[i])):
27                     repe = 0
28                     for z in range(len(directores)):
29                         if(directores[z]==vecDir[i][j]):
30                             repe=1
31                     if repe==0:
32                         directores.append(vecDir[i][j])
33
34
35             raiz.combo2["values"]=directores
36
37
38         elif raiz.combo.get()=="Actor":
39             vecAct=actor()
40             raiz.combo2.place(x=410, y=25)
41             actores=[]
42
43             for i in range(len(vecAct)):
44                 for j in range (len(vecAct[i])):
45                     repe = 0
46                     for z in range(len(actores)):
47                         if(actores[z]==vecAct[i][j]):
48                             repe=1
49                     if repe==0:
50                         actores.append(vecAct[i][j])
51

```

Finalmente, por último, para terminar de explicar esta parte del desarrollo del software, cabe señalar, volviendo al primer párrafo de la explicación, que se controla en cada ComboBox de la izquierda, que se ha introducido algún dato, antes de activar el botón “aceptar”. Si esto no ocurre y el usuario, por equivocación no selecciona ningún parámetro de búsqueda, emitirá el sistema un mensaje de error.

```

else:
    messagebox.showerror("Error", "No se ha seleccionado ninguna opción.")

```

El siguiente paso en el desarrollo del sistema fue pasar a seleccionar una película para mostrar, es decir, implementar el recomendador en sí. Me surgieron varias dudas con cómo llevarlo a cabo. Finalmente decidí establecer un sistema de coeficientes de manera que, al

primer parámetro introducido se asigna el coeficiente 1, al segundo 0.8, el tercero 0.6, 0.4, y 0.2, respectivamente. Lo primero de todo fue implementar el botón “buscar”.

```
1289 raiz.Button7 = ttk.Button(raiz, command=buscar, text = "BUSCAR")  
1290 raiz.Button7.place(x=245, y=225)
```

Como podemos observar, al igual que hemos explicado antes, en la propiedad command del Button, llamamos a la función buscar, que procedemos a enseñar ahora.

Una vez llamada la función comienza el desarrollo del método. Como podemos apreciar en la imagen adjunta, debajo de este párrafo, dependiendo del atributo que el usuario selecciona, vamos aplicando a todas las películas con esa característica, el coeficiente marcado antes. Por ejemplo, en caso de elegir el género War (guerra) como característica primaria, lo primero consiste en guardar War en una variable, para posteriormente ser usada para requerirla en la consulta. Una vez que se realiza la consulta a la base de datos, se buscan todas las películas con ese género, y se les asigna el coeficiente 1, a todas ellas. Este procedimiento se repetiría para todas las características introducidas yendo poco a poco sumando coeficientes. Las películas irían almacenando y sumando coeficientes de manera que cada película terminaría obteniendo una suma concreta de puntos. Por un lado, se guardarían las películas en un orden, en un vector, y, por otro lado, se archivarían los coeficientes de cada una de ellas en el mismo orden. De esta manera, puedo saber en todo momento cual es la puntuación que tiene una película solo con conocer en qué posición se encuentra dentro del vector. De este mismo modo, lo realizamos con los géneros, directores, actores, guionistas y el país, y también para todos los ComboBox situados en la izquierda, que son quienes dan la posibilidad de introducir parámetros.

```
##### COMBO2 #####

if raiz.combo5.get()=="Género":
    var=raiz.combo6.get()
    with driver.session() as session:
        for record in session.run("MATCH (m:Movie)-[Type_Of_Film]-(g:Genre{type:'"+str(var)+"'})"
                                   "RETURN m.title"):
            for i in range (len(vector)):
                if vector[i]==record["m.title"]:
                    coeficientes[i]=coeficientes[i]+0.8
                    true=1
                else:
                    true=0
            if true==0:
                vector.append(record["m.title"])
                coeficientes.append(0.8)
            true=0

if raiz.combo5.get()=="Director":
    var=raiz.combo6.get()
    with driver.session() as session:
        for record in session.run("MATCH (m:Movie)"
                                   "WHERE m.director='"+str(var)+"'"
                                   "RETURN m.title"):
            for i in range (len(vector)):
                if vector[i]==record["m.title"]:
                    coeficientes[i]=coeficientes[i]+0.8
                    true=1
            if true==0:
                vector.append(record["m.title"])
                coeficientes.append(0.8)
            true=0

if raiz.combo5.get()=="Actor":
    var=raiz.combo6.get()
    with driver.session() as session:
        for record in session.run("MATCH (m:Movie)"
                                   "WHERE m.actores CONTAINS '"+str(var)+"'"
                                   "RETURN m.title"):
```

Una vez finalizado este proceso, ya tendríamos el recomendador prácticamente finalizado. Tan solo quedaría buscar dentro de nuestros coeficientes, la suma mayor que hayamos obtenido, encontrar la posición en la cual se halla la suma más grande, y buscar esa misma posición en la lista de películas para obtener la película que el sistema recomienda.

```
##### FINAL #####

for i in range(len(vector)):
    if(orden<coeficientes[i]):
        orden=coeficientes[i]
        pelicula[0]=vector[i]

final.append(pelicula[0])
```

Como vemos, nuestra película final la guardamos en el vector "final". Una vez se ha hecho el proceso de recomendación, pasamos a la fase en la cual terminamos de confeccionar la aplicación, escribimos los mensajes que van a aparecer en la interfaz, y establecemos los últimos detalles. Una idea que se me ocurrió fue mostrar la valoración de la película, atributo que había introducido en Cypher pero que hasta ahora no había utilizado. En un principio, la idea era simple, obtener mediante una consulta a Neo4J los tres números de la crítica de las tres páginas web que seleccioné en el proceso de introducción de datos. Sin embargo, rápidamente surgieron las primeras complicaciones. Python no admitía el formato en el cual había introducido los datos anteriormente. Las notas de las películas estaban metidas de la manera "X'Y" y el apóstrofe daba muchísimos problemas. Aquí surgía el primer contratiempo real, aparte de los obstáculos potenciales que representa el desarrollo de software en sí mismo. Para solucionar esto, tenía dos opciones: o bien cambiar toda la base de datos y modificar las valoraciones de cada película, cosa que llevaría un trabajo enorme, o bien buscar una manera de reemplazar y cambiar el apóstrofe por un punto, para poder operar con cada nota. Finalmente decidí optar por la segunda opción y por ello esto fue lo que he debido hacer.

En esta primera captura de pantalla observamos cómo hago la petición a la base de datos, introduciendo los valores dados en el vector "valoraciones" primero, y en "valoraciones2" luego, pero con las notas divididas ya de las tres primeras iniciales con las que partimos. Como se ve al final, llamamos a un método llamado cambio que transforma de forma recursiva el apóstrofe en un punto.

```
valoraciones = []
valoraciones2 = []

with driver.session() as session:
    i = 0
    for record in session.run("MATCH (m:Movie)"
                              "WHERE m.title = '"+str(final[0])+"'"
                              "RETURN m.critica"):

        valoraciones.append(record["m.critica"])
        valoraciones2.append(str(valoraciones[i]).split(", "))
        i=i+1

valoraciones3 = []
valoraciones3.append(cambio(valoraciones2, 0, valoraciones3))
```

```
def cambio(valoraciones2, z, valoraciones3):
    auxiliar=[]
    if(z<3):
        auxiliar=valoraciones2[0][z].replace("'", ".")
        valoraciones3.append(auxiliar)
        z=z+1
        cambio(valoraciones2, z, valoraciones3)
```

Si bien es cierto que la mayor parte del trabajo estaba hecha, todavía no había llegado a mi propósito inicial, que era lograr sacar la nota media de las tres valoraciones, aunque esto ya fue la parte más fácil del proceso relativo a las notas y valoraciones.

```
critica = 0.0

for i in range(3):
    critica=critica+float(valoraciones3[i])
critica=round(critica/3, 2)
```

Llegados a este punto, tan sólo restaban los últimos detalles de mi interfaz: generar los mensajes de salida y controlar que se haya introducido algún criterio cuando se apriete el botón “buscar”. Para hacer un control de esto último realicé algo extremadamente parecido a un procedimiento anterior:

```
if(final[0]==""):
    messagebox.showerror("Error", "Introduce algún criterio")
nueva()
```

Vemos algo nuevo en la última línea y es la llamada a la función “nueva” que explicaremos más tarde. Por último, añadimos los mensajes que saldrán por pantalla mediante Labels.

```
raiz.label=ttk.Label(raiz)
raiz.label.place(x=65, y=285 )
raiz.label.config(text="Según los criterios introducidos, La película que recomendamos es: " + final[0])
raiz.label.config(font="Verdana 9 italic bold")

raiz.label2=ttk.Label(raiz)
raiz.label2.place(x=65, y=515)
raiz.label2.config(text="La valoración media de esta película es: " + str(critica))
raiz.label2.config(font="Verdana 9 italic bold")

raiz.combo.config(state="disabled")
raiz.combo5.config(state="disabled")
raiz.combo7.config(state="disabled")
raiz.combo9.config(state="disabled")
raiz.combo11.config(state="disabled")
```

Aquí vemos los dos mensajes que mostramos, así como su configuración. En las últimas líneas cambiamos los estados de los ComboBox de la izquierda a disabled. De esta manera, ya no pueden ser usados y evitamos que se realice una nueva búsqueda por encima de la otra.

Luego de finalizar esto decidí que iba a ser interesante poder ver el cartel de la película seleccionada como “ganadora” después de la recomendación. Aquí empezó un proceso de trabajo un poco tedioso y largo ya que debía crear una carpeta e introducir en ella imágenes de todas las películas que estaban metidas en la base de datos, además de cambiar el nombre a cada una de ellas y su tamaño, aunque esto último por equivocación ya que posteriormente supe que podía cambiar y reajustar el tamaño de la imagen al que yo quisiera desde el código. Después de buscar cómo introducir una foto y de varios intentos fallidos, di con la clave de cómo hacerlo:



```
##### IMAGEN PELICULA #####

path = 'C:\\Users\\dalon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Películas\\'+ str(final[0]) + ".jpg"
load = Image.open(path)
render = ImageTk.PhotoImage(load)
img = ttk.Label(raiz, image=render)
img.image = render
img.place(x=295 , y=315)
```

En el path especificamos la ruta en la cual se encuentra la carpeta que contiene las imágenes de los carteles, y añadimos jpg, ya que todas ellas tienen ese formato. De esta manera, especificamos el lugar donde queremos que salga con `img.place` y la foto se mostrará.

Después de acabar con la película en la ventana principal, tocaba ponerse con el libro. Este proceso fue muy sencillo. Una vez teniendo la película sólo tenemos que realizar una consulta a la base de datos y obtener el libro asociado a la película recomendada, así como su escritor. Una vez que tenemos todo esto, sacamos mediante un Label, como antes, la información del libro en el cual se basa la película recomendada.

```
escriptor = []
with driver.session() as session:
    for record in session.run("MATCH (m:Movie)-[Basado_En]-(l:Libro)"
                              "WHERE m.title='"+str(final[0])+"'"
                              "RETURN l.titulo"):
        libro.append(record["l.titulo"])

with driver.session() as session:
    for record in session.run("MATCH (m:Movie)-[Basado_En]-(l:Libro)"
                              "WHERE m.title='"+str(final[0])+"'"
                              "RETURN l.escriptor"):
        escritor.append(record["l.escriptor"])

raiz.label3=ttk.Label(raiz)
raiz.label3.place(x=65, y=550)
raiz.label3.config(text="La película está basada en el libro " + libro[0] + " de " + escritor[0])
raiz.label3.config(font="Verdana 9 italic bold")
```

Finalmente, al igual que hicimos con la película, mostramos la imagen de la portada del libro que previamente hemos introducido todas ellas en otra carpeta distinta:

```
##### IMAGEN LIBRO #####

path = 'C:\\Users\\dalon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Libros\\'+ str(libro[0]) + ".jpg"
load2 = Image.open(path)
render = ImageTk.PhotoImage(load2)
img = ttk.Label(raiz, image=render)
img.image = render
img.place(x=295 , y=585)
```

Quedaban ya solo pues, los últimos retoques de la aplicación. Cómo no se podía realizar una búsqueda sobre otra ya que me daba numerosos fallos y no conseguía solucionarlos, decidí implementar un botón, llamado “nueva búsqueda”, que reiniciaba la aplicación sin necesidad que el usuario hiciera nada más que apretar en él.



```
raiz.Button8 = ttk.Button(raiz, command=nueva, text = "NUEVA BÚSQUEDA")
raiz.Button8.place(x=345, y=225)
```

Al igual que en el resto de botones, el funcionamiento en este tampoco cambia, en la propiedad `command` llamamos a la función "nueva". Anteriormente esta función ya ha sido llamada, en caso de pulsar sobre "buscar" y no haber introducido ningún criterio. Dentro de este documento, más arriba, viene explicado. El único propósito de esta función es reiniciar el programa.

```
def nueva():
    python=sys.executable
    os.execl(python, python, * sys.argv)
```

Una vez finalizado esto, la aplicación ya está en total y correcto funcionamiento y podría presentarse ya. Sin embargo, bajo mi criterio la parte gráfica de la misma estaba bastante pobre. Estéticamente resultaba realmente poco atractiva así que pensé en varias maneras para solventar esto. Tras sopesar varias posibilidades, me decanté finalmente por colocar un fondo de mi aplicación con imágenes icónicas del cine. Además de esto, guardé varias imágenes e ideé una manera de que saliesen distintas imágenes cada vez que se ejecutase la aplicación. El sistema es sencillo: las fotos guardadas en una carpeta están numeradas del 1 en adelante. Después de esto, creé una variable que guardase un número aleatorio entre 1 hasta el número de fotos guardadas. De esta manera, cada vez que la aplicación se lanza, la imagen de fondo se seleccionará aleatoriamente.

```
aleatorio = randint(1, 12)

imagen = Image.open("C:\\Users\\dalon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Fondos\\"+str(aleatorio)+".jpg")
imagen_de_fondo = ImageTk.PhotoImage(imagen)
fondo = tk.Label(raiz, image=imagen_de_fondo)
fondo.place(x=0, y=0, relwidth=1, relheight=1)
```

Unos días después de finalizar todo esto, se me ocurrió la idea de introducir la posibilidad al usuario de consultar la información total de la película recomendada, y su libro asociado. El modo es absolutamente idéntico para el libro como para la película. Lo primero fue crear un botón llamado "información", y rápidamente pensé en que se abriese una ventana nueva donde apareciese toda la información requerida.

```
raiz.Button9 = ttk.Button(raiz, command=infopeli, text = "INFORMACIÓN")
raiz.Button9.place(x=455, y=335)
```

Como vemos, llamamos a la función `infopeli`. Ahí, lo primero que se hace es crear la nueva ventana e introducirle sus parámetros básicos:

```

info = tk.Toplevel(raiz)

info.title("Libro")
info.resizable(width=True, height=True)
info.geometry("950x600")
info.config(bg="grey")

```

Aplicamos a la nueva ventana la misma imagen de fondo que en la ventana principal:

```

imagenFondo = Image.open("C:\\Users\\dalon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Fondos\\"+str(aleatorio))
imagenFondo = imagenFondo.resize((950, 600), Image.ANTIALIAS)
imagen_de_fondo = ImageTk.PhotoImage(imagenFondo)
fondo = tk.Label(info, image=imagen_de_fondo)
fondo.place(x=0, y=0, relwidth=1, relheight=1)

```

Hacemos una consulta a la base de datos, donde le pedimos toda la información que está almacenada acerca de la película recomendada:

```

with driver.session() as session:
    for record in session.run("MATCH (m:Movie)
                              WHERE m.title='"+str(final[0])+"'
                              RETURN m.title, m.actores, m.critica, m.director, m.guion, m.país"):
        titulo.append(record["m.title"])
        prota.append(record["m.actores"])
        crit.append(record["m.critica"])
        direc.append(record["m.director"])
        guion.append(record["m.guion"])
        país.append(record["m.país"])

with driver.session() as session:
    for g in session.run("MATCH (m:Movie)-[Type_Of_Film]-(g:Genre)
                        WHERE m.title='"+str(final[0])+"'
                        RETURN g.type"):
        gen.append(g["g.type"])

```

Y mediante Labels, vamos escribiendo la información extraída en la nueva ventana:

```

info.label1=ttk.Label(info)
info.label1.place(x=425, y=50 )
info.label1.config(text=final[0])
info.label1.config(font="Arial 25 bold")

info.label2=ttk.Label(info)
info.label2.place(x=425, y=115 )
info.label2.config(text="Director: "+direc[0])
info.label2.config(font="Arial 10 bold")

info.label3=ttk.Label(info)
info.label3.place(x=425, y=155 )
info.label3.config(text="País: "+pais[0])
info.label3.config(font="Arial 10 bold")

info.label4=ttk.Label(info)
info.label4.place(x=425, y=195 )
info.label4.config(text="Actores principales: "+prota[0])
info.label4.config(font="Arial 10 bold")

info.label5=ttk.Label(info)
info.label5.place(x=425, y=235 )
info.label5.config(text="Valoraciones: "+crit[0])
info.label5.config(font="Arial 10 bold")

info.label6=ttk.Label(info)
info.label6.place(x=425, y=275 )
info.label6.config(text="Valoracion media: "+str(critica)+" ")
info.label6.config(font="Arial 10 bold")

info.label7=ttk.Label(info)
info.label7.place(x=425, y=355 )
info.label7.config(text="Genero: "+gen[0])
info.label7.config(font="Arial 10 bold")

```

Aquí, se produce el primer detalle interesante de esta función. En la ventana principal, anteriormente, hemos explicado cómo hemos obtenido la nota media de las tres valoraciones que buscamos en la web. Aquí utilizamos esa nota media para, en función de cuál sea, mostrar,

mediante imágenes de estrellitas de valoración almacenadas en una carpeta previamente, dicha nota obtenida.

```
if(critica<=2):
    path = 'C:\\Users\\dalon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Estrellas\\1.jpg'
    load = Image.open(path)
    load = load.resize((84, 16), Image.ANTIALIAS)
    render = ImageTk.PhotoImage(load)
    img = ttk.Label(info, image=render)
    img.image = render
    img.place(x=560, y=275)

elif(critica>2 and critica<=4):
    path = 'C:\\Users\\dalon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Estrellas\\2.jpg'
    load = Image.open(path)
    load = load.resize((84, 16), Image.ANTIALIAS)
    render = ImageTk.PhotoImage(load)
    img = ttk.Label(info, image=render)
    img.image = render
    img.place(x=560, y=275)

elif(critica>4 and critica<=6):
    path = 'C:\\Users\\dalon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Estrellas\\3.jpg'
    load = Image.open(path)
    load = load.resize((84, 16), Image.ANTIALIAS)
    render = ImageTk.PhotoImage(load)
    img = ttk.Label(info, image=render)
    img.image = render
    img.place(x=560, y=275)

elif(critica>6 and critica<=8):
    path = 'C:\\Users\\dalon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Estrellas\\4.jpg'
    load = Image.open(path)
    load = load.resize((84, 16), Image.ANTIALIAS)
    render = ImageTk.PhotoImage(load)
    img = ttk.Label(info, image=render)
    img.image = render
    img.place(x=560, y=275)

else:
    path = 'C:\\Users\\dalon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Estrellas\\5.jpg'
    load = Image.open(path)
    load = load.resize((84, 16), Image.ANTIALIAS)
    render = ImageTk.PhotoImage(load)
```

Una vez hecho esto, mostramos nuevamente el cartel de la película, esta vez en más grande, y añadimos una bandera del país de procedencia en la esquina superior izquierda del cartel.

```

path = 'C:\\Users\\daLon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Películas\\'+ str(final[0]) + ".jpg"
load = Image.open(path)
load = load.resize((350, 495), Image.ANTIALIAS)
render = ImageTk.PhotoImage(load)
img = ttk.Label(info, image=render)
img.image = render
img.place(x=50, y=50)

path = 'C:\\Users\\daLon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Banderas\\'+ str(pais[0]) + ".jpg"
load = Image.open(path)
load = load.resize((85, 59), Image.ANTIALIAS)
render = ImageTk.PhotoImage(load)
img = ttk.Label(info, image=render)
img.image = render
img.place(x=50, y=50)

```

Con el libro hacemos exactamente igual. Creamos un botón “información” para el libro:

```

raiz.Button9 = ttk.Button(raiz, command=infolibro, text = "INFORMACIÓN")
raiz.Button9.place(x=455, y=615)

```

Dentro, vemos la llamada a infolibro, una función, que, lo primero que realiza es crear una nueva ventana:

```

info = tk.Toplevel(raiz)

info.title("Libro")
info.resizable(width=True, height=True)
info.geometry("950x600")
info.config(bg="grey")

```

Establecemos la imagen de fondo, la misma que la anterior:

```

imagenFondo = Image.open("C:\\Users\\daLon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Fondos\\"+str(al
imagenFondo = imagenFondo.resize((950, 600), Image.ANTIALIAS)
imagen_de_fondo = ImageTk.PhotoImage(imagenFondo)
fondo = tk.Label(info, image=imagen_de_fondo)
fondo.place(x=0, y=0, relwidth=1, relheight=1)

```

Realizamos la consulta para obtener el escritor y el país y los mostramos en la aplicación mediante Labels:



```

with driver.session() as session:
    for record in session.run("MATCH (l:Libro)"
                              "WHERE l.titulo='"+str(final[0])+"'"
                              "RETURN l.titulo, l.escriptor, l.pais"):
        titulo.append(record["l.titulo"])
        escrit.append(record["l.escriptor"])
        pais.append(record["l.pais"])

info.label=tkk.Label(info)
info.label.place(x=425, y=50 )
info.label.config(text=libro[0])
info.label.config(font="Arial 25 bold")

info.label=tkk.Label(info)
info.label.place(x=425, y=115 )
info.label.config(text="Escritor: "+escrit[0])
info.label.config(font="Arial 10 bold")

info.label=tkk.Label(info)
info.label.place(x=425, y=155 )
info.label.config(text="País: "+pais[0])
info.label.config(font="Arial 10 bold")

```

Para finalizar, colocamos la imagen de la portada del libro, aunque en más grande y colocamos la bandera del país de origen del libro en la esquina superior izquierda.

```

path = 'C:\\Users\\daLon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Libros\\'+ str(final[0]) + ".jpg"
load = Image.open(path)
load = load.resize((350, 495), Image.ANTIALIAS)
render = ImageTk.PhotoImage(load)
img = tkk.Label(info, image=render)
img.image = render
img.place(x=50 , y=50)

path = 'C:\\Users\\daLon\\Documents\\Daniel\\Clase\\SistemasBI\\Fotos\\Banderas\\'+ str(pais[0]) + ".jpg"
load = Image.open(path)
load = load.resize((85, 59), Image.ANTIALIAS)
render = ImageTk.PhotoImage(load)
img = tkk.Label(info, image=render)
img.image = render
img.place(x=50 , y=50)

```

Una vez terminado esto, la aplicación estaría finalizada. Ha sido un proceso largo, de varios días, tedioso en algunos momentos, aunque estoy bastante satisfecho con el resultado final de la aplicación. A continuación, voy a enseñar cómo han quedado las tres ventanas. La principal, en la cual se encuentra el recomendador, la ventana de la información de la película, y la ventana de la información del libro.

## VENTANA PRINCIPAL

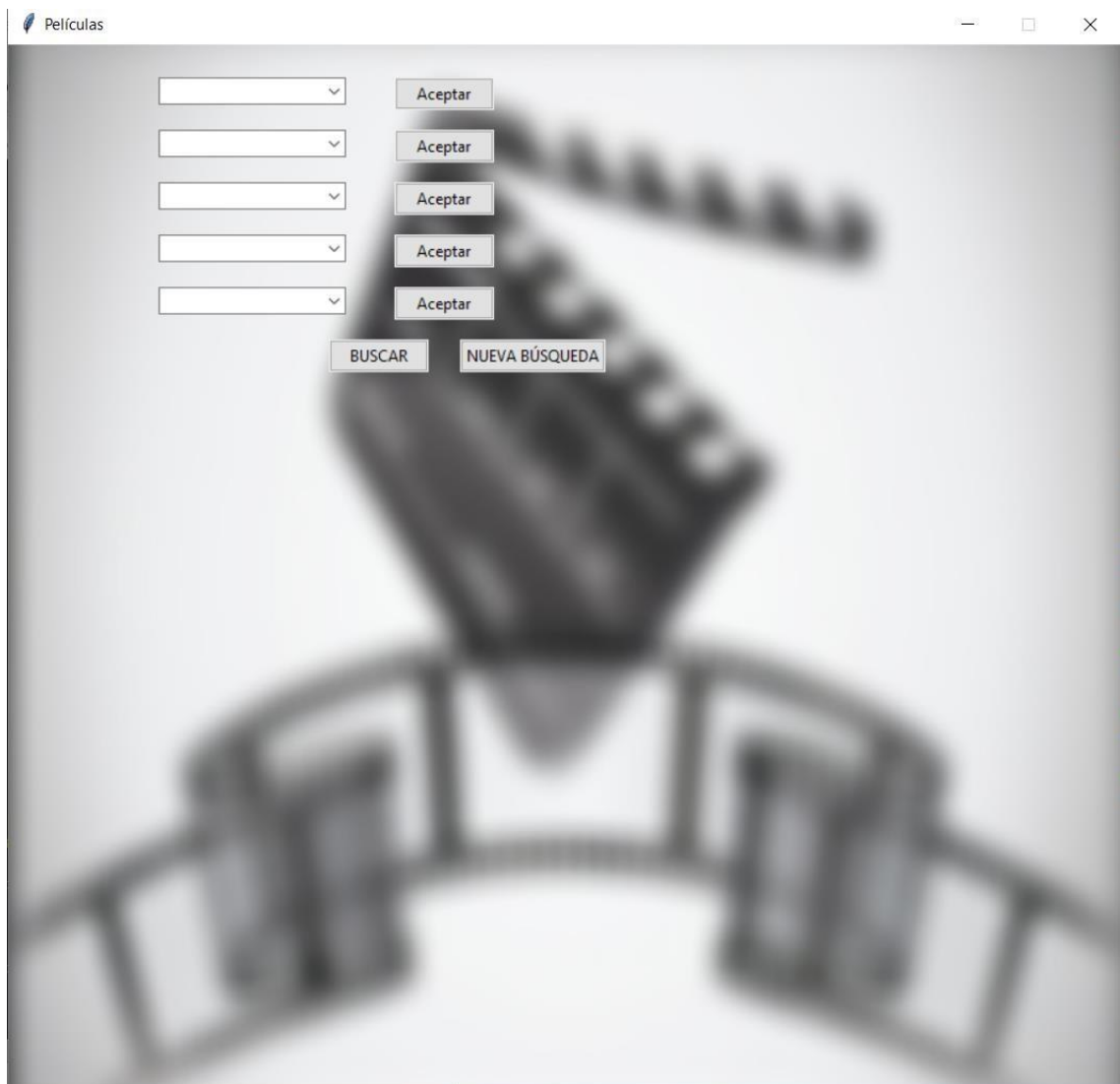


Películas

— □ ×

<input type="text"/>	Aceptar
<input type="text"/>	Aceptar
<input type="text"/>	Aceptar
<input type="text"/>	Aceptar
<input type="text"/>	Aceptar

BUSCAR NUEVA BÚSQUEDA



Películas

Director

Aceptar

Francis Ford Coppola

Aceptar

Aceptar

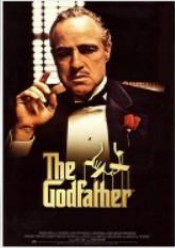
Aceptar

Aceptar

BUSCAR

NUEVA BÚSQUEDA

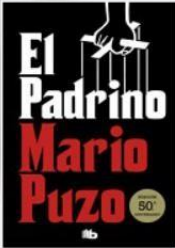
Según los criterios introducidos, la película que recomendamos es: **El Padrino**



INFORMACIÓN

La valoración media de esta película es: 9.6

La película está basada en el libro **El Padrino** de Mario Puzo



INFORMACIÓN

## VENTANA INFORMACIÓN PELÍCULA

27

Películas



**El Padrino**

Director: Francis Ford Coppola

País: EE.UU.

Actores principales: Marlon Brando, Al Pacino, James Caan, Diane Keaton, Robert Duvall

Valoraciones: 10, 9'8, 9

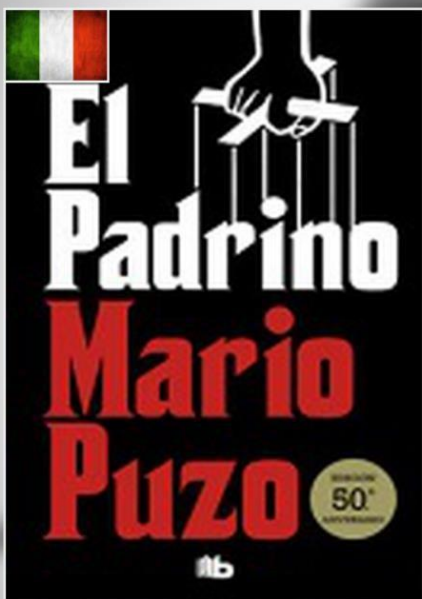
Valoración media: 9.6 ★★★★★

Guionista: Mario Puzo, Francis Ford Coppola

Genero: Drama

## VENTANA INFORMACIÓN LIBRO

Libro



**El Padrino**

Escritor: Mario Puzo

País: Italia