UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Departamento de Sistemas Informáticos y Computación

Universitat Politècnica de València

Una Base de Datos basada en un Modelo Conceptual para la Interpretación del Proteoma del SARS.COV-2

Trabajo Fin de Máster

Máster Universitario en Ingeniería y Tecnología de Sistemas Software

Autor: Runbin Dong

Tutor: Pastor López, Oscar

2020-2021

Resumen

Desde que el SARS-COV-2 se convirtió en pandemia, la comunidad científica ha hecho un gran esfuerzo global para descifrar la genética del virus y entender de esta forma los mecanismos que llevan a la infección y la respuesta del huésped, conocida como Covid-19. Todo este conocimiento generado se ha ido almacenando en repositorios específicos como el Covid l9 UniProtKB, donde se puede consultar informaciones sobre la secuencia del virus y las proteínas que los constituyen. El estudio de estas proteínas y sus interacciones con las proteínas del huésped son las que pueden explicar los mecanismos que el virus utiliza para entrar en las células y predecir la intensidad de la respuesta inmunológica que se puede producir en el organismo infectado. Pero para poder almacenar, analizar y comprender toda esta complejidad de datos de forma correcta es necesario disponer de una base de datos que permita su estructuración en base a una definición ontológica precisa del dominio. Utilizando como base, un modelo conceptual sobre las proteínas desarrollado en el centro de investigación PROS, este trabajo plantea la implementación de una base de datos que permita almacenar de forma estructurada la información actualmente disponible sobre el proteoma del SARS-Cov-2, y relacionar los datos de forma que las consultas realizadas sobre dicha base de datos permitan dar respuesta a problemas y preguntas de investigación actualmente abiertas.

Los laboratorios de investigación del proteoma producen diariamente grandes cantidades de datos sobre las proteínas. La página web existente hoy en día Uniprot almacena datos de manera RDF. Aunque los datos RDF pueden expresar con precisión la información semántica, carecen de la capacidad abstracta de describir cosas concretas, no pueden definir y describir la misma categoría de las cosas, y tienen un enorme espacio de almacenamiento y una gran complejidad temporal de algoritmo de consulta. Debido a las grandes características de los datos de los proteomas, se hace que el volumen de datos de una tabla alcance millones o incluso decenas de millones.

Las bases de datos relacionales tradicionales como MySQL, sin embargo, pueden ser optimizadas mediante la adición de índices, pero una tabla con mucha memoria de almacenamiento, claramente cómo optimizarla es difícil de satisfacer la demanda, aunque se pueda hacer *Sharding*, el costo consumido es también muy alto, y las bases de datos relacionales son ineficientes cuando se trata de consultas de asociación profunda, por lo que las bases de datos relacionales tienen el problema del almacenamiento eficiente y el acceso a datos masivos.

En profundidad la consulta asociada y el motor de computación gráfico, la flexibilidad de los datos, la agilidad de desarrollo, base de datos orientada a grafos surgió.

Neo4j es un motor de base de datos orientada a grafos original, que tiene una estructura de almacenamiento única sin el método de almacenamiento de nodos vecinos de índice y el correspondiente algoritmo de travesía de gráficos, por lo que su rendimiento no se verá afectado con el aumento de los datos. Neo4j tiene un alto rendimiento en consulta.

La extensión natural de la estructura de base de datos orientada a grafos y su formato de datos no estructurado hacen que el diseño de la base de datos de Neo4j sea muy escalable y flexible.

Palabras clave: Sistemas de Información para la BioInformática; Modelado Conceptual; Gestión de Datos del Proteoma; Ciencias de Datos Genómicas

Resumen

Desde que el SARS-COV-2 es va convertir en pandemia, la comunitat científica ha fet un gran esforç global per a descifrar la genética del virus y poder entendre d'esta manera els mecanismos de infecció i la resposta de l'hoste, coneguda com Covid-19. Tot aquets coneiximent generat s'ha almacenat en repositoris cientifics com el Covid-19 UniProtKB, on ja es pot consultar informacions sobre la seqüència del virus i les proteïnes que els constituïxen El estudi d aquestes proteines y le seues interaccion amb les proteines de l hoste son les que poden explicar els mecanismos que el virus utilitza per entrar a les cèl·lula i predecir la intensitat de la resposta inmunológica que es pot produir en el organisme infectat. Però per a poder emmagatzemar, analitzar i comprendre tota la complexitat de dades de forma correcta és necessari disposar d'una base de dades que permeta la seua estructuració basant-se en una definició ontològica precisa del domini. Utilitzant com a base, un model conceptual sobre les proteïnes desenvolupades en el centre d'investigació PROS, este treball planteja la implementació d'una base de dades que permeta emmagatzemar de forma estructurada la informació actualment disponible sobre el proteoma del SARS-Cov-2, i relacionar les dades de forma que les consultes realitzades sobre la dita base de dades permeten donar resposta a problemes i preguntes d'investigació actualment obertes.

Els laboratoris d'investigació del proteoma produïxen diàriament grans quantitats de dades sobre les proteïnes. La pàgina web existent hui en dia Uniprot emmagatzema dades de manera RDF. Encara que les dades RDF poden expressar amb precisió la informació semàntica, careixen la capacitat abstracta de descriure coses concretes, no

poden definir i descriure la mateixa categoria de les coses, i tenen un gran espai d'emmagatzemament i una gran complexitat temporal d'algoritme de consulta. Degut de les grans característiques de les dades dels proteomes, es fa que el volum de dades d'una taula abast milions o inclús desenes de milions.

Les bases de dades relacionals tradicionals com MySQL, no obstant això, poden ser optimitzades mitjançant l'addició d' un índex, però una taula amb molta memòria d'emmagatzemament, clarament com optimitzar-la és difícil de satisfer la demanda, encara que es puga fer Sharding, el cost consumit és també molt alt, i les bases de dades relacionades són ineficients quan es tracta de consultes d'associació profunda, per la qual cosa les bases de dades relacionals tenen el problema de l'emmagatzemament eficient i l'accés a dades massives.

En profunditat la consulta associada i el motor de computació gràfic, la flexibilitat de les dades, l'agilitat de desenvolupament, base de dades orientada a grafos va sorgir

Neo4j és un motor de base de dades orientada a grafos original, ha de tindre una estructura d'emmagatzemament única sense el mètode d'emmagatzemament de nodes veïns d'índex i el corresponent algoritme de travessia de gràfics, per la qual cosa el seu rendiment no es veurà afectat amb l'augment de les dades. Neo4j té un alt rendiment en consulta.

L'extensió natural de l'estructura de base de dades orientada a grafos i el seu format de dades no estructurat fan que el disseny de la base de dades de Neo4j siga molt escalable i flexible.

Palabras clave: Sistemas de Información para la BioInformática; Modelado Conceptual; Gestión de Datos del Proteoma; Ciencias de Datos Genómicas

Abstract

Since SARS-COV-2 became a pandemic, the scientific community has made a great global effort to decipher the genetics of the virus and understanding in this way the mechanisms that lead to infection and the host response, known as Covid-19. All this generated knowledge has been stored in specific repositories such as the Covid19 UniProtKB, where you can find information about the sequence of the virus and the proteins that make it up. The study of these proteins and their interactions with host proteins are those that can explain the mechanisms that the virus uses to enter cells and predict the intensity of the immune response that can occur in the infected organism. But to be able to store, analyze and understand all this complexity of data correctly, it is necessary to have a database that allows its structuring based on a precise ontological definition of the domain. Using as a basis, a conceptual model on proteins developed at the PROS research center, this work

proposes the implementation of a database that allows the information currently available on the SARS-Cov-2 proteome to be stored in a structured way, and to relate the data so that the queries made on that database allow us to respond to problems and research questions that are currently open.

Proteome research laboratories produce vast amounts of data on proteins on a daily basis. Today's existing Uniprot website stores data in an RDF manner. Although RDF data can accurately express semantic information, it lacks the abstract ability to describe concrete things, it cannot define and describe the same category of things, and it has a huge storage space and great temporal complexity of query algorithm. Due to the large characteristics of proteome data, the data volume of a table is made to reach millions or even tens of millions.

Traditional relational databases such as MySQL, however, can be optimized by adding indexes, but a table with a lot of storage memory, clearly how to optimize it is difficult to meet the demand, although Sharding can be done, the cost consumed is also very high, and relational databases are inefficient when it comes to deep association queries, so relational databases have the problem of efficient storage and access to big data.

In-depth associated query and graphical computing engine, data flexibility, development agility, graph-oriented database emerged.

Neo4j is an original graph-oriented database engine, which has a unique storage structure without the index neighbor node storage method and the corresponding graph traversal algorithm, so its performance will not be affected by the data augmentation. Neo4j has a high performance in query.

The natural extension of the graph-oriented database structure and its unstructured data format make Neo4j's database design highly scalable and flexible.

Key words: Information Systems for the Bioinformatics; Conceptual Model; Data Management for the Proteome; Genome Data Science

Tabla de contenidos

Índice de figuras

Índice de Tablas

Introducción

Las proteínas tienen funciones biológicas complejas y delicadas en el proceso de las actividades de la vida. La información de estructura de alto nivel de las proteínas que pueden participar en las actividades de la vida de los organismos incluye la estructura secundaria, la estructura terciaria, la estructura cuaternaria, etc. La estructura de alto nivel de la proteína y la interacción altamente específica entre esta estructura de alto nivel y sus moléculas correspondientes son la base de las diversas funciones de la proteína. Por lo general, una determinada función bioquímica local requiere una cierta estructura local fina de la proteína La formación y los cambios de la estructura de la proteína siguen los principios bioquímicos básicos. Descubrir y extraer la información característica de la estructura de la proteína es la base para comprender la función biológica de la proteína y comprender la interacción entre proteína y proteína.

En este trabajo, los datos de proteínas se almacenan estructurados de acuerdo con el modelo conceptual de proteínas para facilitar a los investigadores el análisis adicional de los datos de proteínas. Los contenidos principales son los siguientes:

Capítulo 1 Objetivos y preguntas de investigación donde se definan los objetivos y las preguntas de investigación.

Capítulo 2 Metodología donde se explique la metodología utilizada para desarrollar el trabajo.

Capitulo 3 Fundamento de investigación relevante donde se presenta las herramientas de desarrollo de software utilizadas en este trabajo.

Capítulo 4 Extracción de datos de Uniprot donde se introduce cómo obtener y organizar los datos de proteínas requeridos del sitio web de Uniprot.

Capitulo 5 Reestructuración de los datos de acuerdo con el Modelo Conceptual de Proteinas donde se introduce como almacenar los datos de proteínas recopilados paso a paso en la base de datos de Neo4j de acuerdo con el modelo conceptual de proteínas.

Capítulo 6 Ejecución de consultas sobre la base de datos donde visualice los datos de proteínas almacenados en la base de datos de Neo4j.

Conclusión donde se resuma el contenido de este trabajo y se responda las preguntas de investigación planteadas.

Capítulo 1 Objetivos y preguntas de investigación

Actualmente, las bases de datos de almacenamiento de proteínas existentes, como UniprotKB, almacenan principalmente datos en RDF. Los datos RDF carecen de la capacidad abstracta para describir cosas específicas, tienen un gran espacio de almacenamiento y el algoritmo de consulta tiene una gran complejidad de tiempo. Para poder almacenar, analizar y comprender correctamente todos estos datos complejos, es necesario contar con una base de datos que permita estructurar definiciones ontológicas precisas basadas en dominios. Tomando como base el modelo conceptual de proteínas desarrollado por el Centro de Investigaciones PROS, este trabajo plantea la implementación de una base de datos que permita almacenar de forma estructurada la información actualmente disponible sobre el proteoma del SARS-Cov-2, y relacionar los datos de forma que las consultas realizadas sobre dicha base de datos permitan dar respuesta a problemas y preguntas de investigación actualmente abiertas. Los objetivos planteados para la consecución del trabajo son los siguientes:

Objetivo 1: Implementar una base de datos que permita almacenar de forma estructurada el conocimiento disponible sobre las proteínas

> **Pregunta de investigación 1:** ¿Qué información es necesaria para representar de forma precisa el conocimiento actual sobre las proteínas?
>
> **Pregunta de investigación 2:** ¿Qué tecnología es la más adecuada para almacenar la información?

Objetivo 2: Poblar la base de datos con la información disponible sobre las proteínas relacionadas con el SARS-CoV-2 y las proteínas del huésped con las que interacciona

> **Pregunta de investigación 3:** ¿Dónde se puede obtener información sobre las proteínas que constituyen en SARS-CoV-2?
>
> **Pregunta de investigación 4:** ¿Cómo se puede almacenar dicha información en la base de datos implementada?

Objetivo 3: Evaluar la utilidad de la base de datos y de la información almacenada para la extracción de conocimiento relevante sobre la COVID-19.

> **Pregunta de investigación 5:** ¿Permite la base de datos almacenar el conocimiento actual generado sobre el proteoma del SARS-CoV-2?
>
> **Pregunta de investigación 6:** ¿Es la base de datos útil para determinar los mecanismos que llevan al desarrollo de la COVID-19?

Capítulo 2 Metodología

Cuando los investigadores biológicos recopilan datos, primero deben aprender a usar la base de datos y completar manualmente la búsqueda de acuerdo con los pasos de operación de la base de datos. Los datos buscados no están necesariamente en el formato deseado, lo que causa grandes inconvenientes a los investigadores biológicos a la hora de preparar los datos. Por lo tanto, se diseña un módulo para rastrear datos automáticamente, que puede descargar los datos de la base de datos de forma rápida y precisa.

Luego, de acuerdo con el artículo **Conceptual Modeling of Proteins Based on UniProt[1]**, se crea las clases de entidades de datos relacionados con proteínas y las relaciones entre las entidades. Luego, los datos rastreados se limpian y almacenan en la clase de entidad correspondiente, y todas las asignaciones de entidades relacionadas se almacenan en la base de datos de Neo4j con los datos de proteínas como cuerpo principal. Finalmente, se realiza una consulta visual de la base de datos a través del Neo4j Browser.

El marco general de este trabajo se muestra en la figura 1:

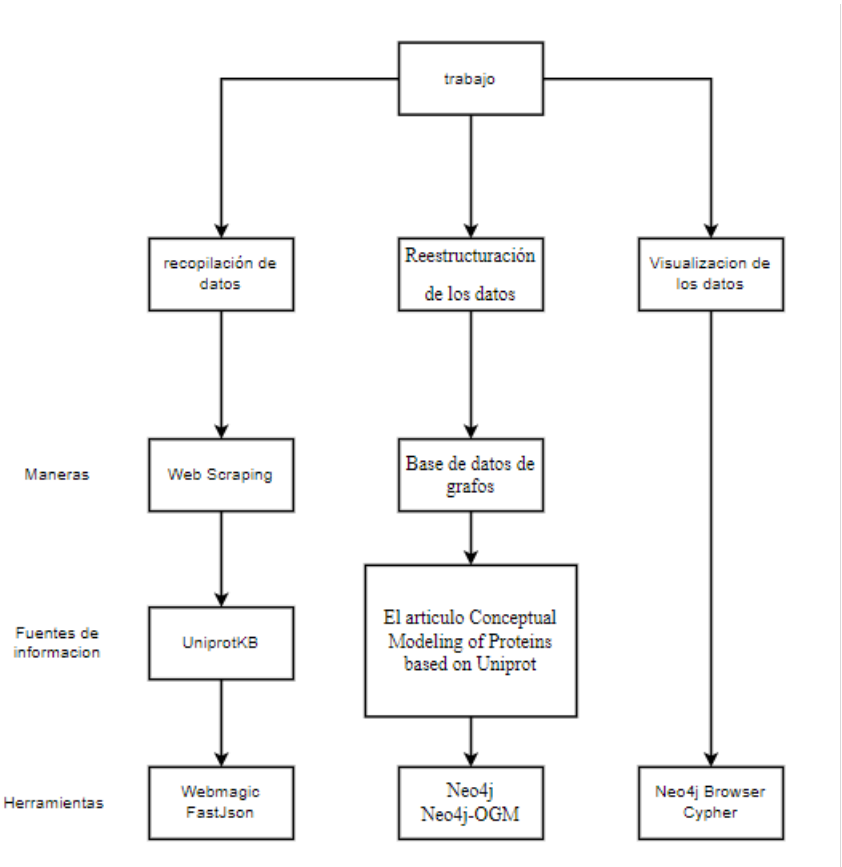

Figure 1 El marco general

Capítulo 3 Fundamento de investigación relevante

3. 1. Web scraping

3. 1. 1. Conceptos de la extracción de la información de páginas web

HTML (Hyper Markup Language) es un lenguaje estándar para hacer páginas web.

Basado en el lenguaje HTML, las computadoras que utilizan diferentes programas de procesamiento de palabras pueden comunicarse libremente, pero no es un lenguaje de programación, sino un lenguaje de marcado. Las etiquetas, también conocidas como marcas, se utilizan para identificar elementos individuales en el navegador. Para distinguir las diferentes partes del navegador, la página web se marca con diferentes tipos de etiquetas, como las etiquetas de propiedad para marcar el tipo de letra, el tamaño de la letra, el color de la letra, etc.

El lenguaje HTML es uno de los tres estándares centrales de la World Wide Web. La World Wide Web es un sistema vinculado por muchos documentos de hipertexto. Los recursos del sistema se identifican por medio de la URL y se devuelven al usuario basándose en el protocolo de transferencia de hipertexto, y el usuario puede hacer clic en el enlace para obtener recursos. El URL es el Localizador Uniforme de Recursos, que es la dirección web que introducimos en la barra de direcciones del navegador; Como tal, el URL es la dirección específica que se asigna a cada uno de los recursos disponibles en la red con la finalidad de que estos puedan ser localizados o identificados.

El HTTP es principalmente un protocolo de comunicación que especifica la forma en que se intercambia información entre un cliente y un servidor. Introducimos la URL para determinar la dirección a visitar, y el navegador extrae el código de la página web del servidor a través del protocolo de transferencia de hipertexto y nos lo devuelve como una página web traducida.

Hay aproximadamente tres tipos de etiquetas en el lenguaje HTML de la web.

La primera es la disposición de la página del lenguaje de etiquetas, esta etiqueta de disposición es principalmente para las diferentes áreas de la página, la información de contenido diferente a etiquetar, a través de esta división de etiquetas, puede ser la disposición local de la página, en términos generales, como <*tabla*>, <*tr*>, <*div*> y otras etiquetas son etiquetas de disposición;

Luego hay algunas etiquetas que contienen Hiperenlace, como <*img*> y otras etiquetas, pueden ser utilizadas para conectar cada página, representando la relación entre las páginas;

La última categoría de etiquetas se utiliza para mostrar características de información, como la etiqueta <*strong*>, que se utiliza principalmente para describir cómo debe mostrarse el contenido, qué estilo y formato debe tener.

Cuando se analiza el archivo de la página web, el documento correspondiente de la página web se compone principalmente del encabezado y el cuerpo.

Se comparan de cabeza a cola con la etiqueta <*html*>, y el contenido intermedio está compuesto por las etiquetas que muestran el contenido principal de la página y las

etiquetas que describen los recursos necesarios para la página. En general, la estructura sintáctica es relativamente simple. Al utilizar una clase de palabra clave predefinida, el contenido de la palabra clave se marca con corchetes en ángulo. Cualquier tipo de etiqueta debe aparecer en pares o puede anidarse, conteniendo múltiples tipos de contenido de la etiqueta.

3. 1. 2. Tecnología Web Scraping basada en Java

Hoy en día, la Internet está llena de contenidos que abarcan la vida, la ciencia y la tecnología, las humanidades, los aspectos militares y otros, y no se puede separar cada campo del soporte de datos. Dichos estos se construye una enorme base de datos de conocimientos.

Se ha convertido en un gran desafío imaginar que la transformación de la cantidad de conocimientos en la Internet en una forma estructurada aportará un gran valor a la búsqueda rápida de datos y al pleno aprovechamiento de la información de los datos. Del mismo modo, la Internet contiene, con mucho, el conjunto de datos más útil, algunos de los cuales están incrustados en la estructura y el estilo del sitio y deben extraerse para su utilización. Por lo tanto, con esos antecedentes, la tecnología de *Web scraping* ha tenido plenas oportunidades de desarrollo. Hasta ahora, esa tecnología se ha actualizado constantemente, no sólo en términos de velocidad, sino también para garantizar la calidad de la información. La tecnología de *Web scraping* basada en Java está especializada en la recopilación de información de Internet, que se origina en la demanda de la gente de descubrimiento y adquisición automáticos de información en línea.

Para resumir esta operación en términos técnicos, es programar para simular el funcionamiento del sitio de solicitud del navegador, el sitio devuelve una variedad de tipos de datos de rastreo al local, y luego realiza las operaciones correspondientes, extrayendo los datos requeridos.

Por lo tanto, el aprendizaje de *Web scraping* puede dividirse en cuatro partes. La primera parte es tener un objetivo claro, y debe saber qué área o sitio web va a buscar. La segunda parte es *Web scraping* o raspado web, se raspa todos los contenidos en la página web destinada. La tercera parte es el análisis, los contenidos raspados de la página web deben ser analizado, y eliminar los datos que no nos sirven; La última es procesar los datos, para almacenarlos y procesarlos en la forma que necesitamos. Dichas cuatro partes son la idea básica de un *web scraping*. En pocas palabras, la *Web scraping* realiza la operación del usuario de obtener datos del navegador manualmente especificando la URL, y finalmente devuelve los datos que el usuario necesita obtener manualmente.

Webmagic[2], un marco de trabajo de *Web scraping* basado en Java, muestra ventajas sobre otros lenguajes. Webmagic es un marco de trabajo de código abierto de rastreador web vertical de Java con el objetivo de simplificar el proceso de desarrollo de rastreador y permitir que los desarrolladores se centren en el desarrollo de funciones lógicas. Adopta un diseño completamente modular con funciones que cubren todo el

ciclo de vida del rastreador (extracción de enlaces, descarga de páginas, extracción de contenidos y persistencia), soporta la obtención de múltiples hilos y la obtención distribuida, y soporta reintentos automáticos, UA/cookie personalizada y otras funciones. Webmagic incluye la extracción de páginas, donde los desarrolladores pueden enlazar y extraer contenido usando selectores CSS, xPaths y expresiones regulares, y soporta llamadas en cadena de múltiples selectores.

3. 2. Base de datos orientada a grafos Neo4j

3. 2. 1. Conceptos básicos de la base de datos orientada a grafo

La base de datos orientada a grafo es un nuevo tipo de base de datos no relacional. Su gráfico se basa en la teoría de grafos tanto el nodo como el borde. Los nodos y bordes de los elementos básicos en la teoría de grafos corresponden a los nodos y relaciones en las bases de datos orientada a grafos. El modelo para una base de datos orientada a grafo consiste en nodos, relaciones y atributos. Principalmente almacena dos tipos de nodos y bordes de datos. Los nodos son entidades, como personas, facturas, películas, libros u otras cosas concretas. Son similares a los registros de una base de datos relacional. Borde (relación): Un concepto, evento o cosa que conecta los nodos. A diferencia de las bases de datos relacionales, estas relaciones se pueden buscar en la base de datos orientada al grafo con sus propios permisos. Un nodo puede tener desde cero hasta más de una propiedad, que existe como un par clave-valor. Las relaciones también pueden tener múltiples propiedades y etiquetas, y una relación consiste en un nodo de inicio y un nodo de parada.

En comparación con las bases de datos orientada a grafo, las bases de datos relacionales siempre han sido populares entre los usuarios, y a menudo se utilizan para construir tablas bidimensionales y sus relaciones, y realizar múltiples operaciones interactivas utilizando las conexiones entre tablas. Pero la desventaja es que, si utilizamos la base de datos relacional para los datos no estructurados, no conseguiremos el efecto deseado, y la asociación entre las tablas reducirá la eficiencia y aumentará el tiempo de funcionamiento. Sin embargo, la base de datos orientada a grafo tiene una gran ventaja a este respecto.

El almacenamiento de la base de datos orientada a grafo también es diferente de la base de datos relacional. Al almacenar nodos, bordes y atributos de la estructura de los gráficos, la base de datos orientada a grafo proporcionará un perfecto lenguaje de consulta de gráficos, soportará varios algoritmos de minería de gráficos y soportará el almacenamiento distribuido. Las bases de datos orientada a grafo comunes incluyen Neo4j, FlockDB y OrientDB.

3. 2. 2. Resumen de base de datos orientada a grafos Neo4j

Neo4j[9] es actualmente la más popular entre las numerosas bases de datos orientada a grafo, con un alto rendimiento y madurez. El modelo se describe basado en el concepto de correlación de gráficos, y los datos se almacenan como nodos y relaciones. Tiene las características de la investigación de **S Jouili[4]**, soporte masivo de datos y consulta rápida de gráficos, etc. A continuación, este trabajo se desarrollará utilizando un software de base de datos orientada a grafo de código abierto.

Neo4j almacena datos estructurados en la Internet en lugar de en tablas. Esta característica puede poner preocupaciones a los usuarios sobre la seguridad de sus datos, pero se está almacenando en la Internet, lo que lo hace más rápido cuando se trata de grandes cantidades de datos, y más eficiente cuando se trata de grandes cantidades de datos de baja estructura que son engorrosos y desordenados.

Cuando se modelan datos alrededor de un gráfico, no importa cuántos datos tenga el gráfico, éste atraviesa nodos y bordes a la misma velocidad, y tiene todas las características de una base de datos madura: cosas ACID, código abierto, modelos de datos complejos, lenguaje declarativo de consulta de gráficos, interfaz amigable, alto rendimiento y basado en JVM.

En características de lo anterior, es mediante la localización de la base de datos de imágenes de construcción de abajo hacia arriba, más rápida para manejar mejor los nodos y la relación: puede soportar grandes conjuntos de datos y ampliar continuamente su capacidad y puede almacenar decenas de miles de millones de entidades, es decir, lo secundario es el alto rendimiento: la fuerza expresiva del lenguaje de consulta de figuras declarativas es rica, la eficiencia de la consulta es alta, y tiene una buena expansibilidad, los usuarios pueden personalizar su propia forma de consulta.

Por último, Neo4j proporciona una verdadera seguridad de los datos mediante transacciones ACID, utilizando transacciones para garantizar que los datos no se pierdan en caso de un fallo de hardware o una caída del sistema.

3. 2. 3. Resumen del lenguaje de consulta Cypher

Cypher[10] es un lenguaje de consulta de bases de datos orientada a grafos, expresivo y eficiente, su estado y función y el equivalente en lenguaje SQL de bases de datos relacionales. Este lenguaje declarativo de consulta de bases de datos orientada a grafos puede consultar y actualizar los datos gráficos de manera eficiente. Para los principiantes, Cypher es relativamente sencillo de usar, pero lo suficientemente potente como para expresar incluso consultas de bases de datos muy complejas en pocas palabras. Esto permite a los usuarios centrarse en lo que están haciendo y no gastar demasiado tiempo en el acceso a la base de datos. Los lenguajes de consulta declarativos heredan muchas prácticas idiomáticas, como “Where” y “Order BY”, Donde la mayoría de las palabras clave vienen del lenguaje SQL. También se ha adoptado la estructura de los lenguajes de bases de datos relacionales, donde las

consultas pueden tener una variedad de combinaciones de declaraciones, declaraciones enlazadas entre sí y conjuntos de resultados intermedios que se pasan entre sí.

Neo4j se compone de nodos y relaciones, los nodos también tienen etiquetas y propiedades, y las relaciones también pueden tener tipos y propiedades. Los nodos representan entidades, y las relaciones conectan un par de nodos. Sin embargo, los nodos y las relaciones están en un nivel bajo, y si se quiere codificar más información, se necesita un patrón, que puede codificar muchos nodos y relaciones en ideas arbitrariamente complejas. Cypher se basa en gran medida en patrón y se adopta las sentencias de tipo SQL para combinar estos patrones para expresar las operaciones esperadas.

3. 2. 4. Resumen de Neo4j-OGM

Un **OGM (Object Graph Mapper)[3]** mapea los nodos y las relaciones en el gráfico con los objetos y las referencias en un modelo de dominio. Las instancias de los objetos se mapean a los nodos mientras que las referencias de los objetos se mapean usando relaciones, o se serializan a las propiedades. Las primitivas JVM se mapean a propiedades de nodos o relaciones. Un OGM abstrae la base de datos y proporciona una forma conveniente de persistir su modelo de dominio en el gráfico y consultarlo sin usar drivers de bajo nivel. También proporciona la flexibilidad al desarrollador de suministrar consultas personalizadas cuando las consultas generadas por Neo4j-OGM son insuficientes.

Neo4j-OGM es una librería de mapeo rápido de objetos-gráficos para Neo4j, optimizada para instalaciones basadas en servidores que utilicen Cypher. Su objetivo es simplificar el desarrollo con la base de datos de gráficos de Neo4j y, al igual que el JPA, utiliza anotaciones en objetos de dominio POJO simples para hacerlo. Centrándose en el rendimiento, Neo4j-OGM introduce una serie de innovaciones, entre ellas:

Escaneo de classpath sin reflejos para un tiempo de inicio mucho más rápido.

Persistencia de profundidad variable para permitirle afinar las solicitudes de acuerdo con las características de su gráfico.

Mapeo inteligente de objetos para reducir las solicitudes redundantes a la base de datos, mejorar la latencia y minimizar los ciclos de CPU desperdiciados.

Vida de sesión definible por el usuario, ayudándole a encontrar un equilibrio entre el uso de la memoria y la eficiencia de las solicitudes del servidor en sus aplicaciones.

Capítulo 4 Extracción de datos de Uniprot

En el capítulo anterior se describen los puntos de conocimiento teórico que intervienen en el presente trabajo, principalmente sobre la construcción del grafo de conocimientos de proteínas utilizado por algunas bases teóricas pertinentes, pero también se describe en detalle el uso de los puntos técnicos pertinentes. Por ejemplo, la tecnología de Web scraping y la base de datos de orientada al grafo Neo4j, etc. Construir un grafo de conocimiento completo es un trabajo complicado, pero las etapas más importantes son la adquisición de conocimientos, la representación de conocimientos, el almacenamiento de conocimientos y la visualización de conocimientos. En este capítulo se describen detalladamente los cuatro pasos principales para construir un grafo del conocimiento: adquisición de datos, procesamiento de datos, almacenamiento de datos y, por último, consultas visuales. Apuntando a la gran cantidad de datos de proteínas en Uniprot, usaremos Web scraping para adquirir y procesar los datos, luego usaremos la base de datos orientada al grafo Neo4j para almacenar, y finalmente usaremos el lenguaje Cypher para la consulta visual para construir un grafo de conocimiento de proteínas sistemático y estandarizado.

4. 1. **Métodos de recopilación de datos de Uniprot**

Uniprot ofrece oficialmente dos formas de obtener datos:

1. Descarga
2. API de consulta

A través del botón "descargar" en la página de inicio de Uniprot, se pueden obtener datos de proteínas en varios formatos. Por ejemplo, Excel, XML, Json, TXT, etc. Sin embargo, dado que Uniprot en sí mismo son datos de proteínas almacenados en formato Json, los datos serán anormalmente desordenados cuando se conviertan a otros formatos, y los datos descargados en formato Json no están completos. Tomemos como ejemplo los datos de Subcellular location en formato Excel, como se muestra en la figura 2, es difícil determinar cómo se define su estructura de datos, lo que ocasiona dificultades en la limpieza de datos.

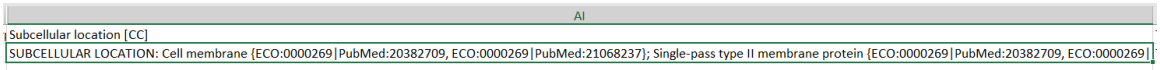

Figure 2 Datos de Subcellular location en formato Excel

El uso de API de consulta para realizar consultas requiere un conocimiento suficiente de UniprotKB. El propio UniprotKB almacena datos en forma de RDF y utiliza SparQL como lenguaje de consulta. Sin saberlo, es difícil para nosotros conocer el nombre de campo específico de la fuente de datos.

Por lo tanto, los dos métodos para obtener datos proporcionados oficialmente por Uniprot no son suficientes para satisfacer nuestras necesidades de desarrollo, por lo que pensamos en utilizar web scraping para extraer datos. Esto tiene muchos beneficios adicionales:

1. Después de que Uniprot actualice los datos, no es necesario volver a descargarlos manualmente, solo es necesario volver a ejecutar el programa.

2. Se usa fuentes de datos en lugar de los métodos que nos proporciona Uniprot, lo que puede reducir algunos costos de aprendizaje.

3. Se puede garantizar que todos los datos mostrados en la página web de Uniprot estén disponibles para nosotros.

4.2. Análisis de fuente de datos

En cuanto al análisis de la fuente de datos, este trabajo toma los datos de Uniprot como fuente de datos para clasificar las proteínas involucradas en COVID-19 a partir de datos masivos. Uniprot Usa el dominio de segundo nivel NOMBRE COVID-19 para clasificar específicamente las proteínas involucradas, simplificando nuestro trabajo de recolección de fuentes de datos. Debido a que Uniprot utiliza una forma semiestructurada para almacenar datos, después de obtener los datos de origen, primero debemos organizar los datos, guardar los datos que necesitamos de forma estructurada y descartar otros datos innecesarios.

4.3. Recopilación de fuente de datos

Para la recopilación de fuentes de datos, se recopila la información de la página web de Uniprot a través del uso de la tecnología de Web scraping. Debido a que el sitio web de Uniprot utiliza la carga dinámica para generar páginas, no podemos obtener ningún dato consultando su código fuente. Podemos usar un rastreo hacia atrás para consultar el objeto XMLHTTPRequest que la página ha creado y devuelto para localizar el URL del archivo Json que la solicitud de Ajax hizo. Se puede hacer consulta de paginación con Uniprot, por defecto es de 25 datos por página, podemos rastrear todos los datos a la vez, o podemos rastrear más de uno a la vez, el rastreo se realiza por lotes. La información de la fuente de datos solicitada por Uniprot se muestra en la figura 3:

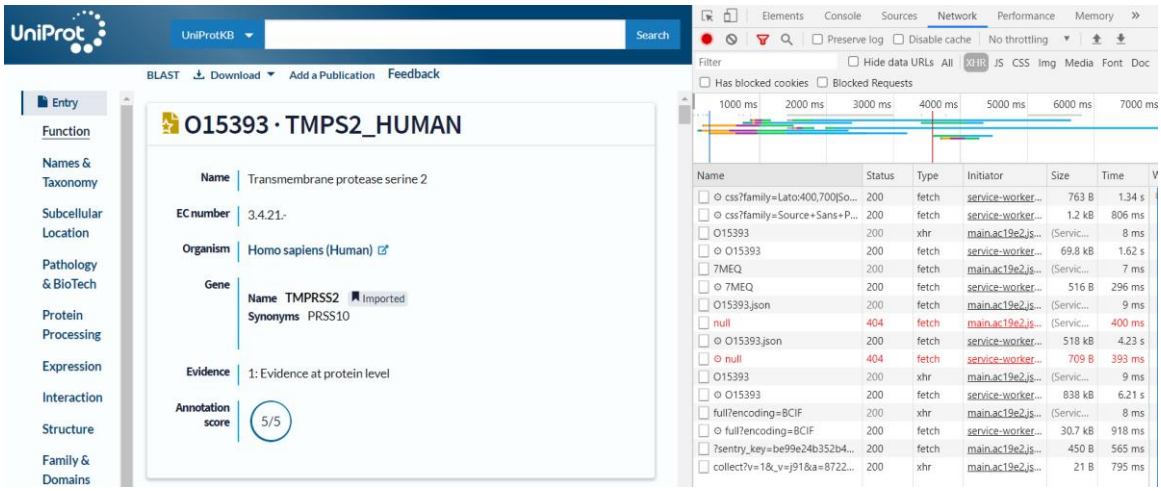

Figure 3 Información de la fuente de datos solicitada por Uniprot

El entorno de desarrollo de este trabajo es realizar un rastreo web basado en Java. Webmagic es su biblioteca de terceros, es potente y fácil de usar, puede ahorrar mucho trabajo, cumple totalmente con los requisitos. Se puede crear fácilmente un manejador de página para esa URL invocando Spider.create(new ListPageProcesser().addUrl(url)),y luego en el método Process de la clase ListPageProcesser, podemos procesar los datos de esa página directamente.

Primero, “limpiamos” las etiquetas HTML, obtenemos los datos de Json, y luego encontramos la lista de datos de proteínas que necesitamos en “results”. Después de obtener un identificador único para cada dato de proteína, cada dato de proteína se rastrea por separado. Finalmente, se crea la clase de entidad de datos imitando la estructura de datos de Uniprot, y los datos se almacenan en la memoria en forma de deserialización, para facilitar el siguiente procesamiento de los datos. Las clases de entidad se muestran en la figura 4.

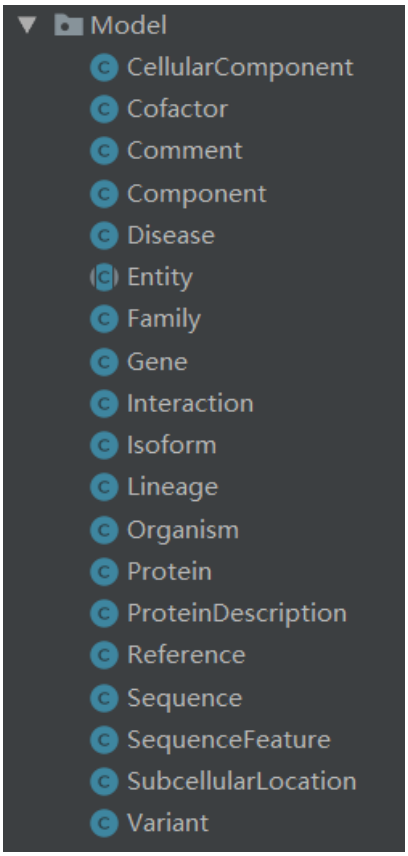

Figure 4 Clases de entidad

Debido a la característica de los datos semi-estructurados de Uniprot, tiene demasiados datos redundantes, necesitamos primero una "limpieza" manual para los datos estructurados, y realizamos por segunda vez para los datos incompletos, como por ejemplo los datos de Isoform, los datos de la proteína que reacciona con las otras proteínas y los datos de la Variante, por lo que necesitamos crear 5 procesadores para procesar los diferentes datos, como se muestra en la figura 5.

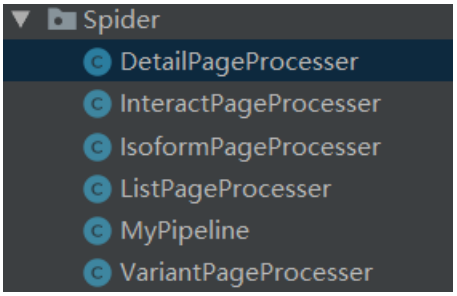

Figure 5 Procesadores para procesar los diferentes datos

Capítulo 5 Reestructuración de los datos de acuerdo con el Modelo Conceptual de Proteinas

El modelo conceptual de proteína se ha reconstruido en el artículo **Conceptual Modeling of Proteins Based on UniProt[1]**. En esta parte del contenido, seguiremos paso a paso el artículo para transferir los datos en Uniprot a la base de datos de gráficos de Neo4j.

El modelo conceptual completo de proteína se muestra en la figura 6 y el resultado de visualización de datos de tipo grafo se muestra en la figura 7:

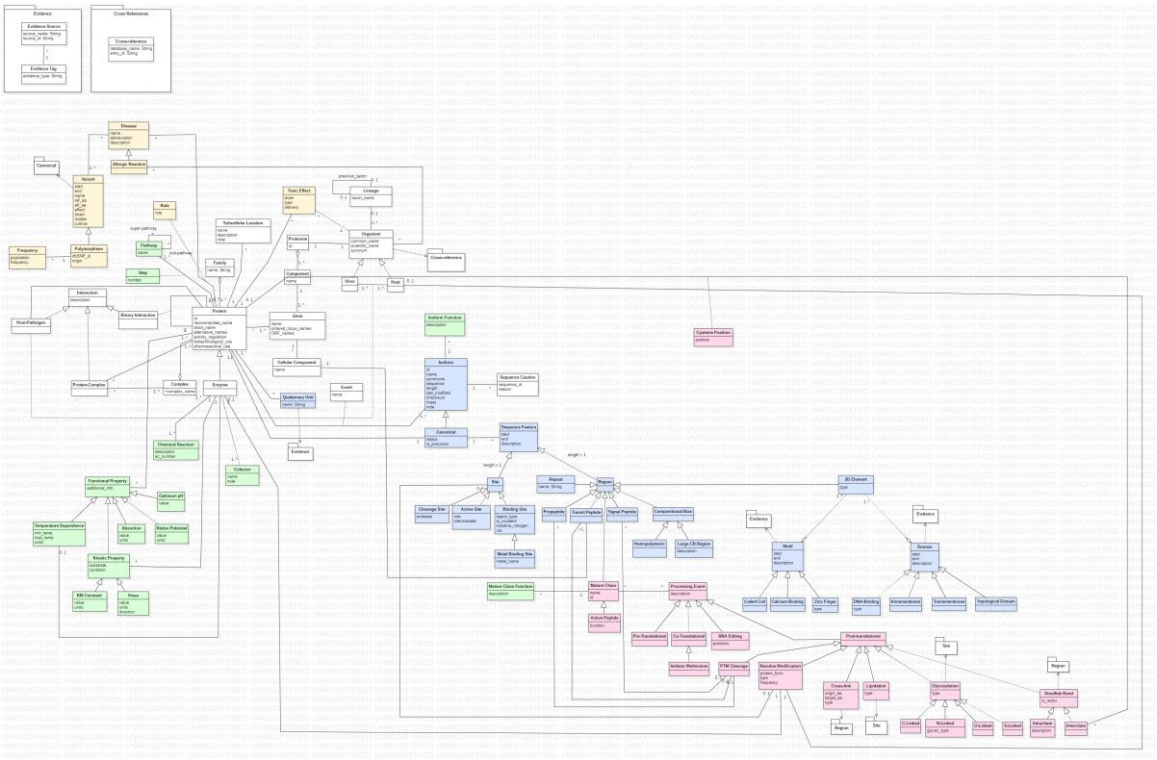

Figure 6 El modelo conceptual completo de proteína

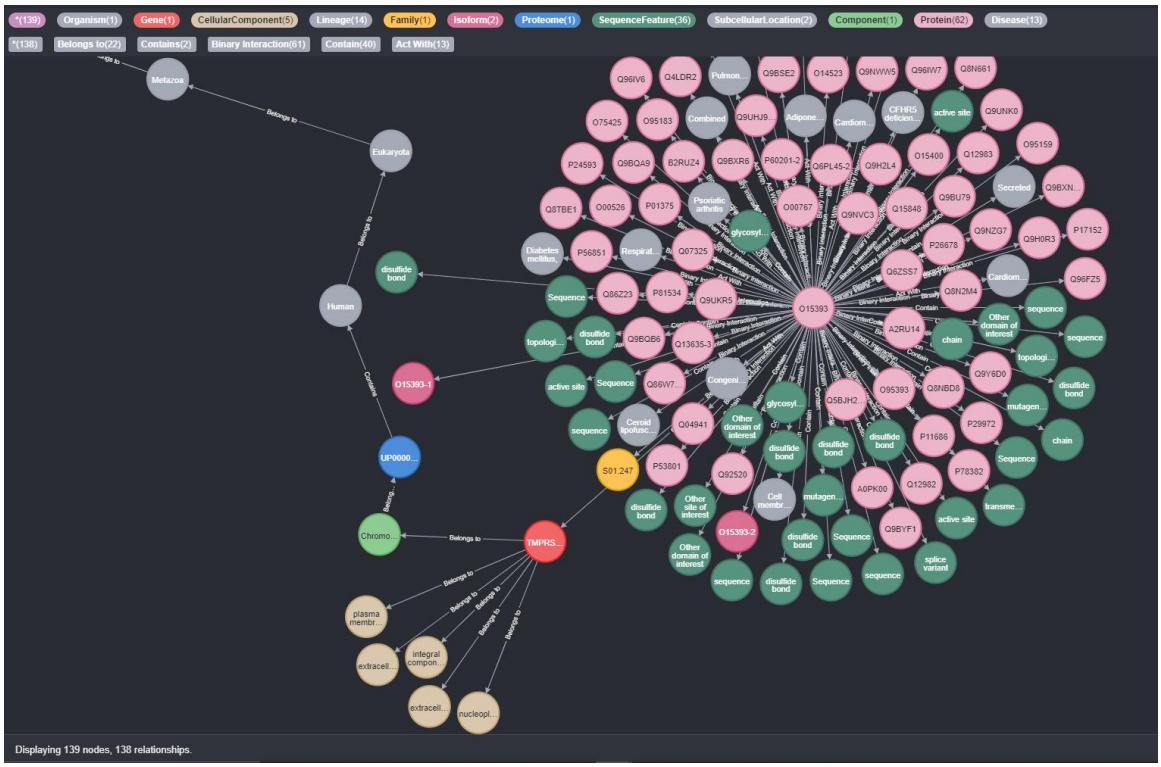

Figure 7 Resultado de visualización de datos de tipo grafo.

5.1. Reestructuración de Proteomes y Information Basica sobre Proteinas

Primero, reconstruimos la parte central de los datos de las proteínas (proteoma, proteína, gen, organismo). La relación entre ellos se puede expresar de la siguiente manera en el diagrama UML:

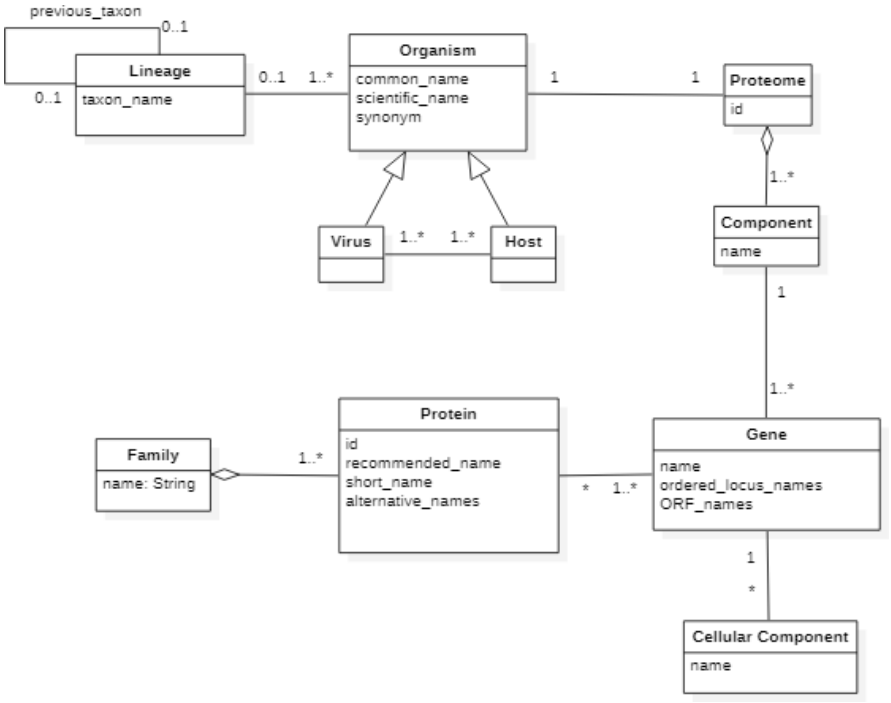

Figure 8 Conceptual model that represents the basic information about proteomes, genes, proteins, and organisms.

Según el modelo conceptual, los datos extraídos de Uniprot se almacenan en Neo4j, y el resultado se muestra en la figura 9:

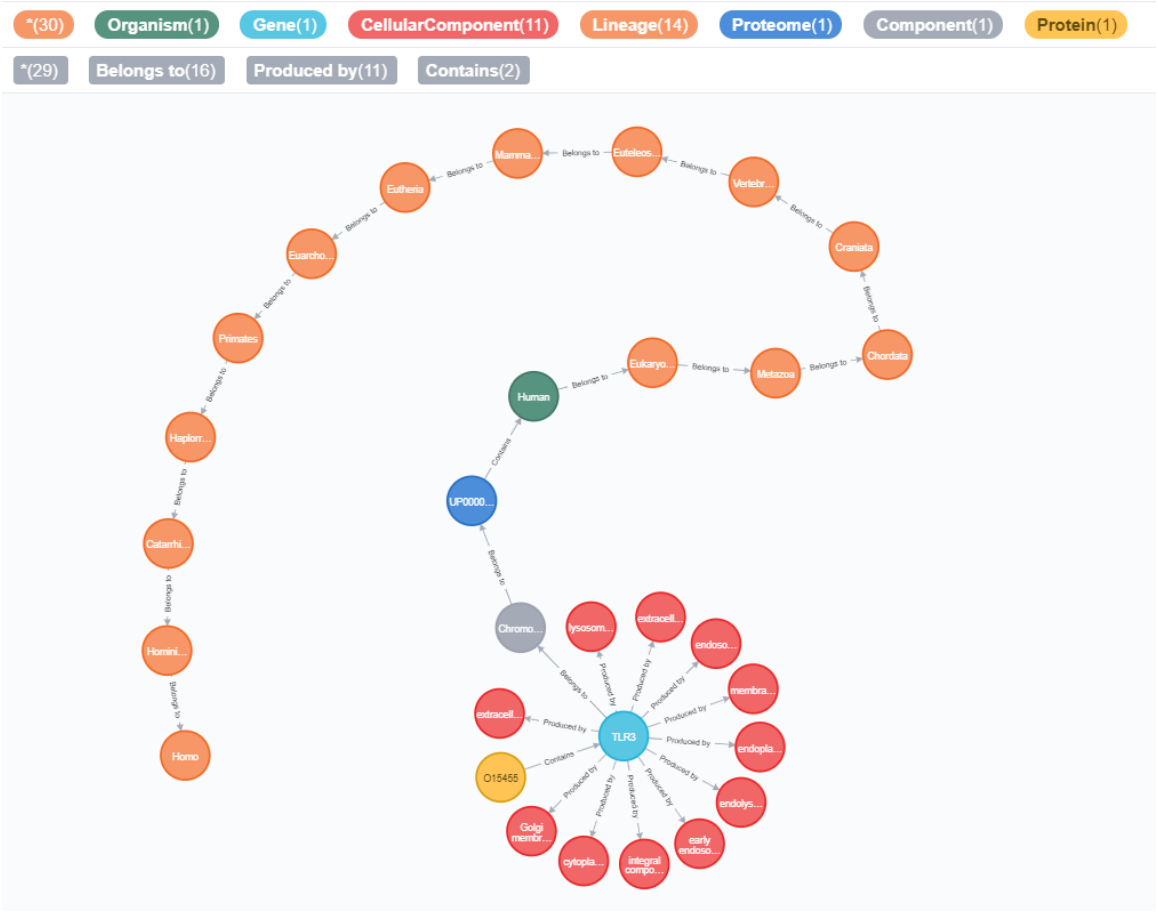

Figure 9 Resultado de la parte de información básica sobre proteínas

En Uniprot, los datos se almacenan en forma de Json y toda la información relacionada se almacena en un enorme archivo Json con cada proteína como núcleo.

Cada proteína puede estar codificada por uno o más genes, por lo que la relación entre proteínas y genes es una relación de muchos a muchos. En Uniprot, el gen de cada proteína es una lista guardada con *"genes"* como clave., donde cada gen contiene los campos *"geneName"* y *"synonyms"*. Como se muestra en la Figura 10:

```
"genes":[
        {
                "geneName":{
                        "evidences":[
                                {
                                        "evidenceCode":"ECO:0000312",
                                        "source":"HGNC",
                                        "id":"HGNC:11876"
                                }
                        ],
                        "value":"TMPRSS2"
                },
                "synonyms":[
                        {
                                "value":"PRSS10"
                        }
                ]
        }
],
```


Figure 10 Estructura de datos genéticos de Uniprot

Por tanto, solo se necesita expresarlo en forma de clase según el modelo conceptual como se muestra en la figura 11:

```
@NodeEntity
public class Gene extends Entity
{
    @Id
    public String geneName;
    public String synonyms;
    public List<String> orfNames;

    @Relationship(type = "Belongs to", direction = Relationship.OUTGOING)
    public Component component;
    @Relationship(type = "Produced by", direction = Relationship.OUTGOING)
    public List<CellularComponent> cellularComponents = new ArrayList<>();
```

Figure 11 Clase de entidad genética creada en base a un modelo conceptual

Compare la diferencia de nombre de atributo entre la fuente de datos de *Gene* de Uniprot y el modelo conceptual en forma de tabla, como se muestra en la tabla 1:

Comparación de los nombres de los atributos de las clases de proteínas y genes	
Nombres de atributo de Uniprot	Nombres de atributo del modelo

	conceptual
genes.geneName.value	Gene.geneName
genes.Synonyms.value	Gene.synonyms
recommendedName	Protein.recommendedName
shortName	Protein.shortName
alternativeNames	Protein.alternativeNames

tabla 1 Comparación de los nombres de los atributos de las clases de proteínas y genes

El organismo es un objeto almacenado con *"organism"* como clave, incluyendo *"scientificName", " commonName"* y la información de *"lineage"*. El linaje se almacena capa por capa del linaje al que el organismo pertenece al linaje superior en forma de una lista. Como se muestra en la Figura 12.


```
"organism":{
        "scientificName":"Homo  sapiens",
        "commonName":"Human",
        "taxonId":9606,
        "lineage":[
                "Eukaryota",
                "Metazoa",
                "Chordata",
                "Craniata",
                "Vertebrata",
                "Euteleostomi",
                "Mammalia",
                "Eutheria",
                "Euarchontoglires",
                "Primates",
                "Haplorrhini",
                "Catarrhini",
                "Hominidae",
                "Homo"
        ]
},
```


Figure 12 Estructura de datos de organism de Uniprot

En Uniprot, cada dato de proteína contiene una información de organismo, que obviamente es redundante. Solo necesitamos guardarla en Neo4j con *"scientificName"* como clave, de esta forma, el mismo organismo solo necesita ser guardada una vez en la base de datos. La clase de entidad *Organism* que creamos se muestra en la Figura 13.

```
@NodeEntity
public class Organism extends Entity
{
    @Id
    public String scientificName;
    public String commonName;

    @Relationship("Belongs to")
    public Lineage lineage;
```

Figure 13 La clase de entidad Organismo

Al comparar la fuente de datos Uniprot y el modelo conceptual en forma de tabla, la diferencia en los nombres de atributo de las clases *Organism* y *Lineage* se muestra en la tabla 2. El campo *"lineage"* se almacena en el campo *"organism"* en forma de matriz en Uniprot, pero cada *Lineage* en el modelo conceptual Ambos son un solo objeto de datos:

Comparación de los nombres de los atributos de las clases de Organism y Lineage	
Nombres de atributo de Uniprot	Nombres de atributo del modelo conceptual
organism.scientificName	Organism.scientificName
organism.commonName	Organism.commonName
organism.lineage	Lineage.name

tabla 2 Comparación de los nombres de los atributos de las clases de Organism y Lineage

5.2. Reestructuración de Interacciones entre proteínas

El modelo conceptual y los resultados de la consulta de esta parte se muestran en las figuras 14 y 15:

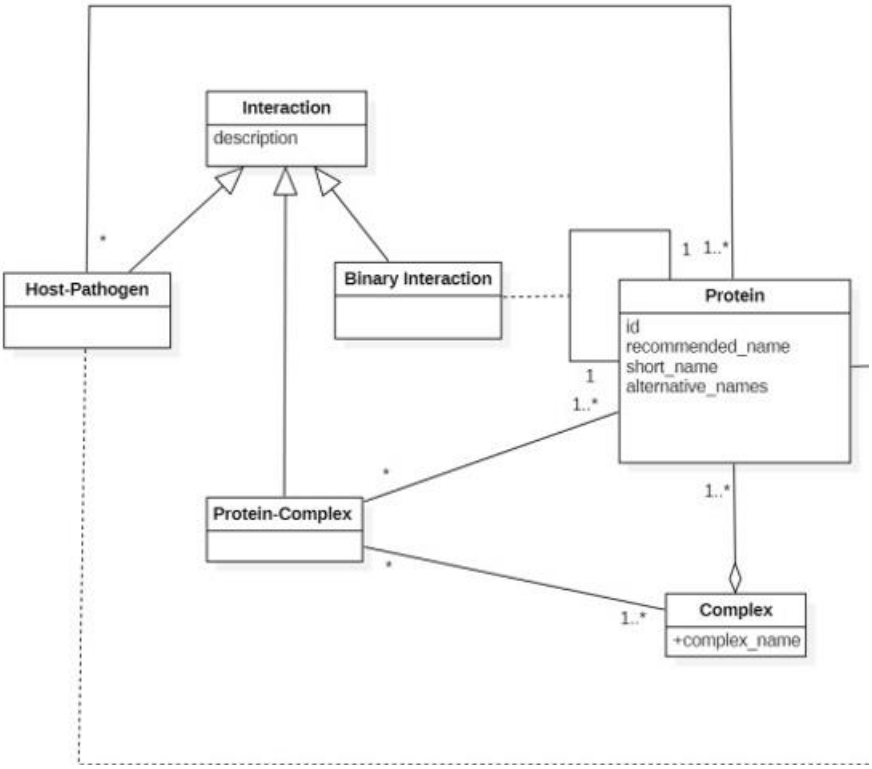

Figure 14 Conceptual model that represents the interactions among proteins.

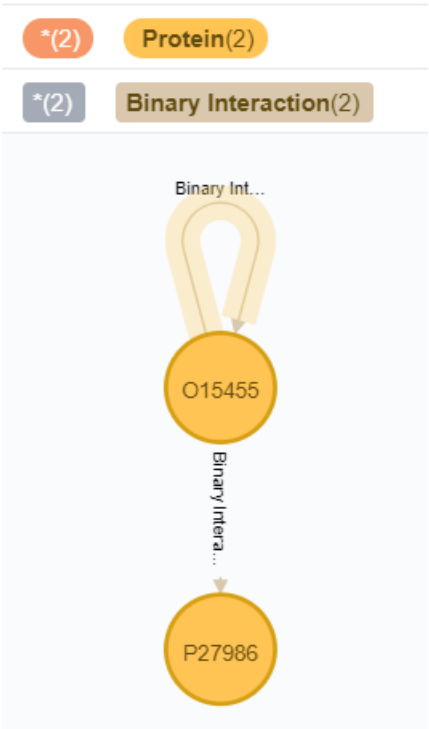

Figure 15 Resultado de la parte de Interacción entre proteínas

Uniprot almacena el contenido de esta parte en el campo *"comments"*. En este campo se almacenan muchos tipos de datos. Se distinguen por el campo *"commentType"*. Si *commentType* es *"INTERACTION"*, habrá un campo de *"interactions"* en la estructura que almacena brevemente la información de todas las demás proteínas que interactúan con la proteína en forma de lista. Como se muestra en la Figura 16.


```
"commentType":"INTERACTION",
"interactions":[
    {
        "interactantOne":{
            "uniProtkbAccession":"O15393",
            "intActId":"EBI-12549863"
        },
        "interactantTwo":{
            "uniProtkbAccession":"Q9BYF1",
            "geneName":"ACE2",
            "intActId":"EBI-7730807"
        },
        "numberOfExperiments":3,
        "organismDiffer":false
    },
```


Figure 16 Estructura de datos de interacciones de Uniprot

Esta información es demasiado breve para apoyar nuestra reconstrucción de esta parte del modelo conceptual, por lo que observamos todos los enlaces solicitados por la página de Uniprot. En estos enlaces, encontramos que solicitó una URL adicional específicamente responsable de obtener información de interacción de proteínas. Pero en este nuevo archivo Json, simplemente cambia la estructura de almacenamiento y el contenido almacenado todavía solo incluye la información de la proteína que interactúa. Como se muestra en la Figura 17.


```
"interactions":[
    {
        "accession1":"O15393",
        "accession2":"Q9BYF1",
        "gene":"ACE2",
        "interactor1":"EBI-12549863",
        "interactor2":"EBI-7730807",
        "organismDiffer":false,
        "experiments":3
    },
```


Figure 17 Estructura de datos de interacciones de Uniprot

Cuando almacenamos en la base de datos Neo4j, solo necesitamos agregar un campo en la clase *Protein* para representar la relación con otras proteínas. Como se muestra en la Figura 18.

```
@Relationship(type = "Binary Interaction", direction = Relationship.OUTGOING)
public List<Protein> interactions = new ArrayList<>();
```

Figure 18 representación de interacciones de proteínas

Posteriormente, se realizó un segundo *scraping* según el nombre de la proteína con la que reaccionó, y la información de la proteína también se almacenó en la base de datos de Neo4j. Si es necesario en investigaciones futuras, se puede rastrear de forma recursiva para formar una cadena de reacción para el almacenamiento.

La diferencia en los nombres de los atributos de la fuente de datos de *Interaction* de Uniprot y del modelo conceptual se compara en forma de tabla como se muestra en la tabla 3:

Comparación de nombres de atributos de la clase Interaction	
Nombres de atributo de Uniprot	Nombres de atributo del modelo conceptual
interactions.interactor1	Protein1.recommendName
interactions.interactor2	Protein2.recommendName

tabla 3 Comparación de nombres de atributos de la clase Interaction

5.3. Protein Structure

El modelo conceptual y los resultados de la consulta de esta parte se muestran en la figura 19 y 20:

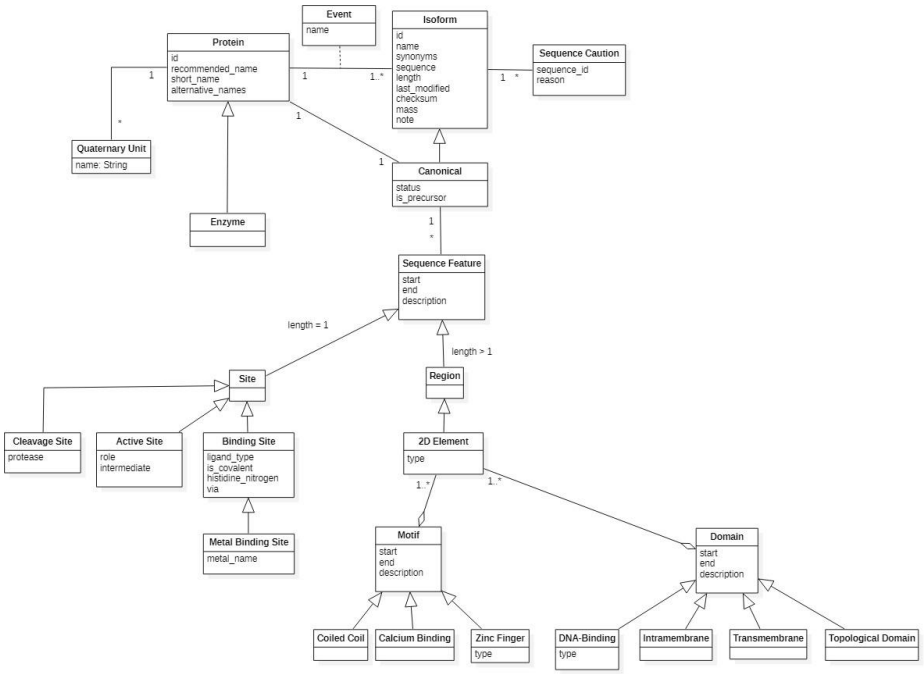

Figure 19 Conceptual model that represents the sequence features and the different elements that constitute the three-dimensional structure of the proteins.

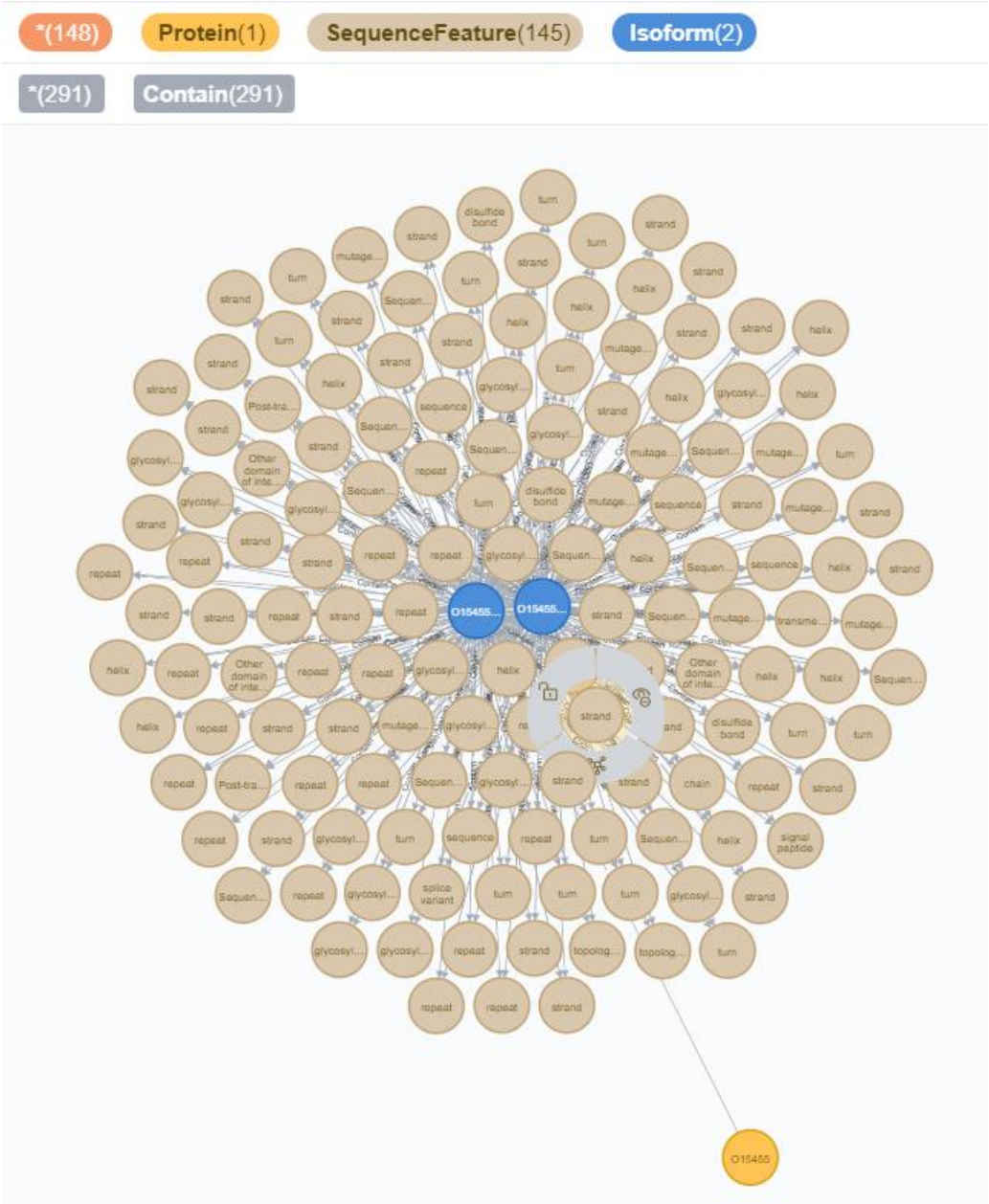

Figure 20 Resultados de la consulta de la parte Sequence features

Los datos de isoformas también se almacenan en el campo *"comments"*. Cuando su tipo de comentario es *"PRODUCTOS ALTERNATIVOS"*, habrá un campo de *"isoforms"*. A través del campo *"isoformIds"* de cada isoforma, necesitamos recuperar sus datos de proteínas, porque los datos de cada isoforma se almacenan en forma de proteína. Finalmente, obtuvimos la longitud, el peso y otros datos en el campo *"sequence"* de cada isoforma. Como se muestra en la Figura 21.

```
"sequence":{
        "value":"MALNSGSPPAIGPYYENHGYQPENPYPAQPTVVPTVYEVHPAQYYPSPVPQYAPRVLTQASNPVVCTQPKSPSGTVCT
SKTKKALCITLTLGTFLVGAALAAGLLWKFMGSKCSNSGIECDSSGTCINPSNWCDGVSHCPGGEDENRCVRLYGPNFILQVYSSQRKSWHPVCQDDWNEN
YGRAACRDMGYKNNFYSSQGIVDDSGSTSFMKLNTSAGNVDIYKKLYHSDACSSKAVVSLRCIACGVNLNSSRQSRIVGGESALPGAWPWQVSLHVQNVHV
CGGSIITPEWIVTAAHCVEKPLNNPWHWTAFAGILRQSFMFYGAGYQVEKVISHPNYDSKTKNNDIALMKLQKPLTFNDLVKPVCLPNPGMMLQPEQLCWI
SGWGATEEKGKTSEVLNAAKVLLIETQRCNSRYVYDNLITPAMICAGFLQGNVDSCQGDSGGPLVTSKNNIWWLIGDTSWGSGCAKAYRPGVYGNVMVFTD
WIYRQMRADG",
        "length":492,
        "molWeight":53859,
        "crc64":"C05B5531C8A311C7",
        "md5":"75A0BF8A517DF367DC18159AA37F84D3"
},
```


Figure 21 Estructura de datos de Sequence de Uniprot

Las *Sequence Cautions* de cada isoforma también se almacenan en el campo *"comments"*, y su *commentType* es *"SEQUENCE CAUTION"*. Como se muestra en la Figura 22.


```
{
        "commentType":"SEQUENCE CAUTION",
        "sequenceCautionType":"Miscellaneous discrepancy",
        "sequence":"AAD22395.1",
        "note":"Contaminating sequence. Sequence of unkno
t.",
        "evidences":[
                {
                        "evidenceCode":"ECO:0000305"
                }
        ]
},
```


Figure 22 Estructura de datos de Sequence caution de Uniprot

Sequence Feature se almacena en el campo *"features"*, tiene muchos campos redundantes, solo necesitamos extraer *"start"*, *"end"*, *" desceiption"* y *"type"*, donde el campo *"type"* expresa el tipo al que pertenece esta *Sequence Feature*. Como se muestra en la Figura 23.

```
"features":[
        {
                "type":"chain",
                "location":{
                        "start":{
                                "value":1,
                                "modifier":"EXACT"
                        },
                        "end":{
                                "value":880,
                                "modifier":"EXACT"
                        }
                },
                "description":"Nuclear pore complex protein Nup98",
                "featureId":"PRO_0000019929"
        },
```


Figure 23 Estructura de datos de Sequence features de Uniprot

Para crear una asociación entre *Isoform*, *Sequence Caution* y *Sequence Feature*, solo necesitamos agregar dos listas a la clase *Isoform*. Como se muestra en la Figura 24, 25 y 26.

```
@NodeEntity
public class Isoform extends Entity
{
    @Id
    public String id;
    public String name;
    public String sequence;
    public int sequenceLength;
    public int sequenceMass;
    public String status;

    @Relationship(type = "Contain", direction = Relationship.OUTGOING)
    public List<SequenceCaution> sequenceCautions = new ArrayList<>();

    @Relationship(type = "Contain")
    public List<SequenceFeature> sequenceFeatures = new ArrayList<>();
```

Figure 24 Clase de entidad de Isoform

```
@NodeEntity
public class SequenceCaution
{
    @Id
    public String sequenceId;
    public String reason;
```

Figure 25 Clase de entidad de Sequence caution

```
@NodeEntity
public class SequenceFeature extends Entity
{
    public String type;
    public int start;
    public int end;
    public String description;
    public String alternativeSequence;
```

Figure 26 Clase de entidad de Sequence feature

Compare las diferencias de nombre de atributo entre la fuente de datos Uniprot y el modelo conceptual relacionado con la clase *Isoform* en forma de tabla como se muestra en la tabla 4:

Comparación de nombre de atributo de clase Isoform, SequenceCaution, SequenceFeature	
Nombres de atributo de Uniprot	Nombres del atributo del modelo conceptual
sequence.value	Isoform.sequence
sequence.length	Isoform.sequenceLength
sequence.molWeight	Isoform.sequenceMass
comments.note	SequenceCaution.reason
features.location.start	SequenceFeature.start
features.location.end	SequenceFeature.end
features.location.description	SequenceFeature.description
features.alternativeSequence	SequenceFeature.AlternativeSequence
features.type	SequenceFeature.type

tabla 4 Comparación de nombre de atributo de clase Isoform, SequenceCaution, SequenceFeature

5.4. Protein Processing Events

Los datos de *PTM* también están en el campo de comentarios. Cuando el campo *"commentType"* es *"PTM",* podemos obtener información relevante del campo *"texts"*. Como se muestra en la figura 27.


```
⊟{
        "texts":⊟[
                ⊟{
                        "evidences":⊟[
                                ⊟{
                                        "evidenceCode":"ECO:0000269",
                                        "source":"PubMed",
                                        "id":"17114460"
                                }
                        ],
                        "value":"Phosphorylated at serine residues. Phosphorylation i
regions is required for cytoplasmic translocation followed by secretion (PubMed:1

                }
        ],
        "commentType":"PTM"
```


Figure 27 Estructura de datos de PTM de Uniprot

Lo almacenamos en la clase *Protein*.

5.5. The Function of Proteins

Function también se almacena en *"comments",* cuando *commentType = "FUNCTION",* la información relacionada se almacena en el campo *"texts"* en forma de texto. Como se muestra en la figura 28.

```
{
    "texts":[
        {
            "evidences":[
                {
                    "evidenceCode":"ECO:0000269",
                    "source":"Reference",
                    "id":"Ref.71"
                }
            ],
            "value":"
nfection) Critical for entry of human coronaviruses SARS-CoV and SARS-CoV-
as human coronavirus NL63/HCoV-NL63. Regulates the expression of the pro-
ACE2 and CTSL through chromatin modulation"
        }
    ],
    "commentType":"FUNCTION"
},
```


Figure 28 Estructura de datos de Function de Uniprot

5.6. Biological Pathways and Subcellular Locations

El modelo conceptual y los resultados de la consulta de esta parte se muestran en las figuras 29 y 30:

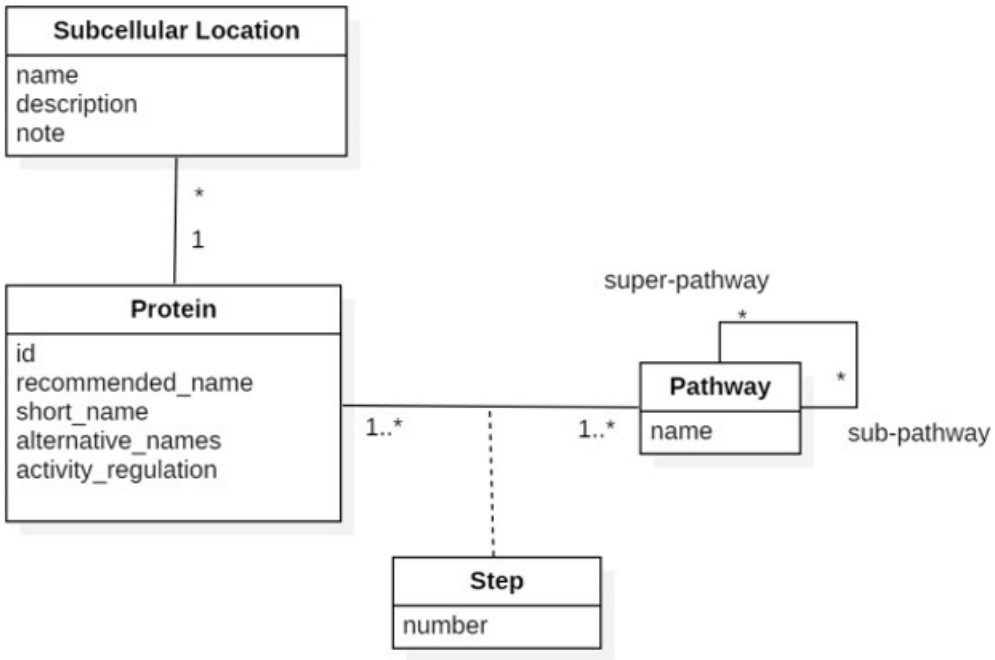

Figure 29 Conceptual model that describes the locations where proteins perform their function

Figure 30 Resultados de la consulta de la parte Subcellular location

La información de *Pathways* se almacena en el campo "uniProtKBCrossReferences", que almacena información que hace referencia a otras bases de datos. Conecte los datos de cada campo de *"database"* como *"Reactome"* en orden, que son las vías de la proteína. Como se muestra en la figura 31.

```
{
        "database":"Reactome",
        "id":"R-HSA-1236974",
        "properties":[
                {
                        "key":"PathwayName",
                        "value":"ER-Phagosome  pathway"
                }
        ]
},
{
        "database":"Reactome",
        "id":"R-HSA-140342",
        "properties":[
                {
                        "key":"PathwayName",
                        "value":"Apoptosis  induced  DNA  fragmentation"
                }
        ]
},
```


Figure 31 Estructura de datos de Pathway de Uniprot

Luego crear la relación entre ellos en Protein. Como muestra en la figura 32.

```
@Relationship(type = "Participate", direction = Relationship.OUTGOING)
public List<Pathway> pathways = new ArrayList<>();
```

Figure 32 Relación entre Pathway y Proteina

La información de *Subcellular locations* se almacena en el campo de *"comments"*. Cuando *commentType = "SUBCELLULAR LOCATION"*, podemos obtener *"location"*, *"description"* y *"note"* de todas las *Subcellular locations* en el campo *"subcellularLocations"*. Como se muestra en la figura 33.


```
"subcellularLocations":[
        {
                "location":{
```


Figure 33 Estructura de datos de Subcellular location de Uniprot

Luego crear la relación entre ellos en nuestra clase *Protein*. Como se muestra en la figura 34.

```
@Relationship(type = "Has evolved to")
public List<SubcellularLocation> subcellularLocations = new ArrayList<>();
```

Figure 34 Relación entre Subcellular location y Protein

Comparación de la fuente de datos Uniprot y las diferencias de nombre de atributo relacionadas con *Pathway* y *SubcellularLocation* del modelo conceptual en forma de tabla, como se muestra en la tabla 5:

Comparación de los nombres de los atributos de las clases Pathway y SubcellularLocation	
Nombres de atributo de Uniprot	Nombres de atributo del modelo conceptual
uniProtKBCrossReferences.properties.value	Pathway.name
comments.subcellularLocations.location	SubcellularLocation.location
comments.subcellularLocations.topology	SubcellularLocation.topology
comments.subcellularLocations.note	SubcellularLocation.note

tabla 5 Comparación de los nombres de los atributos de las clases Pathway y SubcellularLocation

5.7. Involvement in Disease: Variants and Polymorphisms

El modelo conceptual de esta parte se muestra en la figura 35 y el resultado de consulta se muestra en la figura 36.

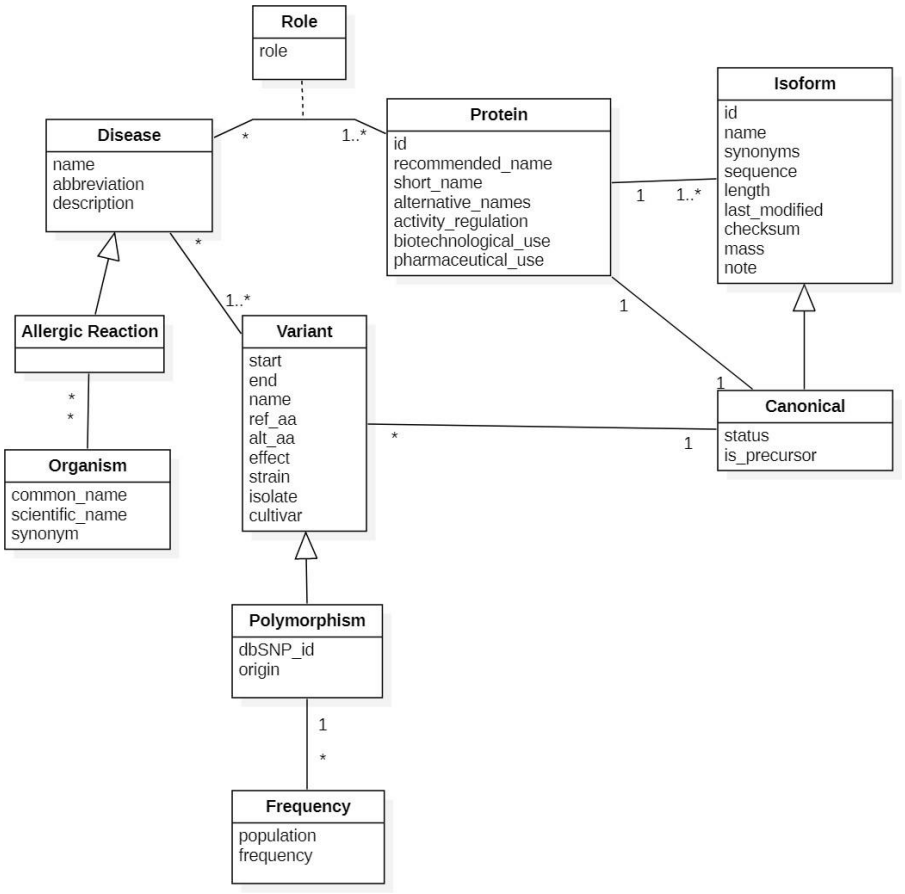

Figure 35 Conceptual model that describes the variants that may occur in the protein sequence

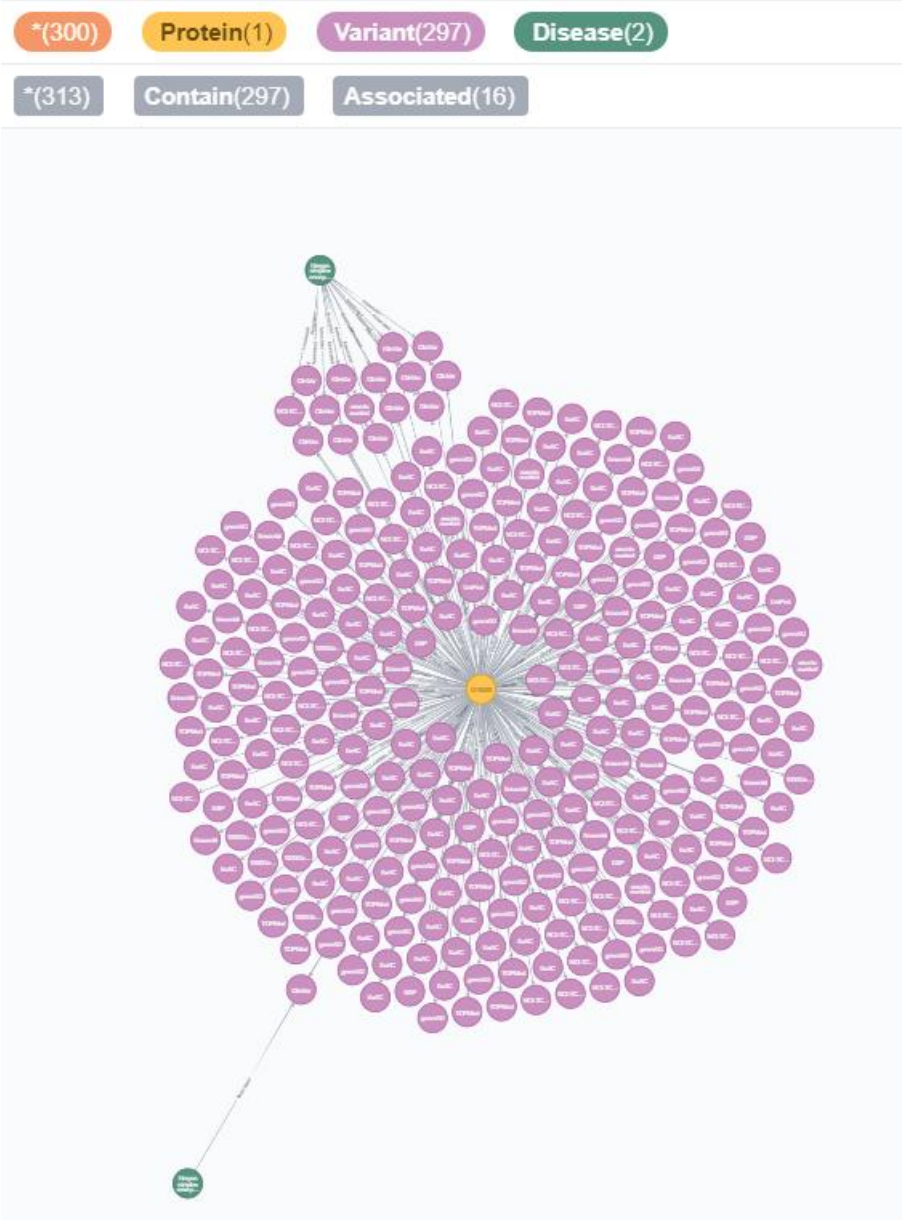

Figure 36 Resultado de consulta de la parte Variant

En la fuente de datos de Uniprot, los datos relacionados con *Variant* se obtienen solicitando otra URL. Su estructura de fuente de datos se muestra en la figura 37:

```
{
        "type":"VARIANT",
        "ftId":"VAR_084539",
        "alternativeSequence":"R",
        "begin":"74",
        "end":"74",
        "xrefs":[
                {
                        "name":"UniProt",
                        "id":"VAR_084539",
                        "url":"https://uniprot.org/annotation/VAR_084539"
                }
        ],
        "evidences":[
```


Figure 37 Estructura de datos de Variant de Uniprot

No todas las variantes tienen información relacionada con *Disease*, si existe, se almacenará en el campo *"association"*. Como se muestra en la figura 38.


```
"association":[
        {
                "name":"Herpes simplex encephalitis 1 (IIAE1)",
                "dbReferences":[
                        {
                                "name":"MIM",
                                "id":"610551",
                                "url":"https://www.omim.org/entry/610551"
                        }
                ],
```


Figure 38 Estructura de datos de Disease de Uniprot

Las clases de entidad que creamos en base al modelo conceptual son las figuras 39 y 40:

```java
public class Variant extends Entity
{
    @Relationship(type = "Contain", direction = Relationship.INCOMING)
    public Protein protein;
    @Relationship(type = "Associated", direction = Relationship.INCOMING)
    public List<Disease> diseases = new ArrayList<>();

    public int begin;
    public int end;
    public String name;
    public String consequenceType;
    public String wildType;
    public String mutatedType;
}
```

Figure 39 Clase de entidad Variant

```java
@NodeEntity
public class Disease extends Entity
{
    @Id
    public String name;
    public String abbreviation;
    public String description;

    @Relationship(type = "Associated", direction = Relationship.INCOMING)
    public Protein protein;
}
```

Figure 40 Clase de entidad Disease

Compare los nombres de los atributos de la fuente de datos Uniprot y el modelo conceptual relacionado con las clases *Variant* y *Disease* en forma de tabla, como se muestra en la tabla 6:

Comparación de nombres de atributos de clases de Variant y Disease	
Nombres de atributo de Uniprot	Nombres del atributo del modelo conceptual
begin	Variant.begin
end	Variant.end
name	Variant.name
consequenceType	Variant.consequenceType
wildType	Variant.wildType
mutatedType	Variant.mutatedType
association.name	Disease.name

description	Disease.description

tabla 6 Comparación de nombres de atributos de clases de Variant y Disease

5.8. Almacenamiento de base de datos orientada a grafos

Se almacena los datos utilizando el marco de Neo4j OGM, que pueden ser filtrados y correlacionados por medio de anotaciones. En el estudio de **Ryan Norton[11]** se dice que Neo4J OGM (Object-Graph Mapping) es una biblioteca Java pura que puede conservar objetos de dominio anotados en la base de datos de gráficos de Neo4J (2017). **Fabien Chevalier[6]** por su parte manifiesta que esta herramienta proporciona lectura y/o escritura fácil para objetos Java hacia o desde bases de datos gráficas. Neo4J OGM usa un enfoque centrado en el código, donde las anotaciones almacenan la información que utiliza el marco para saber cómo se debe serializar el objeto (2016). Esta librería ha sido elegida por el conjunto de tecnologías montadas en el presente trabajo.

En cuanto al campo inútil rastreado, se puede añadir la anotación @Transient para informar al sistema de que el campo no necesita ser almacenado; Para los datos duplicados, se pueden evitar las adiciones repetidas utilizando la anotación @Id, que incluye, pero no se limita a la palabra clave ÚNICA en la sentencia Cypher; Para los objetos correlacionados, se puede añadir la anotación @Relationship para establecer la relación entre los objetos. Finalmente, cada objeto de la proteína se almacena, y todos los objetos internos asociados se almacenarán en la base de datos.

A continuación, analizamos más detalladamente las etiquetas presentadas, explicamos y justificamos su uso dentro del proyecto. Los ejemplos de estas anotaciones pueden observarse en la figura 41.

Figure 41 Ejemplo de la etiquetación de elementos de código.

En esta figura se puede ver los campos anotados con las etiquetas dichas anteriormente. La clase también se etiqueta con *@NodeEntity* para enseñar a la librería de Neo4j que esta clase pertenece a la base de datos y que trabajaremos sobre esta. Según la documentación oficial de Neo4j los objetos POJO anotados con esta etiqueta se representarán como nodos en el gráfico. Cada nodo y relación persistente en el gráfico debe tener una identificación. Neo4j-OGM usa esto para identificar y reconectar la entidad al gráfico en la memoria. El identificador puede ser una identificación principal o una identificación gráfica nativa (2021).

El campo *primaryAccession* tiene la etiqueta *@Id* que se refiere a la identificación principal. La identificación principal es aquella que pone el usuario sobre los datos necesarios dentro de código. Sin embargo, existe otro tipo de etiquetación de objetos. Se refiere a la identificación nativa, que usa la herramienta Neo4j para marcar los datos generados por la base de datos de Neo4j cuando un nodo o relación se guarda por primera vez.

Junto con la etiqueta de identificación se puede usar opcionalmente otra etiqueta *@GeneratedValue*. Según la documentación oficial de la herramienta Neo4J este marcador pertenece a la identificación nativa. Básicamente sirve para indicar a la ejecución de código la identificación del gráfico nativo. Esta identificación se asigna automáticamente al guardar la entidad en el gráfico y el código de usuario nunca debe asignarle un valor, si no el proyecto provoca errores.

Por su parte el campo *secondaryAccessions* tiene la etiqueta *@Transient*. Esta etiqueta según la documentación oficial sirve para indicar a la ejecución que los valores puestos están exentos de persistencia, es decir, no deben ser almacenados en la BBDD. Básicamente esto sirve para no cargar la base de datos con los datos que no tienen nada que ver con la información necesaria. Por ejemplo, si tenemos una variable técnica que

sirve para activar los interruptores, o una variable que guarda los valores u objetos para pasarlo después a otras partes de código, la presencia de estas variables dentro de la base de datos puede ser maliciosa por varias razones: estas variables pueden descubrir el funcionamiento de código para la gente que no debería verlo (clientes), puede provocar errores por ser valores inesperados a la hora de procesar los datos y simplemente ensucian la BBDD. Aunque esta etiqueta es opcional, en el presente trabajo su uso es justificado por la minimización de posibles problemas que pueden aparecer durante el desarrollo.

La variable genes dispone de la etiqueta *@Relationship*. Esta etiqueta sirve para especificar el tipo de relación para una variable deseada y su dirección. El tipo es lo que se pone dentro de la base de datos indicando qué relación tiene el objeto. Esto sirve básicamente para poder diferenciar visualmente las relaciones entre objetos usando la aplicación de escritorio para visualizar la BBDD. La dirección sirve para indicar la jerarquía de los objetos y entrelazarlo entre sí. Si la relación es "*OUTGOING*" (saliente) indica que el objeto que tiene esta etiqueta es el objeto madre que está relacionado con los objetos hijos. La ruta de objetos en este caso empieza desde madre y termina en uno de sus hijos. Si la relación es "*INCOMING*" (entrante) esto indica que el objeto de la etiqueta es hijo que está relacionado con el objeto (o varios objetos) madre. También existe la dirección "*UNDIRECTED*" (indirecto) que indica que el objeto, aunque está en la BBDD, no tiene relaciones con otros registros de la base de datos. Este tipo de dirección no está en el trabajo presente por la carencia de la necesidad de aplicarla sobre los datos de estudio nuestro.

5.9. La estructura de código.

Nuestro proyecto tiene la estructura de 3 diferentes directorios teniendo diferentes clases para diversas fases de la ejecución de código y la clase **Main** para lanzar el proyecto en la ejecución. Estos directorios son:

1. Modelo
2. Neo4j
3. Web scraping

Modelo

Este directorio tiene todas las data clases que luego se convertirán en los datos dentro de la BBDD de grafos. Como se puede observar en la figura anterior que nos enseña la clase *Protein* que pertenece al directorio Modelo, estas clases usan las etiquetas proporcionadas por Neo4j para ajustar los datos antes de pasarlos en la BBDD. Básicamente los valores en estas clases y sus etiquetas sirven como los campos futuros que tendremos dentro de la BBDD. Como se verá posteriormente, a la hora de describir los datos obtenidos por el navegador de Neo4j, el nombre de clase da el nombre al conjunto de nodos. Es decir, si los datos analizados se generan usando esta clase (*Protein*), los nodos procesados serán de tipo que se llama igual que esta clase (*Protein*).

Neo4j

Este directorio tiene las clases abstractas que se usan para construir los datos de la data clases del directorio anterior. Estas clases sirven para aplicar los métodos de acceder a los datos, organizarlos y pasarlos a la BBDD y, además, sirven para lanzar la sesión de conexión con esta misma base de datos. Se observa en la figura de estructura del proyecto (figura 43), que existe una clase abstracta genérica *GenericDAO* (figura 42). Aquella clase sirve para facilitar el desarrollo de sistema permitiendo a otras clases heredar de esta para conseguir mejor legibilidad de código, simplificar su mantenimiento y subir el nivel de la abstracción de los elementos de este código.

```
public abstract class GenericDAO<T extends Entity> {

    protected static Session session;

    protected GenericDAO()
    {
        if(session == null)
        {
            ConfigurationSource properties = new ClasspathConfigurationSource( propertiesFileName: "ogm.properties");
            org.neo4j.ogm.config.Configuration configuration = new org.neo4j.ogm.config.Configuration.Builder(properties).build();
            SessionFactory sessionFactory = new SessionFactory(configuration, ...packages: "Model");
            session = sessionFactory.openSession();
        }
    }

    protected abstract Class<T> getEntityType();

    public void delete(Long id) { session.delete(session.load(getEntityType(), id)); }

    public void createOrUpdate(T entity) { session.save(entity); }

    public long filterCount(Iterable<Filter> filters){
        return session.count(getEntityType(), filters);
    }

    public T find(Long id) { return session.load(getEntityType(), id); }

    public Collection<T> find(Filters filters) {
        Collection<T> results = session.loadAll(getEntityType(), filters);
        return results;
    }
```

Figure 42 Ejemplo de la clase genérica GenericDAO.

Web scraping

Esta carpeta contiene dos tipos de clases: procesadores de página y una clase Pipeline que usamos luego para lanzar un *web scraping* (véase el apartado **3.1.** donde se describen las tecnologías tipo *web scraping*). El objetivo de las clases de procesadores es contener la lógica de la *web scraping*, según cual la herramienta puede procesar los datos, sacados desde el sitio web UniProt. La clase de Pipeline nos permite configurar un Pipeline customizado que utilizará la *web scraping* para lanzar los hilos sacando los datos deseados.

Por su parte la clase **Main** contiene unas líneas de código que lanzan la clase create() de la librería *web scraping* usando el procesador de páginas y el pipeline customizado en diferentes hilos aplicando los conceptos de la programación asíncrona.

En este proyecto se usa la tecnología **Maven de The Apache Software Foundation[12]** para adquirir las librerías y herramientas necesarias para el desarrollo de código. Los creadores describen su tecnología como una herramienta de comprensión y gestión de proyectos de software. Basado en el concepto de un modelo de objetos de proyecto (POM), Maven puede administrar el *build* de un proyecto, su

reporting y la documentación a partir de una pieza central de información (Porter, s.f.). Para configurar las tecnologías de proyecto, se cambia el archivo de Maven pom.xml pasando allí la información de dependencias requeridas en el formato XML.

La estructura completa de código se puede observar en la figura siguiente (figura 43):

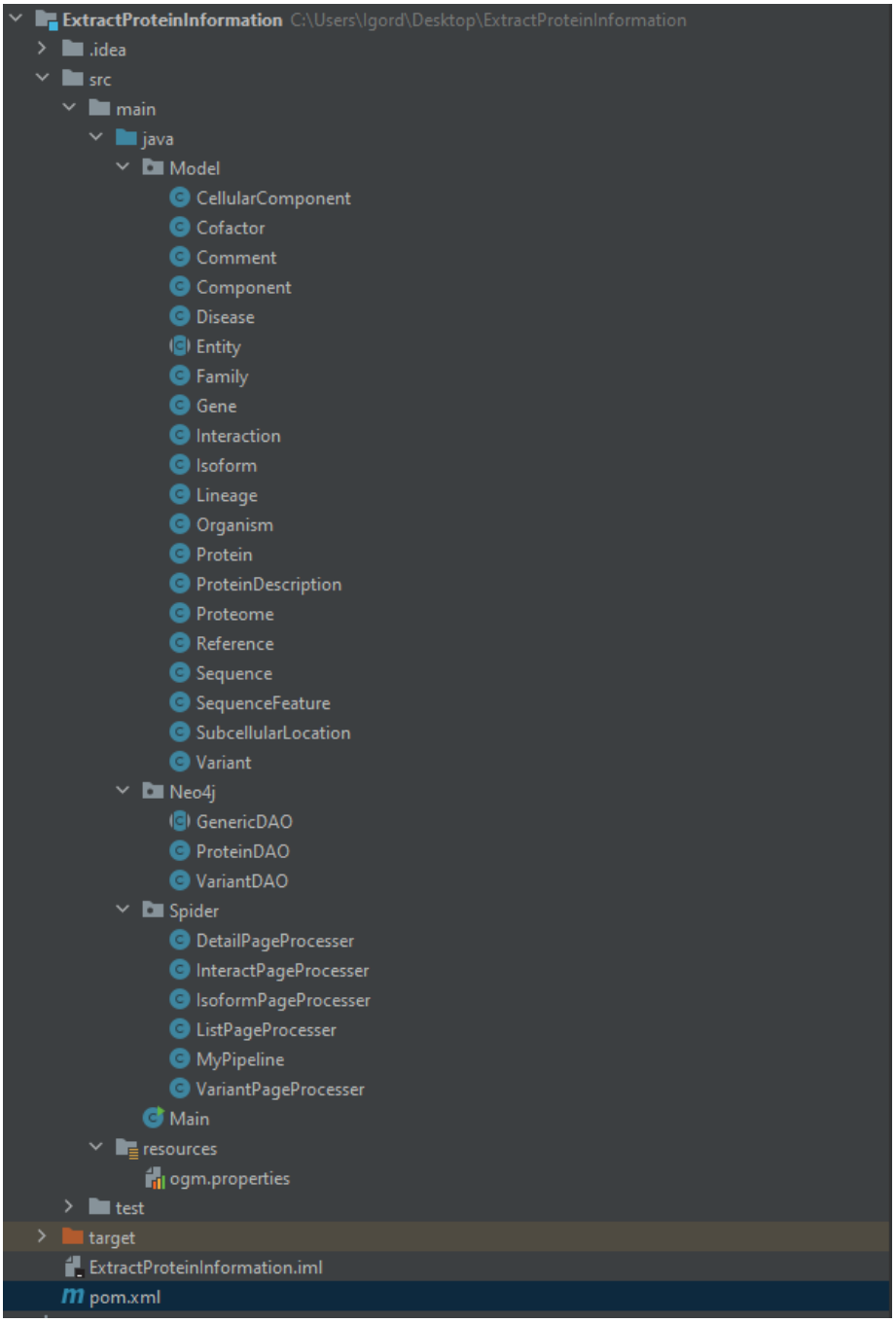

Figure 43 La estructura del proyecto.

5.10. Preparación de sistema y sus herramientas.

Antes de empezar a ejecutar las consultas en la Base de datos se debe lanzar la ejecución de código y el servidor para la base de datos. El servidor se pone en marcha desde la aplicación de escritorio Neo4j Desktop® que sirve para visualizar los datos de las BBDD basados en grafos en diferentes formatos: es posible presentar el contenido de la base de datos como un grafo donde se observan todas las relaciones y direcciones entre entidades, una tabla que es básicamente la presentación de datos en el formato JSON si tienen los datos por dentro, o de formato tabla Excel si las peticiones son especificadas y la salida de ejecución tiene los objetos definidos sin llevar los parámetros adicionales. A pesar de esto, hay un formato de texto, que también dispone de datos en el formato JSON/tabla donde todas las entradas son básicamente *strings* y un formato de código, donde los datos se revelan como un conjunto de objetos con sus propios argumentos y parámetros. El último tipo de la presentación de datos es lo que se puede observar desde la siguiente figura (figura 44):

Figure 44 Representación de datos tipo código.

Además, se debe tomar en cuenta que a veces las peticiones a la base de datos son bastante específicos y concretos que a veces no permite a la Neo4j generar el grafo. En este caso se puede seguir usando el resto de los formatos que propone la herramienta de visualización para trabajar con los datos necesarios.

El servidor de la base de datos lo desplegamos en local, es decir, no usamos la conexión ajena sino conectamos el servidor al localhost. La aplicación de Neo4j (junto con la librería homónima) nos ofrece diferentes tipos de conexión. La que nos interesa se establece por el controlador Bolt que usa el protocolo homónimo de comunicación Bolt. En la documentación de creadores del protocolo Neo Technology, Inc. se dice, que el protocolo de red Bolt es un protocolo cliente-servidor ligero y altamente eficiente, diseñado para aplicaciones de bases de datos. El protocolo está orientado a declaraciones, lo que permite al cliente enviar peticiones, cada una de las cuales consta de un solo *string* y un conjunto de parámetros escritos. El servidor responde a cada solicitud con un mensaje de resultado y un flujo opcional de registros de resultados.

Después de conectarse satisfactoriamente al servidor en local, la base de datos se pone como activa y es posible lanzar la consola de consultas sobre la base de datos, como se puede observar en la figura 45.

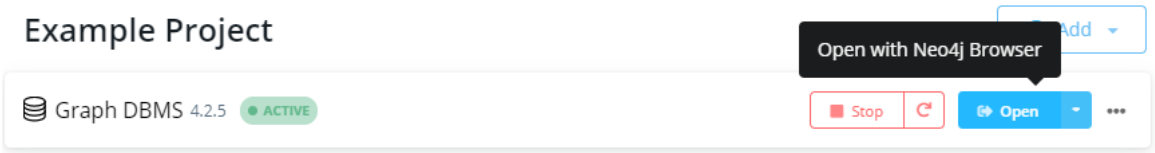

Figure 45 Ejemplo del servidor satisfactoriamente lanzado.

Cuando el servidor ya está en marcha y la consola ya está disponible para usar, se necesita ejecutar el código de proyecto para recibir los datos desde el sitio web de UniProt. Primero, la ejecución lanza los hilos para mapear los valores, referencias, relaciones y direcciones para nuestra base de datos de la manera correcta y necesaria (figura 46):

```
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - mapping related entity
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - visiting: Model.SequenceFeature@7c76180
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - Model.SequenceFeature@7c76180, has not changed
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - mapping references declared by: Model.SequenceFeature@7c76180, currently at depth 2
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - trying to map relationship between Model.Protein@1335f4d6 and Model.SequenceFeature@7c76180
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - context-add: (74)-[-1675:Contain]->(60)
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - Model.Protein@1335f4d6: mapping reference type: Binary Interaction
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - context-del: (74)<-[:Binary Interaction]-()
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - Model.Protein@1335f4d6: mapping reference type: Belongs to
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - context-del: (74)-[:Belongs to]->()
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - linking to entity Model.Family@5147af27 in one direction
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - mapping related entity
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - visiting: Model.Family@5147af27
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - Model.Family@5147af27, has not changed
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - mapping references declared by: Model.Family@5147af27, currently at depth 2
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - trying to map relationship between Model.Protein@1335f4d6 and Model.Family@5147af27
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - context-add: (74)-[-1676:Belongs to]->(21)
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - Model.Protein@1335f4d6: mapping reference type: Contains
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - context-del: (74)-[:Contains]->()
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - linking to entity Model.Gene@44a252a5 in one direction
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - mapping related entity
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - visiting: Model.Gene@44a252a5
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - Model.Gene@44a252a5, has not changed
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - mapping references declared by: Model.Gene@44a252a5, currently at depth 2
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - Model.Gene@44a252a5: mapping reference type: Belongs to
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - context-del: (1)-[:Belongs to]->()
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - Model.Gene@44a252a5: mapping reference type: Belongs to
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - context-del: (1)-[:Belongs to]->()
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - trying to map relationship between Model.Protein@1335f4d6 and Model.Gene@44a252a5
19:33:12.029 [pool-3-thread-1] DEBUG org.neo4j.ogm.context.EntityGraphMapper - context-add: (74)-[-1677:Contains]->(1)
19:33:12.030 [pool-3-thread-1] DEBUG org.neo4j.ogm.drivers.bolt.transaction.BoltTransaction - No current transaction, starting a new one
19:33:12.032 [pool-3-thread-1] DEBUG org.neo4j.ogm.drivers.bolt.transaction.BoltTransaction - Native transaction: org.neo4j.driver.internal.InternalTransaction@177a2c1e
19:33:12.032 [pool-3-thread-1] DEBUG org.neo4j.ogm.drivers.bolt.transaction.BoltTransaction - Committing native transaction: org.neo4j.driver.internal.InternalTransaction@177a2c1e
19:33:12.034 [pool-3-thread-1] DEBUG org.neo4j.ogm.transaction.Transaction - Thread 38: Commit transaction extent: 0
19:33:12.034 [pool-3-thread-1] DEBUG org.neo4j.ogm.transaction.Transaction - Thread 38: Committed
```

Figure 46 Salida de la ejecución de hilos mapeando las entradas de la BBDD.

Después de que la ejecución mapea los valores necesarios, pasa a lanzar los hilos para descargar los datos del sitio web que nos interesa usando las peticiones tipo URI indicando en la cadena los objetos y parámetros deseados. La herramienta que se usa para adquirir los datos desde el sitio web es una librería *web scraping* de Codecraft. **Heaton[7]** en su artículo subraya que *web scrapings* son programas que pueden visitar sitios web y seguir hipervínculos. Al usar uno, puede trazar rápidamente todas las páginas contenidas en un sitio web (2021). Esto justifica el uso de dicho recurso en nuestro trabajo porque lo usamos para descargar los datos para analizar desde el sitio web concreto. La salida de la ejecución del *web scraping* usado en el proyecto está en la figura 47:

```
20:06:02.272 [pool-2-thread-1] INFO us.codecraft.webmagic.Spider - Spider 127.0.0.1 closed! 1 pages downloaded.
get page: https://www.ebi.ac.uk/uniprot/api/covid-19/uniprotkb/accession/O15393
20:06:07.286 [pool-1-thread-1] INFO us.codecraft.webmagic.Spider - Spider 127.0.0.1 closed! 1 pages downloaded.
get page: https://www.ebi.ac.uk/uniprot/api/covid-19/uniprotkb/search?facets=reviewed%2Cmodel_organism%2Cother_organism%2Cproteins_
20:06:12.291 [main] INFO us.codecraft.webmagic.Spider - Spider 127.0.0.1 closed! 1 pages downloaded.
20:06:12.295 [Finalizer] DEBUG org.apache.http.impl.conn.PoolingHttpClientConnectionManager - Connection manager is shutting down
20:06:12.296 [Finalizer] DEBUG org.apache.http.impl.conn.DefaultManagedHttpClientConnection - http-outgoing-2: Close connection
20:06:12.296 [Finalizer] DEBUG org.apache.http.impl.conn.PoolingHttpClientConnectionManager - Connection manager shut down
20:06:12.296 [Finalizer] DEBUG org.apache.http.impl.conn.PoolingHttpClientConnectionManager - Connection manager is shutting down
20:06:12.296 [Finalizer] DEBUG org.apache.http.impl.conn.DefaultManagedHttpClientConnection - http-outgoing-3: Close connection
20:06:12.296 [Finalizer] DEBUG org.apache.http.impl.conn.PoolingHttpClientConnectionManager - Connection manager shut down
```

Figure 47 Ejemplo de ejecución de web scraping y terminación de las conexiones al sitio web.

Además, en la figura anterior se puede observar que después de descargar los datos, la ejecución para las conexiones al sitio web y quita los managers de conexiones.

A partir de este momento la base de datos está disponible para trabajar y analizar los datos que lleva. La ejecución de código Java ya no se necesita por cumplir el requisito deseado: adquirir los datos del recurso UniProt y meterlos en el servidor de Neo4j previamente configurado y puesto en marcha.

Capítulo 6 Ejecución de consultas sobre la base de datos

6.1. Visualización y explicación de datos

Para aplicar las consultas necesarias sobre nuestra base de datos basada en grafos, se debería usar el lenguaje de consultas adecuado para la herramienta de visualización de datos. En caso de este trabajo, el lenguaje este es Cypher. Esta herramienta de consultas es de Neo4j desarrollada especialmente para consultar los datos basados en grafos. En la documentación oficial de Neo4j se manifiesta que Cypher es el lenguaje de consulta de gráficos de Neo4j que permite a los usuarios almacenar y recuperar datos de la base de datos de grafos. Neo4j quería que la consulta de datos de gráficos fuera fácil de aprender, comprender y usar para todos, pero también incorporar el poder y la funcionalidad de otros lenguajes estándar de acceso a datos. Esto es lo que Cypher pretende lograr. (s.f.)

Cuando el servidor está lanzado y los datos se han subido a la BBDD, aplicamos la petición usando **Cypher match(n) return n** para comprobar que la base de datos funciona correctamente devolviendo los datos de búsqueda. Esta petición devuelve el primer dato encontrado y en nuestro caso sirve solo para asegurarse del funcionamiento del servidor. A continuación, se puede observar la figura con el ejemplo de grafo sacado aplicando esta petición (figura 48):

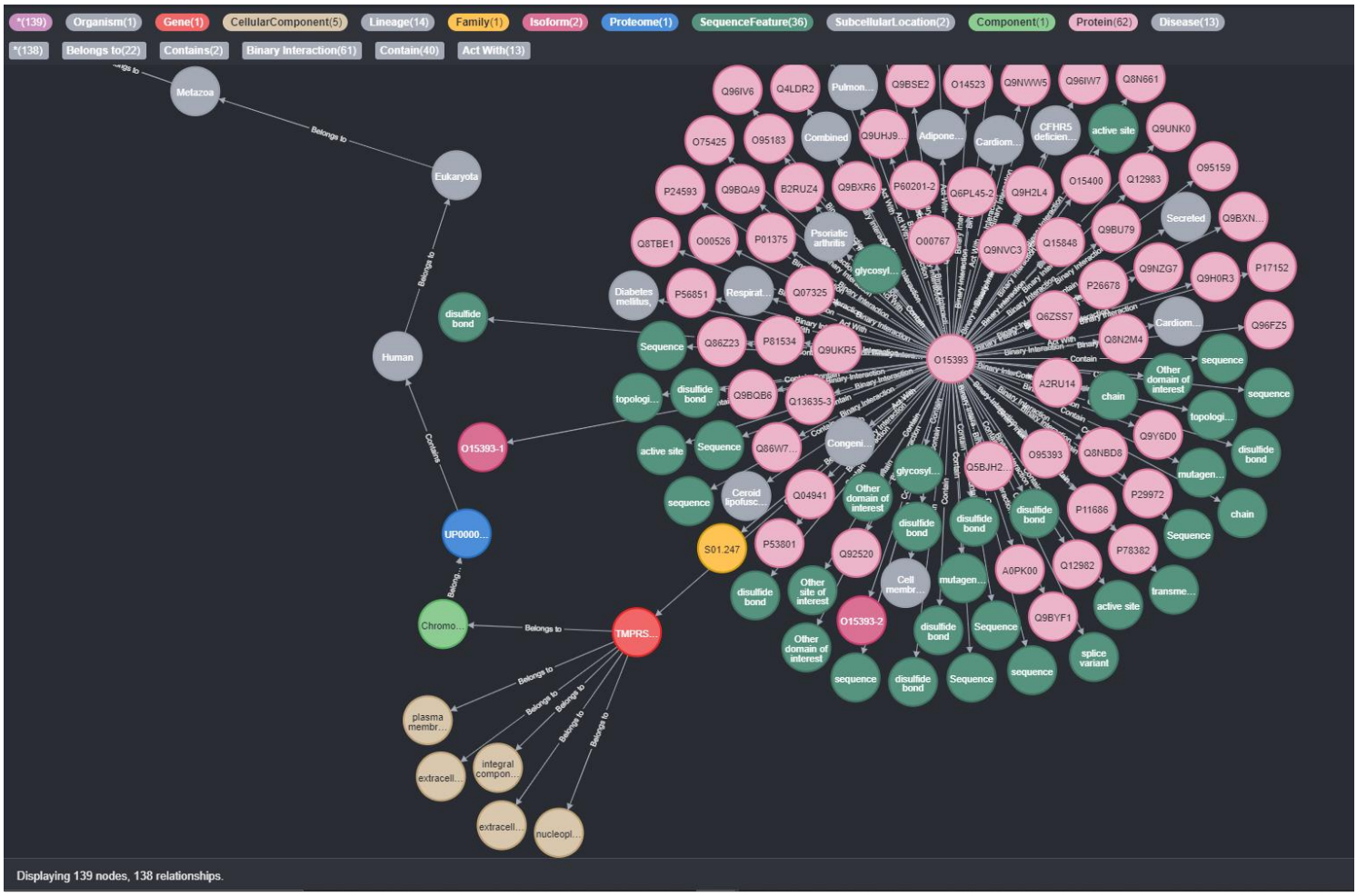

Figure 48 Ejemplo de visualización de datos de tipo grafo.

En el ejemplo sacado anteriormente se puede observar diferentes nodos de diferentes tipos de objetos. Por ejemplo, si analizamos el gene TMPRSS2 (la burbuja roja clara a la izquierda de la burbuja amarilla) veremos que tiene dos tipos de relaciones *contains* y *belongs to*. Además, este objeto tiene los dos tipos de direcciones: entrante (INCOMING) y saliente (OUTGOING). ¿Cómo se observa esto? La flecha que apunta al dicho gene tiene el tipo dirección INCOMING porque está relacionado como nodo hijo con el nodo madre (O15393, la burbuja rosada en el centro de nodos) y su tipo de relación es *contains*, que significa que el nodo madre contiene al nodo de gene. Al mismo tiempo el nodo analizado es el nodo madre, porque tiene la relación de la dirección saliente ya que la flecha sale de este nodo y apunta al nodo Chromosome 21 (la burbuja verde), aunque el tipo de esta relación es *belongs_to* (pertenece a).

6.2. Consultas aplicadas sobre la BBDD

Después de preparar el sistema, descargar los datos necesarios, lanzarlos en el servidor de Neo4j y comprobar el funcionamiento correcto del proyecto, se puede pasar para plantear las preguntas que nos interesen en el contexto de este estudio. Se verán más

adelante la facilidad de uso de la base de datos orientada a grafos y la profundidad de análisis que ofrece el uso de esta tecnología, aplicando las peticiones de Cypher.

Las preguntas claves a la base de datos orientada a grafos de este estudio son las siguientes:

1. ¿Qué proteínas del virus le ayudan a entrar en las células de la persona infectada?

2. ¿Qué proteínas del infectado interaccionan con las proteínas del virus?

3. ¿En qué tejidos se contienen estas proteínas de la persona infectada?

A continuación, se analizarán las peticiones hechas a la base de datos, se analizará el output producido por estas peticiones y se enseñarán los ejemplos de la salida de la base de datos. **¿Qué proteínas de una persona infectada ayudan al virus entrar en las células de esta persona?**

Para responder a la primera pregunta preparamos la petición a la base de datos. La petición que nos interesa es la siguiente:

```
mat
                protein_name, s.location as subcellular
```

Para preparar esta petición usamos la palabra reservada **match** que busca los resultados que coinciden con el argumento, pasado dentro de paréntesis. En nuestro caso es **n:Protein**, donde n es la variable asignada para los resultados de tipo proteína. Luego, dentro de corchetes rectangulares indicamos el tipo de relación que nos interesa, que es **Contain** (contiene) y la dirección de flecha que enseña a Cypher que nodos de proteínas son los nodos madre, es decir, son los nodos salientes (desde estos nodos sale la relación y entra en nodos de ubicación subcelular). Luego indicamos los nodos hijos que son **s:SubcellularLocation**, donde asignamos el valor obtenido a la variable s y después, retornamos los valores sacados dando los nombres adecuados para las columnas de la tabla.

neo4j$ match(n:Protein)-[:Contain] → (s:SubcellularLocation) return n.recommendedNam…

Table | Text | Code

	protein_name	subcellular
25	"ATP-dependent RNA helicase DDX1"	"Cytoplasm, cytosol"
26	"ATP-dependent RNA helicase DDX1"	"Cytoplasmic granule"
27	"ATP-dependent RNA helicase DDX1"	"Mitochondrion"
28	"1-phosphatidylinositol 3-phosphate 5-kinase"	"Early endosome membrane"
29	"1-phosphatidylinositol 3-phosphate 5-kinase"	"Cytoplasmic vesicle, phagosome membrane"
30	"1-phosphatidylinositol 3-phosphate 5-kinase"	"Late endosome membrane"

Figure 49 Salida de ejecución de petición realizada sobre las proteínas y ubicación celular.

Los valores sacados perfectamente responden a la pregunta planteada. Las proteínas (la columna izquierda) tipo, por ejemplo, “ATP-dependent RNA helicase DDX1” ayudan el virus SARS-CoV-2 entrar en las células en la ubicación subcelular tipo “Cytoplasm, cytosol”.

¿Qué proteínas del virus interaccionan con las proteínas de un infectado?

```
match(p1:Protein)-[r:`Binary Interaction`]->(p2:Protein) return p1 as protein_source, r
                                    as interact_with, p2 as protein_target
```

Esta pregunta requiere la siguiente petición para sacar los datos necesarios:

Igual que en la petición anterior se usa la palabra **match** para buscar los resultados que coinciden con el argumento **p1** de tipo **Protein**. La relación necesaria la buscamos pasando a los corchetes argumento **r** de tipo **Binary Interaction**. En este caso usamos la variable para aquel argumento para ser capaz de manejar el resultado de la manera necesaria (en caso de esta petición asignando el valor a la relación permite cambiar el nombre del resultado después de obtener la salida de ejecución de esta petición. Es decir, si el valor tiene la variable asignada, luego se puede cambiar los atributos de esta variable. En nuestro caso r as interact_with cambia el nombre del resultado de la relación que tiene el tipo Binary Interaction). Luego se indica la dirección con la ayuda de flecha, marcando **p1:Protein** como el nodo madre y

p2:Protein como el nodo hijo. Al ejecutar la petición, la base de datos devuelve el grafo (a pesar de la tabla, texto y código), que será analizado más adelante. La base de datos devuelve el grafo (a pesar de la tabla, texto y código), que será analizado más

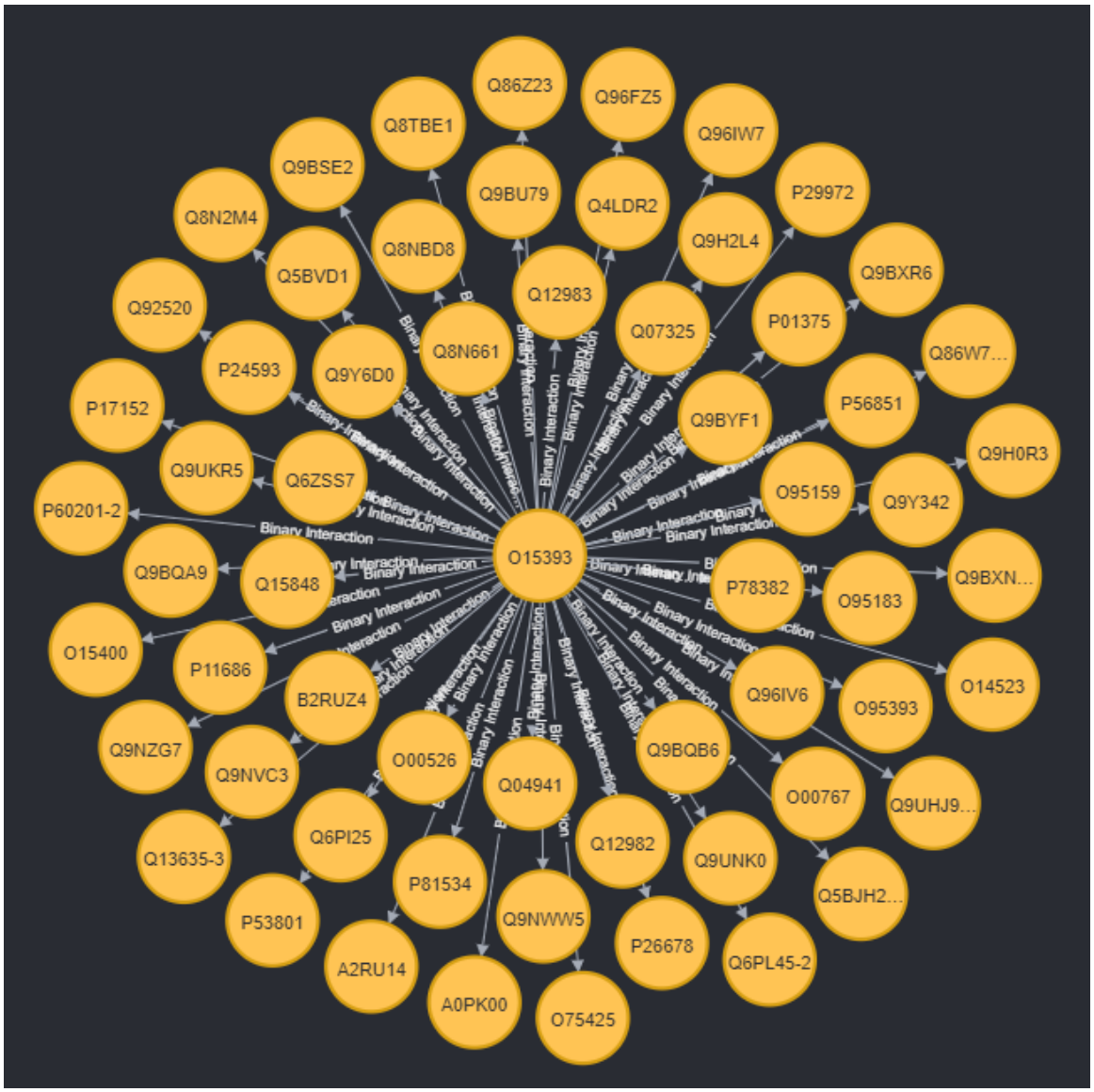

Figure 50 El grafo como salida de ejecución de la petición proteínas e interacción binaria.

adelante.

En la salida de tipo tabla aparecen 61 resultados de proteína de virus que interaccionan con la proteína humana O15393 (UniProt Consortium, 2006), en el grafo es el nodo central. Todas las proteínas de virus que aparecen en la figura 50, conectados con la proteína O15393 por la interacción binaria interaccionan con esta proteína humana.

```
match(p:Protein)-[:Source]->(c:CellularComponent) return p.recommendedName as
protein_name, c.name as component
```

¿En qué tejidos se contienen las proteínas de una persona infectada?

La siguiente petición sirve para responder a la pregunta planteada:

Lo mismo que en las peticiones anteriores, en esta usamos la palabra reservada match para encontrar la variable p del tipo *Protein* (nodo madre) que tiene la relación de tipo *Source* con la variable c de tipo *CellularComponent* (nodo hijo). La petición devuelve los datos que pueden observarse en la figura 51:

En la salida de la ejecución obtenemos la respuesta para la pregunta que nos interesa. Por ejemplo, las proteínas tipo "*Prohibitin*" están en los tejidos tipo "*Mitochondrial inner membrane*". Esto se comprueba por varios autores de diferentes revistas médicas (**Ande, Xu, Mishra, 2017[5]**).

Figure 51 La salida de tipo tabla de ejecución de petición sobre los tejidos y proteinas.

Conclusión

De acuerdo con las necesidades reales de la investigación biológica, este artículo realiza el almacenamiento y la consulta visual de datos de proteínas basados en la base de datos Neo4j y logra los objetivos y las preguntas de investigación planteadas en el artículo anterior.

Objetivo 1: Implementar una base de datos que permita almacenar de forma estructurada el conocimiento disponible sobre las proteínas.

En el Capítulo 5, seguimos el artículo **Conceptual Modeling of Proteins Based on UniProt[1]** para crear las clases de entidades de datos del modelo conceptual de

proteínas paso a paso y almacenar estas clases de entidades en la base de datos de Neo4j a través del mapeo OGM.

Pregunta de investigación 1: ¿Qué información es necesaria para representar de forma precisa el conocimiento actual sobre las proteínas?

Esta Pregunta de investigación se resuelve utilizando el modelo conceptual basado en UniprotKB que ya ha sido desarrollado.

Pregunta de investigación 2: ¿Qué tecnología es la más adecuada para almacenar la información?

Los resultados sacados, (es decir, la salida de la ejecución de peticiones) son correctos, esto se comprueba en varias fuentes científicas citadas en este trabajo. Se puede reconocer que el funcionamiento del sistema es correcto y esperado. Aunque otros tipos de bases de datos como bases de datos de documentos (MongoID), bases de datos orientados en objetos (Realm) y etc. también pueden representar los datos sacados correctamente, la mayoría no ofrecen la visualización clara de los resultados de las peticiones, no permiten recolectar los datos en diferentes formatos tipo tabla, texto, código. También, según la ingeniera de Neo4j aquellas bases de datos están diseñadas para atravesar datos almacenados muy rápidamente y recuperar resultados en milisegundos, que algunos paradigmas no pueden ofrecer por ser las tecnologías antiguas o por tener algunas limitaciones específicas. ( )

Además, hay que tener en cuenta que el uso de una base de datos basada en grafos está justificado solo en contexto de datos masivos, complejos, que tienen muchas relaciones entre sí, como en caso de este estudio. El desarrollo de una BBDD así es costoso y requiere muchos conocimientos técnicos, aunque el resultado es más que satisfactorio. Si los datos son pequeños, de tipos primitivos (valor-clave) y no tienen muchas relaciones, se recomienda encontrar otra solución de desarrollo de base de datos. Sin embargo, las posibilidades que puede ofrecer una BBDD de grafos son muy útiles, beneficiosos y prácticos para los estudios científicos.

Objetivo 2: Poblar la base de datos con la información disponible sobre las proteínas relacionadas con el SARS-CoV-2 y las proteínas del huésped con las que interacciona.

La base de datos de gráficos es diferente de la base de datos relacional, la base de datos relacional necesita definir la estructura de los datos antes de almacenar los datos específicos, cuando se almacenan los datos, debe almacenarse de acuerdo con la estructura definida. La base de datos de gráficos se almacena directamente en la estructura de los datos almacenados cuando se almacenan datos específicos. Entonces resolvimos el Objetivo 1 y el 2 juntos.

Pregunta de investigación 3: ¿Dónde se puede obtener información sobre las proteínas que constituyen en SARS-CoV-2?

UniprotKB

Pregunta de investigación 4: ¿Cómo se puede almacenar dicha información en la base de datos implementada?

Primero extrajimos los datos de Uniprot. Luego, de acuerdo con el artículo **Conceptual Modeling of Proteins Based on UniProt** 错误!未找到引用源。, se crearon las clases de entidades de datos relacionados con proteínas y las relaciones entre las entidades. Luego, los datos extraídos se limpiaron y almacenaron en la clase de entidad correspondiente, y todas las asignaciones de entidades relacionadas se almacenaron en la base de datos de Neo4j con los datos de proteínas como cuerpo principal.

Objetivo 3: Evaluar la utilidad de la base de datos y de la información almacenada para la extracción de conocimiento relevante sobre la COVID-19.

Pregunta de investigación 5: ¿Permite la base de datos almacenar el conocimiento actual generado sobre el proteoma del SARS-CoV-2?

Pregunta de investigación 6: ¿Es la base de datos útil para determinar los mecanismos que llevan al desarrollo de la COVID-19?

Estos objetivos y preguntas se han verificado en la sección 6.2.

Los principales contenidos del trabajo finalizado son los siguientes:

1. Comprenda cómo la base de datos de Uniprot almacena datos de proteínas, se usa web scraping para descargar información de proteínas y limpiar la información.

2. De acuerdo con el modelo conceptual de la proteína, las clases de entidad de los datos de la proteína y las relaciones entre las entidades se crean en el programa Java, y la información de la proteína se almacena en la memoria.

3. Se crea un mapeo de almacenamiento entre la clase de entidad y Neo4j, y almacene los datos en la memoria en la base de datos de Neo4j.

4. Se utiliza el Neo4j Browser para consultar visualmente la base de datos y verificar la usabilidad de la base de datos con Neo4j-OGM.

Este trabajo proporciona una nueva base de datos de proteínas que está altamente visualizada y expresa intuitivamente la relación entre los datos relacionados con las proteínas. Es conveniente que los futuros investigadores utilicen la base de datos para el análisis de datos de proteínas.

Bibliografía

[1]. León Palacio, A.; Pastor López, O. (2020). Conceptual Modeling of Proteins Based on UniProt. http://hdl.handle.net/10251/145619

[2]. Webmagic, a simple but scalable crawler framework. http://webmagic.io/docs/en/

[3]. Neo4j – OGM Object Graph Mapper https://neo4j.com/developer/neo4j-ogm/

[4]. S. Jouili and V. Vansteenberghe, "An Empirical Comparison of Graph Databases," 2013 International Conference on Social Computing, 2013, pp. 708-715, doi: 10.1109/SocialCom.2013.106. https://ieeexplore.ieee.org/document/6693403

[5]. Ande, S. R., Xu, Y. X. Z., & Mishra, S. (2017). Prohibitin: a potential therapeutic target in tyrosine kinase signaling. Signal Transduction and Targeted Therapy, 2(1). https://doi.org/10.1038/sigtrans.2017.59

[6]. Chevalier, F. (2016). European Semantic Web Conference. AutoRDF - using OWL as an object graph mapping (OGM) specification language, 151–155. https://link.springer.com/content/pdf/10.1007%2F978-3-319-47602-5.pdf

[7]. Heaton, J. (2021, 25 marzo). Programming a Spider in Java. Developer.Com. Recuperado 14 de abril de 2021, de https://www.developer.com/java/programming-a-spider-in-java/#:%7E:text=Introduction,Web%20sites%20and%20follow%20hyperlinks.&text=Java%20has%20built%2Din%20support,an%20HTML%20parser%20built%20in.

[8]. Neo Technology, Inc. (s. f.). Bolt Protocol. Bolt Protocol. Recuperado 13 de abril de 2021, de https://boltprotocol.org/

[9]. Neo4j, & Reif, J. (2021, 15 marzo). How Do You Know If a Graph Database Solves the Problem? Neo4j Graph Database Platform. https://neo4j.com/developer-blog/how-do-you-know-if-a-graph-database-solves-the-problem/

[10]. Neo4j. (s. f.). Cypher Query Language - Developer Guides. Neo4j Graph Database Platform. Recuperado 15 de abril de 2021, de https://neo4j.com/developer/cypher/

[11]. Norton, R. D. (2017). Dynamic Database Schemas and Multi-Paradigm Persistence Transformations.

[12]. Porter, B. (s. f.). Maven – Welcome to Apache Maven. Apache Maven Project. https://maven.apache.org/

[13]. Tutorial - OGM Library. (2021). Neo4j Graph Database Platform. https://Neo4j.com/docs/ogm-manual/current/tutorial/

[14]. UniProt Consortium. (2006, 3 octubre). TMPRSS2 - Transmembrane protease serine 2 precursor - Homo sapiens (Human) - TMPRSS2 gene & protein. https://www.uniprot.org/uniprot/O15393