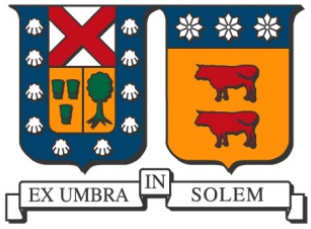
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“LENGUAJE DE CONSULTA VISUAL PARA LA BASE DE DATOS OPEN SOURCE MILLENNIUMDB”

RODRIGO JAVIER DÍAZ DEL VALLE

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Carlos Buil
Profesor Correferente: Jose Luis Martí

Enero - 2024

DEDICATORIA

A quienes me han apoyado en el transcurso de este trabajo y me han soportado hasta el final

AGRADECIMIENTOS

Agradezco a mi familia por aguantarme durante este tiempo e impulsarme a terminar el trabajo.

También agradezco a mi profesor guía, quien me brindó el último impulso y se preocupó por mí durante la realización de este trabajo.

RESUMEN

Resumen— Con la nueva base de datos de grafos, MillenniumDB, en desarrollo por la IMFD, es necesario la implementación de una interfaz que facilite su uso tanto para usuarios nuevos como expertos. Para lograr esto, se utiliza la interfaz RDFExplorer, la cual permite la visualización y consultas de grafos de Wikidata. La implementación se logra cambiando la BD SPARQL de RDFExplorer por MillenniumDB, modificando y añadiendo nuevas funcionalidades que permiten realizar nuevas consultas a través de esta BD.

Palabras Clave— Bases de Datos de Grafos, MillenniumDB, Interfaz de usuario, Integración de RDFExplorer.

ABSTRACT

***Abstract*—** With the new graph database, MillenniumDB, under development by the IMFD, it's essential to implement an interface to facilitate its use for both novice and expert users. To achieve this, the interface RDFExplorer is used, which allows the vizualisation and queries of graphs from Wikidata. The implementation is achieved changing RDFExplorer DB SPARQL for MillenniumDB, modifying and adding new functionalities that allow new queries through this DB.

***Keywords*—** Graph Databases, MillenniumDB, User Interface, RDFExplorer Integration.

GLOSARIO

API: *Application Programming Interface*
BD: Base de Datos
DGQL: *Domain Graph Query Language*
IMFD: Instituto Milenio Fundamentos de los Datos
IRI: *Internationalized Resource Identifier*
RDF: *Resource Description Framework*
SPARQL: *SPARQL Protocol and RDF Query Language*
URI: *Uniform Resource Identifier*
URL: *Uniform Resource Locator*
VQG: *Visual Query Graphs*
W3C: *World Wide Web Consortium*
XML: *eXtensible Markup Language*

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Con la evolución continua de las bases de datos, han surgido modelos para atender diversas demandas de almacenamiento y gestión de datos. Entre las diversas estructuras de datos existente, tenemos los datos altamente interrelacionados, como los presentes en las redes sociales. Para abordar la complejidad de este tipo de estructuras, se crearon las bases de datos de grafos como una solución cuyo objetivo es la identificación y exploración de relaciones entre datos, a la vez que permite una mejor visualización de estos mismos datos para su exploración.

Una base de datos de grafos posee un lenguaje de consultas especifico para el manejo de datos. El patrón más simple y común que se utiliza en las consultas de grafos, es el patrón de grafos estructurados como los datos utilizados, en el cual se pueden reemplazar las constantes por variables para realizar la búsqueda de datos.

Dentro del contexto de BDs de grafos, se encuentra en proceso de desarrollo MillenniumDB, un sistema innovador que busca mejorar las consultas de datos mediante la incorporación de nuevas funciones que permitan una exploración de datos más avanzada y a la vez sencilla, ampliando así la profundidad de la información que se puede obtener.

Actualmente MillenniumDB enfrenta un problema crucial, la falta de una interfaz que permita la visualización de las consultas realizadas. Esta falta representa un obstáculo tanto para investigadores interesados en su funcionamiento como para potenciales usuarios que deseen explorar su utilidad. Por este motivo, se ha planteado la necesidad de integrar una interfaz existente en el modelo de MillenniumDB.

RDFExplorer ha demostrado ser una opción efectiva para los nuevos usuarios debido a su accesibilidad y facilidad de uso. Sin embargo, la adaptación de este plantea el desafío de convertir las consultas, redactadas en un lenguaje distinto al de MillenniumDB, y ajustar los funcionamientos de la interfaz para satisfacer las especificaciones de este modelo.

En este documento se examinara en detalle que compone a MillenniumDB para su integración con la interfaz de RDFExplorer. También se vera la estructura de RDFExplorer para averiguar que cambios seran necesarios para integrar a MillenniumDB.

Para el caso de MillenniumDB se investigará principalmente su modelo y la estructura que posee, además del lenguaje de consultas utilizado para obtener los datos. Y en el caso de RDFExplorer se investigará como maneja los datos y el lenguaje de consultas que utiliza para poder cambiar el lenguaje de consultas utilizado al de MillenniumDB.

Finalmente para verificar el buen funcionamiento de esta integración, se realizaran *tests* con cada posible conexión que genere un tipo de consulta diferente, demostrando si la implementación fue exitosa o si tiene problemas que se deben resolver.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

1.1. Contexto y Situación Actual

Hoy en día existen varios modelos de bases de datos que cumplen diversas funciones, para este caso se trabajara con la base de datos MillenniumDB, una base de datos de grafos la cual se encuentra en proceso de desarrollo por parte de la IMFD. MillenniumDB trabaja con datos con varias relaciones, y almacena estas relaciones como nodos, los cuales están relacionados a otros nodos o valores específicos, para unir las conexiones se utilizan las ”propiedades”, las cuales dan coherencia a la conexión. Esto le permite a MillenniumDB trabajar con datos altamente relacionados, como redes sociales, sistemas de recomendación, etc.

Actualmente MillenniumDB carece de una interfaz que permita trabajar con los datos que pueda utilizar, por lo que no existe una forma rápida y simple de realizar consultas y visualizar los resultados de estas para sus usuarios. La posibilidad de tener una interfaz permitiría mayor facilidad en el aprendizaje y *testeo* de la base de datos MillenniumDB, mejorando la accesibilidad, facilitando la construcción de las consultas y trayendo consigo la posibilidad de atraer a usuarios nuevos.

Para el caso de MillenniumDB se ha decidido utilizar una interfaz previamente construida, la cual deberá ser adaptada al lenguaje de consultas utilizado por nuestra base de datos, esta interfaz es RDFExplorer. La decisión de su uso se debe al buen resultado que el aprendizaje y uso por parte de los usuarios a demostrado [Vargas *et al.*, 2019], pero la implementación implica que se debe comprender su funcionamiento e idealmente mantener sus funciones básicas para su uso al momento de su integración.

Finalmente se debe probar el funcionamiento de la integración de MillenniumDB a RDFExplorer, para lo cual se presenta el desafió de levantar una base de datos de MillenniumDB y conectar RDFExplorer con esta. Para esto se debe revisar con que datos trabaja RDFExplorer y como integrar los datos a MillenniumDB para el correcto uso e implementación, una vez realizado esto se debe verificar el funcionamiento de la interfaz.

1.2. Objetivos

1.2.1. Objetivos generales

Adaptar la interfaz existente de RDFExplorer a MillenniumDB para la visualización de datos.

1.2.2. Objetivos específicos

- Analizar el modelo de datos al que accede MillenniumDB.
- Analizar el lenguaje de MillenniumDB e identificar los operadores más importantes del lenguaje.
- Implementar una representación visual del modelo de datos de MillenniumDB.

CAPÍTULO 2
MARCO CONCEPTUAL

2.1. RDF

RDF es un modelo estándar de la W3C para la representación de datos *web*. Su función principal es permitir la descripción de recursos en la internet de una manera estructurada y formal, lo que facilita el intercambio y la reutilización de metadatos o información sobre los datos. RDF se utiliza para codificar metadatos de forma que puedan ser entendidos y utilizados por diferentes sistemas.

RDF utiliza XML como una sintaxis común para el intercambio y procesamiento de metadatos. La utilización de XML ofrece independencia de plataforma, validación y la capacidad de representar estructuras complejas. Al aprovechar las características de XML, RDF ofrece una estructura que expresa de manera comprensible semánticas que permiten la codificación, intercambio y procesamiento consistente de metadatos estandarizados. Además RDF facilita la publicación tanto de vocabularios legibles por humanos como procesables por máquinas [Miller, 2001].

El modelo RDF se basa en la idea de utilizar triples, compuestos por sujeto, predicado y objeto, para representar información. Estos triples conforman grafos dirigidos donde los nodos son recursos y los bordes representan relaciones entre estos recursos. Viendo en mas detalle los componentes de la estructura tenemos [Ora, 1999]:

- **Recursos**: Todos los objetos descritos por expresiones RDF son llamados recursos. Un recurso puede ser una página web entera como un documento HTML, también puede ser parte de una página web, por ejemplo un HTML específico o elemento XML en la fuente del documento. Un recurso puede ser la colección completa de páginas, por ejemplo todo un sitio web. Los recursos pueden ser objetos no accesibles directamente a través de la Web, por ejemplo un libro impreso. Los recursos siempre son nombrados por una URI más un identificador opcional. La ventaja de utilizar este formato es que cualquier cosa puede tener un URI.
- **Propiedades**: Describen las relaciones entre dos recursos o entre recursos y valores concretos (literales). Los literales se utilizan para representar información directa asociada a las propiedades y pueden incluir cadenas de texto, números, fechas o booleanos. Las propiedades establecen conexiones entre los recursos y los valores asociados a través de triples. Estas propiedades pueden considerarse como los rasgos distintivos de un recurso específico, como el título de un libro, el autor, la fecha de publicación, etc.

2.1.1. RDF – Ejemplo

Utilizando un ejemplo para entender mejor cómo funciona RDF y entender como se realizan las declaraciones RDF para formar estructuras de relaciones, realizaremos un análisis de un ejemplo de grafo.

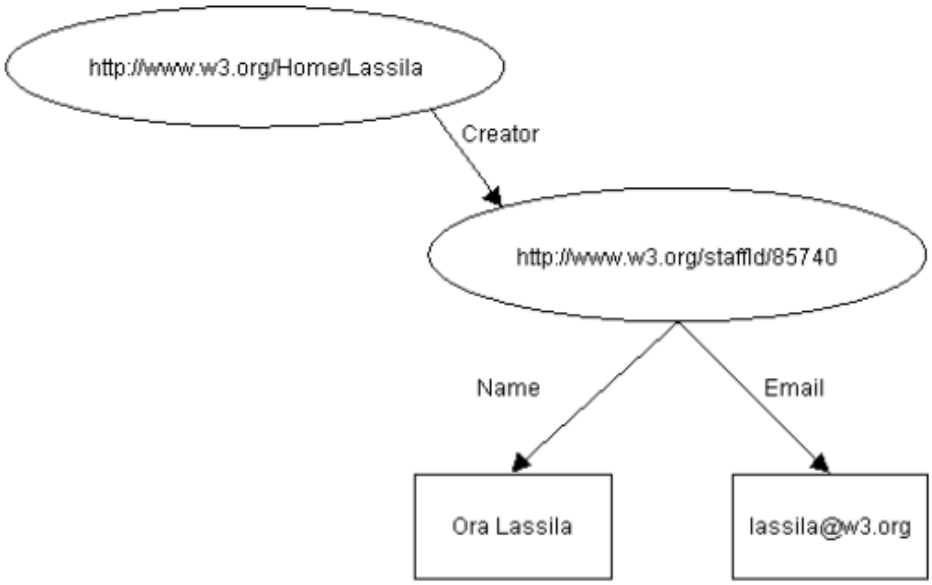


Figura 1: Ejemplo de un grafo con el modelo RDF.
Fuente: https://www.w3.org/TR/1999/REC-rdf-syntax-19990222/

Podemos ver en la Figura 1, donde los nodos circulares corresponden a recursos, las flechas corresponden a propiedades y los nodos cuadrados corresponden a valores literales. Para este caso podemos ver la relación entre dos recursos que ocurre en la parte superior de la figura, donde la URI terminada en “/home/Lassila” es el sujeto (recurso), “*Creator*” el predicado (propiedad) y la URI terminada en “/staffId/85740” es el objeto (valor). También podemos observar dos declaraciones entre un recurso (URI “/staffId/85740”) y 2 propiedades conectadas a un literal cada uno, la propiedad ”*Name*” con el valor literal ”Ora Lassila” y la propiedad ”Email” con el valor literal ”lassila@w3.org”.

Los triples con valores literales cumplen el objetivo de describir al recurso de donde provienen. En este caso podemos leer el triple como: “La persona referida con la ID del empleado 85740 es llamada Ora Lassila y posee el correo electrónico lassila@w3.org. El recurso http://www.w3.org/Home/Lassila fue creado por esta persona”.

2.2. SPARQL

Posterior al lanzamiento del modelo RDF fue necesario resolver el siguiente problema, como realizar las consultas de los datos RDF. Desde entonces han surgido diversos diseños e implementaciones que buscaron dar respuesta a este problema. En 2004, el *RDF Data Access Working Group* (Grupo de Trabajo de Acceso a Datos RDF) liberó el primer borrador de trabajo público de un lenguaje de consultas para RDF, llamado SPARQL. En enero del 2008, SPARQL pasó a ser el lenguaje de consultas recomendado por la W3C [Arenas *et al.*, 2007].

RDF es un modelo de datos de grafos etiquetados dirigidos, y SPARQL es un lenguaje de consultas que busca coincidencias en el grafo según las especificaciones de búsqueda. Las consultas realizadas por SPARQL están compuestas por tres partes. La parte de *pattern matching*, que incluye varios componentes de asignación de patrones de grafos, como partes opcionales, unión de patrones, anidación, filtrar valores de posibles coincidencias y la posibilidad de escoger el dato de origen para coincidir por un patrón. La parte de *solution modifier*, una vez que el *output* del patrón ha sido computado (de la forma de una tabla de valores de variables), permite modificar estos valores aplicando operadores clásicos, como proyección, distinción, orden y límite. Finalmente, el *output* de una consulta SPARQL puede ser de distintos tipos: consultas de si/no, selección de valores de las variables que coincide con el patrón, construcción de un nuevo dato RDF en base a estos valores, y descripción de recursos [Arenas *et al.*, 2007].

En resumen, SPARQL es el lenguaje estándar para consultar y manipular datos estructurados en formato RDF. Permite buscar patrones dentro de grafos utilizando una sintaxis similar a SQL, pero adaptada a grafos. Las consultas SPARQL constan de patrones de triples (sujeto, predicado, objeto), y estas consultas pueden ser tan simples o complejas como se necesite para recuperar la información deseada de manera precisa, además permite filtrar datos, combinar patrones, realizar operaciones de agrupación y ordenación, entre otras opciones.

2.3. RDFExplorer

Como se mencionó anteriormente, para el trabajo en RDF se recomienda el uso de SPARQL, sin embargo el manejo y llamado de datos a través de un lenguaje de consultas puede ser complicado, y en la realidad, la gente no siempre utiliza el lenguaje de consultas de una base de datos para realizar todas sus consultas, ya que se pueden utilizar interfaces que permiten un manejo más fácil o específico de una BD. Teniendo RDF y un lenguaje de consultas funcional SPARQL, es necesario una interfaz que pueda visualizar y facilitar el uso de RDF.

Existen varias soluciones de interfaz con distintos objetivos, de entre estos trabajos podemos destacar las soluciones de búsqueda y navegación, los que permiten al usuario buscar información a través de palabras clave y potencialmente filtrar o modificar los resultados seleccionando facetas o siguiendo caminos (por ej., Tabulator, Explorator, Visinav, etc). Este

acercamiento al problema está limitado en términos de los tipos de consultas que se quieren expresar, por ejemplo esto no permite ciclos. Otros tipos de interfaces se enfocan en visualizar y resumir los datos (por ej., DBpedia Atlas, LinkedGeoData Browser, etc), estos pueden proveer vistas generales de los datos más que explorarlos o consultar por nodos o entidades específicas. Otros sistemas combinan la navegación/exploración y visualización, generalmente siguiendo el paradigma de navegación basada en grafo (por ej., RDF Visualiser, Fenfire, etc), esto permite enfocarse en un nodo específico y explorar su vecindario en el grafo, sin embargo no permite generalizar estas exploraciones en consultas [Vargas *et al.*, 2019].

RDFExplorer tiene una interfaz que busca permitir a usuarios con poca experiencia realizar consultas y explorar grafos RDF, los datos que utiliza esta interfaz son los que se encuentran en wikidata. RDFExplorer se basa en un lenguaje de consulta visual formulado con respecto a un grafo de consultas visual.

2.3.1. Grafo de Consultas Visual para RDFExplorer

Para entender cómo funciona un Grafo de Consultas Visual, primero se define I como un conjunto de IRIs (siendo esto una generalización de un URI el cual permite un aumento en el uso de caracteres que no soporta un URI.), L un conjunto de literales y V un conjunto de consultas variables, tenemos lo siguiente:

Un grafo de consultas visual (VQG por sus siglas en inglés) es definido como un grafo dirigido, con aristas o etiquetadas $G = (N, E)$, con nodos N y aristas E. Los nodos del VQG son conjuntos finitos de IRIs, literales y/o variables: $N \subset I \cup L \cup V$. Las aristas del VQG son un conjunto finito de triples, donde cada triple indica a una arista dirigida entre dos nodos con una etiqueta tomada del conjunto de IRIs o variable: $E \subset N \times (I \cup V) \times N$.

Definición 1.1: Denotamos a var(G) como el conjunto de las variables que aparecen en $G = (N, E)$, en los nodos o etiquetas de aristas: var$(G) := \{v \in V | v \in N$ or $\exists n_1, n_2 : (n_1, v, n_2) \in E\}$.

Definición 1.2: Dejando que G = (N,E) denote el estado actual del VQG; un VQL esta definido en las siguientes cuatro operaciones atómicas:

- Inicializar un nuevo nodo variable: $\nu(G) := (N \cup \{v\}, E)$ donde $v \notin var(G)$.
- Añadir un nuevo nodo constante: $\nu(G, x) := (N \cup \{x\}, E)$ donde $x \in (I \cup L)$.
- Inicializar una nueva arista entre dos nodos con una variable arista-etiquetada: $\epsilon(G, n_1, n_2) := (N, E \cup \{(n_1, v, n_2)\})$ donde $\{n_1, n_2\} \subseteq N \; and \; v \notin var(G)$.
- Añadir una nueva arista entre dos nodos con una IRI arista-etiquetada: $\epsilon(G, n_1, n_2, x) := (N, E \cup \{(n_1, x, n_2)\})$ donde $\{n_1, n_2\} \subseteq N \; and \; x \notin I$.

Cabe destacar que para los operadores (G) y (G,n_1,n_2), la variable no está especificada, donde en vez una variable arbitraria puede ser automáticamente generada. No importa qué variables se escojan, ya que las variables añadidas son frescas, en el VQG resultante siempre se obtendrán valores únicos; en la práctica, el sistema genera nombres únicos para cada variable [Vargas *et al.*, 2019].

2.3.2. Interfaz de RDFExplorer

Ya explorada la teoría para trabajar con el VQG, el cual trabajara con SPARQL para las consultas, se debe ver como se visualizan los datos en RDFExplorer. La interfaz de RDFExplorer está compuesta por seis componentes principales, los cuales son mostrados en tres paneles, estos son el panel de búsqueda (ubicado a la izquierda), un editor de consultas visuales (ubicado en el centro), y una vista con el detalle del nodo (ubicado a la derecha), esto se puede observar en la Figura 2.

Para utilizar la interfaz, el usuario comienza con un lienzo en blanco, el usuario es quien se encarga de agregar los nuevos nodos, ya sean constantes o no. El usuario puede seleccionar un nodo constante utilizando el panel de búsqueda, buscando por palabras clave, este panel sugiere nodos los cuales pueden ser arrastrados al lienzo, o bien puede comenzar inicializando un nuevo nodo variable haciendo clic derecho sobre el lienzo y escogiendo la opción *"New variable"*. El usuario puede seguir añadiendo nodos variables o constantes siguiendo las mismas instrucciones.

Con un nodo creado en el lienzo, es posible añadir una arista o también llamada propiedad, seleccionando un nodo con el clic derecho y usando la opción *"New property"*, la cual genera la arista y la conecta automáticamente a un nodo variable. También se puede crear la arista arrastrando una propiedad desde la lista que aparecerá a la derecha al hacer clic a un nodo, lo cual genera una arista a un nodo constante. Finalmente arrastrando una flecha del nodo de origen a un nodo objeto creado previamente, generará un triple con una variable arista-etiqueta. Las propiedades se muestran como una caja anidada dentro del nodo de origen, luego se genera una lista de sugerencias de IRIs que, de ser posible, generaran resultados no vacíos.

Además de poder crear propiedades que conecten dos nodos, es posible crear propiedades que conecten a un nodo constante de valor literal, el cual es representado como otra caja anidada dentro del nodo de origen, debajo de su propiedad que la une.

Al hacer clic en un nodo constante, se puede observar el contenido en más detalle, en caso de que el nodo sea una variable, se presentan posibles valores para esa variable. En caso de que el nodo sea una constante, se muestran los datos del nodo organizados según las propiedades del tipo de dato. También se pueden convertir nodos variables en nodos constantes y viceversa para permitir al usuario una mayor exploración. Todas estas opciones, permiten al usuario explorar el grafo y recibir *feedback* de los resultados obtenidos.

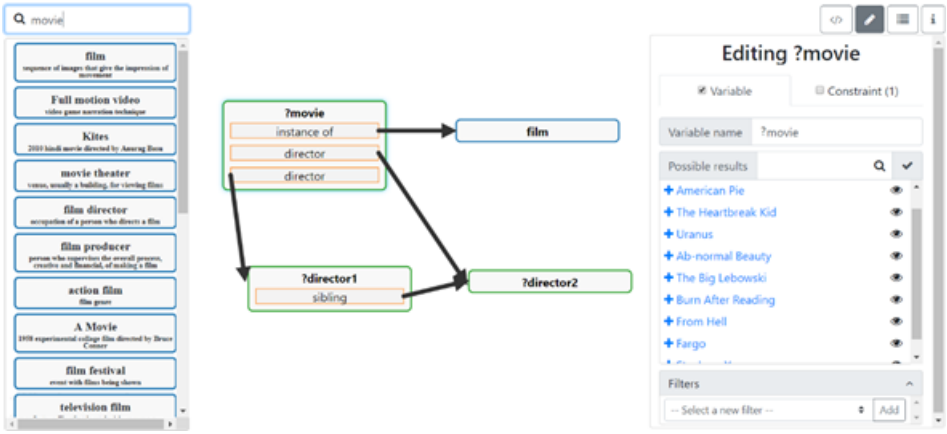


Figura 2: Ejemplo de una consulta visual. Busca a directores de cine, hermanos, que hayan trabajado juntos.

Fuente: https://rdfexplorer.org/

2.3.3. Consultas con SPARQL

Para comprender de mejor manera el uso y la lógica detrás de los grafos en RDFExplorer, se debe explorar el funcionamiento de SPARQL que permite realizar las consultas para obtener los posibles valores de un nodo variable. Para esto veremos la sintaxis de SPARQL, lo cual es necesario para como se generan las soluciones que da RDFExplorer. Primero veremos la estructura que generalmente veremos con SPARQL para RDFExplorer, cabe destacar que existen muchas otras funciones que no serán utilizadas en RDFExplorer:

```
PREFIX ...
SELECT ...
WHERE {
...
    OPTIONAL {
    ...
    }
}
LIMIT Number
```

En la estructura presentada, PREFIX corresponde a prefijos utilizados por las URIs de las constantes, de esta forma acortamos las URIs haciéndolas más legibles. La clausula SELECT corresponde a las variables que queremos obtener sus resultados, en RDFExplorer esto corresponde al nodo variable seleccionado. La clausula WHERE corresponde a las condiciones que se deben cumplir al buscar esta variable, se encarga de definir los patrones de triples que se utilizan para buscar y filtrar información en el grafo RDF, entregando los resultados de las variables que satisfagan los triples. La clausula OPTIONAL se utiliza cuando queremos obtener variables que pueden ser necesarios solo si existen, en caso de no existir no detendrá

la búsqueda y no considerara lo pedido aquí. Finalmente la clausula LIMIT limita la cantidad de datos retornados según el numero ingresado, su utilidad radica en la capacidad de reducir tiempos de búsqueda limitando la cantidad de datos recuperados, lo que resulta en una consulta más eficiente al enfocarse únicamente en un número específico de resultados en lugar de buscar la totalidad de los datos disponibles.

2.4. *Domain Graph*

El *Domain Graph* (o grafo de dominio en español) asigna *IDs* a la conexión entre dos nodos de un grafo con una propiedad. Asumiendo un universo *Obj* de objetos (*IDs*, strings, etc) tenemos que un *Domain Graph* consiste en un conjunto finito de objetos $O \subseteq Obj$, y un *mapeo* parcial: $\gamma : O \to O \times O \times O$, de esta forma podemos representar una conexión entre dos nodos de la forma $\gamma(e) = (n_1, t, n_2)$, en la Figura 3 podemos ver un ejemplo de conexión entre nodos, los cuales representan que Michelle Bachelet tuvo la posision de presidenta de Chile, a este caso en concreto le podemos asignar una *ID* e. Esto significa que un arco (n_1, t, n_2) tiene *ID* e, es del tipo t y une el nodo de origen n_1 con el nodo n_2. A este modelo lo podemos definir de la siguiente manera:

$$DomainGraph(source, type, target, eid)$$

siendo *eid* la *ID* unica para la relación $(source, type, target)$, *source* es el nodo de origen, *type* corresponde a la propiedad del arco generado y *target* es el nodo objeto al que apunta la propiedad [Vrgoc *et al.*, 2021].

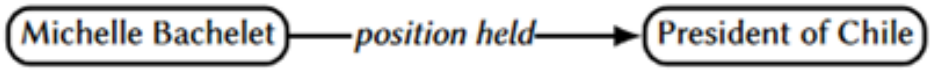


Figura 3: Ejemplo de una relación entre dos nodos, asignando un *eid*, tenemos un modelo $DomainGraph$.
Fuente: *MillenniumDB: A Persistent, Open-Source, Graph Database*

Una ventaja del *Domain Graph* es que subsuma al modelo de grafo de RDF, debido a que el modelo RDF utiliza triples de la misma forma, donde el triple $(Recurso, arista, valor)$ es equivalente al modelo $DomainGraph(source, type, target, eid)$ sin la *eid*, por lo que para añadir los valores de RDF simplemente le podemos asignar una *eid* para que cumpla con el modelo.

2.5. *Property Graph*

Los *Property Graph* permiten a nodos y aristas ser anotados con etiquetas y pares propiedad-valor, como se muestra en la Figura 4, podemos observar que lo que serían aristas a literales ahora forman parte del nodo como una etiqueta, este tipo de grafo tiene ventaja para datos repetidos, esto se debe a que los casos que se repitan datos, da una opción que permite diferencias los casos y agregarles datos que ayudan a la información que se quiere entregar, esto se puede ver en el caso del par propiedad-valor (*order*, "2") de la arista e_2.

2.5.1. Relación de *Property Graph* con *Domain Graph*

El sistema de anotación del *Property Graph* es compatible con los *Domain Graph*, por ejemplo podemos anotar el par propiedad-valor (*first name*, "Michelle") del nodo n_1 como una arista $\gamma(e_3) = (n_1, \textit{first name}, \text{Michelle})$, el par propiedad-valor (*order*, "2") de la arista e_2 se convierte en $\gamma(e_4) = (e_2, \textit{order}, \text{"2"})$, y siguiendo con estas transformaciones podemos relacionarlo con el *Domain Graph*. Sin embargo, dado el legado del *Property Graph*, existen potenciales incompatibilidades entre estos tipos de grafos, por ejemplo *strings* como *"male"*. etiquetas como *"human"*, etc, ahora son nodos en el grafo, lo que genera nuevos caminos que pueden afectar los resultados de una consulta.

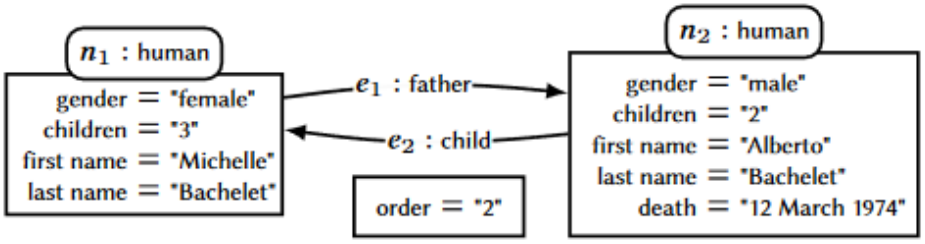


Figura 4: Un *Property Graph* con dos nodos y dos aristas, la propiedad en e_2 indica que es el segundo hijo

Fuente: *MillenniumDB: A Persistent, Open-Source, Graph Database*

El sistema combinado de *Property Domain Graph* posee mayor compatibilidad para la idea que logra *Property Graph*, con esta implementación podemos generar triples involucrando las aristas y de esta forma realizar consultas parecidas a SPARQL apuntando a las etiquetas.

2.6. MillenniumDB

MillenniumDB es una base de datos, persistente y *open source* en desarrollo, que apunta a mejorar la teoría y practica para los sistemas de BD de grafos, siendo basada en bases teóricas

sólidas y las técnicas más avanzadas, mientras que es aplicable en situaciones prácticas. El diseño e implementación de MillenniumDB se basa en los siguientes objetivos:

- **Generalidad:** La BD apunta a apoyar varios modelos de BD de grafos y lenguajes de consultas a través de la generalización e implementación de técnicas para una configuración más general
- ***State-of-the-art*:** Con el objetivo de alcanzar lo último en rendimiento, MillenniumDB incorpora y combina las últimas técnicas desarrolladas en literaturas investigativas
- **Teóricamente bien fundado:** MillenniumDB provee semánticas claras para las interfaces que tiene por debajo, y prioriza técnicas que proveen garantía teórica
- **Modularidad**: Para permitir extensibilidad, MillenniumDB sigue un diseño modular, que desacopla un sistema de BD de grafos en componentes con interfaces claras y definidas
- ***Open source*:** El código para MillenniumDB se encuentra disponible bajo una licencia *Open source*, con el objetivo de que pueda extenderse y ser reusado por otros investigadores y practicantes.

2.6.1. MillenniumDB - Modelo de datos

MillenniumDB implementa una simple extensión del modelo de *Domain Graph* llamado *Property Domain Graphs*, el cual da permisos para anotaciones externas, por ejemplo añadir etiquetas y par propiedad-valor a nodos y aristas. Escrito de forma formal, si L es un conjunto de etiquetas, P es un conjunto de propiedades, y V es un conjunto de valores, se puede definir al *Property Domain Graphs* de la siguiente manera:

Definición 2.1: Un *Property Domain Graphs* es definido como una tupla $G = (O, \gamma, lab, prop)$, donde:

- (O, γ) es un *Domain Graph* con O como el conjunto finitos de objetos y γ la función que representa la conexión entre 2 nodos a través de su *eid*
- lab: $(O \to 2^L)$ es una función asignando un conjunto finito de etiquetas a un objeto
- prop: $O \times P \to V$ es una función parcial asignando un valor a cierta propiedad de un objeto

Asumiendo que por cada objeto $o \in O$, existe una cantidad finita de propiedades ´ $\in P$ tal que prop(o, p) es definido.

Con este modelo, además de la relación $DomainGraph$ vista en el modelo de *Domain Graph*, se añaden dos nuevas relaciones:

$$Labels(\text{object, label})$$
$$Properties(\text{object, property, value})$$

Donde el conjunto de *(object, property)* de la relación $Properties$ es la llave primaria de la segunda relación, con la primera relación permitiendo múltiples etiquetas por objeto.

2.6.2. MillenniumDB - Lenguaje de consultas

Siguiendo al objetivo de generalizar, existen una gran variedad de lenguajes de consultas que se han propuesto para trabajar con grafos durante años, mientras que las sintaxis de los lenguajes de consultas creados varían ampliamente, comparten una base común en términos de consultas primitivas. En específico, todo lenguaje de consulta soporta diferentes tipos de patrones de grafos, que transforman un grafo en una relación (o también dicho, una tabla) de resultados. La forma más simple de patrón de grafo es una *Basic Graph Pattern*, la cual de forma más común, es un patrón de grafos estructurado como los datos, pero que también permite variables para reemplazar constantes. Luego tenemos los *Navigational Graph Patterns*, que permiten especificar caminos de expresiones que coinciden caminos de largo arbitrario en los grafos. También se tiene *Relational Graph Patterns*, el cual, notando que un patrón de grafo convierte un grafo en una relación, permite el uso de operadores relacionales para manipular los resultados de uno o más patrones de grafo en una sola relación. Finalmente, los lenguajes de consulta pueden soportar otras características.

Entonces para lograr este objetivo y soportar los múltiples modelos de grafos, MillenniumDB debe utilizar un lenguaje de consulta que le permita soportar un número de lenguajes de consultas de grafos, sin embargo no existe un lenguaje que permita aprovechar el modelo de *Property Domain Graph*, por lo que MillenniumDB implementa un lenguaje de consultas llamado DGQL, el cual se asemeja a Cypher pero adaptado al modelo de la BD, también añade características de SPARQL comúnmente utilizadas para consultas de knowledge graphs como Wikidata. Una consulta sencilla de DGQL toma la siguiente forma:

```
MATCH Patrones
WHERE filtros
RETURN Variables
```

Esta consulta básica expresa una búsqueda que cumpla con el patrón insertado en la clausula MATCH, luego filtra los resultados obtenidos colocando condiciones en la clausula WHERE,

y finalmente RETURN expresa las variables consultadas de las cuales queremos obtener sus resultados. En el Anexo A podemos encontrar un ejemplo mas detallado de su uso durante el trabajo, con todas las clausulas utilizadas y el detalle de su uso para los casos que se iran a revisar.

2.7. Wikidata

Para este trabajo se utilizará el conjunto de datos de Wikidata, un proyecto de Wikipedia que apunta a disponibilizar los datos recopilados por los usuarios de Wikipedia en una gran base de datos más accesible para consultas por otras aplicaciones, esto lo diferencia de Wikipedia, la cual recopila información pero esta no es ordenada para el uso de una maquina.

Wikidata utiliza un modelo de datos simple, los datos de un objeto están almacenados de la forma *Property-value* (Propiedad-valor), esto significa que cada recurso de Wikidata es un nodo que posee aristas en la forma de propiedades asociadas a un valor que seria otro nodo, este valor al que se conecta puede ser tanto otro recurso como un valor literal. Por ejemplo para el objeto ”Roma” la propiedad ”población” puede tener un valor ”2.777.979”. Como los valores también pueden ser un recurso, es ideal para generar bases de datos de grafos, ya que permite busquedas a través de las relaciones entre recursos y literales.

Para el caso en que un arco de un triple de Wikidata, es posible que posean pares propiedad-valor asociado a sus arcos, a estos valores se les llaman *Qualifiers* por Wikidata. Por ejemplo en el mismo caso de la población romana, este ejemplo corresponde a la población del 2010, para evitar tener que crear un par propiedad-valor para cada año, donde la propiedad se defina como ”Población año 2010” y se asocie al valor definido anteriormente, podemos utilizar un *Qualifier* que indique el año que se midió la población de este par, de esta manera al revisar los valores de población existentes, podemos saber cuando se tomo esta medición. También se pueden incluir más de un *Qualifier*, como por ejemplo el metodo utilizado para la medición de la población [Vrandečić y Krötzsch, 2014]. Se puede ver este mismo caso en la Figura 5.

Figura 5: Ejemplo de par *Property-value* para el objeto ”Roma”
Fuente: *Communications of the ACM*

Cabe destacar que Wikidata para sus recursos utiliza *IDs* únicas e *IDs* únicas para sus propie-

dades, ambos ubicados en sus URLs, los valores de las propiedades se conforman de recursos o literales.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

3.1. Analizando el modelo de MillenniumDB

Para empezar debemos analizar el modelo de MillenniumDB, el cual se vio que utiliza un lenguaje de consultas llamado DGQL. Para comprender su uso y utilidades, se investigó el repositorio de MillenniumDB[1], el cual contiene la base de datos libre para su uso. En este repositorio encontramos las instrucciones y la ayuda para su uso, sin embargo encontramos uno de los primeros problemas, el cual es que actualmente MillenniumDB nos ofrece dos opciones para crear una base de datos, la primera opción es creando la BD con SPARQL y la segunda opción utilizando MQL.

Originalmente se habla de un lenguaje de consulta utilizado, llamado DGQL, pero como MillenniumDB aun se encuentra en desarrollo, nos encontramos con esta situación. Para esto analizando los documentos y las opciones que nos proveen las dos bases de datos, vemos que el caso de crear la base de datos con SPARQL directamente resulta en algo similar a SPARQL, el problema de esto es que no cumple con los objetivos del mencionado DGQL, ya que creando la BD con esta opción solo nos permite utilizar triples para las consultas de grafos, por lo que descartamos esta opción.

La segunda opción, que es crear la base de datos con MQL, utiliza un modelo *"quad model"* utilizando un sistema similar al visto en DGQL, donde los nodos tienen pares propiedad-valor y cada conexión contiene una *eid* que la identifica, por lo que de esta opción se pueden extraer los *Qualifiers* de Wikidata, o como se menciona en DGQL, las propiedades de los arcos $\gamma(e)$.

En este modelo existen 4 tipos de objetos:

1. Literales: este objeto incluye los clásicos valores básicos: enteros, *strings*, flotantes y *booleanos*.
2. Nodos: incluye los recursos que pueden tener etiquetas o propiedades. Los nodos se dividen en 2 tipos:
    2.1 Nodos nombrados: son aquellos que tienen un identificador al ingresarlos a la base de datos.
    2.2 Nodos anónimos: corresponde a aquellos nodos que no poseen una identificación al ingresarlos a la base de datos, en estos casos MillenniumDB creara un identificador auto-generado.

[1] https://github.com/MillenniumDB/MillenniumDB

3. Arcos: un arco es un objeto que se relaciona con otros objetos, según los siguientes atributos:

    - Un identificador auto-generado
    - *from*: cualquier objeto sin incluir este
    - *to*: cualquier objeto sin incluir este
    - *types*: un conjunto de nodos nombrados que puede ser vacio.
    - *properties*: un conjunto de pares llave-valor donde la llave es un *string* y el valor un literal. Las llaves no pueden repetirse.

Para utilizar este modelo, cada valor debe calzar con un formato alfanumérico, a excepción de valores *true* o *false*. En caso de tener nodos anonimos, estos tomaran un valor que sigue la expresión regular _a[1-9][0-9]*.

3.2. Análisis y Adaptación de RDFExplorer

Para nuestro problema ya se tiene previsto el uso de RDFExplorer, pero antes de empezar su uso, se investigó sobre el lenguaje de programación utilizado, en este caso principalmente JavaScript para la implementación de la BD y HTML en el caso de necesitar ajustar otros detalles del *front-end*.

La integración de MillenniumDB a RDFExplorer implica una serie de pasos, comenzando por las modificaciones de la interfaz. Esto debido a que se necesita integrar la capacidad de obtener las propiedades asociadas a los arcos generados en un triple, la forma que se quiere obtener para representar esta idea se ve en el ejemplo de la Figura 6, donde *"order"* representa que *?y* es el primer hijo de *?x*. Esta forma se debe a la necesidad de representar de alguna manera las propiedades pertenecientes al arco del que proviene, para esto se puede utilizar la propiedad del arco como un "nodo del arco" con el cual podemos agregar las propiedades y comprender a que arco pertenece. Con esto mantenemos la simplicidad de RDFExplorer y es fácil de entender que esta es una conexión distinta.

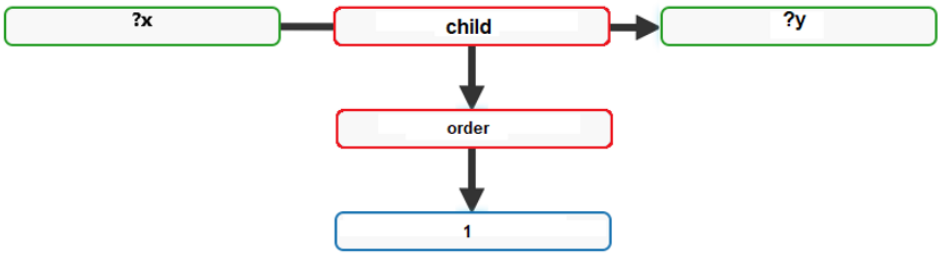


Figura 6: Ejemplo prototipo de propiedad-valor de un arco.
Fuente: Elaboración propia

Fuera del requisito anterior, es necesario realizar el cambio en el lenguaje de consultas para las conexiones que se realizan al armar los grafos.

Previo al uso de RDFExplorer para nuestra solución, se debió investigar el lenguaje de programación utilizado en la interfaz, se utiliza JavaScript para la implementación de la BD y del *back-end*, y HTML para el *front-end*, aunque este último requiere de menos cambios debido al uso de la libreria *D3* en JavaScript para construir los grafos de manera interactiva.

3.2.1. Adaptación de las Propiedades y Literales

Para integrar MillenniumDB a la interfaz de RDFExplorer, primero se debió realizar un ajuste en la estructura de los nodos, originalmente los nodos contienen las propiedades dentro de ellos, formando parte de la estructura del nodo, de esto se quiere extraer la propiedad fuera de este. Esto supuso un nuevo problema, ya que para evitar tener que reconstruir todo el código, se debió mantener la unión del nodo con la propiedad, debido a esto, las propiedades se mueven en conjunto al nodo, pero de todas maneras se logró ”individualizar” a la propiedad, separándola y permitiendo moverla libremente sin estar adherida a su nodo padre.

Además de separar la propiedad del nodo, se altero la posición inicial al crear el objeto propiedad, el nodo ya no se expande verticalmente para ajustar las propiedades como lo hacia originalmente, y se cambiaron las dimensiones para tener similitud con la forma de los nodos para que quede claro al usuario que puede moverse, pero se mantuvo el color para diferenciarlo fácilmente. Otros cambios agregados fue la inclusión de una linea entre el nodo original y su propiedad, no se utilizó una flecha para dar a entender que esta conexión continua con el nodo objetivo.

Este primer cambio se puede ver en la Figura 7, la cual compara el antes y después de la implementación.

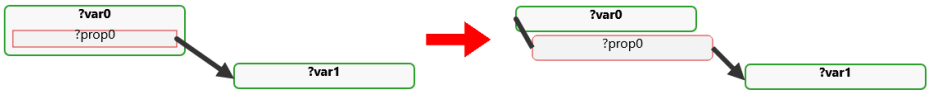


Figura 7: Diferencia entre la estructura original y la nueva de un triple.
Fuente: Elaboración propia

Para el caso de los literales, se realizaron pocos cambios, esto se debe a que los literales no solo están ligados al nodo del que proviene la propiedad, sino que además es parte de la propiedad que lo invoca, por lo que el trabajo de separarlo es mas costoso, difícil y realmente no influye mucho en lo que se quiere realizar, ya que los triples tipo Nodo-Propiedad-Literal no pueden generar una conexión mas larga, y la idea de obtener los datos del arco utilizando la propiedad como origen sigue funcionando. La nueva estructura de un triple con Literal se

ve en la Figura 8. Además de esto, la propiedad sigue moviéndose independiente, pero el literal siempre esta adherido.

Figura 8: Diferencia entre la estructura original y la nueva de un triple.
Fuente: Elaboración propia

3.3. Implementación de MillenniumDB

Luego de estos ajustes en el código, con los que además se comprendió de mejor manera el funcionamiento RDFExplorer, se inició la implementación de MillenniumDB al sistema, para lo cual se debió levantar un servidor con la base de datos de Wikidata. Esto involucró los siguientes pasos:

- Aprender a inicializar una base de datos en MillenniumDB con datos de prueba de Wikidata.
- Cambiar las consultas SPARQL al lenguaje de MillenniumDB.
- Conectar la BD con RDFExplorer.

3.3.1. Ingresando los datos de prueba

Para utilizar el modelo de MillenniumDB en RDFExplorer, debemos utilizar los mismos tipos de datos que RDFExplorer usa. RDFExplorer trabaja estrechamente con Wikidata, por lo tanto para el trabajo utilizaremos su BD, sin embargo la base de Wikidata contiene demasiada información, y para el desarrollo de este trabajo se utilizara la base de datos de forma local. Para evitar tiempos de espera muy largos o limitaciones por el espacio de la memoria, se utilizará una base de datos reducida de Wikidata para evitar estos problemas.

Reducir los datos de Wikidata no genera problemas en el problema que se busca solucionar, ya que si la adaptación de la interfaz RDFExplorer para MillenniumDB funciona con este sub-conjunto de datos de Wikidata, no hay razón por la cual no funcione con el conjunto total de datos de Wikidata.

Una vez obtenida esta base de datos reducida, debemos adaptar la información al modelo de MillenniumDB. Como se explicó anteriormente, esta base se creará con la opción de MQL, esta opción tiene ciertas restricciones a la hora de importar datos a la BD.

La primera restricción corresponde a los nodos, estos deben tener una *ID* la cual puede ser un nodo nombrado o anónimo generado automáticamente. Los nodos pueden opcionalmente contener una lista de etiquetas y/o una lista de propiedades del tipo llave-valor.

La segunda restricción corresponde a la definición de los arcos, cada arco debe cumplir con 5 condiciones para poder integrarse:

1. El arco debe contener un nodo identificador o un literal.
2. Este nodo o literal debe ser seguido por una dirección, indicada de la manera ->o <-.
3. El segundo nodo o literal del cual se origina o se dirige.
4. Un tipo de arco, debe ser un nodo nombrado
5. En caso de tener propiedades, integrar esta lista de propiedades

Para el caso de Wikidata nos interesa cumplir con la definición de los arcos, ya que todos los datos están formados por conexiones y no poseen etiquetas.

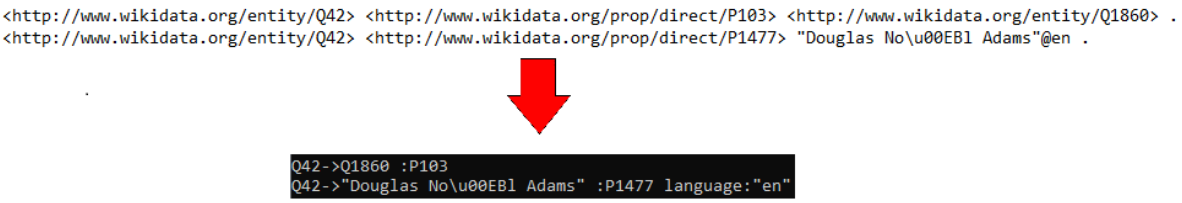


Figura 9: Diferencia entre la estructura original y la nueva de un triple.
Fuente: Elaboración propia

El archivo de los datos de Wikidata tiene un formato legible para SPARQL, sin embargo esto no funciona con el modelo MQL, para adaptar los datos utilizamos el Github de MillenniumDB *benchmark*[2], el cual contiene un código que permite modificar los datos de Wikidata a algo legible por el modelo MQL.

Los cambios que realiza el código utilizado para la transformación los podemos ver en la Figura 9, donde vemos 2 ejemplos de triples de SPARQL, los cuales son adaptados para cumplir con el modelo utilizado por MillenniumDB según las condiciones dadas por la definición de los arcos.

[2]https://github.com/MillenniumDB/benchmark

3.3.2. Inicializando MillenniumDB

Para levantar la base de datos localmente, descargamos repositorio de MillenniumDB, luego debemos ejecutar la opción que crea los archivos ejecutables llamada *"create_db_mql"* con el archivo de texto con la base de datos reducida de Wikidata adaptada para su uso. Con esto generamos una base de datos utilizable por MillenniumDB y sera almacenado en una carpeta para su uso.

Una vez generado los archivos, ejecutamos a través de la terminal el archivo *"server_mql"* en conjunto a la ruta de la carpeta de la base que generamos. De esta manera la base de datos esta levantada y funcionando.

Una vez levantada la BD, para realizar las consultas se debe ejecutar un comando desde la consola de la forma "*query_mql <consulta.txt*". Esto significa que el archivo ejecutable *query_mql* se ejecuta utilizando *consulta.txt* como *input*. Esto nos retornara un *string*, el cual su primera linea son los resultados de las variables consultadas seguido de todos los resultados para el conjunto de variables consultadas.

La ejecución de las consultas crea un nuevo problema, ya que se ejecuta como un comando de consola en el servidor y no se realiza directamente al servidor que se encuentra levantado. Por culpa de esto, surgieron restricciones en las conexiones que no permitian el ingreso de comandos por temas de seguridad. Para enfrentar este problema, se creó un código de python ubicado en la misma carpeta donde se levantó el servidor, el cual recibe la solicitud de la consulta que un cliente quiere realizar, luego este programa ejecuta el comando de la consola con la consulta como input, para finalmente enviar el resultado de la consulta de vuelta al cliente. Este proceso se ve en el diagrama de secuencia de la Figura 12.

3.3.3. Implementando el lenguaje de consultas de MillenniumDB para consultas base

Ahora empezando la integración de MillenniumDB y utilizando su nuevo lenguaje de consultas como reemplazo para SPARQL, primero se debieron investigar como RDFExplorer utiliza SPARQL para realizar sus consultas y como maneja el resultado de estas, para esto se investigo el código de RDFExplorer que genera las consultas.

En el código encontramos que RDFExplorer genera las consultas por partes. Para esto en primer lugar al seleccionar un nodo que contiene una variable, se guardan todos los nodos que interactúan con este y los que afectan los otros resultados en un conjunto de triples. RDFExplorer también genera triples que utiliza para obtener la etiqueta con el nombre del recurso.

Una vez se tienen todos los triples necesarios para realizar la consulta, RDFExplorer genera las consultas de la siguiente manera:

1. Se genera la parte de la consulta donde se busca la variable y su etiqueta (SELECT DISTINCT variable varEtiqueta).
2. Se escriben todas las relaciones existentes con los triples almacenados en la clausula WHERE, además se filtran los nodos según si son IRIs o literales para diferenciar los resultados.
3. A traves de la clausula OPTIONAL, busca la etiqueta *varEtiqueta* con el nombre de la variable en caso de existir, de no ser asi retorna vacío.
4. Finalmente con LIMIT se limitan los resultados obtenidos según la configuración que se tenga

Este sistema debe ser adaptado al lenguaje de consultas de MillenniumDB. Para entender como se realiza el proceso que crea las consultas se utilizaron varios grafos en RDFExplorer y se analizó las consultas que generó, para entenderlo mejor utilizaremos el grafo de la Figura 10, de esta manera se logró comprender como se debió adaptar el lenguaje de MillenniumDB.

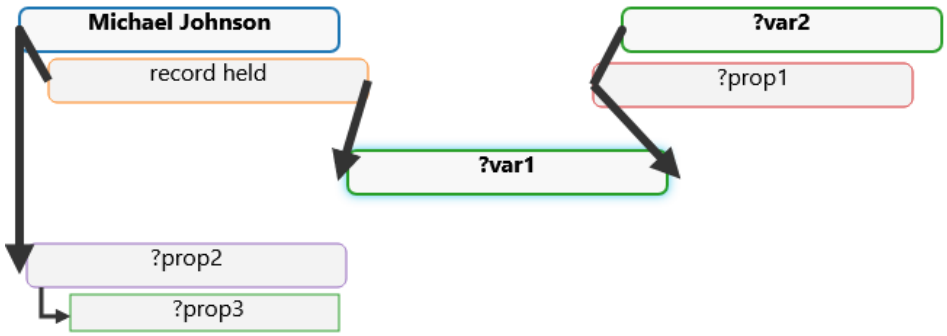


Figura 10: Grafo de ejemplo creado en RDFExplorer adaptado a MillenniumDB.
Fuente: Elaboración propia

El caso mencionado sera dividido en distintas partes para comprender como se forma la consulta en el código, y de esta forma comprender los requerimientos para su construcción en MillenniumDB.

3.3.4. Caso de camino simple

Primero veremos como se forma una consulta de una conexión simple. utilizando como referencia el grafo de la Figura 10, tomando el triple (Michael Johnson, *record held*, ?var1) para nuestro caso simple.

Cuando se hace clic en un nodo o propiedad variable, RDFExplorer comienza en el momento su búsqueda con SPARQL. Revisando la consulta generada en el código, obtenemos la siguiente consulta para el caso propuesto:

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?var1 ?var1Label WHERE {
  wd:Q190924 wdt:P1000 ?var1 .
  OPTIONAL {
    ?var1 rdfs:label ?var1Label .
    FILTER (lang(?var1Label) = "en")
  }
} LIMIT 10
```

- **"PREFIX"** son prefijos utilizados por el API de wikidata para abreviar URIs largos.
- **SELECT DISTINCT ?var1 ?var1Label** selecciona los resultados distintos para *?var1* y *?var1Label*, siendo el ultimo la etiqueta correspondiente al ID de los resultados de *?var1*.
- En la primera linea después del comando **WHERE** selecciona los valores de la propiedad P1000 para el elemento con ID Q190924 en Wikidata, los resultados son retornados en *?var1*.
- El comando **OPTIONAL** intenta obtener la etiqueta en ingles para los valores de *?var1*
- **LIMIT** limita los resultados obtenidos a 10 en este caso, de esta forma se demora menos la busqueda y no sobrecarga de resultados a la interfaz.

Adaptar esta consulta a MillenniumDB resulta en algo más sencillo, ya que se ignoran ciertos comandos debido a que MillenniumDB no tiene una API con la que obtener los datos directamente de Wikidata, en cambio, la BD utilizada esta construida con las IDs de los elementos y propiedades, y los valores que pueden ser literales o IDs. Debido a esto, se deben obtener los datos a traves de otra API proveniente de Wikidata[3], con la cual podemos obtener toda la información de una entidad con su ID.

Con lo considerado anteriormente, utilizando el lenguaje de consultas de MillenniumDB obtenemos el siguiente resultado

```
MATCH (Q190924)-[?e0 :P1000]->(?var1)
WHERE ?var1 is not string
RETURN DISTINCT ?var1
LIMIT 10
```

- **MATCH** busca todos los resultados donde el nodo de origen sea Q190924 con la propiedad P1000. *?e0* tomara los valores de ID de arco y *?var1* los valores del nodo objeto

[3]https://www.wikidata.org/w/api.php

- **WHERE** fija las condiciones de la busqueda, en este caso se buscan los *?var1* que no sean *string*, ya que las IDs no son consideradas un *string* en la BD de Wikidata para MillenniumDB, pero no hay un filtro que tome exactamente la ID, y es conocido que los literales en la BD son *strings*.

- **RETURN DISTINCT** es el comando que permite retornar los valores distintos pertenecientes a *?var1*

- Al igual que SPARQL, **LIMIT** limita la cantidad de resultados obtenidos a 10

Con esto obtenemos la ID del nodo objeto, utilizando la API de Wikidata para obtener la información del nodo obtenido, podemos filtrar directamente lo pedido por la API al idioma ingles, y de esta manera se obtiene la etiqueta faltante.

3.3.5. Caso de camino complejo

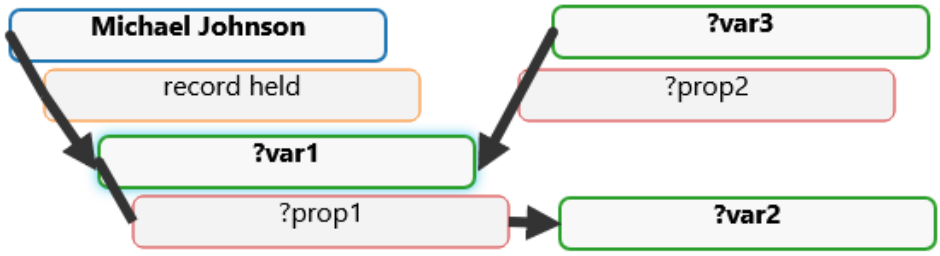


Figura 11: Grafo complejo, con más conexiones.
Fuente: Elaboración propia

En este caso se vera como se tratan grafos con caminos más complejos, siendo los objetivos de la consulta nodos apuntados por mas de una entidad, o nodos con 1 o más propiedades. Como este caso difiere mucho del original, se utilizará la Figura 11 como referencia. Para este y los siguientes casos solo se explicara las lineas de comando nuevas.

```
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
PREFIX wikibase: <http://wikiba.se/ontology#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?var1 ?var1Label WHERE {
  wd:Q190924 wdt:P1000 ?var1 .
  ?var1 ?prop1 ?var2 .
  FILTER isIRI(?var2)
  ?var3 ?prop2 ?var1 .
  FILTER isIRI(?var1)
```

```
  ?prop1tmp wikibase:directClaim ?prop1 .
  ?prop2tmp wikibase:directClaim ?prop2 .
  OPTIONAL {
    ?var1 rdfs:label ?var1Label .
    FILTER (lang(?var1Label) = "en")
  }
} LIMIT 10
```

Para casos mas complejos, simplemente se toma cada triple o conexión que hay entre 2 nodos, y se integra a la consulta. En SPARQL se ve cada condición relacionada con la variable que se busca, en este caso *?var1*. Ademas la linea *"?prop1tmp wikibase:directClaim ?prop1 ."*, que también se repite para ?prop2, cumple la función de que no exista una secuencia de relaciones o nodos intermedios.

Las variables que reciben el filtro *"isIRI"* permite asegurarse que los resultados obtenidos no son literales.

Esta consulta esta escrita de la siguiente manera en MillenniumDB:

```
MATCH (?var3)-[?e0 :?prop2]->(?var1),
 (Q190924)-[?e1 :P1000]->(?var1),
 (?var1)-[?e2 :?prop1]->(?var2)
WHERE ?prop2 is not string
AND  ?var3 is not string
AND  ?var1 is not string
AND  ?prop1 is not string
AND  ?var2 is not string
RETURN DISTINCT ?prop2
LIMIT 10
```

En el caso del lenguaje de consultas de MillenniumDB, no es necesario asegurarse que no haya una secuencia entre los nodos, ya que el comando exige que se conforme por una conexión en cada caso. Se verifica que cada variable y propiedad no sea *string*, que equivale a decir que no sea un literal o al filtro *"isIRI"* del caso de SPARQL para las variables.

3.3.6. Caso valor literal

Para este caso volvemos a utilizar la Figura 10 de ejemplo. Cuando se buscan valores del tipo literal, se realiza una consulta distinta, debido a que el resultado literal difiere mucho de un ID, ya que este ultimo puede extenderse a un grafo más largo, en cambio los literales son generalmente solo texto que no puede generar un camino más largo en el grafo.

SPARQL genera la siguiente consulta para obtener el literal en el ejemplo:

```
PREFIX wd: <http://www.wikidata.org/entity/>
SELECT DISTINCT ?prop3 WHERE {
  wd:Q190924 ?prop2 ?prop3 .
  FILTER isLiteral(?prop3)
} LIMIT 10
```

En este caso las diferencias son que en vez de una variable *var* que representa un nodo, tenemos otra variable con nombre *prop* al final del triple, esto es solo un cambio para una mejor comprensión humana en la interfaz, pero SPARQL sigue viendo una variable cualquiera. Para expresar que *?prop3* es un literal, se utiliza la linea ***FILTER isLiteral(?prop3)***, de esta forma evitamos tomar resultados de otros nodos.

Para el caso de MillenniumDB tenemos lo siguiente:

```
MATCH (Q190924)-[?e0 :?prop2]->(?prop3)
WHERE ?prop3 is string AND
 ?prop2 is not string
RETURN DISTINCT ?prop3
LIMIT 10
```

En este caso, se busca en la propiedad objetivo *prop3* que contenga un *string*, esto debido a que los literales almacenados en MillenniumDB están conformados de *strings*, y como se mencionó, se quiere evitar tomar los resultados no literales o IDs.

3.3.7. Conectando MillenniumDB con RDFExplorer

Una vez levantado los servidores de MillenniumDB y de RDFExplorer, se deben cambiar los códigos correspondientes a las búsquedas de las variables, para que se formen los comandos como se vio anteriormente, para esto se reutilizaron algunas partes del código original y se logró automatizar la formación de las consultas.

Luego de esto se debe resolver el problema de la base de Wikidata que utiliza MillenniumDB, esta base no tiene las etiquetas de las *IDs* de los nodos y propiedades. Para resolver este problema, se implemento una API proveída por Wikidata la cual permite obtener los recursos asociados a la ID de los datos. De los los datos recibidos por Wikidata, solo se utiliza lo necesario para que RDFExplorer continué funcionando normalmente. El resultado se ve expresado en la Figura 12.

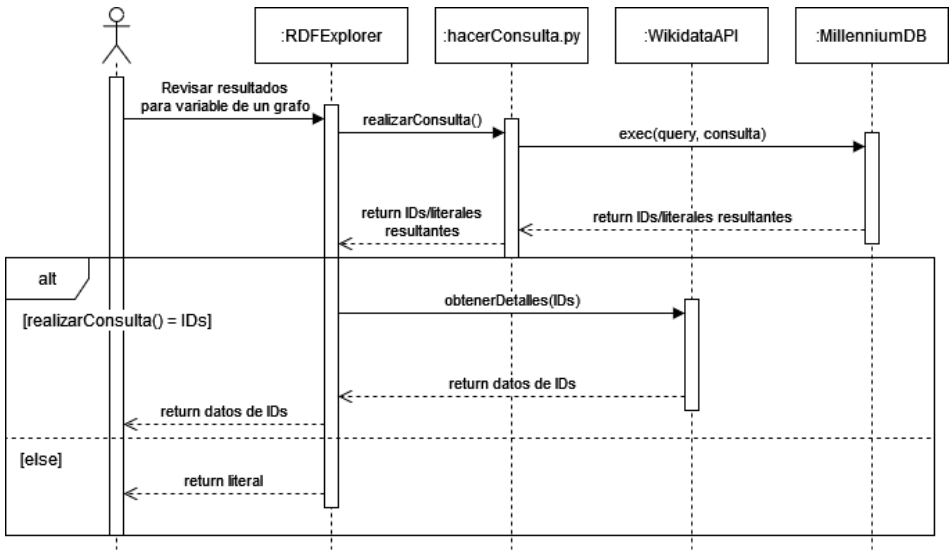


Figura 12: Diagrama de secuencia para las consultas en RDFExplorer con MillenniumDB. Si el resultado de la consulta es un literal, no se llama a la API de Wikidata.
Fuente: Elaboración propia

Con esto realizado, ya generamos las consultas originales de SPARQL de forma correcta y se obtienen los mismos resultados o similares según si existen en la BD reducida de MillenniumDB, con esto se logró adaptar la primera parte de MillenniumDB.

3.4. Implementación de las propiedades de arcos en RDFExplorer

Ya realizado los cambios anteriores, ahora se debe implementar las propiedades de los arcos o los llamados *Qualifiers* en Wikidata, para RDFExplorer. Esto es posible en MillenniumDB debido a la existencia de las anteriormente mencionados *eid* o *ID* del arco, y como se vio en los otros casos, tenemos las variables *"e"* en las consultas de MillenniumDB que no fueron utilizadas, las cuales representan la *ID* del arco.

Con la *ID* de los arcos, es posible consultar por las propiedades de un arco de Wikidata, sin embargo durante el trabajo se encontró un problema más allá de nuestras capacidades, la base de datos reducida de Wikidata esta construida para ser usada en SPARQL, por lo que los arcos no tienen sus *Qualifiers* o propiedad-valor del arco, por lo que no se pueden realizar pruebas con estos datos. Para la construcción de las siguientes consultas se utiliza la sintaxis conocida por la publicación de [Vrgoc *et al.*, 2021].

3.4.1. Consulta a las propiedades del arco

Desde este punto, las consultas realizadas por SPARQL a Wikidata no tienen como obtener los datos de un arco, si bien se puede realizar la consulta a los *Qualifiers* de los datos a traves de la API de consultas de Wikidata[4], esto requiere de consultas mucho más complejas.

Utilizando MillenniumDB estas consultas resultan más faciles, con el uso de las *IDs* de los arcos, pero debido a que la base de datos reducida utilizada no cuenta con conexiones a los arcos, no es posible obtener los *Qualifiers* a través de MillenniumDB. Además MillenniumDB al estar aun en desarrollo, al momento del trabajo no existe información para ingresar datos de prueba para obtener estos datos, debido a esto, lo que se presentará a continuación, esta basado principalmente en la forma que se realizan las consultas según lo visto en el trabajo de [Vrgoc *et al.*, 2021].

Para obtener los *Qualifiers* de los arcos, se debe mostrar en la interfaz un grafo que sea capaz de representarlo, para esto se crearon 2 conexiones nuevas que utilizan los mismos nodos y propiedades integrados en RDFExplorer, a lo que se le agrega una nueva referencia a su ”propiedad padre”, el cual corresponde a la propiedad de donde proviene el *Qualifier*.

3.4.2. Consulta Arco-Propiedad-Recurso

Se creó una nueva conexión para el caso de una consulta arco-propiedad-valor cuyo valor represente un recurso URI, para esto se utiliza la misma conexión regular que agrega una propiedad a un nodo en RDFExplorer, pero en este caso se crea desde la propiedad del arco al que queremos consultar, podemos observar esta opción para la propiedad en el menú de la Figura 13, en el cual se agregaron 2 funciones que no poseían. Para este caso se utiliza en la función *”New property”* o nueva propiedad. Normalmente esto crea un nuevo par propiedad-valor conectado a un nodo, en cambio en este caso se crea este par conectado al arco, representado por una conexión con la propiedad de este arco.

Al crear una nueva propiedad desde este menú, se genera una conexión propiedad-valor para el *Qualifier* como se muestra en la Figura 14, en el cual los nuevos nodos para el *Qualifier* se forman desde la propiedad del arco, obteniendo un resultado similar a la forma que se buscó realizar al inicio en la Figura 6.

Para obtener la consulta que funcione con el ejemplo mostrado, se necesita agregar una nueva función que permita hacer una consulta con el arco como nodo de origen, se utilizan las variables *e* vistas anteriormente en las otras consultas de MillenniumDB. Esta variable es utilizada solo para estos casos, ya que representa la *ID* del arco que se crea, y al utilizarlo como un nodo de la forma: $(e, propiedad, valor)$, permite consultar las propiedades del arco *e*.

[4]https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries#Working_with_qualifiers

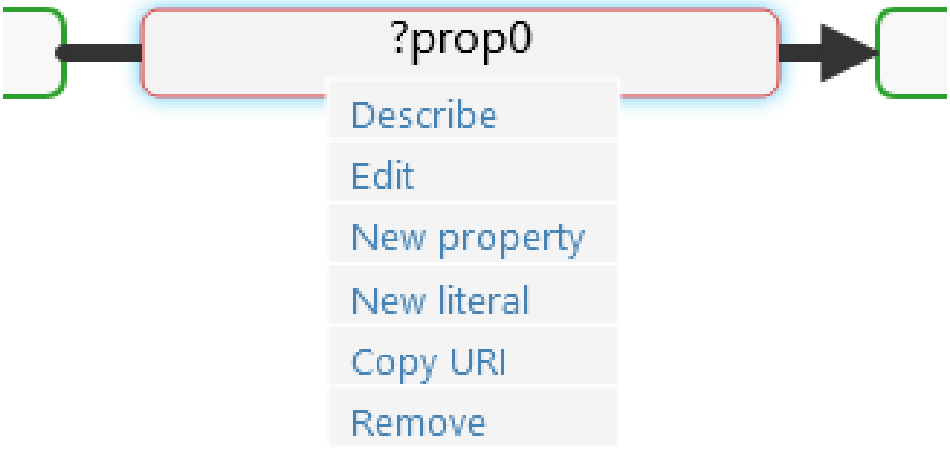


Figura 13: Menú de opciones para la propiedad de un triple.
Fuente: Elaboración propia

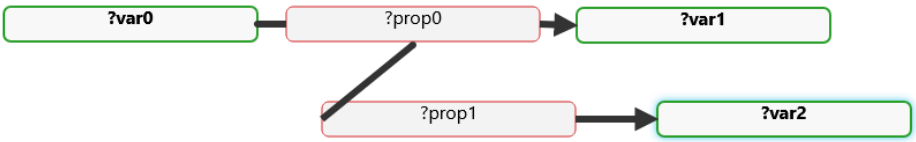


Figura 14: *Qualifier* creado desde el arco, genera una propiedad-valor generada desde el arco.

Fuente: Elaboración propia

Como RDFExplorer no puede formular la consulta a *Qualifiers* de Wikidata, al momento de generar un *Qualifier*, la propiedad-valor generada utiliza el nodo de origen del arco como nodo de origen. Para la Figura 14 esto significa que en la conexión $(arco, prop1, var2)$, en realidad el valor $arco$ es *var0* y genera la consulta $(?var0) - [?e0\ :?prop1] - > (?var2)$. Para arreglar este problema, al crear el *Qualifier* desde la propiedad del arco, la propiedad del *Qualifier* se le es asignado la propiedad del arco como "propiedad padre", lo que después permite reverenciarlo para obtener la *ID* del arco, para luego reemplazar el valor *var0* por *e*. Finalmente esta consulta queda de la siguiente forma:

```
MATCH (?var0)-[?e1 :?prop0]->(?var1),
 (?e1)-[?e0 :?prop1]->(?var2)
WHERE ?var2 is not string
AND  ?var0 is not string
AND  ?prop1 is not string
AND  ?prop0 is not string
AND  ?var1 is not string
RETURN DISTINCT ?var2
```

```
LIMIT 10
```

Esta consulta obtiene los posibles *IDs* de *var2*, esta consulta es igual para el caso de *prop1*, sin embargo como se mencionó, esto esta basado solo en la teoría y actualmente la base de datos no permite obtener resultados.

3.4.3. Consulta Arco-Propiedad-Literal

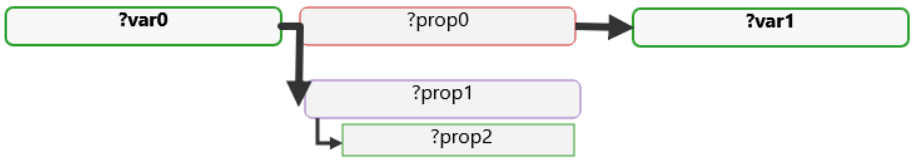


Figura 15: *Qualifier* creado desde el arco, genera una propiedad-valor generada desde el arco.

Fuente: Elaboración propia

Para el caso que el valor del *Qualifier* corresponda a un literal, se utiliza la función *"New literal"* desde la propiedad del arco como se ve en la Figura 13. Cuando se crea un nuevo *Qualifier* en el cual el valor del par propiedad-valor es un literal, sucede un proceso similar al mencionado en la solución anterior, en el cual para este caso se crea un literal asociado al arco, también se aplican los cambios en la consulta para obtener solo resultados que cumplan la condición para ser literales. En este caso sucede el mismo problema respecto al valor del nodo origen mencionado anteriormente, este problema se resuelve de igual manera, buscando cual es su "propiedad padre", extrayendo su *ID* de arco, y utilizándolo para arreglar la consulta. El grafo obtenido al generar el *Qualifier* se presenta en la Figura 15. La consulta realizada para este caso es la siguiente:

```
MATCH (?var0)-[?e1 :?prop0]->(?var1),
 (?e1)-[?e0 :?prop1]->(?prop2)
WHERE ?prop2 is string
AND  ?var0 is not string
AND  ?prop1 is not string
AND  ?prop0 is not string
AND  ?var1 is not string
RETURN DISTINCT ?prop2
LIMIT 10
```

La consulta obtenida es similar a la anterior, con la excepción del filtro mencionado, el cual pide que los resultados de *prop2*, nuestro objetivo, retorne solo *strings*. Nuevamente esto solo se basa en la teoría y al momento de realizar el trabajo no hay como comprobarlo.

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

Para la validación de esta solución se realizaran diferentes *tests* con el objetivo de comprobar que MillenniumDB funcione correctamente al momento de realizar las consultas y que RDFExplorer muestre los resultados correctamente, para esto desde la interfaz se deben mostrar los dos elementos mencionados.

Los elementos que se deben validar son las diferentes consultas, verificando que no existan errores o se manejen correctamente, para esto se realizara una lista de consultas distintas validando cada parte que puede ser consultada. También se debe verificar que las variables se actualicen correctamente al cambiar los datos del grafo y revisar que no hayan problemas si una consulta no tiene ningún resultado. Se debe considerar que los datos disponibles corresponden a los datos reducidos de Wikidata, por lo que los resultados pueden diferir un poco entre la interfaz RDFExplorer original y la adaptada para MillenniumDB. Los *tests* a realizar se presentan en la Figura 16 basado en las formas que se buscan en los *tests* de [Vargas *et al.*, 2019] y formas nuevas, con lo cual se busca demostrar que MillenniumDB funcione correctamente.

En los casos de *testeo* mostrados, los círculos indican un nodo o valor, las flechas representan propiedades y los círculos punteados representan el arco entre 2 nodos. Para cada caso se comprobara que cada parte del grafo obtenga un resultado, incluyendo si es posible los casos con literales, y luego comprobar si existen los casos a través de la página web de RDFExplorer[5].

Cabe destacar nuevamente los problemas surgidos por restricciones de la conexión entre la interfaz RDFExplorer y MillenniumDB, para lo cual se creó un programa en el servidor de MillenniumDB, que recibe las *queries* y realiza las consultas a MillenniumDB.

4.1. Resultados de los *tests*

Para cada caso se intento obtener una respuesta en cada nodo y propiedad, o en la Figura 16, cada ”circulo” o ”flecha”, cada uno comenzando solamente como una variable. Con cada consulta los valores entregados obtenian un valor que tuviese algun valor que logre satisfacer los grafos según las conexiones presentes, las excepciones son los *tests* 7 y 8, que veremos más tarde.

[5]https://rdfexplorer.org/

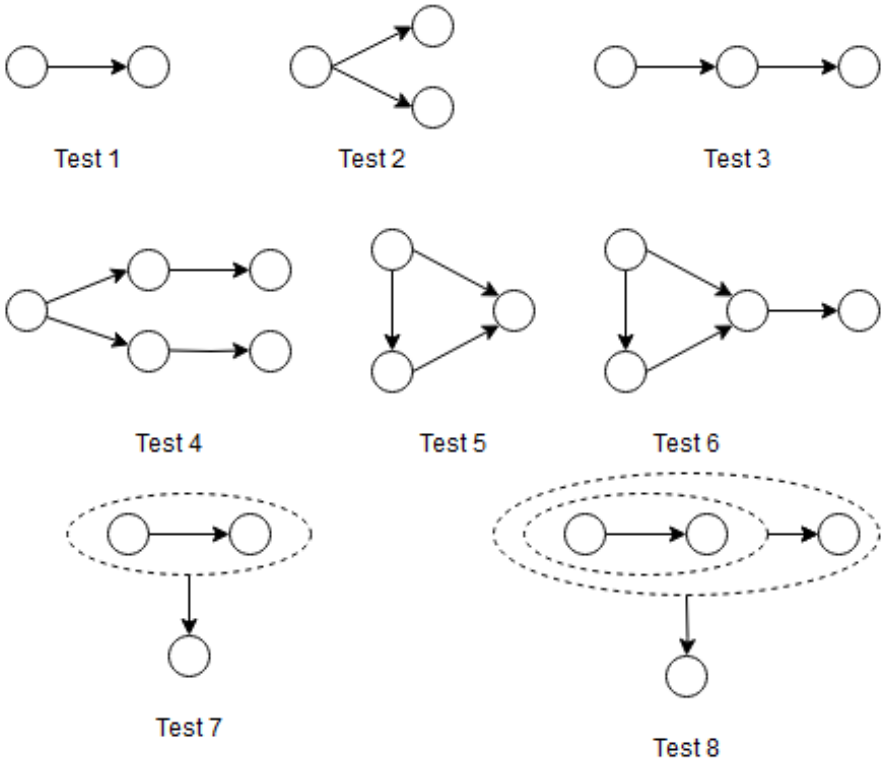


Figura 16: *Tests* para la validación.
Fuente: Elaboración propia

4.1.1. Test 1

El objetivo del primer *test* es comprobar la funcionalidad más básica de RDFExplorer en conjunto a MillenniumDB, esta es la base de todas las consultas que se realizan por lo que es lo más importante de que funcione, una vez verificado su buen funcionamiento, se puede proceder a crear grafos de mayor complejidad.

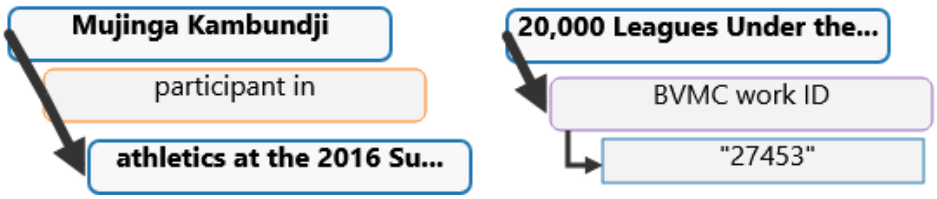


Figura 17: Grafo obtenido de *test* 1.
Fuente: Elaboración propia

La Figura 17 muestra el resultado obtenido donde todas las variables tienen un valor generado por los resultados de las consultas por MillenniumDB. Durante esta prueba se encontró

1 problema.

El problema encontrado durante la validación sucede cuando se realiza la consulta al literal cuando solo este es la variable. Cuando se realiza la *query* a MillenniumDB, este no entrega ningún resultado, pero si se cambia el estado de la propiedad o el nodo de origen a variable, el literal si encuentra resultados, esto se repite para todas las situaciones similares que sigan. El resto de los casos funcionan correctamente.

4.1.2. Test 2

Con el segundo *test* se busca verificar que las consultas funcionen con el contexto de otros valores obtenidos en distintos caminos, de esta manera podemos saber si efectivamente se consideran o no las otras variables.

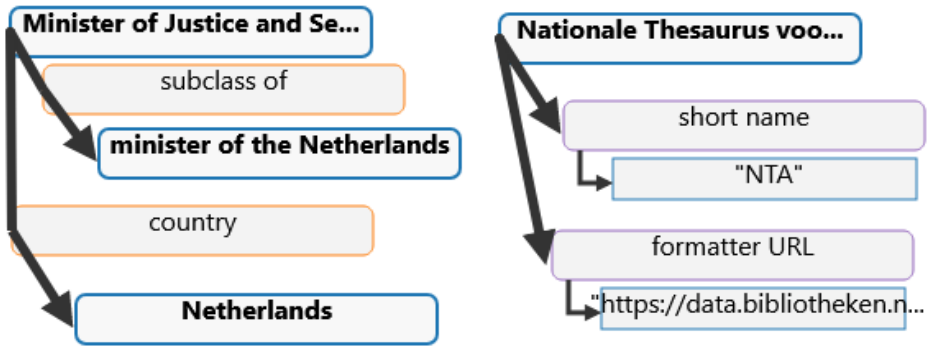


Figura 18: Grafo obtenido de *test* 2.
Fuente: Elaboración propia

En este caso se logra encontrar todas las consultas para cada caso, visible en la Figura 18, pero durante este *test* se logró encontrar un error en el que no se guardaban las etiquetas de las *IDs* buscadas, gracias a eso, ahora la interfaz es capaz de identificar cuando una *ID* no tiene etiqueta, reemplazando el valor por su *ID*.

Además se pueden observar lentitud en los procesos más complejos, lo cual también ocurre por parte de la página web de RDFExplorer al manejar muchos datos a la vez o cuando se actualizan los datos.

4.1.3. Test 3

Para este *test* se busca comprobar que funcionen caminos más largos, demostrando que los nodos de origen afecten en los resultados de los nodos con caminos más largos.

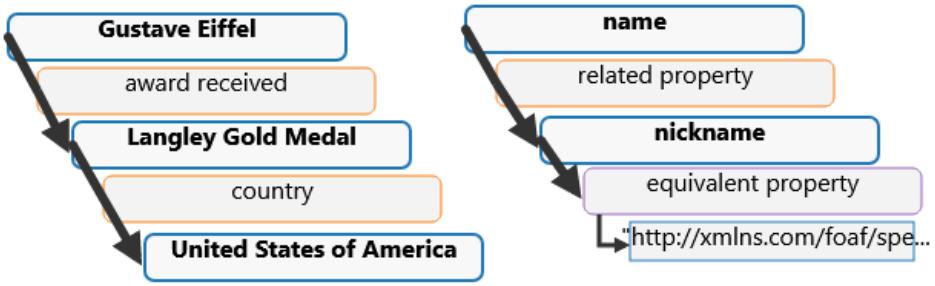


Figura 19: Grafo obtenido de *test* 3.
Fuente: Elaboración propia

Para el *test* 3 o la Figura 19 se lograron obtener valores a través de las *queries* para todas las conexiones sin ningún problema además de los ya antes mencionados.

4.1.4. Test 4

Con el *test* 4 se busca realizar una consulta más compleja juntando las ideas del *test* 2 y 3.

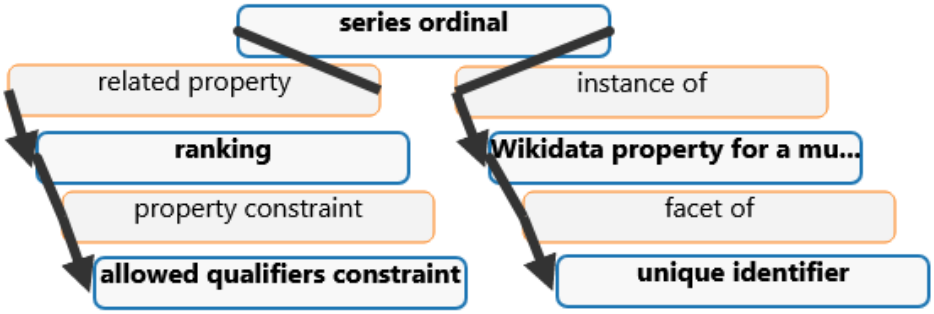


Figura 20: Grafo obtenido de *test* 4.
Fuente: Elaboración propia

Para el *test* 4 o la Figura 20 se lograron obtener valores a través de las *queries* para todas las conexiones sin ningún problema además de los ya antes mencionados.

4.1.5. Test 5

Con el *test* 5 se busca revisar la capacidad de realizar consultas donde existe mayor interacción entre pocos nodos.

En el *test* 5 o Figura 21 se nota más el tiempo de espera para realizar las *queries*, por lo que se puede asumir que a mayor complejidad, es más probable que se intensifiquen los problemas

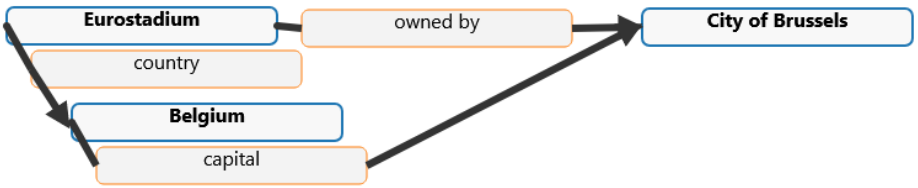


Figura 21: Grafo obtenido de *test* 5.
Fuente: Elaboración propia

de latencia.

4.1.6. Test 6

Similar al *test* 4, en este *test* se mezclan los *test* 5 y 3 para verificar el resultado de una consulta compleja, con resultados más dificiles de encontrar.

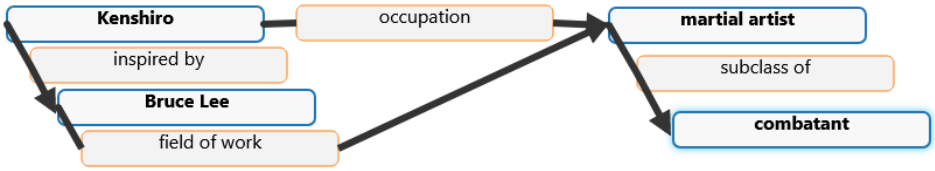


Figura 22: Grafo obtenido de *test* 6.
Fuente: Elaboración propia

En primera instancia en el *test* 6 o Figura 22, encontramos un error al intentar obtener el ”ultimo” valor de las conexiones (siguiendo las flechas). Esto permitió el arreglo de 1 problema propio de MillenniumDB, esto se debía a que ante la falta de algunos datos, MillenniumDB llena automáticamente nodos faltantes con nodos anónimos cuyo nombre se genera automáticamente, debido a esto, cuando buscamos los objetos de Wikidata a través de la API que utilizamos, este nodo anónimo interrumpe la búsqueda al no ser un código valido, y genera este error. Para arreglar esto, filtramos los resultados para eliminar los nodos anónimos, eliminando los resultados problemáticos.

Luego de arreglar los errores, se logra formar el grafo utilizando las *queries* sin tener otros problemas.

4.1.7. Test 7 y 8

Para el caso de los *tests* 7 y 8, se busca trabajar con la nueva implementación de MillenniumDB a traves de los *IDs* de los arcos, con los cuales se buscan los *Qualifiers* de un arco. En el caso del *test* 8 se busca un *Qualifier* dentro de un *Qualifier*.

Como no es posible obtener resultados de las *queries* de los *tests* 7 y 8, solo podemos teorizar los resultados que deberian obtenerse. Considerando la Figura 16, para el caso del *test* 7 deberian obtenerse las posibles propiedades o valores del *Qualifier* del arco, en cambio para el caso 8, lo que se quiere comprobar es si hay algun *Qualifier* dentro de otro, de ser asi este caso deberia entregarnos un resultado positivos, de lo contrario podemos conocer las limitaciones de las busquedas en Wikidata y en un futuro limitar los *Qualifiers* para reducir la complejidad de los grafos evitando la creación de los *Qualifiers* dentro de otros.

CAPÍTULO 5
CONCLUSIONES

5.1. Resultados de la implementación entre RDFExplorer y MillenniumDB

El trabajo realizado tuvo como objetivo principal la integración de la base de datos MillenniumDB a la interfaz de RDFExplorer, esta selección realizada principalmente por su simpleza y buenos resultados en el uso para nuevos usuarios, siendo una herramienta util para sacar provecho de las bases de datos. Además que el lenguaje de consultas de MillenniumDB resultó ser bien compatible a RDFExplorer debido a su similitud a SPARQL.

La integración de una base de datos nueva no es un trabajo simple como cambiar las consultas a algo que acepte la nueva BD, esto se debió principalmente a que RDFExplorer trabaja estrechamente con la API de Wikidata de consultas, la cual maneja internamente los datos y facilita mucho más la extracción de los datos, esto no fue fácil de adaptar debido a la separación de MillenniumDB con Wikidata, ya que MillenniumDB no trabaja con toda la información de Wikidata en una consulta, por lo que se debieron realizar pasos extras para incluir los datos faltantes a traves de otra API de Wikidata, la cual solicitó información que debe ser manejada cautelosamente para no tener errores y retornar los datos pedidos correctamente.

Aparte de los problemas de la integración con Wikidata para integrar MillenniumDB a RDFExplorer, se tuvieron que resolver muchos problemas propios de la base de datos, debido a que aun se encuentra en desarrollo, MillenniumDB carece de varias funciones e incluso es difícil encontrar la información necesaria para hacerlo funcionar correctamente, esto resulto en extensas búsquedas con pruebas y errores cuya solución no fue posible encontrar en linea. Debido a esto soluciones que pudiesen parecer triviales tomaba más tiempo del necesario, relantizando el trabajo e incluso llevando a la posibilidad de resolver un problema incorrectamente para después tener que abordar el problema de nuevo.

Otra dificultad al momento de cambiar las bases de datos es el como realizar las consultas, más allá de como formar las *queries*, para trabajar con el sistema que incorpora las *IDs* de los arcos, las consultas no se ejecutaban de una forma convencional debido a que es necesario mandar un comando a través de una terminal, y entre los servidores no es posible enviar esta ejecución al terminal, comando negado por la posibilidad de vulnerar la seguridad del servidor, lo cual llevó la integración del código intermedio, que debió manejar las interacciones entre la interfaz y la base de datos, como se mencionó anteriormente, fue difícil encontrar soluciones en linea que se pudiesen aplicar a la BD en desarrollo para este caso en concreto.

Además de los ajustes en la base de datos, RDFExplorer presentó otras dificultades respecto a la adaptación de la interfaz, este trabajo implicó la modificación importante de partes del código original que componía RDFExplorer, esto demostró bastante dificultad y esfuerzo en

la comprensión de un código no escrito por uno, donde fue importante probar que realizaba cada cambio en la estructura del código. Debido a esto, también existe un limite en el que algunos cambios podían ser muy grandes y cambiar completamente el esqueleto del código, esto significó que además de comprender y cambiar el código, se debe intentar hacer lo posible por mantener los datos que se manejan en el mismo estado que se utilizan en otras partes, limitando la capacidad para los cambios a realizar, esto se puede ver por ejemplo, en el hecho de que un nodo con su propiedad siguen estando intrínsecamente unidos y la posición del nodo en la interfaz afecta al de la propiedad, u otro caso donde el literal no se puede separar de la propiedad sin realizar muchos cambios en la estructura del nodo, la propiedad y el mismo literal.

Otro problema durante la realización de este trabajo, surge en los extras que tenía originalmente RDFExplorer, como la capacidad de filtrar las variables con un buscador o filtros para la consulta, todos estos casos de limitaciones donde se tuvieron que cambiar o directamente eliminar funcionalidades de RDFExplorer son debido a que la adaptación de una interfaz a algo para lo que no fue construido originalmente, restringe muchas funcionalidades originales por la falta de un sistema que pueda manejarse con MillenniumDB.

5.2. Objetivos realizados

Durante el transcurso de este trabajo, el objetivo general fue ”Adaptar la interfaz existente de RDFExplorer a MillenniumDB para la visualización de datos”, para lo cual se debió analizar el modelo de MillenniumDB previo a su implementación, analizar el lenguaje de consultas que utilizaba MillenniumDB, identificando sus operadores más importantes y finalmente implementar la interfaz para representar el modelo de datos de MillenniumDB, para lo cual fue utilizado RDFExplorer.

El modelo de datos de MillenniumDB fue principalmente analizado durante el marco teórico antes de comenzar el trabajo en RDFExplorer, dejando los problemas del análisis del lenguaje de consultas y su implementación para el proceso de integración.

Durante el trabajo se investigó el uso de las consultas de MillenniumDB, analizando la estructura de las consultas para su implementación en RDFExplorer, esto resulto mayormente bien a excepción de algunos *bugs* para casos específicos. Los principales problemas se encuentran en la búsqueda de los datos para los pares propiedad-valor de los arcos, debido a la falta de una base de datos que satisfaga tal consulta. Debido a esto no podemos verificar satisfactoriamente que se haya implementado una representación visual del modelo de datos de MilllenniumDB para este caso en especifico.

5.3. Efectos de la implementación

Con esta primera implementación, el objetivo principal fue establecer una interfaz que facilite el uso de MillenniumDB. Esta interfaz esta diseñada para permitir la realización y prueba de distintas consultas de manera sencilla y comprensible. La elección de RDFExplorer como plataforma se basa en los estudios que demuestran ser una interfaz más amigable, especialmente para usuarios nuevos en el sistema. Con esta implementación también se busca fomentar su uso al ofrecer una interfaz más intuitiva y accesible. Con esto se facilita el descubrimiento de las funcionalidades de MillenniumDB, lo que podría despertar el interés de usuarios y permitirles explorar a fondo las capacidades de esta BD. Además, al utilizar una interfaz que ha demostrado ser más amigable, se espera que los usuarios se sientan más cómodos para realizar pruebas y experimentar con distintas consultas, facilitando así la adopción y comprensión de las características de MillenniumDB.

Al integrar MillenniumDB con la interfaz RDFExplorer, se espera no solo facilitar la experiencia de los usuarios finales, sino también ayudar en el proceso de desarrollo para los investigadores. Esta integración proporciona un entorno de fácil acceso para *testear* consultas y las funcionalidades de MillenniumDB. Los desarrolladores pueden utilizar RDFExplorer para realizar pruebas, lo que les permite experimentar con distintas consultas y funciones. Esto puede agilizar el desarrollo al ofrecer una herramienta práctica para verificar el funcionamiento de las características implementadas, identificar posibles errores y realizar ajustes de manera más sencilla.

5.4. Trabajo a futuro

Debido a las limitaciones mencionadas, actualmente no es posible la verificación completa del funcionamiento de MillenniumDB en RDFExplorer, por lo que se debe encontrar alguna forma de integrar en la base de datos arcos que contengan propiedades, de esta manera se pueden revisar las consultas sin solución actual. También deben arreglarse los *bugs* encontrados durante el trabajo, que si bien no afectan al trabajo en su funcionamiento, afectaran a futuro en la satisfacción del uso de RDFExplorer con MillenniumDB. Los *bugs* y mejoras que pueden ser implementados son los siguientes:

- Arreglar las conexiones de los literales, ya que bajo ciertas condiciones, al crear una segunda propiedad con valor literal, este no crea la flecha que los conecta, también en el caso de que el valor literal de un triple $(arco, propiedad.valor)$ con valor literal, la flecha es distinta al caso de un valor del tipo *ID* de Wikidata.
- Mejorar la independencia en el movimiento de las propiedades, las cuales se arrastran en conjunto a los nodos de donde se originan.
- Agregar independencia del movimiento en literales, los cuales no solo están ligados a la propiedad que los origina, sino que están ligados a su nodo que origina el triple. Este

caso debe verse con mayor detalle dependiendo si puede o no ser mejor que estén unidos con la propiedad, ya que desde un literal no se puede realizar otra relación desde o hacia este, por lo que mantenerlo estrictamente unido a su literal.

- Optimizar el código que crea las flechas, ya que actualmente realiza varias operaciones similares que difieren segun del origen y objetivo de la flecha.
- Puede existir la posibilidad de obtener el nombre correspondiente a las *IDs* invocadas desde la misma base de datos de MillenniumDB, pero esto requiere mayor investigación, trabajo y manejo de los resultados para satisfacer los requerimientos de RDFExplorer al retornarlos para evitar tener que rehacer mucho código.

Como podemos ver, existen muchas mejoras que se pueden realizar y problemas que no se pueden solucionar con la situación actual de la base de datos, sin embargo esto no significa que en un tiempo más pueda estar implementado un sistema o se pueda trabajar en alguna forma de extraer los datos de Wikidata de tal manera que podamos transformarlos en algo más util para MillenniumDB, implementando *IDs* para los arcos y sus respectivos pares propiedad-valor o *Qualifiers*.

ANEXOS

Anexo A: Sintaxis de las consultas de MillenniumDB

Este anexo aborda la sintaxis del lenguaje de consultas de MillenniumDB, basado en el escrito por [Vrgoc *et al.*, 2021] con algunos cambios para las consultas utilizadas. El lenguaje de MillenniumDB esta basado principalmente en los lenguajes de consultas para grafos SPARQL y Cypher. La estructura general de las consultas de MillenniumDB utilizadas en este trabajo es la siguiente:

```
MATCH MatchPattern
WHERE Condition
RETURN RetOptions
LIMIT Number
```

Para este caso, comenzamos formando la estructura del grafo que buscamos, que puede ser un patrón de grafo básico o navegacional utilizando MATCH. Usamos WHERE para filtrar los resultados, restrigiendo y limitando los valores que pueden tomar las variables que utilizamos en MATCH. RETURN es donde seleccionamos las variables cuyos resultados queremos obtener, en este trabajo se trabaja con un resultado, pero se pueden obtener combinaciones de resultados en caso de ser necesario. LIMIT limita la cantidad de resultados obtenidos segun el numero que seleccionamos, principalmente para evitar consumir muchos recursos para obtener todos los resultados posibles.

Para entender bien el funcionamiento de la sintaxis del lenguaje de consultas de MillenniumDB, utilizamos el ejemplo de un grafo básico del mismo trabajo:

```
MATCH (Q190924)-[?e0 :P1000]->(?var1)
WHERE ?var1 is not string
RETURN DISTINCT ?var1
LIMIT 10
```

Evaluando esta consulta, en MATCH tenemos las *IDs* Q190924 que representa a Michael Johnson, y P1000, que representa la propiedad record sostenido". Por lo tanto lo que se quiere calzar, es que records sostiene Michael Johnson, los cuales se almacenan en la variable *var1*, además la *ID* de los posibles arcos son almacenados en *e0*. La clausula WHERE limita los valores de la variable *var1* a valores que no sean *strings*, que corresponden a valores literales. Los valores que no son strings son aquellos que tienen URIs, a los cuales nos conectamos a traves de su *ID*. La clausula RETURN DISTINCT retorna todos los resultados de

var1 que sean diferentes entre si, de esta manera evitamos la repetición de resultados que puedan causarse. Finalmente con la clausula LIMIT limitamos los resultados a solo 10, de esta manera evitamos relantizar el proceso de búsqueda como se mencionó anteriormente.

REFERENCIAS BIBLIOGRÁFICAS

[Arenas *et al.*, 2007] Arenas, M., Gutierrez, C., y Pérez, J. (2007). An extension of sparql for rdfs. pp. 1–20.

[Miller, 2001] Miller, E. J. (2001). An introduction to the resource description framework. *Journal of library administration*, 34(3-4):245–255.

[Ora, 1999] Ora, L. (1999). Resource description framework (rdf) model and syntax specification. *http://www. w3. org/TR/REC-rdf-syntax/*.

[Vargas *et al.*, 2019] Vargas, H., Buil-Aranda, C., Hogan, A., y López, C. (2019). Rdf explorer: A visual sparql query builder. pp. 647–663.

[Vrandečić y Krötzsch, 2014] Vrandečić, D. y Krötzsch, M. (2014). Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85.

[Vrgoc *et al.*, 2021] Vrgoc, D., Rojas, C., Angles, R., Arenas, M., Arroyuelo, D., Aranda, C. B., Hogan, A., Navarro, G., Riveros, C., y Romero, J. (2021). Millenniumdb: A persistent, open-source, graph database. *arXiv preprint arXiv:2111.01540*.