Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Sistemas Computacionales

Sumarización de grafos basada en consultas

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
MAESTRA EN SISTEMAS COMPUTACIONALES

Presenta: **ING. PABLO CÉSAR GUZMÁN CURIEL**

Asesor **DR. LUIS FERNANDO GUTIÉRREZ PRECIADO**

Tlaquepaque, Jalisco. diciembre de 2020

AGRADECIMIENTOS

Quiero agradecer al Instituto Tecnológico y de Estudios Superiores de Occidente (ITESO) por los recursos proporcionados para el desarrollo de este trabajo de obtención de grado. Adicionalmente, al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico recibido a través de la beca número 640699 y al alumno Genaro de La Campa por apoyar en el desarrollo de la interfaz gráfica con D3.

DEDICATORIA

Dedico este trabajo de obtención de grado a mi familia y amigos.

RESUMEN

Con la gran cantidad de datos generada en la actualidad, redes sociales, portales de ventas, los analistas de información han puesto el interés en el análisis de información de big data, la cual puede ser representada en grafos, pero surge el problema de ser visualizada de manera adecuada para tomar acciones o entender la tendencia respecto a la información del grafo.

Para dicho problema, en el trabajo propuesto se desarrolla una herramienta interactiva en la cual el usuario pueda definir sus agrupaciones o sumarizaciones para visualizar y analizar patrones o tendencias. Dicha herramienta representa un grafo sumarizado que facilita el análisis y permite realizar acciones de manera interactiva sobre supernodos.

Para visualizar los datos sumarizados por medio de consultas se utilizaron dos bases de datos, una de una de películas y otra de Twitter, en las cuales se realizaron diferentes consultas para observar las agrupaciones y las relaciones de las agrupaciones previamente definidas, también de forma dinámica realizar operaciones sobre dichas agrupaciones.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE ACRÓNIMOS Y ABREVIATURAS

REST	Transferencia de estado representacional (en inglés *representional state transfer*)
RESTful	Servicios Web que se ajustan al estilo de arquitectura *REST*
JSON	Notación de objetos de JavaScript (en inglés *JavaScript Object Notation*)
API	Interfaz de programación de aplicaciones (en inglés *application programming interface*)
SQL	Lenguaje de consulta estructurada (en inglés *Structured Query Language*)
UI	Interfaz de usuario (en inglés *User Interface*)
ID	Identificador
ACID	Atomicidad, Consistencia, Aislamiento y Durabilidad (en inglés *Atomicity, Consistency, Isolation and Durability*)
Framework	Esquema o entorno de trabajo para el desarrollo de software
Plug-in	Componente de software que añade una función especifica a un programa existente
MCL	Algoritmo para clusters en grafos (en inglés *Markov cluster agorithm*)

1. INTRODUCCIÓN

Los datos masivos generados por empresas o aplicaciones requieren de un análisis rápido y eficiente para dar solución a sus necesidades, ya sea una empresa de *marketing*, tienda en línea o una aplicación de red social, estas tienen la necesidad de ver con claridad los patrones y comportamiento de sus datos, de tal manera puede, remediar problemas o puede aplicar estrategias de negocio para cambiar el comportamiento de dichos patrones.

Las herramientas como visualización de grafos vienen a ayudar a los analistas o expertos en *big data* a tener una representación gráfica de los datos recabados, pero simplemente ver los grafos en gran escala no es de gran ayuda. Necesitan aplicar algoritmos para que la información contenida en el grafo les muestre los patrones y comportamiento que ellos desean visualizar de una manera fácil y rápida. Para este problema el presente trabajo propone una herramienta interactiva el la cual un usuario o analista pueda definir sus agrupaciones o sumarizaciones para realizar los análisis correspondientes.

1.1. Justificación

La cantidad de datos generados por una empresa puede ser visto como un problema o una solución a sus necesidades.

Solución por que es posible tomar decisiones de acuerdo a la información obtenida o problema ya que por la gran cantidad de datos que se generan por segundo si no se gestionan de manera adecuada dificulta su análisis. Los individuos o empresas al no analizar bien esta información caen en el riesgo de no tener retroalimentación y acciones inmediatas para la toma de decisiones.

Existen herramientas populares como VAST-VIGOR, TULIP o GEPHI las cuales no cuentan con una interfaz fácil de usar que permita sumarizar y visualizar adecuadamente un grafo con miles o millones de nodos y relaciones.

Con la generación de información masiva de grafos, y un grafo teniendo una gran cantidad de nodos y relaciones, se dificulta ser manejable para su visualización y análisis.

1.2. Objetivo del Proyecto

El objetivo es desarrollar una herramienta que facilite analizar información masiva de grafos a través de la sumarización, permitiendo identificar patrones y relaciones a macro escala basadas en consultas a la base de datos neo4j.

La herramienta permite al usuario la construcción de un grafo sumarizado a partir de consultas definidas por el mismo. Se asume que el usuario conoce previamente la estructura del grafo y que el mismo ya está cargado en la base de datos Neo4j.

Sin esta herramienta de sumarización basada en consultas, sería sumamente complejo agrupar por características personalizadas y los analistas estarían perdidos en un mundo de datos. Por ello se necesita tener un conocimiento del dominio sobre el cual se harán las sumarizaciones para que la información representada tenga sentido y significado.

1.3. Propuesta

La propuesta es una herramienta que realice los cálculos necesarios para construir supernodos y súper aristas que son generados a partir de las consultas definidas, y a su vez mantener en memoria el estado del grafo sumarizado.

Una vez cargado el grafo sumarizado en la herramienta se pueden realizar funciones de agregación a los supernodos, generar un histograma de un supernodo, desagrupar un supernodo por atributo, y ver la intersección entre nodos.

Con una herramienta de sumarización basada en consultas y teniendo en Neo4j un grafo con miles de nodos, los analistas de datos pueden explorar, hacer consultas en tiempo real. Los analistas tienen una perspectiva visual de las tendencias o comportamiento de los datos, de esta misma manera al permitir observar dichos patrones y comportamiento pueden tomar decisiones o acciones inmediatas para influir en un cambio de comportamiento de sus datos de ser necesario.

2. ESTADO DEL ARTE O DE LA TÉCNICA

Para el análisis de información existen herramientas que permiten visualizar información de distintas formas, entre ellas encontramos, VIGOR, TULIP, Gephi, y el propio neo4j.

2.1. VIGOR

VIGOR, es una herramienta que se compone de 4 áreas principales, la vista ejemplar, donde se visualiza la consulta textual realizada por el usuario y que soporta filtrado por valor, ver Figura 1.

En la vista de gráfico de fusión se despliega el subgrafo resultante de la consulta realizada por el usuario. Además, da una representación rápida de cuáles nodos aparecen con más frecuencia. La vista embebida de subgrafo es la que sumariza todos los resultados reduciéndolos a un clúster coloreado basado en características similares. El explorador de características que despliega la distribución de cada tipo de nodo incluido en los resultados [1].

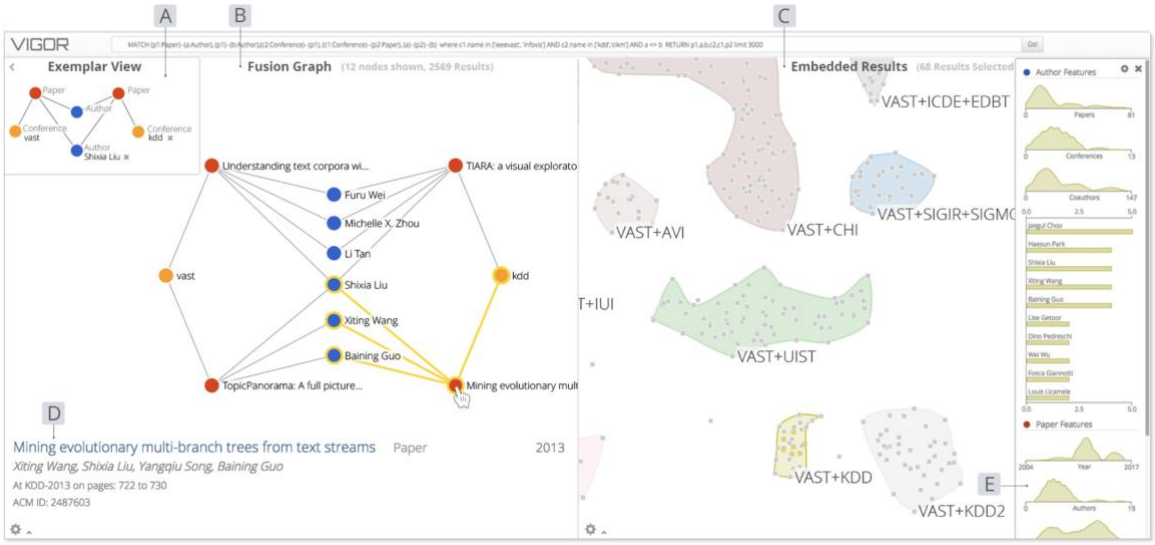

Figura 1: Interfaz gráfica de VIGOR [1].

2.2. Gephi

Gephi es un software de fuente abierta líder en visualización y exploración de grafos. El propósito de la herramienta es ayudar a los usuarios a descubrir patrones e hipótesis en filtrados y rutinas de visualización [2].

Respecto al análisis post-visualización, se pueden descargar plug-ins para clustering tipo MCL y se puede incorporar visualización para mapas geográficos con GeoLayout. Usuarios pueden visualizar como evoluciona un grafo en el tiempo manipulando la línea de tiempo embebida en gephi [3].

Gephi soporta casi todos los tipos de gráficas de redes las cuales incluyen redes complejas, redes jerárquicas, redes dinámicas y redes temporales. En la Figura 2 podemos observar un ejemplo de un grafo y sus relaciones en Gephi. Gephi incluye funciones listas para usarse, como visualización en tiempo real,

algoritmos de diseño que permiten darle forma al grafo y métricas (camino mas corto, coeficiente de agrupación, modularidad etc.) [4].

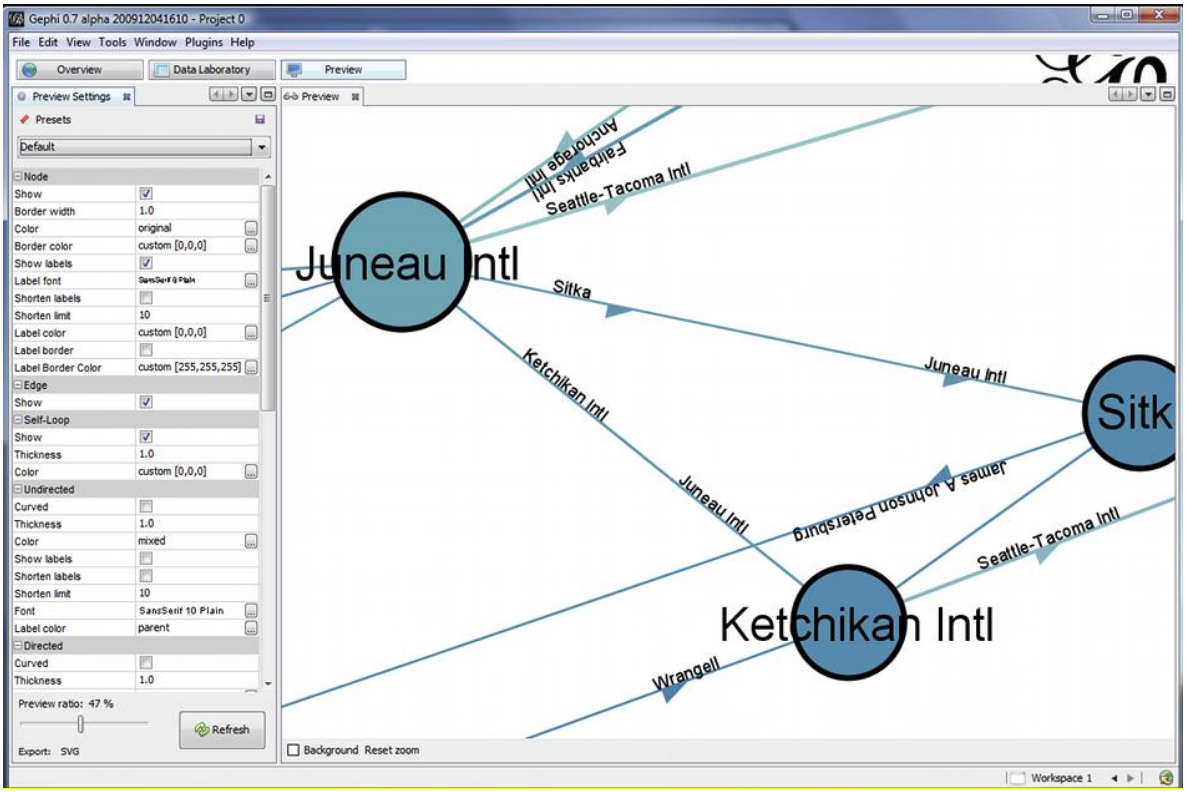

Figura 2: Interfaz gráfica de Gephi [2].

2.3. TULIP

Es una herramienta que permite navegar de manera eficiente por jerarquías de grafos y grafos anidados, a bajo nivel TULIP soporta la creación de grafos anidados o sub-grafos traslapados. TULIP permite manipular data sets que consisten en relaciones y entidades que son guardadas en memoria. Las características clave de TULIP son [5]:

- Algoritmos para probar si un grafo es plano o calcular la cuantificación uniforme de un set de valores [5].
- Minimizar la cantidad de memoria usada mientras se provee operaciones en un data set. Soporta un número no restringido de atributos en elementos de un grafo. Incluye funciones de agregación a entidades o relaciones, en análisis de post-visualización encontramos algoritmos para elementos

bi-conectados o fuertemente conectados o algoritmos para encontrar árboles que se extienden o anidados [5].

- Para clustering de sus nodos usa algoritmos como “Lovian” que identifica subgrupos que están fuertemente conectados. También usa un plugin llamado “Equal value” para formar subgrupos de nodos con el mismo valor a cierta propiedad en un subgrupo [5].

En la Figura 3 observamos un ejemplo de la interfaz grafica de TULIP.

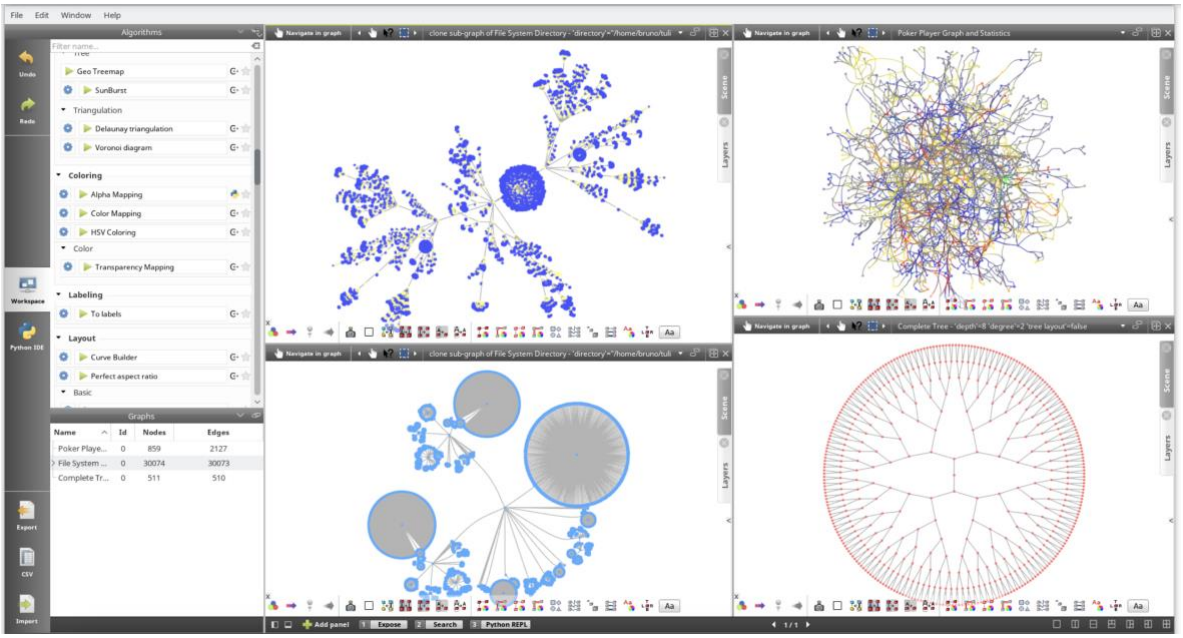

Figura 3: Interfaz gráfica de TULIP [5].

2.4. Neo4j

Los nodos y aristas virtuales no existen en el grafo de neo4j, estas solo son regresados por una consulta y pueden ser usados para representar solo una proyección del grafo.

Solo pueden ser usados para visualizar información, como por ejemplo agregar relaciones en una sola, o colapsar nodos intermedios en una relación.

Se pueden combinar entidades reales o virtuales, como ejemplo crear una relación virtual entre dos nodos reales o crear una relación virtual de un nodo real a uno virtual.

Consideraciones especiales de nodos y aristas virtuales es que tienen IDs negativos y no se pueden hacer consultas sobre los nodos virtuales [6].

Como podemos ver en el ejemplo de Figura 4 se muestran las consultas para nodos y relaciones virtuales en neo4j.

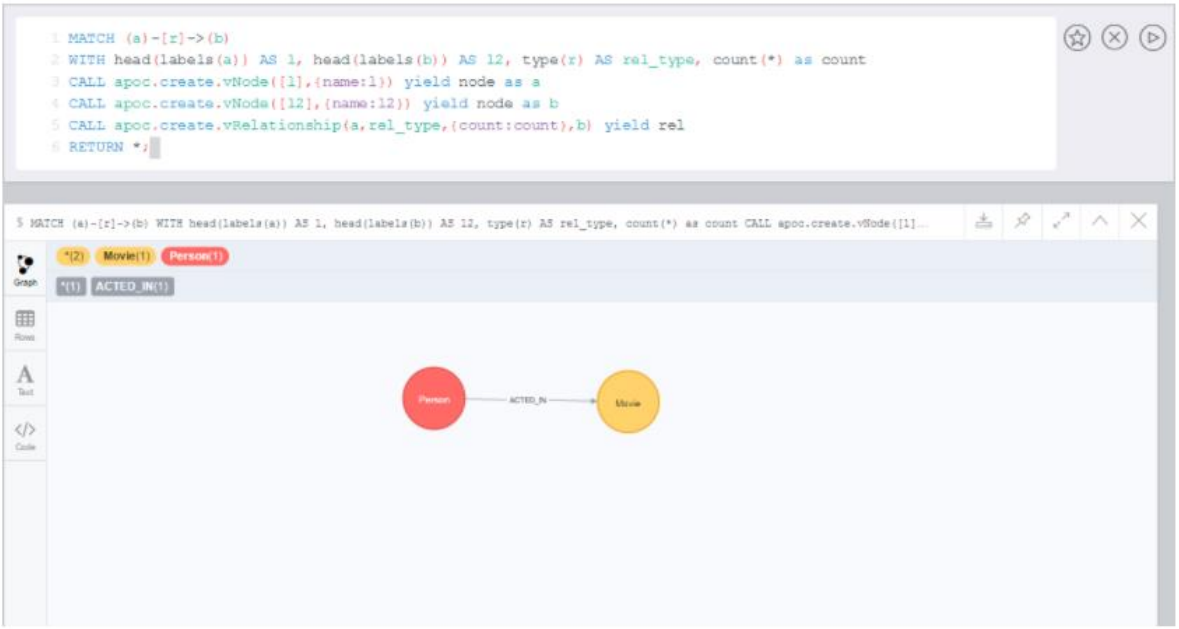

Figura 4. Ejemplo de nodo y relación virtual. [7]

Observamos que VIGOR es que está enfocada a desplegar información y agrupamiento de similitudes de los nodos, pero no se despliega información de las aristas en la vista ejemplar, ni en el gráfico de fusión y el punto de entrada solo este habilitado para recibir una consulta especificando el tipo de nodos, no se puede tener nodos mixtos ni múltiples consultas.

Por otro lado, encontramos TULIP, que nos permite navegar por grafos jerárquicos y grafos anidados, TULIP puede hacer agrupaciones de nodos por propiedades similares, pero al igual que VIGOR, en los gráficos no podemos observar una clara agrupación de las aristas y tampoco se muestra una interfaz donde de manera interactiva se pueda agrupar por combinación características complejas que el usuario quiera definir.

Gephi puede trabajar con información en tiempo real con ayuda de APIs de streaming y puede hacer clustering con la ayuda de plug-ins, pero no se puede realizar una sumarización en tiempo real conectado a una base de datos ya que trabaja con información previamente cargada o proveída por medio de streaming.

En Neo4j encontramos que la UI nativa por si sola no nos proporciona una herramienta para realizar agrupaciones, estas se tienen que realizar por medio de nodos y aristas virtuales, el usuario tiene que conocer previamente las relaciones que existen y construir la sumarización del grafo por medio de consultas haciendo de esto un proceso complicado, terminando con consultas muy extensas y difíciles de comprender.

Por lo tanto, es clara la necesidad de una herramienta que incluya sumarización basada en múltiples consultas donde se pueden tener supernodos heterogéneos, visualizar peso y etiquetas de súper aristas, operaciones interactivas como reagrupamiento de supernodo por atributo, histograma, intersecciones entre supernodos y operaciones de agregación.

Antes de continuar con la propuesta necesitamos revisar los conceptos requeridos para comprenderla.

3. MARCO TEÓRICO/CONCEPTUAL

Para comprender la solución propuesta es necesario adentrarnos con los siguientes conceptos, *Grafo, neo4j. cypher, sumarización, supernodo y super aristas.*

3.1. Grafo

Un **grafo** es una representación de datos con dos distintos elementos, nodos y arista. Nodo o también conocido como vértice, representa el concepto de una entidad o dato, mientras la arista, representa una relación o enlace de un nodo a otro. Los vértices ilustran las relaciones que existen entre los datos. Es importante comprender las características de los nodos y aristas como la etiqueta o tipo de un nodo. Las etiquetas de un nodo son objetos que son semánticamente homogéneos, un nodo representa una clase de objetos que comparten mismo tipo de relaciones y atributos. La etiqueta de una arista nombra el tipo de relación entre los nodos en el esquema de la base de datos. Así como las etiquetas los nodos contienen propiedades o también conocidos como atributos, una propiedad o atributo describe las características de una etiqueta de nodo, tales como nombre, fecha, u otra característica descriptiva. Una arista dirigida va en una dirección de un nodo a otro, y una arista bidireccional va en ambas direcciones entre los nodos [8].

Como los nodos una arista o relación puede tener propiedades, en la mayoría de los casos estas propiedades son medidas cuantitativas como pesos, costo, distancia, rating, intervalos de tiempo, o fuerza. En neo4j las relaciones son direccionales [9].

3.2. Almacenamiento y consulta de grafo

Neo4j, es una base de datos que cumple con los principios ACID de las bases de datos relacionales, pero con un sistema nativo de almacenamiento y procesamiento para grafos. Está escrita en Java y es accesible para software escritos en otros lenguajes a través de lenguaje de consultas cypher por medio de un endpoint transaccional http o por medio del protocolo binario “bolt [10].

Hacer una consulta a neo4j usando APIs de java puede ser tedioso, se necesita visitar toda la gráfica y evadir nodos que no son iguales a lo que se está buscando. Cambiar la consulta consiste en rehacer el código, cambiarlo y construirlo de nuevo, la razón es que se está usando un lenguaje imperativo para hacer la búsqueda del patrón, lenguajes imperativos no funcionan adecuadamente para esta tarea. **Cypher** es un lenguaje declarativo de consultas usado para base de datos Neo4j. Declarativo significa que se enfoca en aspectos del resultado en vez de métodos o maneras de obtener el resultado, dado que es expresivo y legible para humanos [11] .

Cypher es la herramienta más poderosa para la consulta de Neo4j. Cypher fue diseñado para atacar el balance correcto para ser declarativo y fácil para usuarios que vienen de un entorno de SQL [12].

Cypher tiene una similitud con las consultas de SQL, que consiste de cláusulas, palabras clave y expresiones como predicados y funciones, ejemplo: **WHERE, ORDER BY, SKIP LIMIT, AND, p.unitPrice > 10.** A diferencia de SQL, cypher se trata de expresar patrones del grafo, donde se añade una cláusula **MATCH** para hacer coincidir esos patrones en tus datos [13].

A continuación, mostramos un ejemplo sencillo de una consulta con una relación.

Como se observa en el siguiente ejemplo cuando se conoce el tipo de relación y se quiere hacer una consulta que coincida, se especifica el tipo de relación en la consulta.

MATCH (wallstreet:Movie { title: 'Wall Street' })<-[:ACTED_IN]-(actor) RETURN actor.name

Regresa todos los actores que **ACTED_IN** 'Wall Street' [14].

3.3. Sumarización

La sumarización de un grafo es un método valioso para el procesamiento en memoria de grandes grafos, comprime un grafo de gran escala a uno compacto y mantiene las propiedades subyacentes del grafo. Dicha versión compacta facilita un eficiente procesamiento de varios fenómenos de la vida real identificando comunidades, marketing viral y análisis de tendencias [15].

El objetivo principal de la sumarización es de reducir el tamaño de la entrada de datos de grafo para que su análisis pueda ser realizado eficientemente. Las técnicas de sumarización se encuentran con el reto de procesar grandes cantidades de datos. La sumarización requiere de la extracción de información importante o de interés, sin embargo, la definición de interesante por sí misma es subjetiva y el identificar información interesante por lo general requiere un conocimiento del dominio, así como las preferencias del usuario. El límite entre interesante y poco interesante puede ser difícil de determinar de una manera basada en principios; por lo general se decide tomando en cuenta la compensación entre tiempo, espacio e información conservada en la sumarización, así como la complejidad de soluciones cartográficas obtenidas de la sumarización [16].

3.4. Supernodo y súper arista

Particionar, agrupamiento o detección de comunidades se usa para obtener una representación de un grafo al agregar todos los nodos que pertenece a una comunidad a un nodo virtual llamado **supernodo** y enlazarlos con **súper-aristas** con la suma del peso de las aristas originales [16].

Con estos conceptos es posible comprender la arquitectura del API de sumarización propuesta.

4. Modelo Propuesto

Como se había mencionado previamente hace falta una herramienta que tenga una agrupación del grafo basado en características personalizadas, reagrupación de supernodos basado en atributos, operaciones de agregación en supernodos e información de intersecciones entre supernodos, con este objetivo proponemos una API con las características mencionadas.

4.1. Modelo de sumarización basado en consultas

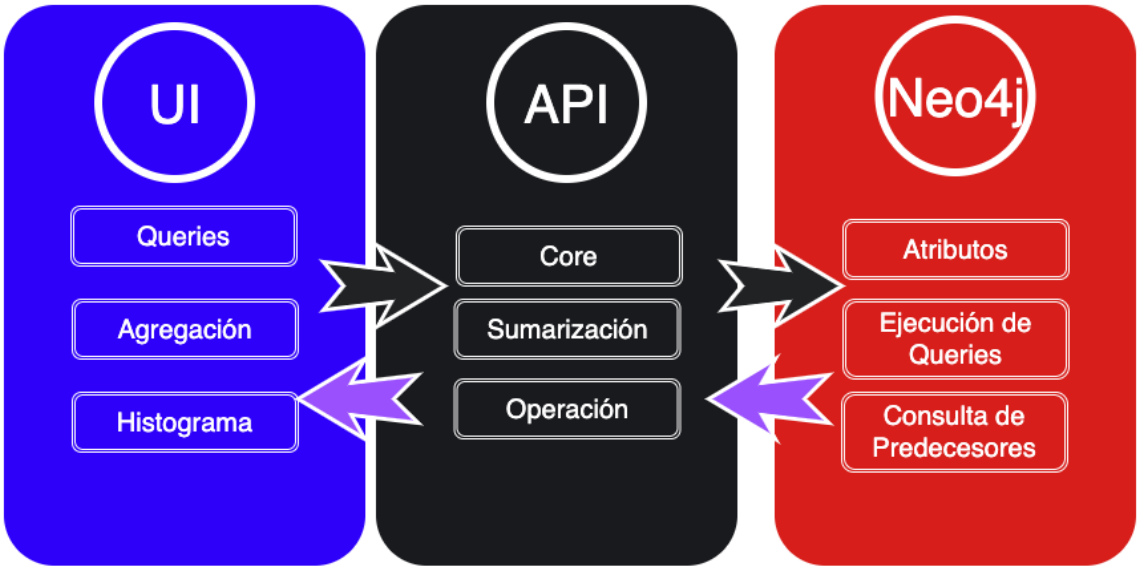

Figura 5: Modelo de sumarización.

Como se presenta en la Figura 5, el módulo de la interfaz gráfica **UI**, nos ayuda a visualizar el grafo sumarizado, en dicha interfaz se ingresan las consultas las cuales ya cuentan con las características o patrones que quieren ser analizados la cual corresponde al sub módulo de **consultas**, una vez obtenidos los resultados por parte de la API podemos realizar operaciones sobre el grafo, el sub módulo de **agregación** nos permite realizar operaciones de agregación sobre los supernodos, dichas operaciones son máximo, mínimo y conteo, el sub módulo de **histograma** nos da una representación gráfica de un atributo numérico del supernodo donde la superficie de cada barra es proporcional a la frecuencia de los valores del atributo, por otra parte se cuenta con una operación de re-agrupamiento la cual consiste en seleccionar un atributo del supernodo y hacer la descomposición en grupos de nodos por dicho atributo, esta funcionalidad es útil ya que permite dar más información visual al poder hacer un análisis más exhaustivo de los nodos formados por el reagrupamiento del atributo.

El módulo de **API** es donde se encuentra el procesamiento de los datos, este módulo se compone de tres piezas claves que son core, sumarización y operación. En el módulo **core** es donde encontramos las estructuras de los datos que van a ser enviados a la UI en formato json, tenemos el modelo para las aristas, nodo y grafo, así mismo aquí encontramos las estructuras que son aceptadas para las peticiones de la UI, tales son la estructura para peticiones de consultas, histogramas, intersecciones, funciones de agregación y descomposición por atributos. El submódulo de **sumarización** se encuentra el procesamiento de las consultas, la cual es la encargada de construir los supernodos y super aristas, cuando se está haciendo el cálculo de los supernodos se hacen consultas a la base de datos para calcular los predecesores de cada grupo generado con el objetivo de calcular las aristas en operaciones subsecuentes. El submódulo de **operación** encontramos plantillas que consisten en consultas predefinidas las cuales contienen fragmentos

que son reemplazados para obtener información de las operaciones solicitadas por UI, como: histograma, funciones de agregación, reagrupación de nodos.

En el módulo de **Neo4j**, es la parte que nos permite extraer los datos, se tiene la inicialización del controlador para neo4j para permitir las transacciones entre la API y neo4j. El submódulo de atributos nos permite sacar el tipo dato de cada atributo del supernodo que posteriormente será usado para funciones de agregación. El submódulo llamado **ejecución de queries** es donde se tiene la lógica para establecer la conexión y realizar la ejecución de consultas, antes de hacer la ejecución de las consultas definidas por el usuario este submódulo tiene la tarea realizar operaciones de limpieza escapando caracteres especiales y organizar la captura del conjunto de resultados. El submódulo de **predecesores** cuenta con una consulta especial que es la encargada del cálculo de predecesores de cada nodo, esta acción se hace para llevar un historial de los nodos dirigidos, calculando quien es su predecesor. Este historial de predecesores es fundamental para la parte de sumarización mencionada en el módulo de API.

A continuación, describimos detalladamente las estructuras usadas para mantener en memoria el grafo sumarizado, los modelos para transferir datos al módulo gráfico, y el proceso detallado para realizar la sumarización de un grafo dadas consultas predefinidas.

4.1.1. Proceso de sumarización

Se construye un grafo sumarizado a partir de recibir las consultas en un formato esperado, las consultas deben entregar como resultado una columna con colecciones de IDs o una columna de IDs, a las cuales llamamos consulta multigrupo o consulta individual respectivamente, si no se cumple con dicho formato la consulta es descartada, ejemplo Figura 6. Una vez detectado si la consulta es válida se ejecuta y genera estructura **grupo para consulta individual** o lista de **grupos para consulta multigrupo**. Un grupo a este momento contiene la lista de **IDs** asociada al grupo, un tipo, una etiqueta y atributos Figura 7. si en la consulta se detecta una palabra clave **MATCH** se intenta extraer el tipo, si la consulta es más compleja e incluye estructuras mixtas y no se puede detectar un tipo, el tipo del grupo queda como *'UNKNOWN*". Para poder asignar una etiqueta al grupo es necesario poner un comentario al final de cada consulta, si se omite el comentario, el atributo de etiqueta para el grupo queda como "*LABEL_UNDEFINED*".

```
{

  "queries":
  [
    "MATCH(p:Person)-[:ACTED_IN]->(m:Movie) WITH p, count(m) AS rel RETURN rel AS TITLE, collect(id(p)) AS IDS //NUM_ACTED",
    "MATCH(m:Movie) WHERE m.released >= 1975 AND m.released <= 1995 RETURN id(m) AS ID //1975-1995 Movies",
    "MATCH(m:Movie) WHERE m.released > 1995 AND m.released <= 2000 RETURN id(m) AS ID //1995-2000 Movies",
    "MATCH(m:Movie) WHERE m.released > 2000 AND m.released <= 2012 RETURN id(m) AS ID //2000-2012 Movies"
  ]

}
```

Figura 6: Ejemplo de una consulta multigrupo y tres consultas individuales.

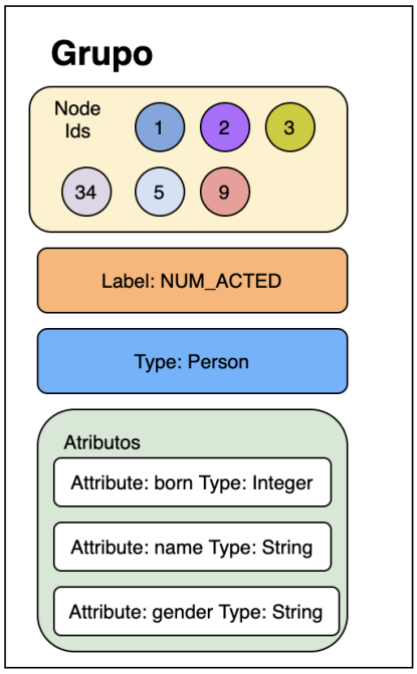

Figura 7: Ejemplo de estructura de un grupo.

4.1.1.1. Creación de supernodos

Una vez que se tienen construidos los grupos, por cada grupo, se crea un **supernodo**, el supernodo contiene el **grupo** previamente creado, un **color** asignado de acuerdo con el tipo de grupo, y un "hashmap" de **predecesores**, cada supernodo se agrega a un "hashmap" de **supernodos** con un único identificador, ver Figura 8.

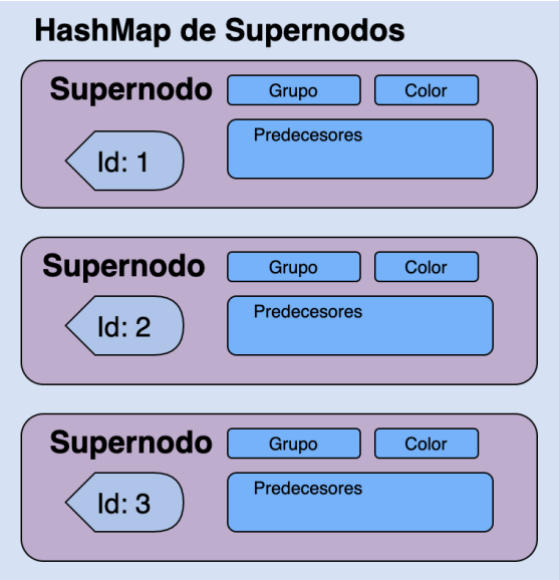

Figura 8: HashMap de supernodos.

En un grafo dirigido cuando se tiene una arista de un nodo **"x"** hacia un nodo **"y"**, el predecesor de **"y"** se considera **"x"**,

El atributo de predecesores de cada supernodo se evalúa cuando se construye el supernodo,

Para los **IDs** contenidos en la estructura grupo de un **supernodo** se consultan a neo4j para extraer los IDs **predecesores** y ser almacenados en el **supernodo**, como las aristas pueden tener un tipo asociado, una vez obtenidos los predecesores son evaluados por tipo, por tal motivo se tiene un hashmap de **predecesores** en el cual la llave es el tipo de arista y el valor son los ids de los predecesores pertenecientes a al tipo de arista, ver Figura 9.

La evaluación de **predecesores** es una parte esencial en el proceso de sumarización ya que forman parte del proceso para el cálculo de super aristas.

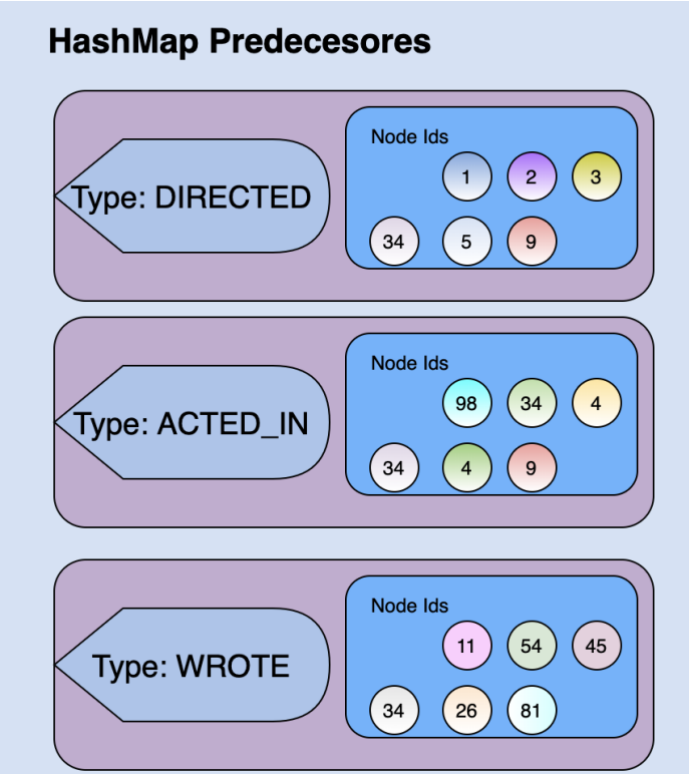

Figura 9: Estructura de predecesores pertenecientes a un supernodo, donde las aristas son del tipo DIRECTED, ACTED_IN, y WROTE.

Hasta este momento un **grupo** contenido en un **supernodo** no se tiene información sobre los **atributos** pertenecientes al grupo, para esto se cuenta con un proceso que por medio de consultas de neo4j predefinidas se hace el cálculo de atributos y tipo las cuales son asignadas al grupo del supernodo, ver Figura 7.

Como se mencionó anteriormente cada **supernodo** es asignado a un hashmap con un identificador, ver Figura 8, hasta el momento todas las estructuras son internas y manejadas solamente por el API, se necesita un modelo para transferir información del supernodo sumarizado en formato JSON a la interfaz gráfica. Para generar el JSON se tiene una estructura **"Node Model "** que a partir de cada **supernodo** se le asignan los atributos, el **"Node Model"** contiene el identificador del **supernodo** con el que fue asignado al hashmap, el peso del supernodo, que es la cantidad de ids que tiene el grupo perteneciente al **supernodo**, la etiqueta del grupo perteneciente al supernodo, el color del grupo perteneciente al supernodo y los atributos que pertenecen al grupo del supernodo, ver Figura 10.

Figura 10: Representación en formato JSON del Modelo de un supernodo.

4.1.1.2. Creación de súper aristas

Las **super aristas** se construyen a partir de los **supernodos** de donde se extraen los **predecesores,** ver Figura 8. El cálculo se hace de la siguiente manera, observando la Figura 11, donde se representa a tres **supernodos** de un grafo ya sumarizado de con **supernodos** tipo **movie** y **person**.

El cálculo de las super aristas se hace en un ciclo para todos los **supernodos**.

Para el **supernodo 0,** tenemos **predecesores** 7, 9 y 10 para la arista con atributo "FOLLOWS", se compara nuevamente los identificadores de los grupos de **supernodo 1** y **supernodo 2** para analizar qué grupo de un supernodo contiene a los predecesores, el supernodo 2 contiene a los predecesores 7, 9 y 10 para el atributo "FOLLOWS" y se genera una arista de **supernodo 2** a **supernodo 0** con un peso de valor 3 y de tipo **"FOLLOWS".**

Para el **supernodo 1**, con **predecessor 1, 2** para las aristas con atributo **"DIRECTED"** y **predecesor 8** que aparece dos veces para la arista **"PRODUCED"**, se comparan los identificadores de los predecesores con los identificadores de los grupos de **supernodo 0** y **supernodo 2**, para analizar qué grupo de un **supernodo** contiene a los **predecesores**, el **supernodo 0** contiene a los **predecesores** 1, y 2 para el atributo **"DIRECTED"** y se genera una arista de **supernodo 0** a **supernodo 1** con un peso de valor 2 y de tipo **"DIRECTED",** el **supernodo 2** contiene a los **predecesores** 8 para el atributo "**PRODUCED**" y se

genera una arista de **supernodo 2** a **supernodo 1** con un peso de valor 2 ya que el predecesor 8 aparece 2 veces, el tipo de la arista generada es "**PRODUCED**".

Como **supernodo 2** no contiene predecesores no se hace ningún análisis, en la Figura 12 se encuentra la representación de las super aristas generadas.

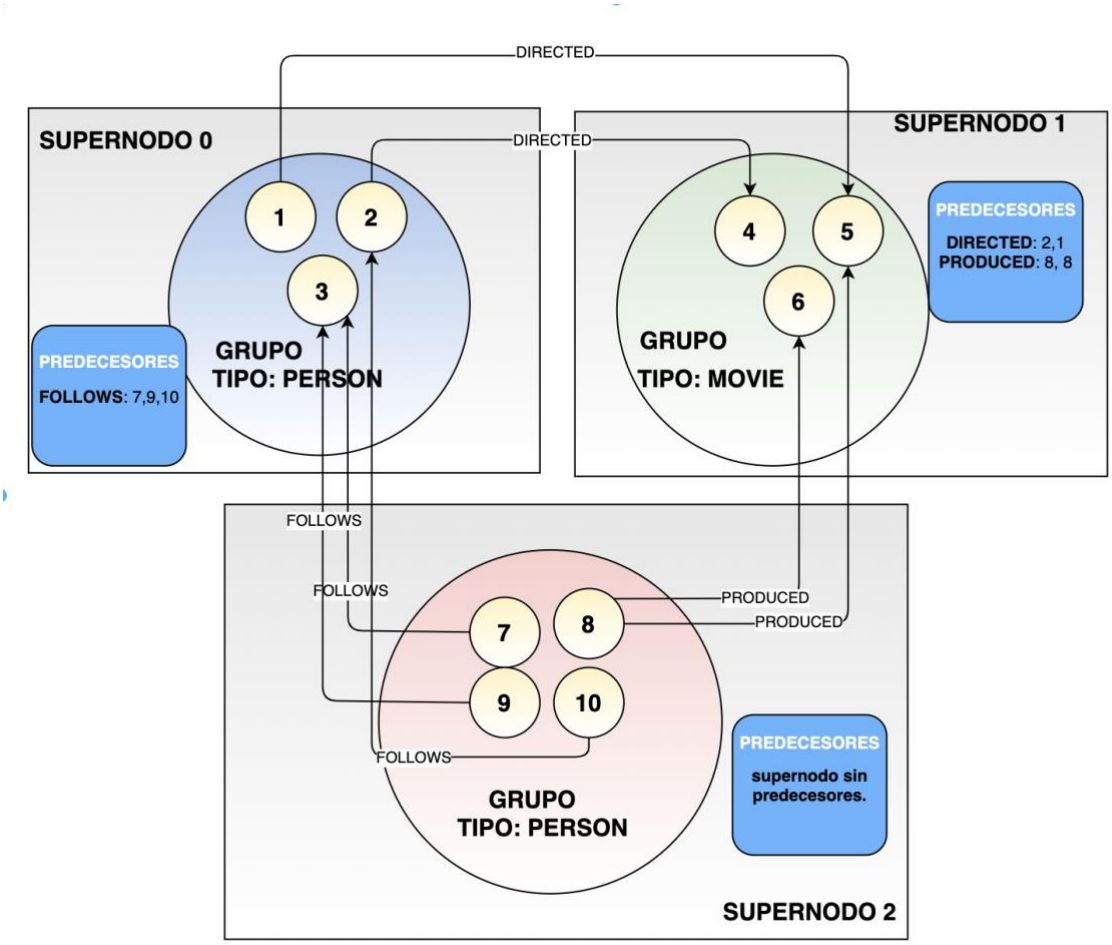

Figura 11: Creación de súper aristas a partir de supernodos.

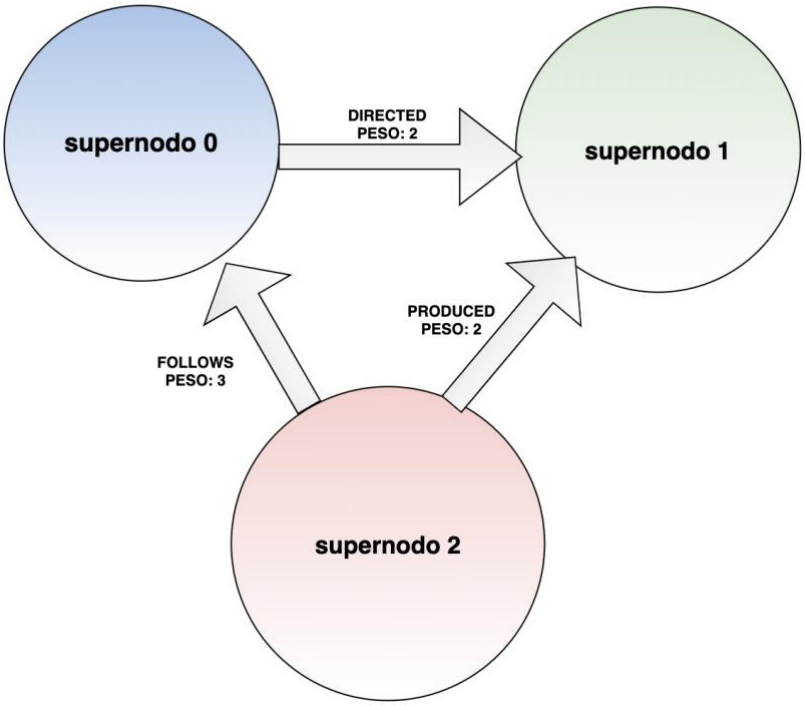

Figura 12: Súper aristas generadas para supernodos.

Tenemos una estructura llamada **"EdgeMap"**, la cual guarda todas las **superaristas** calculadas en una lista de objetos **"Edge Model",** que al igual que el **"Node Model"** es una estructura para regresar a la interfaz gráfica las super aristas en formato json, ver Figura 14. Los atributos de **"Edge Model"**, son fuente, el objetivo, el peso y la etiqueta, ver Figura 13 donde se tiene representado la estructura generada del ejemplo de la creación de aristas de la Figura 12.

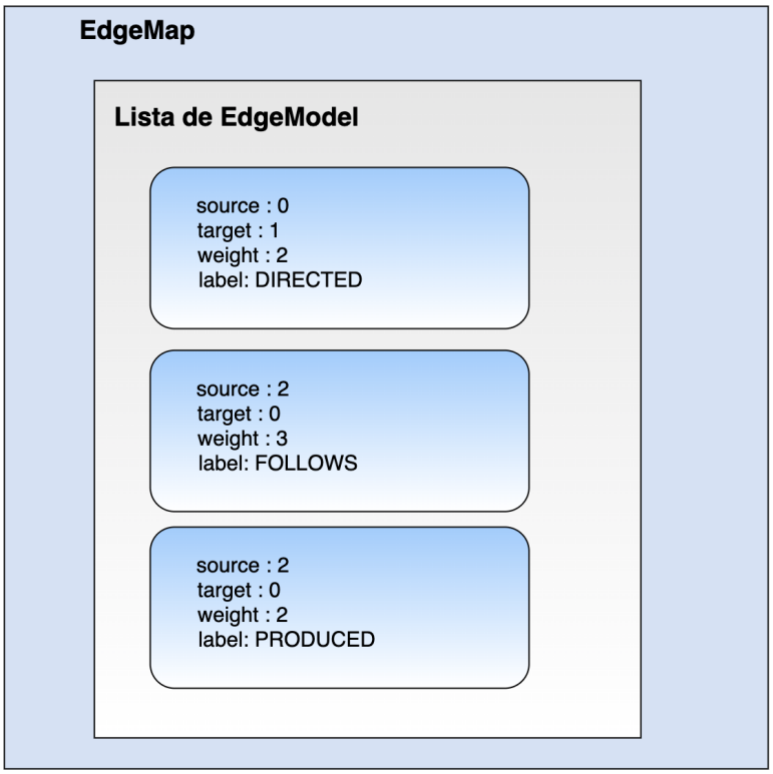

Figura 13: Estructura EdgeMap que lleva registro de las super aristas del grafo sumarizado.

```
{
  "links": [
    {
      "source": 0,
      "target": 1,
      "weight": 2,
      "label": "DIRECTED"
    },
    {
      "source": 2,
      "target": 0,
      "weight": 3,
      "label": "FOLLOWS"
    },
    {
      "source": 2,
      "target": 0,
      "weight": 2,
      "label": "PRODUCED"
    }]
}
```

Figura 14: Representación de formato JSON de la lista de super aristas

4.1.1.3. Cálculo de intersecciones entre dos nodos

Se calcula el porcentaje de Ids de la intersección entre dos supernodos, dicha intersección es guardada en una lista de objetos llamado **"Intersection Model**", este modelo contiene el id del **supernodo** en el que nos enfocamos, el id del nodo objetivo, y el porcentaje de ids que contiene el nodo en el que nos enfocamos respecto al nodo objetivo.

En el ejemplo de 3 supernodos sumarizados representados en la Figura 15, se verifica las siguientes intersecciones.

- Intersección de **supernodo 0** con **supernodo 1**
- Intersección de **supernodo 0** con **supernodo 2**
- Intersección de **supernodo 1** con **supernodo 2**

Dando como resultado el modelo representado en la Figura 16, haciendo la interpretación del primer elemento en el modelo, el nodo foco es **supernodo 0** y el nodo objetivo es **supernodo 1,** donde se observa que hay tres elementos de intersección, lo cual representa que el **supernodo 0** tiene un 60% de sus elementos con intersección respecto al **supernodo 1**.

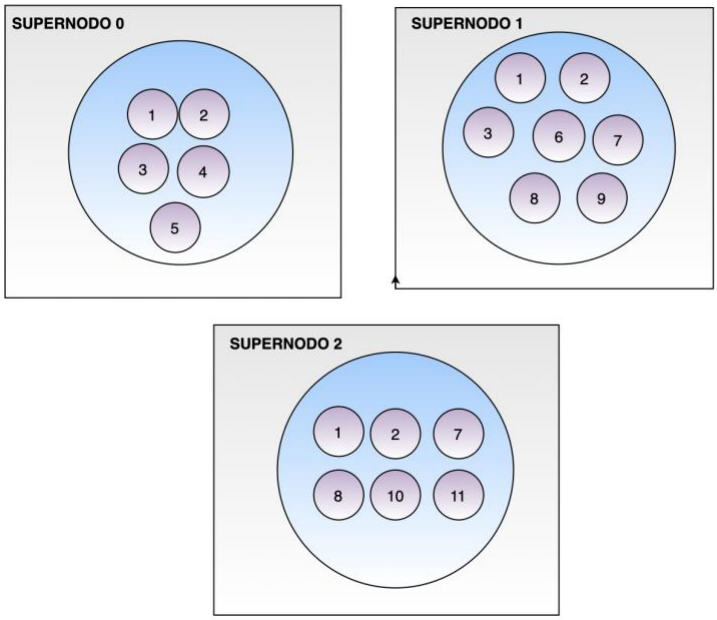

Figura 15: Representación de tres supernodos sumarizados.

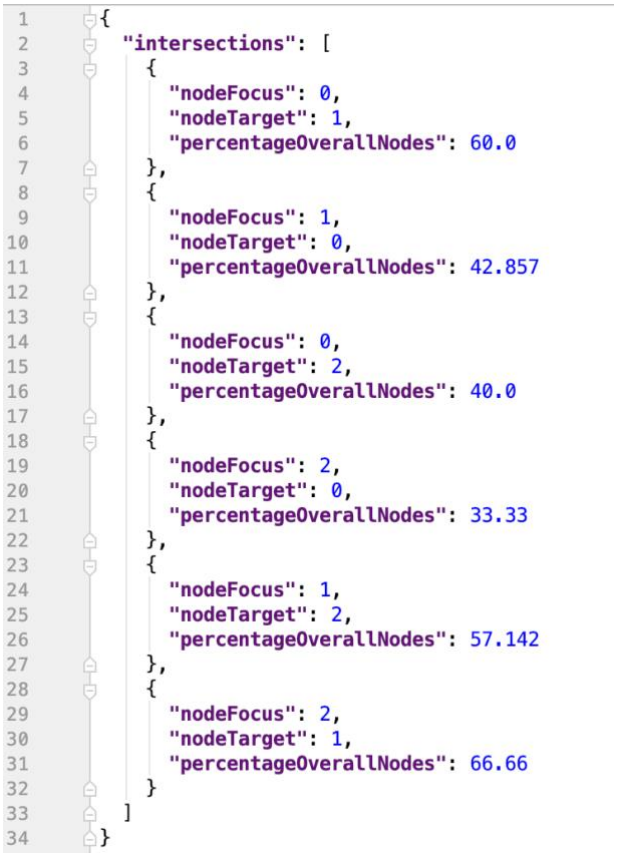

Figura 16: Modelo en formato json de las intersecciones.

Hasta este punto ya tenemos calculados la lista de "Node Model" que son los supernodos y la lista de "Edge Model" que son las super aristas, y la intersección entre supernodos en "Intersection Model", estas estructuras son regresadas por la API en formato JSON a la interfaz gráfica.

4.1.2. Proceso de reagrupamiento

El proceso de reagrupamiento por atributo comienza desde la interfaz gráfica, cuando se selecciona un **supernodo** y un se selecciona un atributo. Se hace una llamada a la API con el id del **supernodo** a reagrupar por el atributo seleccionado.

Del HashMap en memoria de supernodos Figura 8, se obtiene por Id el supernodo a reagrupar, del **grupo** del **supernodo** se obtienen los ids de nodos pertenecientes al **supernodo** y la etiqueta, con estos datos se hace la siguiente consulta predefinida.

```
MATCH(a) WHERE id(a) IN [{0}] RETURN a.{1} AS TITLE, collect(id(a)) AS IDS //{2}
```

Donde el parámetro 0 contiene los Ids de nodos contenidos por el supernodo, parámetro 1 es el atributo y parámetro 2 es la etiqueta.

Una vez ejecutado la consulta anterior se obtiene como resultado una lista de Grupos con ids, y se hace el mismo proceso descrito en la sección **"Creación de supernodos",** los nuevos supernodos se agregan al HashMap de **supernodos** descrito previamente y se borra del hashMap el **supernodo** que se le hizo re-agrupación, como se muestra en la Figura 17, se reagrupa el **supernodo** con id 2, y como resultado de la reagrupación por atributo se agregan dos nuevos **supernodos** con id 4 y 5.

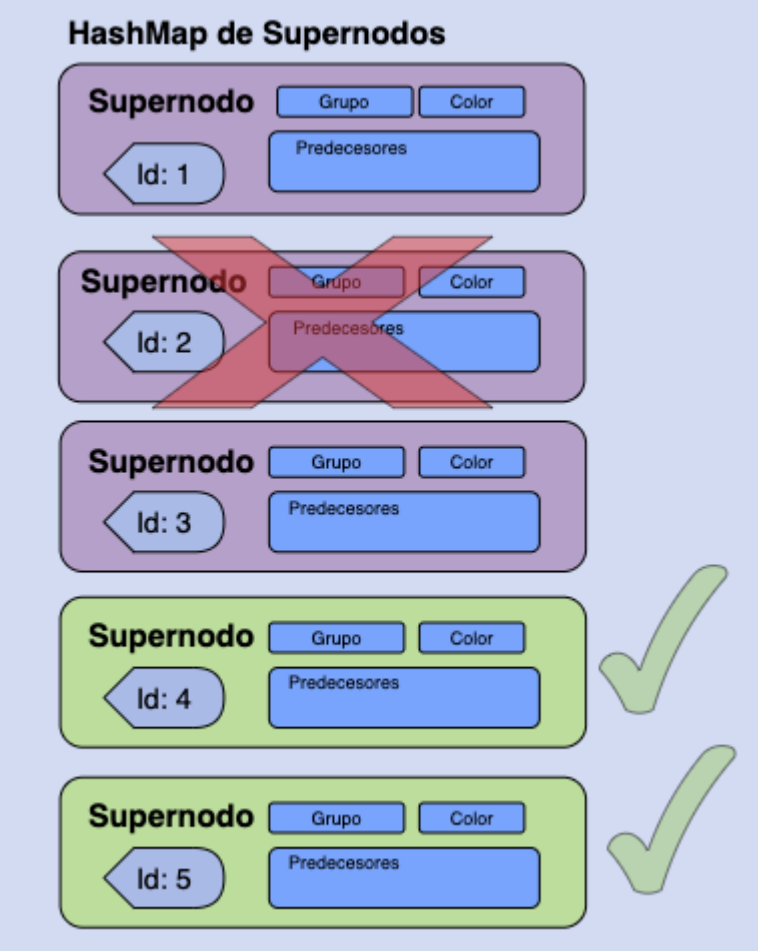

Figura 17: Reagrupación de supernodos.

4.1.2.1. Reagrupación de súper aristas

Después de tener la reagrupación y la generación de nuevos supernodos se procede al reagrupamiento de las aristas. En la estructura **EdgeMap** donde se tienen todas las super aristas en memoria, Figura 13.

Con el id del supernodo a reagrupar, se analiza la lista de **EdgeModel** para identificar aristas donde aparece el supernodo como fuente(source) o como objetivo (target). Se realizan los siguientes pasos para el cálculo de las nuevas súper aristas generadas a partir del reagrupamiento.

- En la super arista donde el supernodo reagrupado fue fuente(source), se recalcula las súper aristas tomando el id de los **supernodos** objetivo (target), se realiza el mismo procedimiento visto en "Creación de super aristas" con la única diferencia que el cálculo de súper aristas es realizado solamente entre el Ids de supernodos objetivo (target) y los nuevos **supernodos** agregados.
- En la super arista donde el supernodo reagrupado fue objetivo(target), se recalcula la súper arista donde tomando el id de los **supernodos** fuente (source), se realiza el mismo procedimiento visto en "Creación de super aristas" con la única diferencia que el cálculo de super aristas es realizado solamente entre los nuevos **supernodos** agregados y los Ids de **supernodos** fuente (source).
- Borrar las super aristas de memoria de la lista de **EdgeModel,** que contienen como fuente (source) y objetivo(target) el id del supernodo reagrupado.
- Actualizar la lista de de **EdgeModel** con las nuevas super aristas calculadas.

En la Figura 18 vemos la estructura interna de un grafo sumarizado y en la Figura 19, se muestra la reagrupación del **supernodo 0,** por el atributo nacionalidad, donde se generan dos nuevos supernodos, **supernodo 3 y supernodo 4.**

- Se calcular nuevas super aristas entre supernodo 3 y supernodo 4 y el supernodo reagrupado tenía como objetivo el **supernodo 1**.
- Se calcular nuevas super aristas entre supernodo 3 y supernodo 4 y el supernodo reagrupado tenía como fuente (source) el **supernodo 2**
- Se borran las súper aristas de memoria
- Se actualiza la lista de **EdgeModel** con 3 nuevas súper aristas.

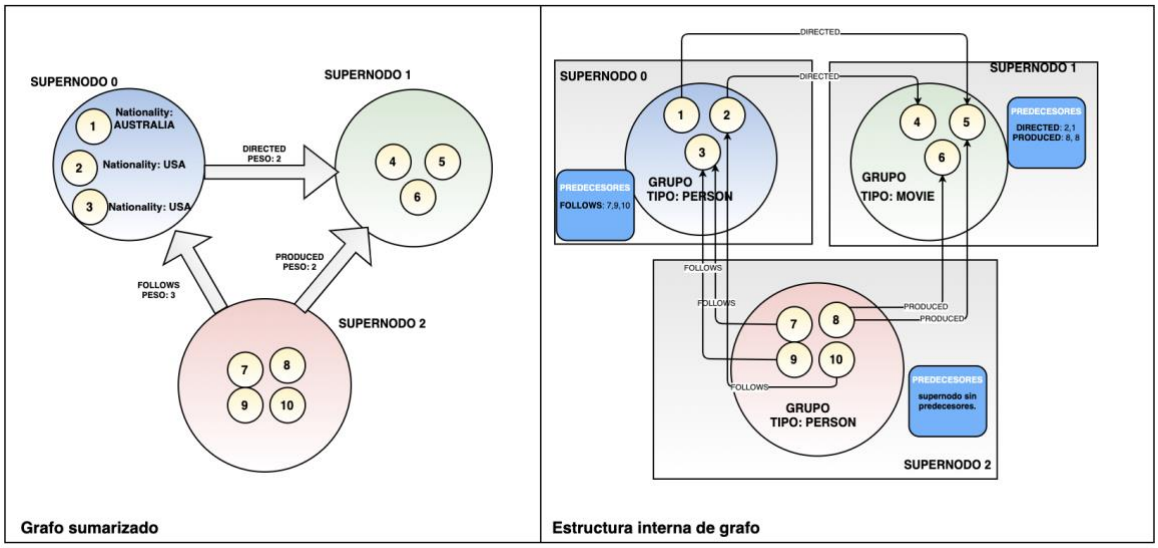

Figura 18: Grafo sumarizado y estructura interna.

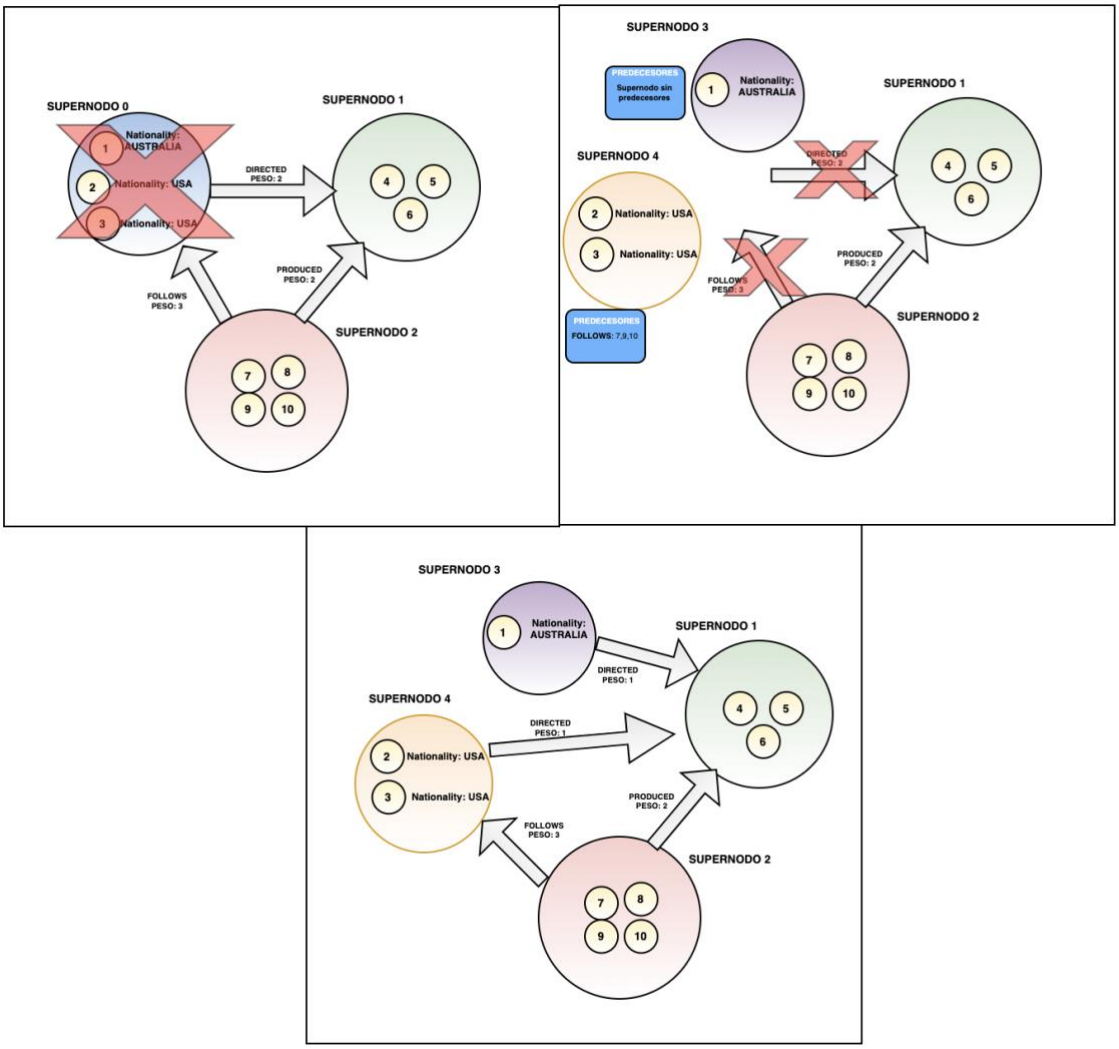

Figura 19: Recalculo de súper aristas cuando se reagrupa supernodo 0 por atributo de nacionalidad.

La API solo regresa el modelo con los nuevos **supernodos** generados y las nuevas aristas generadas del reagrupamiento.

4.1.3. Operaciones de agregación

La API cuenta con un end-point para realizar funciones de agregación a un supernodo en específico. Se recibe las funciones operación con el modelo **AgregateFunctionRequest** que cuenta con el siguiente formato mostrado en Figura 20, donde se especifica que tipo de operación se quiere realizar, el identificador del supernodo, en el caso de COUNT se debe de especificar el valor que se va a contar, y el tipo del atributo.

Figura 20: Modelo para realizar operación de agregación.

Una vez recibida la operación de agregación se hace la consulta a neo4j y se regresa un string con el resultado obtenido.

La consulta que se realiza las siguientes consultas predefinidas dependiendo de la operación.

```
COUNT = MATCH(n'{'{1}:{2}'}') WHERE id(n) IN [{0}] RETURN count(*) AS RESULT
MAX = MATCH (n) WHERE id(n) IN [{0}] RETURN max(n.{1}) AS RESULT
MIN = MATCH (n) WHERE id(n) IN [{0}] RETURN min(n.{1}) AS RESULT
```

Donde el parámetro 0 contiene los Ids de nodos contenidos por el supernodo, parámetro 1 es el atributo, parámetro 2 es el valor.

4.1.4. Operaciones de histograma

La API cuenta con un end-point para realizar funciones de histograma a un supernodo en específico. Se recibe la operación con el modelo **HistogramRequest** que cuenta con el siguiente formato mostrado en la Figura 21. El número de bloques representa el número de barras del histograma, el id es el identificador del supernodo al cual se quiere realizar el histograma, atributo la variable del histograma, y tipo es el tipo de dato del atributo, solo se puede realizar la operación de histograma a atributos numéricos.

Figura 21: Modelo para operación de histograma.

Una vez recibida la operación de histograma se calculan los valores máximo y mínimo del atributo. El valor máximo menos el mínimo dividido entre el número de bloques, nos da el tamaño o el rango de cada barra en el histograma.

Se realiza un ciclo las veces indicada por el número de bloques para calcular cada barra del histograma. Para el cálculo de cada barra se realiza la siguiente consulta predefinida

```
MATCH(n) WHERE id(n) IN [{0}] AND n.{1} >= {2} AND n.{1} <= {3} RETURN count(*) AS RESULT;
```

Donde el parámetro 0 contiene los Ids de nodos contenidos por el supernodo, parámetro 1 es el atributo, parámetro 2 es el inicio del bloque o barra y el parámetro 3 es el valor final para la barra o bloque.

Cada resultado de cada bloque se guarda en un modelo llamado **histograma** que tiene dos atributos valor y área, donde área es el rango del bloque/barra y valor es la frecuencia o cantidad de elementos que existen en el rango. Al final del ciclo se tiene una lista de objetos histograma con todos los bloques.

Para el resultado mostrado en la Figura 22, que es la operación de la Figura 21 para un supernodo de un grafo previamente sumarizado.

Se calculó un valor máximo de 1996 y valor mínimo de 1930, y el tamaño de cada bloque es de 11, y obtenemos una lista de 6 bloques/barras con la frecuencia para cada barra.


```
[
  {
    "area": "From  1930 to 1941",
    "value": "13"
  },
  {
    "area": "From  1942 to 1952",
    "value": "21"
  },
  {
    "area": "From  1953 to 1963",
    "value": "28"
  },
  {
    "area": "From  1964 to 1974",
    "value": "27"
  },
  {
    "area": "From  1975 to 1985",
    "value": "6"
  },
  {
    "area": "From  1986 to 1996",
    "value": "1"
  }
]
```


Figura 22: Modelo en formato json de la lista de objeto histograma.

5. Implementación

Una vez que revisemos las estructuras y el proceso para la sumarización, veremos con más detalle la implementación.

5.1. Herramientas

Se usó dropwizard version 1.3.9, que es un framework de java para servicios web RESTFul, así como el controlador neo4j 1.0.4 para permitir la conexión a la base de datos, y el controlador neo4j JDBC versión 3.0.1, el cual permite la ejecución de comandos tipo cypher sobre JDBC. JDBC es la especificación oficial de java donde se define como se debe de cómo un cliente puede conectarse a la base de datos. El código se realizo en lenguaje java con versión 1.8.0_22

Como parte de la implementación de dropwizard contamos con un archivo config.yml, el cual contiene la configuración necesaria para la base de datos.

Los parámetros mas importantes de la configuración de dropwizard son los siguiente:

- El controlador JDBC
- El usuario de la base de datos
- Las credenciales
- El host donde se encuentra la base de datos
- Número máximo de conexiones abiertas
- El tiempo máximo de espera de una pila de conexiones antes de lanzar una excepción

Por parte de la base de datos se instaló la versión 3.5.17 de neo4j y el plugin apoc-3.5.0.8-all.jar, para el funcionamiento del plug-in mencionado se modificó el archivo de configuración de neo4j con el siguiente declaración, **dbms.security.procedures.unrestricted=apoc.* .**

5.2. Secuencia en el proceso de sumarización y reagrupamiento

A continuación, mostramos el proceso de sumarización y reagrupamiento el cual esta compuesto de los siguientes procedimientos, proceso de consultas y construcción de supernodos, cálculo de aristas basada en supernodos y la construcción del objeto GraphModel.

5.2.1. Procesamiento de consultas y construcción de supernodos

Como observamos en la secuencia de la Figura 25, en la clase "**GraphResource**" encontramos todos los end-points del servicio, cada end-point está encargado en una acción en específico, el punto de entrada para la sumarización es el método "**buildGraph**" que corresponde al end-point *<hostName>/v1/queries*.

Una vez recibida las consultas se manda llamar el proceso **processQueries** del servicio "**Graph**", el proceso es el encargado de ejecutar las operaciones necesarias para la sumarización de acuerdo con las consultas de entrada. Se comienza con el subproceso "**buildSuperNodes**", el cual detecta si es una consulta multigrupo o una consulta individual (ver Figura 6) para consulta individual o multigrupo se llaman procesos de la clase **"GraphDao",** la cual es la encargada de ejecutar la consulta a neo4j y dar el formato a cada record regresado por dicha consulta.

Las consultas multigrupo e individual deben de cumplir con la siguiente expresión regular, de lo contrario serán descartadas.

- Expresión regular multigrupo : "(.*)RETURN (.*) AS TITLE, collect(.+) AS IDS(.*)"
- Expresión regular individual: "(.*)RETURN (.*) AS ID(.*)"

Para la consulta "**individual**" y "**multi grupo**" se usan expresiones regulares sobre la consulta y se extrae el tipo y la etiqueta.

- Expresión regular para tipo: "^MATCH\\((\\w+):(\\w+)\\)(.*)"
- Para extraer la etiqueta se obtiene del último comentario de la consulta.

El proceso de "**sumarizacionMultiRecordQuery**" nos da como resultado ya el mapeo de los récords a una lista de objetos **Grupo**, ver Figura 23.

El proceso de "**sumarizationSingleRecordQuery**" nos da como resultado un objeto **Group**, ver Figura 23

Una vez obtenidos todos los grupos sobre las consultas de entrada, se construyen dos listas la **lista de NodeModel,** ver Figura 23, que es el modelo usado para la respuesta en formato json de los súper nodos generados, y la l**ista de objetos SuperNode**, ver Figura 24, que es el objeto que se mantiene en memoria, cuando se agregan los elementos de **SuperNode** a la lista de **Graph**, se hace el cálculo de predecesores usando la siguiente consulta predefinida.

```
MATCH (n)-[r]->(p) where id(p) IN[{0}] RETURN  id(n) as ID, type(r) as TYPE;
```

Donde el atributo **0** es reemplazado por los ids del objeto **Group**, de esta manera se obtienen los predecesores de cada objeto **superNode** directamente de neo4j.

Cada objecto **superNode** agregado en la lista del objeto **Graph** incrementa el atributo **nodeId**, que lleva registro del último índice de los generados.

Hasta este punto en la secuencia se tiene el objeto **Graph** con la lista de **supernodos** ya sumarizados y el **nodeId** con el último índice generado para el id de los supernodos.

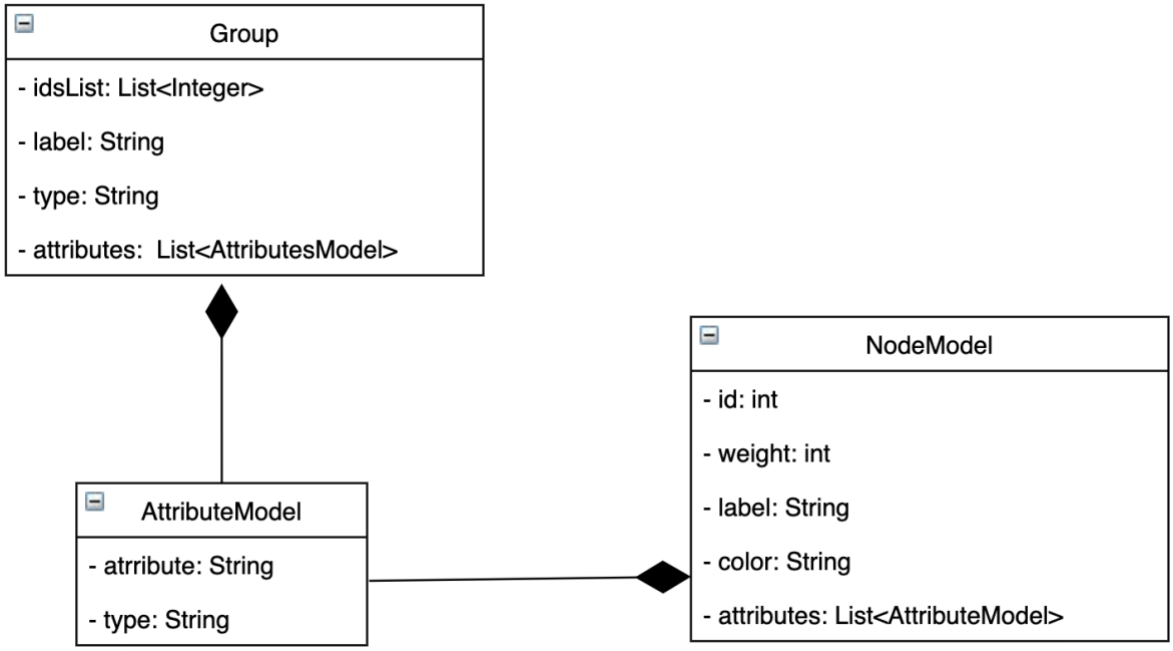

Figura 23: UML Group, AtrributeModel y NodeModel.

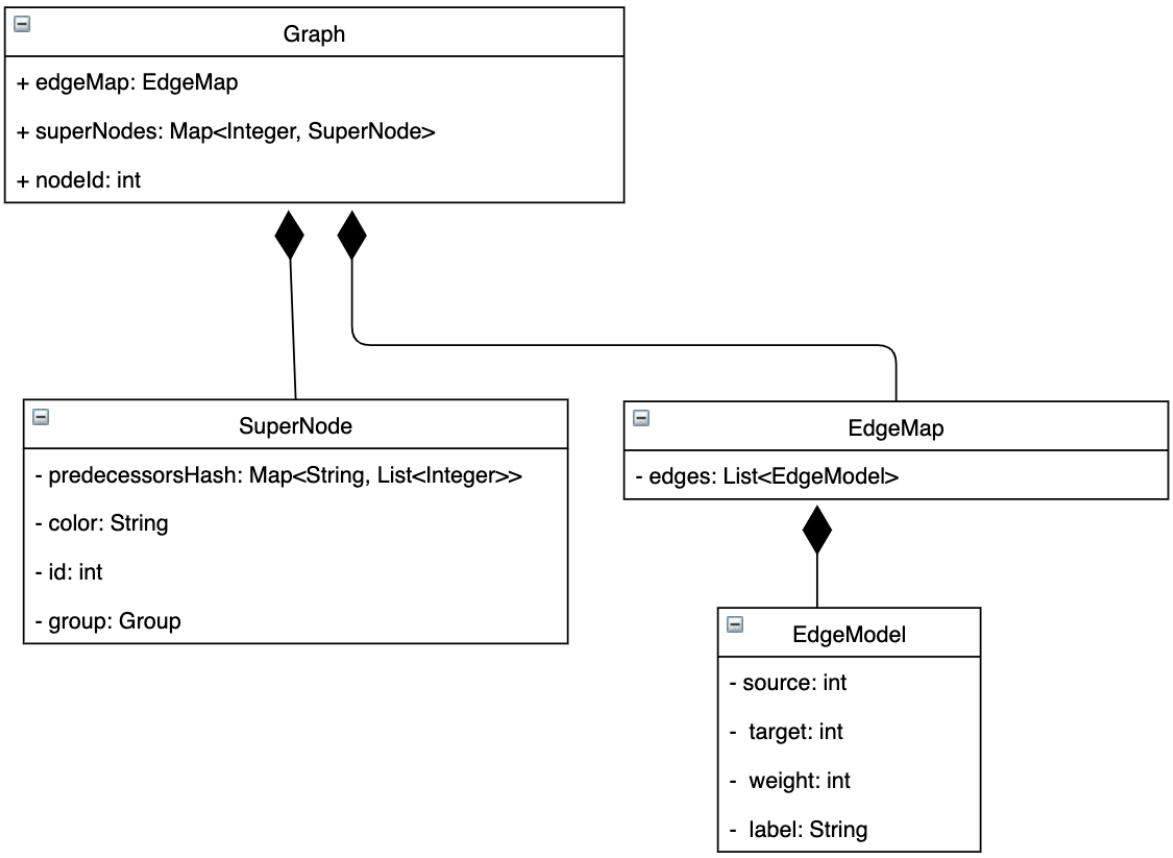

Figura 24: UML de Graph, SuperNode, EdgeMap y EdgeModel.

5.2.2. Cálculo de aristas basada en la lista de supernodos

Como se observa en la secuencia de la Figura 25. una vez calculados los supernodos, se pasa al proceso "buildEdgeMap". Este proceso crea la instancia del objeto **EdgeMap** y ejecuta el proceso "buildEdgeMappings" que es el proceso descrito en la sección **"Creación de súper aristas"**.

Al término de la ejecución del proceso se tiene el atributo **EdgeMap** de **Graph** con toda la lista de súper aristas.

5.2.1. Construcción de objeto GraphModel

Después de hacer el cálculo de súper aristas en la secuencia de la Figura 25 se ejecuta el proceso **calculateIntersections** descrito en la sección **"Calculo de intersecciones entre dos nodos",** que es donde se construye el objeto "**IntersectionsModel**" mostrado en la Figura 26.

Como ya se tienen el objeto **IntersectionsModel** y listas de **EdgeModel** y **NodeModel**, se construye el objeto **GraphModel** el cual es la respuesta de la aplicación en formato json. En la Figura 26 observamos los componentes de **GraphModel.**

5.3. Secuencia de reagrupamiento

Observando la secuencia de la Figura 25 en el recurso **GraphResource** encontramos el método **reGroup** que corresponde al end-point *<hostName>/v1/regroup.*

Usando el modelo **RegroupRequest** mostrado en la Figura 27 se hace la llamada al método **reGroup** de **GraphResource** que a su vez manda llamar el método **reGroup** del servicio **Graph,** este método cuenta con la implementación del modelo descrito en la sección **"Proceso de Reagrupamiento".**

Se hace la llamada a **GraphDao** para reagrupar el id del **supernodo** de la petición por atributo y obtener una lista de objetos **Group** que son los nuevos **supernodos** de dicho reagrupamiento. Por cada objeto **Group** de la lista se hace un ciclo y se ejecuta el método llamado **buildAndAddSuperNode**, el cual es el responsable de generar un objeto **superNode** con sus respectivos atributos como id, predecesores y color, y posteriormente agregar cada **superNode** a la map en memoria del objeto **Graph** y generar una lista de **NodeModel**, ver Figura 23 y Figura 24.

Finalmente se manda llamar el proceso de **regroupEdges** donde se realiza el cálculo de las nuevas super aristas y actualiza el atributo **edges** del objeto **edgeMap** , y también regresa solamente las nuevas aristas generadas por el reagrupamiento en una lista de **EdgeModel.**

Al final del proceso regroup de **Graph** regresa un objeto **GraphModel** con la lista de **nodeModel** y **edgeModel** que contiene solamente las aristas y supernodos generados por el reagrupamiento, ver Figura 26.

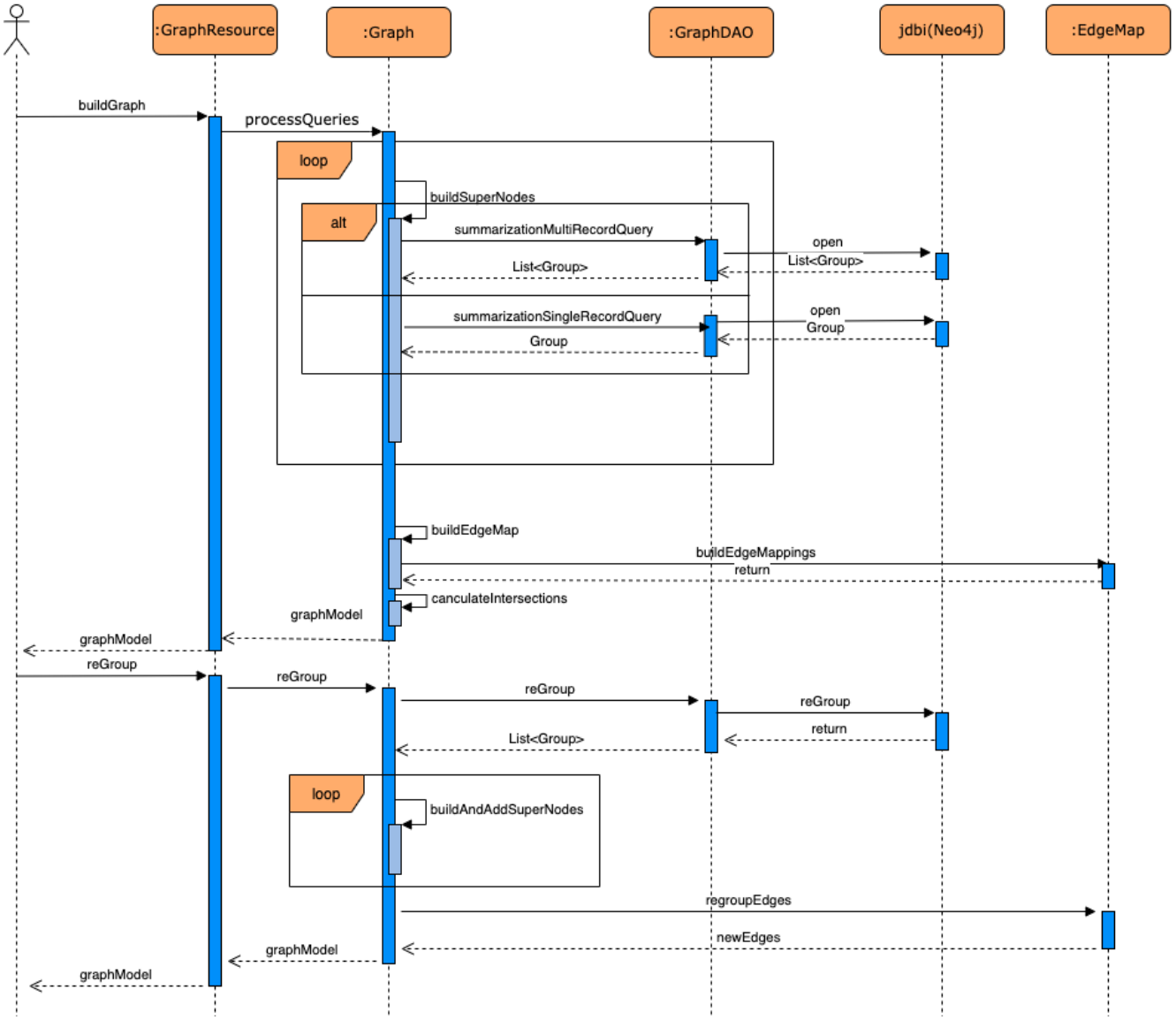

Figura 25: Secuencia de sumarización y reagrupamiento por atributo.

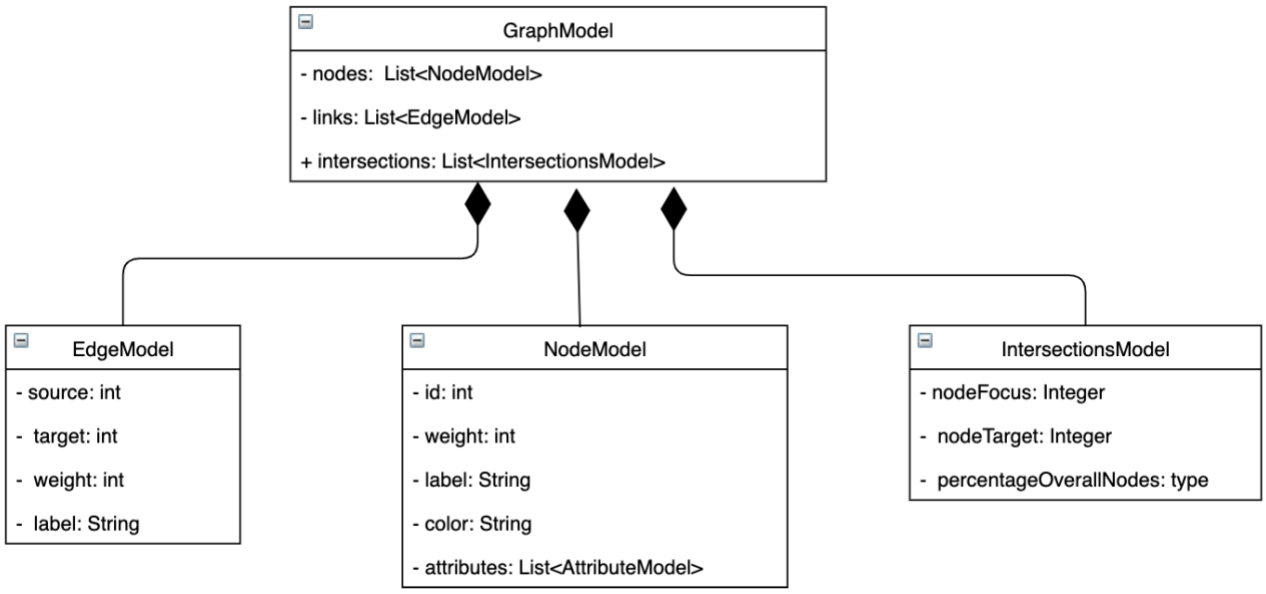

Figura 26: UML de GraphModel , EdgeModel, NodeModel y IntersectionsModel.

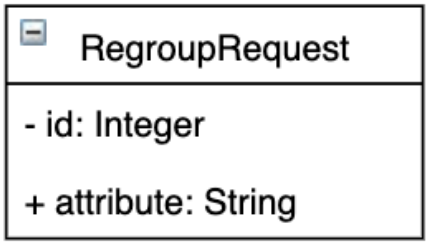

Figura 27: UML de RegroupRequest.

5.4. Secuencia de funciones de agregación

Observando la secuencia de la Figura 28, en **GraphResource** encontramos el método **aggregateFun** que corresponde al end-point *<hostName>/v1/aggregateFunction*, el cual recibe como parámetro el modelo **AggregateFunctionrequest**, ver Figura 29,

El modelo **AggregateFunctionrequest** cuenta con el id del supernodo al cual se le quiere aplicar la función de agregación, el atributo al cual ejecutar la operación, tipo de operación de agregación, valor si la función de agregación requiere como parámetro y el tipo de dato del atributo, en la sección **"Operaciones de Agregación"** se describe detalladamente las consultas que se tienen predefinidas para la ejecución.

Se manda llamar al método **calculateAggregateFunction** del servicio **Graph,** el cual se encarga de extraer los ids del supernodo a realizar la operación de agregación y también dar formato al atributo valor, si es un string lo formatea con comillas simples para poder ser evaluado correctamente en la consulta, si es del tipo Integer no se hace ninguna modificación, posteriormente se hace la llamada al proceso **evaluateAggregateFunciton** en este proceso se detecta que tipo de función de agregación se va a realizar y se obtiene el modelo predefinido de la función, de este modelo predefinido se reemplazan, los ids, atributo, valor para después realizar la ejecución de la consulta.

La consulta regresa un string que es regresado a las diferentes capas hasta generar la respuesta al servicio que hizo la llamada.

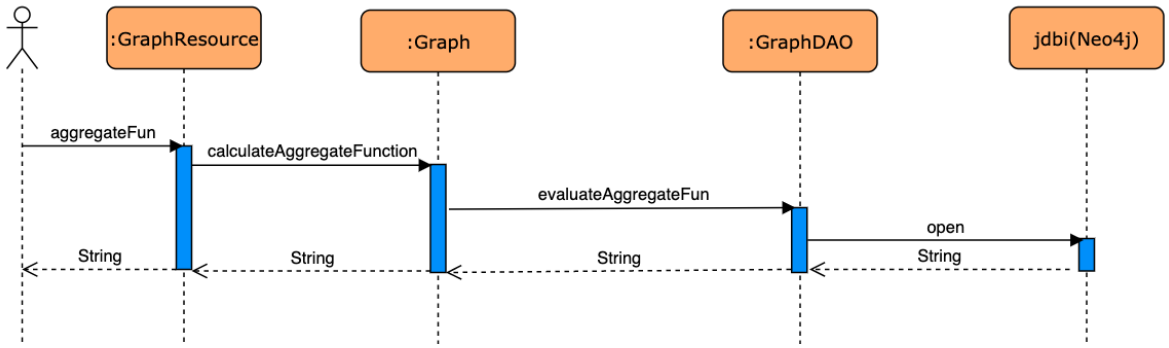

Figura 28: Secuencia de función de agregación.

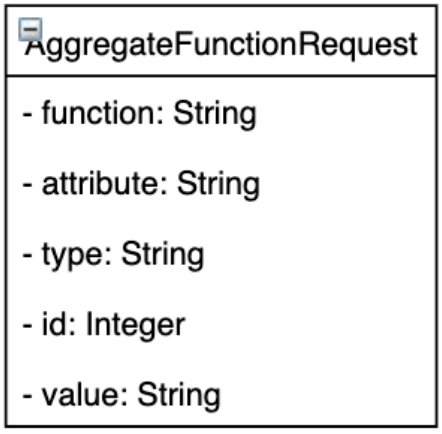

Figura 29: UML de AggregateFunctionRequest.

5.5. Secuencia para operación de histograma

Como observamos en la secuencia de la Figura 30,el punto de entrada es **GraphResource** donde encontramos el método **calculateHistogram** que corresponde al end-point

<hostName>/v1/aggregateFunction el cual recibe como parámetro el modelo **HistogramRequest**, ver Figura 32. El modelo cuenta con los atributos de número de bloques que son las barras del histograma, el id del nodo, el atributo del supernodo por el cual se va a realizar el histograma, y el tipo de dato del atributo.

La descripción del algoritmo implementado en el proceso **calculateHistogram** está descrito en la sección **"Operaciones de Histograma".**

Observando la secuencia de la Figura 30, vemos que las dos llamadas a los métodos **calculateAgregateFunction** del servicio **Graph** son para el cálculo del valor máximo y mínimo del histograma.

Con el máximo, mínimo y número de barras que se especifica en la petición se calcula el tamaño de cada barra del histograma, con esta información se usa la consulta predefinida descrita en la sección **"Operaciones de Histograma"** para el cálculo de elementos de cada bloque, está llamada se hace en un bucle llamando al método **getHistogramCount** el número de veces definido por número de bloques que se hizo en la petición.

El resultado entregado por cada ejecución de **getHistogramCount** es el tamaño del bloque, para ello se guarda cada resultado en una lista de objetos de tipo **Histogram**, ver la Figura 31, este objeto mantiene el rango de cada bloque y el valor obtenido de la consulta a la base de datos. Finalmente, el servicio regresa la lista de objetos de tipo Histograma.

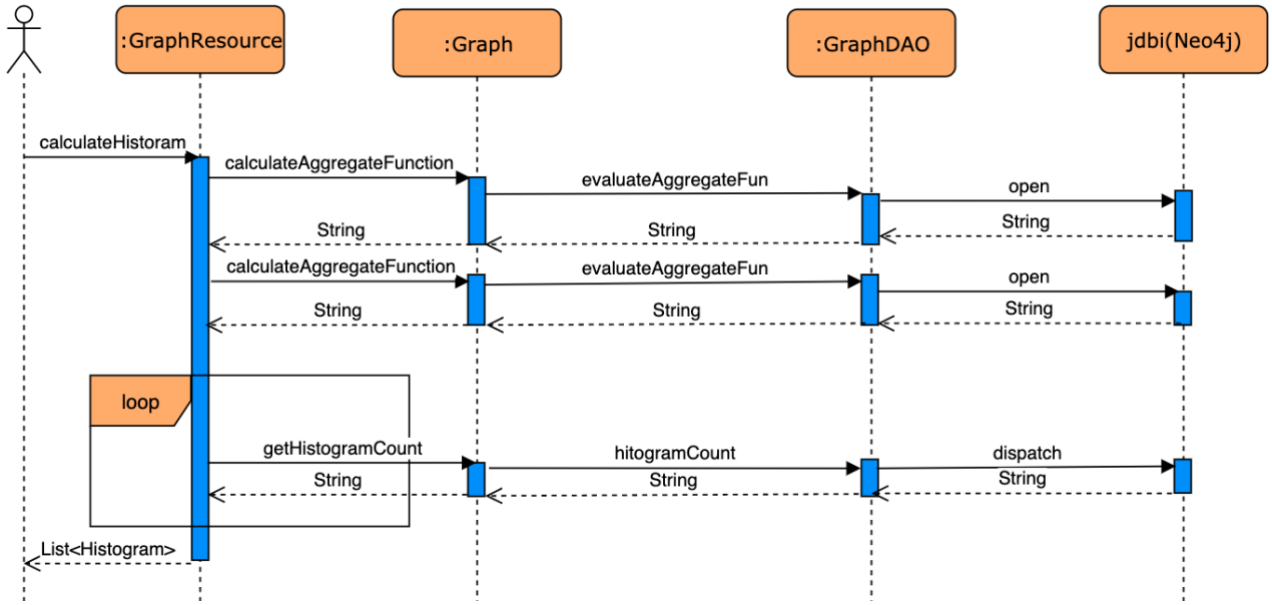

Figura 30: Secuencia de cálculo de histograma para un supernodo.

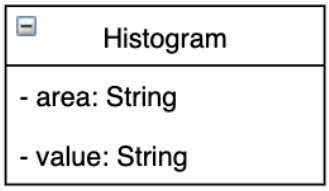

Figura 31: UML de Histogram.

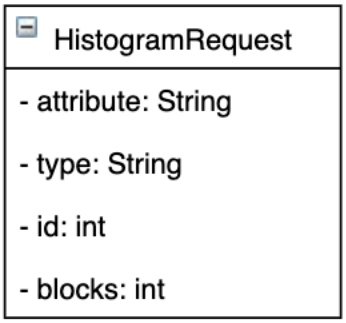

Figura 32: UML de HistogramRequest.

Una vez que ya se comprendido la implementación a bajo nivel podemos realizar los experimentos y ver los resultados.

6. EXPERIMENTOS Y RESULTADOS

Una vez visto el modelo e implementación haremos experimentos con dos datasets, uno de menor escala de películas con 171 nodos y 253 relaciones, ver Figura 33, y otro de gran escala de twitter con 50,081 nodos y 156,449 relaciones, ver Figura 34 , para desplegar la información de la herramienta de sumarización se usó una aplicación web con librerías de d3.js.

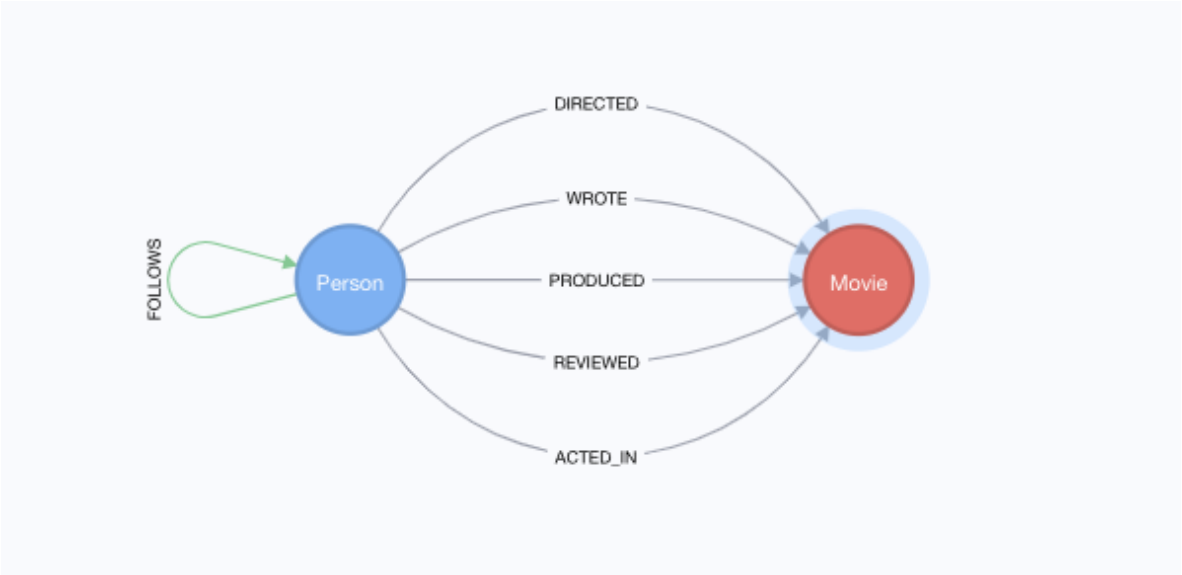

Figura 33: Schema de Base de datos de Personas y Películas .

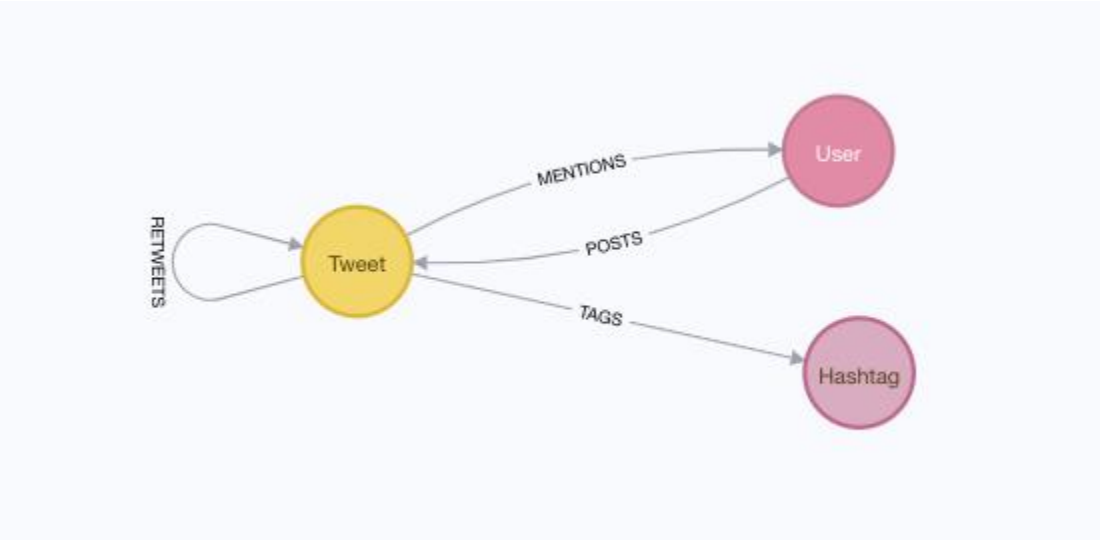

Figura 34: Schema de base de datos de Twiiter, User y Hashtag.

Para realizar sumarización basado en consultas se debe tener un conocimiento previo sobre los atributos y tipos de nodos que contiene el dataset, y el tipo de información que se requiere agrupar.

6.1. DataSet Películas y Personas

6.1.1. Agrupación y reagrupación con consultas sencillas

Con la base de datos de personas y películas, queremos agrupar las películas con ingresos de 500 millones de dólares y una segunda agrupación de personas que actuaron en películas con título de ‘Jerry Maguire’, 'The Da Vinci Code' o 'The Replacements' y observar la relación de aristas entre estas dos agrupaciones, hacerlo desde neo4j resulta una tarea complicada.

En la Figura 35 observamos las películas que tienen un ingreso de 500 Millones de dólares en neo4j, Observando este grafo no obtenemos la información significativa, de acuerdo al planteamiento inicial, **con la interfaz de neo4j no podemos obtener un análisis claro de información de las relaciones y agrupaciones, con actores y directores.**

Figura 35: Películas con ingresos de 500 Millones de dólares en neo4j.

Ahora bien, si intentamos usar nodos y aristas virtuales tendríamos que hacer los siguientes pasos. Nodo virtual para agrupación de películas con ingreso de 500 millones de dólares, nodo virtual para agrupación de personas que actuaron en ciertas películas, y crear aristas virtuales para las relaciones entre nodos virtuales, a continuación, se enumeran las consultas necesarias.

Consulta para nodo virtual de Personas que hicieron revisión de películas con título específico:

```
MATCH(p:Person)-[:REVIEWED]->(m:Movie)
WHERE m.title = 'Jerry Maguire' OR m.title = 'The Da Vinci Code'OR m.title = 'The Replacements' OR
m.title = 'Jerry Maguire' OR m.title = 'Cloud Atlas'
WITH count(DISTINCT id(p)) as count
RETURN apoc.create.vNode(['Person'], {label:"Person", weight:count}) as person
```

Consulta para nodo virtual de películas con ingresos de 500 millones de dólares:

```
MATCH(m:Movie)
WHERE m.revenue = 500 WITH count(*) as count
RETURN apoc.create.vNode(['Movies'], {label:"revenue 500",weight:count}) as movies
```

Consulta para calcular el peso del atributo de arista revisión (REVIEW) de agrupación de personas hacia agrupación de películas:

```
MATCH(p:Person)-[r:REVIEWED]->(m:Movie)
WHERE (m.title = 'Jerry Maguire' OR  m.title = 'The Da Vinci Code' OR m.title = 'The Replacements' OR
m.title = 'Jerry Maguire' OR  m.title = 'Cloud Atlas')
WITH p MATCH(p:Person)-[r:REVIEWED]->(m:Movie {revenue:500})
WITH DISTINCT id(p) as id return count(id)
```

Consulta para calcular el peso del atributo de arista FOLLOWS de la agrupación de personas:

```
MATCH(p:Person)-[r:REVIEWED]->(m:Movie)
WHERE (m.title = 'Jerry Maguire' OR  m.title = 'The Da Vinci Code' OR m.title = 'The Replacements' OR
m.title = 'Jerry Maguire' OR  m.title = 'Cloud Atlas')
WITH p MATCH(p:Person)-[r:FOLLOWS]->(p2:Person)
WITH DISTINCT id(p) as id return count(id)
```

Una vez obteniendo los nodos virtuales y consultas de los pesos para las aristas virtuales se tiene que unir las cuatro consultas en una consulta compleja para poder ser visualizada. Si necesitamos hacer una visualización

Si del nodo virtual que se tiene de películas con ingreso de 500 millones de dólares, queremos descomponer ese nodo virtual en agrupaciones de nodos por el atributo de “compañía”, nosotros como usuarios o analistas de información no podemos realizar operaciones directas sobre el nodo virtual, por lo cual se necesita crear las consultas para generar los nodos virtuales que deseamos observar, en este ejemplo sencillo, se enlistan las consultas necesarias.

- Generar consulta para nodo virtual de películas con revenue de 500 mdd de la compañía sony
- Generar consulta para nodo virtual de películas con revenue de 500 mdd de la compañía universal
- Generar las consultas necesarias para los enlaces virtuales que se quieren observar, en este caso una consulta para la agrupación de artistas con la etiqueta de “reviewed” que se relaciona del nodo virtual de personas hacia el nuevo nodo virtual de películas de la compañía sony.

Como observamos se debe tener un dominio a nivel experto del lenguaje cypher para lograr agrupar la información en nodos y aristas virtuales, además cabe resaltar que no se puede realizar ninguna operación de cypher sobre nodos o aristas virtuales.

Usando la herramienta de sumarización y una interfaz gráfica, hace esta tarea sea una operación sencilla, donde solo se tiene que realizar las dos consultas para las agrupaciones una donde se obtiene el supernodo de películas con ingresos de 500 Millones de dólares y otra para supernodo de personas que hicieron revisión de ciertas películas, a continuación, se muestran las dos consultas:

- `MATCH(m:Movie) WHERE m.revenue = 500 RETURN id(m) AS ID //revenue 500`
- `MATCH(p:Person)-[:REVIEWED]->(m:Movie) WHERE m.title = 'Jerry Maguire' OR m.title = 'The Da Vinci Code' OR m.title = 'The Replacements' OR m.title = 'Jerry Maguire' OR m.title = 'Cloud Atlas' RETURN DISTINCT id(p) AS ID //target person`

Para poder comprender las relaciones y agrupaciones observamos el reslutado de la Figura 36, donde obtenemos dos supernodos.

El peso de ambos supernodos es de 3 y observamos que el grupo de personas está ligeramente ligado con una revisión del grupo de películas con ingresos de 500.

El supernodo de personas tiene un auto lazo con peso de 2 que indica que dos personas dentro de ese supernodo siguen a otra.

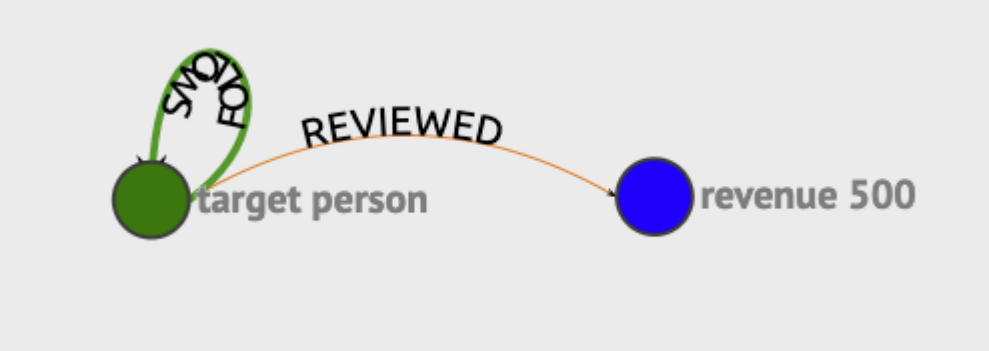

Figura 36: Sumarización de personas y películas por ingreso.

Como analistas necesitamos mas detalles de como esta compuestos internamente los dos supernodos sumarizados, para ello se pueden realizar una operación sobre el supernodo que se llama reagrupamiento por atributo, al hacer el reagrupamiento por el atributo por género del supernodo de personas observamos que se genera dos nuevos supernodos y las relaciones de dicha reagrupación son calculadas nuevamente las cuales son mostradas en la Figura 37.

Lo mismo hacemos con el supernodo de películas, hacemos un reagrupamiento por su atributo de compañía y observamos el resultado en la Figura 38.

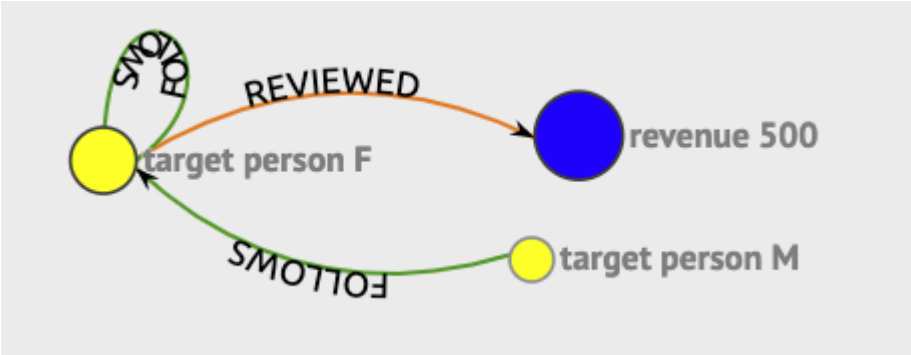

Figura 37: Split de supernodo de personas por atributo de género.

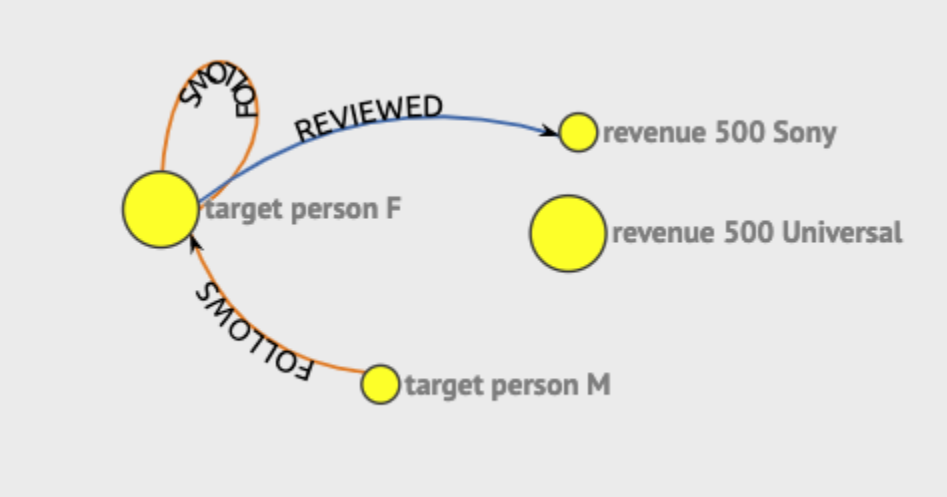

Figura 38: Split de supernodo de revenue 500 por atributo compañía.

6.1.2. Agrupación y reagrupación, funciones de agregación e histograma con consultas complejas.

En la base de datos de películas y personas queremos hacer la agrupación de personas por el número de películas que han actuado, y agrupar por las películas en tres rangos de fecha, **y observar la influencia de las personas con mayor o menor número de películas sobre los grupos de películas por fecha**. Para ello realizamos las siguientes cuatro consultas.

- `MATCH(p:Person)-[:ACTED_IN]->(m:Movie) WITH p, count(m) AS rel RETURN rel AS TITLE, collect(id(p)) AS IDS //NUM_ACTED`

- MATCH(m:Movie) WHERE m.released >= 1975 AND m.released <= 1995 RETURN id(m) AS ID //1975-1995 Movies
- MATCH(m:Movie) WHERE m.released > 1995 AND m.released <= 2000 RETURN id(m) AS ID //1995-2000 Movies
- MATCH(m:Movie) WHERE m.released > 2000 AND m.released <= 2012 RETURN id(m) AS ID //2000-2012 Movies

Obteniendo como resultado el grafo sumarizado de la Figura 39.

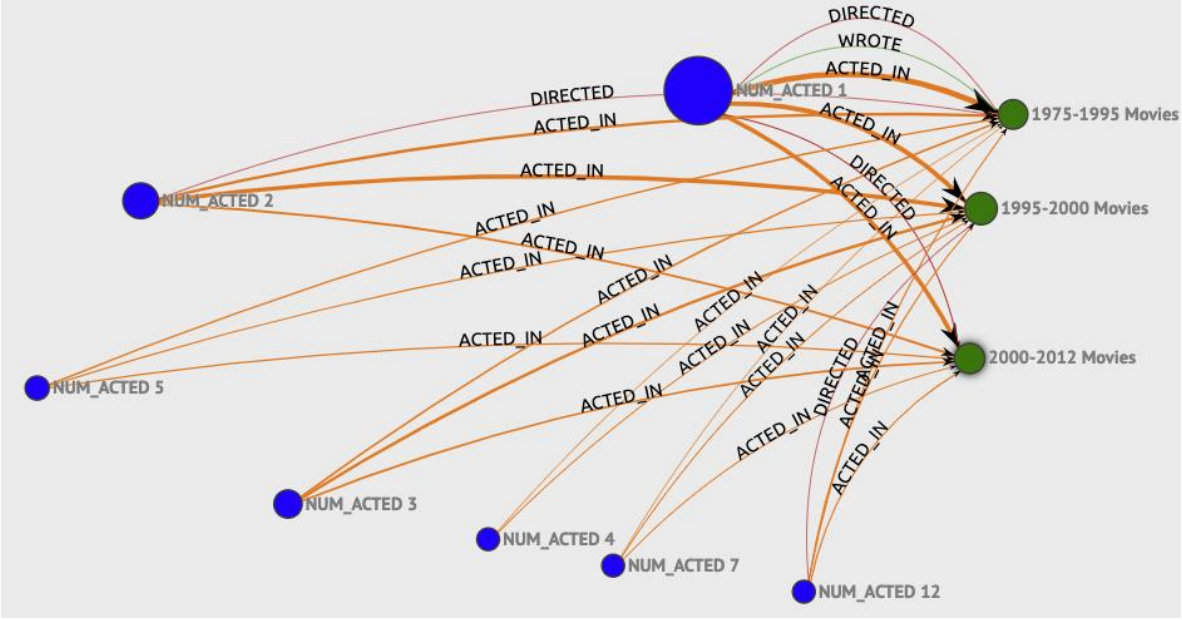

Figura 39: Grafo sumarizado de agrupación de personas por número de películas en las que actuaron y agrupación de películas por fecha.

Con los resultados obtenidos observamos que tenemos un gran grupo donde las personas que participaron en solo una película es grande, si nos posicionamos en el nodo donde el número de películas en las que actuaron es 1 y la arista dirigida hacia películas de 1975-1995, podemos ver sus características mostradas en la Figura 40.

Podemos observar de manera fácil los grupos de actores formados por la cantidad de películas en las que han actuado y en sus relaciones observamos un patrón de que tan fuerte o débilmente tienen influencia sobre grupos de películas de cierta época.

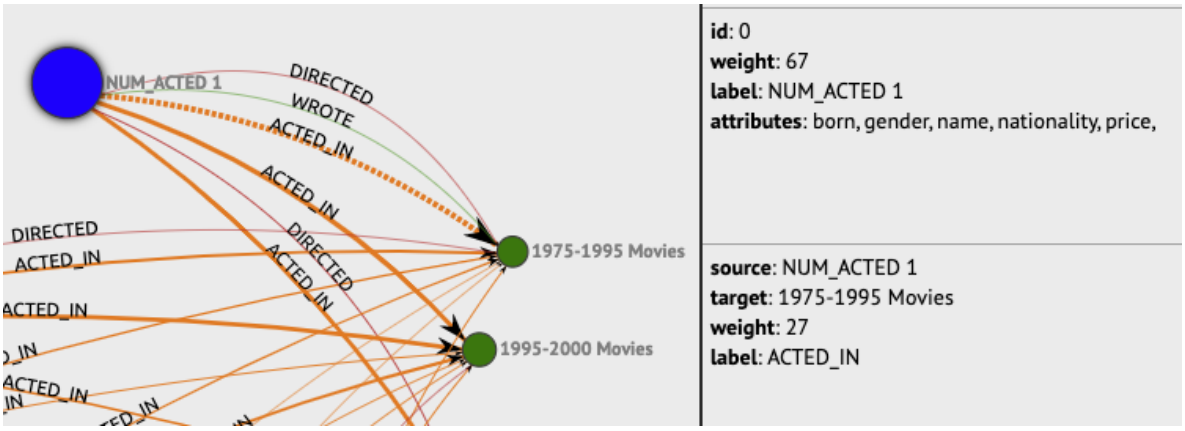

Figura 40: Características del supernodo de personas con número de películas que actuaron es uno y su relación con el supernodo de películas 1975-1995.

Para poder comprender lo que sucede en el supernodo "1975-1995 Movies" y poder analizar que patrón se tiene en dicho supernodo, interactivamente queremos observar supernodos agrupados por número de estrellas del supernodo de películas de 1975-1995, aplicamos la operación de reagrupamiento del nodo de películas 1975-1995 por el atributo de número de estrellas y obtenemos como resultado tres nuevos nodos, donde al final de la etiqueta de los nuevos supernodos tenemos una descripción del grupo generado, también observamos las nuevas aristas calculadas hacia los nuevos grupos, Figura 41 y Figura 42.

Con este análisis podemos concluir que las películas con cantidad de 2 estrellas son las que tienen mas influencia para los actores que participaron solo en una película.

Figura 41: Operación de reagrupamiento por atributo estrellas.

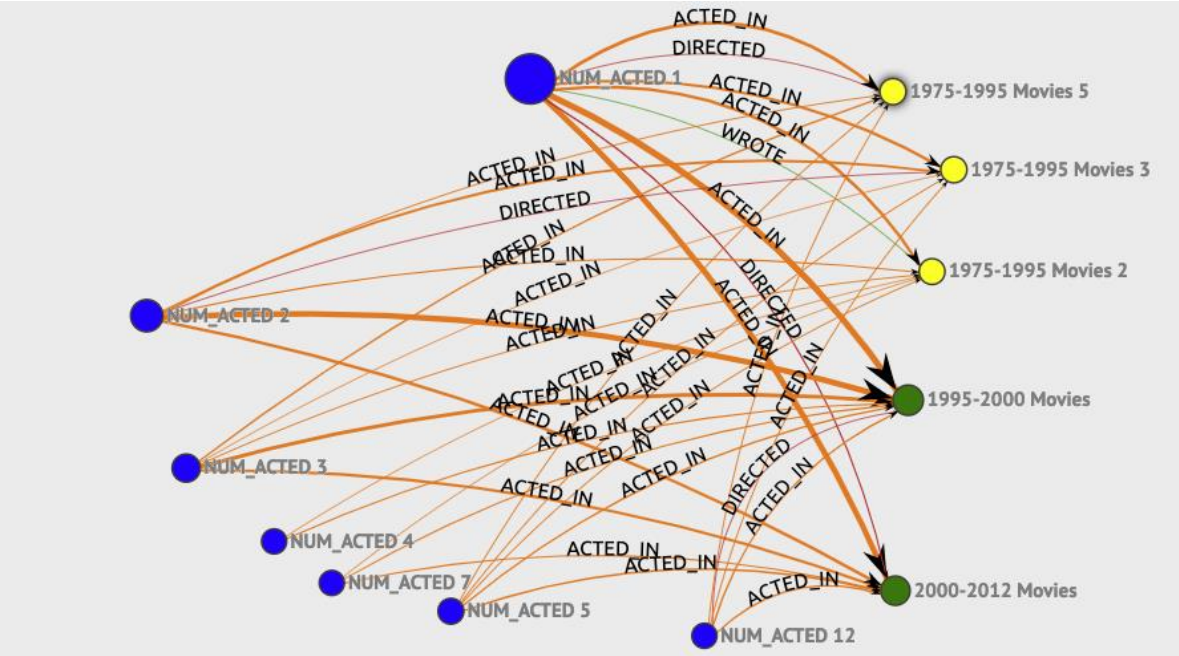

Figura 42: Resultado de reagrupamiento por atributo estrellas de supernodo peliculas 1975-1995.

Como analistas necesitamos comprender mas información de algún patrón que compone al supernodo de personas que actuaron en solo una película para ello necesitamos obtener información sobre la distribución e información sobre el atributo de año de nacimiento de las personas del supernodo **NUM_ACTED 1,** utilizamos las funciones de agregación e histograma. Queremos saber cuántos elementos nacidos en 1956 contiene, fecha máxima, mínima y una distribución de 10 bloques de las edades, ver Figura 43, Figura 44 y Figura 45.

Con esta información podemos concluir que este supernodo esta influenciado por personas en un rango de edades de 1955 a 1972

Figura 43: Selección del supernodo NUM_ACTED 1 y atributo born.

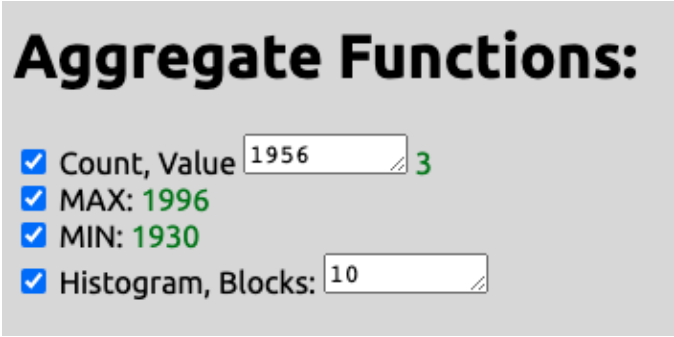

Figura 44: Funciones de agregación, contar valor de fecha 1956 e histograma con 10 bloques para atributo born

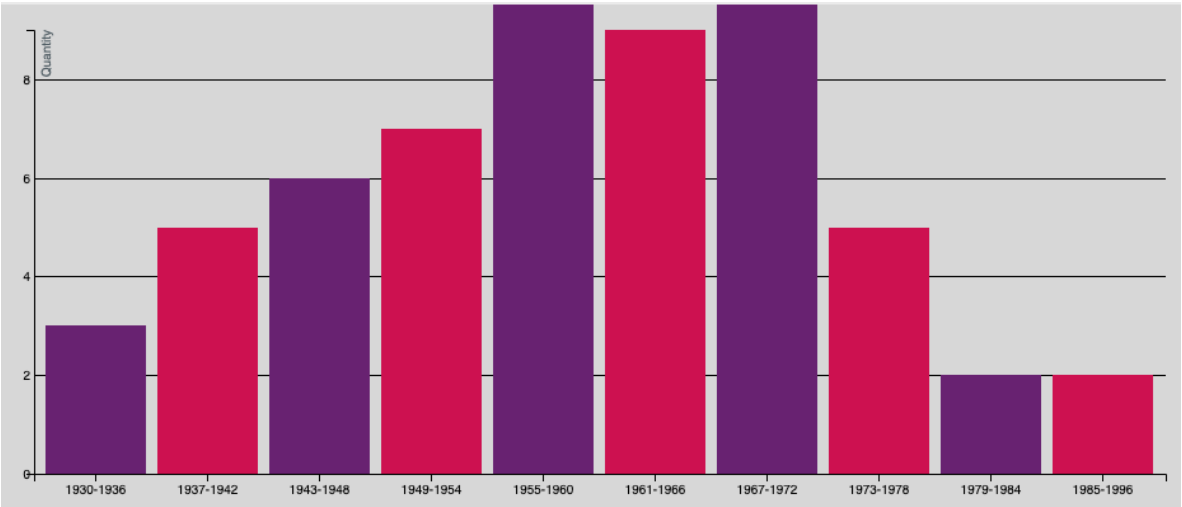

Figura 45: El resultado del histograma con 10 bloques del atributo born.

6.1.3. Intersecciones entre supernodos

En la base de datos de películas y personas necesitamos observar la agrupación de personas por el atributo de “price”, este atributo representa el nivel de ingresos de cada persona en una escala de 1 al 8, siendo el 8 el máximo nivel de ingresos, otra agrupación de numero de actores por numero de películas actuadas y una agrupación de películas por el atributo “stars”, este atributo representa el nivel en que la película está evaluada en una escala de 1 al 5, siendo 5 la mejor evaluación. Para ello realizamos las siguientes consultas en la herramienta de sumarización.

- MATCH(a:Person) RETURN a.price AS TITLE, collect(id(a)) AS IDS //PERSON BY PRICE

- MATCH(p:Person)-[:ACTED_IN]->(m:Movie) WITH p, count(m) AS rel RETURN rel AS TITLE, collect(id(p)) AS IDS //NUM_ACTED",
- MATCH(a:Movie) RETURN a.stars AS TITLE, collect(id(a)) AS IDS //MOVIE BY STARS"

De estas tres consultas obtenemos un grafo sumarizado con supernodos de películas y personas representando las agrupaciones correspondientes mostradas en la Figura 46.

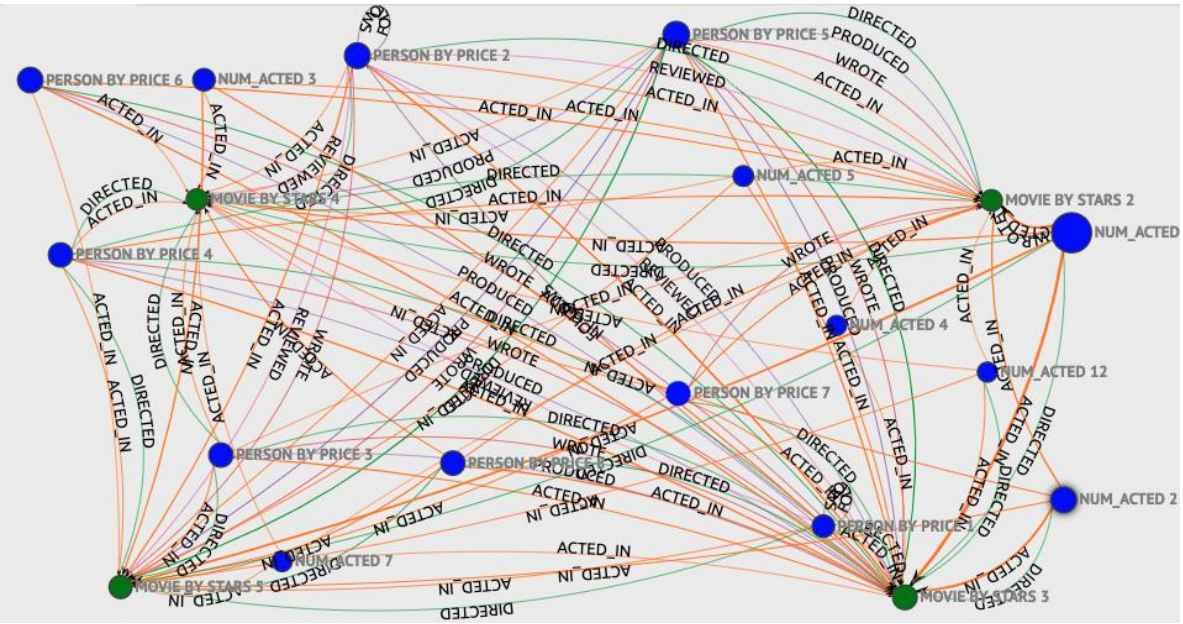

Figura 46: Sumarización de grafo de personas por "price" y "numero de películas actuadas", y películas por numero de "stars".

Como analistas necesitamos comprender la información que los supernodos de personas comparten entre si, de esta manera podemos observar como están distribuidos los nodos de personas en los supernodos de ingresos y supernodos personas con cantidad de películas actuadas.

Para poder visualizar esta información y ver los patrones, la herramienta de sumarización nos entrega información de la intersección entre supernodos si deseamos visualizar los nodos que comparten dos supernodos.

Observando la Figura 47, que contiene la representación de grafo sumarizado de la Figura 46, observamos que cada nodo es una gráfica de pastel y con ayuda de la paleta de colores de la Figura 48, si nos enfocamos

en el nodo con identificador 2, observamos que el color morado representa el porcentaje de los nodos que son únicos para el supernodo, el color rosa nos representa los nodos que comparten el supernodo 2 con respecto al supernodo 8, el color verde representa los nodos que comparten el supernodo 2 con respecto al supernodo 10 y así sucesivamente.

- El supernodo con id 2 representa información de las personas que con un nivel de ingreso de 5.
- El supernodo con id 8 representa la información de las personas que actuaron en solo una película.
- El supernodo con id 10 representa la información de las personas que actuaron en 3 películas.

Con esta información de la Figura 47, podemos ver de una manera rápida y fácil la relación en los datos que la mayoría de las personas que tienen un ingreso de nivel 5 solo han actuado en una película. Lo cual muestra claramente un indicador que por su ingreso de nivel alto solo fueron contratados para una película.

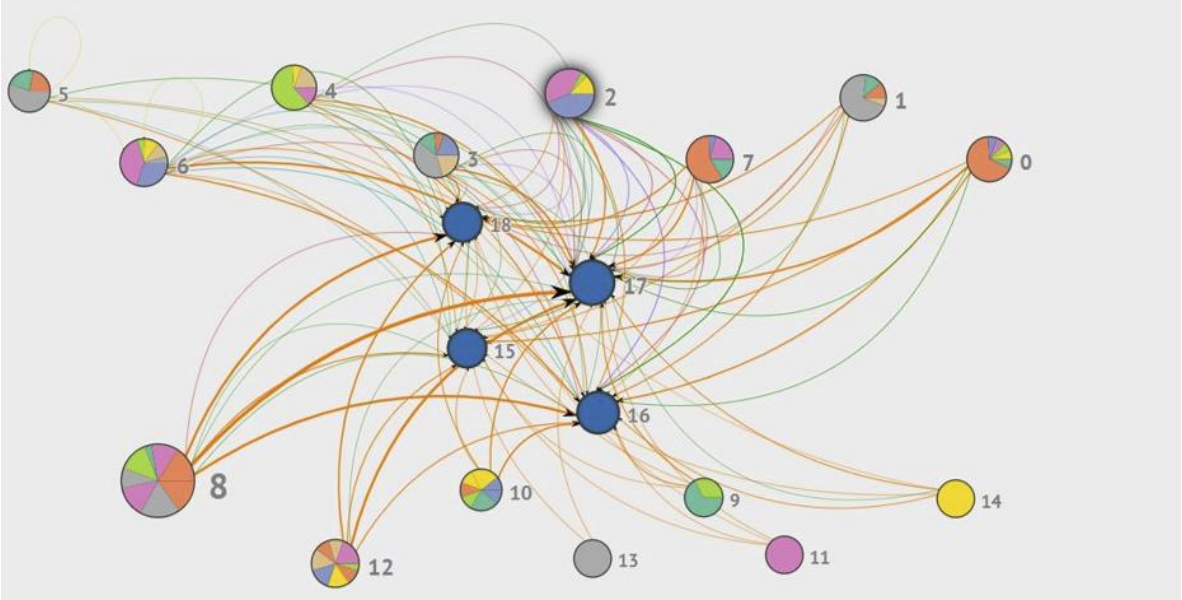

Figura 47: Gráfica de pastel de supernodos representando intersecciones.

Figura 48: Paleta de colores representando color de nodo e intersecciones.

6.2. DataSet de Twitter, User y Hashtag

En la base de datos de Twitter, User y Hashtag queremos agrupar los usuarios por zona horaria y **analizar la tendencia de los posts y re-tweets sobre la desaparición de un joven.** Hacemos las siguientes consultas para las agrupaciones.

- MATCH(a:User) where a.time_zone CONTAINS 'Mexico'OR a.time_zone CONTAINS 'Guadalajara'OR a.time_zone CONTAINS 'Monterrey' RETURN a.time_zone AS TITLE, collect(id(a)) AS IDS //timeZone
- MATCH(a:Tweet)-[:TAGS]->(h:Hashtag) where h.text CONTAINS 'DondeEstaMancera' OR h.text CONTAINS 'ManceraDondeEstaMarcoAntonio' OR h.text CONTAINS 'DóndeestáMancera' OR h.text CONTAINS 'DóndeEstáMiguelÁngelMancera'RETURN h.text AS TITLE, collect(id(a)) AS IDS// TWEET

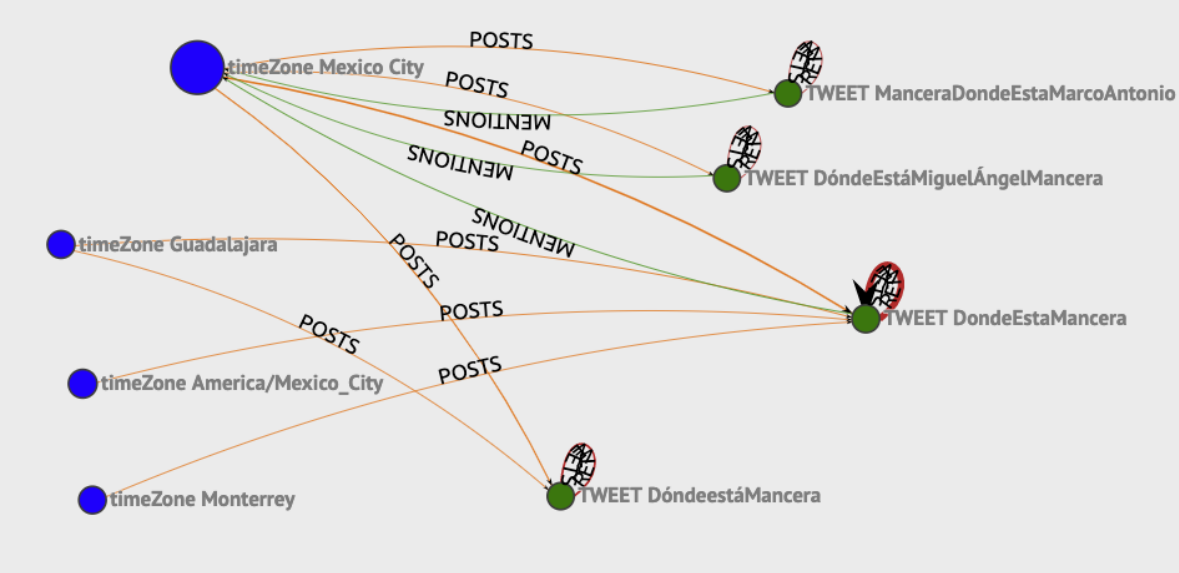

Figura 49: Grafo sumarizado con agrupaciones de usuarios por zona horaria y agrupaciones de tweets por contenido de texto.

En la Figura 49, observamos los resultados, donde vemos las agrupaciones de cuatro zonas horarias que definimos en la consulta las cuales tienen una influencia sobre el tweet con texto “DondeEstaMancera”, así mismo también observamos que fue el tweet con más re-tweets simplemente por observar el peso de la arista RETWEETS. Con la información desplegada en el grafo vemos que fue tendencia de reTweet

“DondeEstaMancera”, y vemos que las 4 zonas horarias que definimos tuvieron la misma influencia respecto a “POSTS”.

Como analistas una información útil y es saber los “influencers” con los que cuenta un grupo, para ello, de forma interactiva queremos ver la distribución de seguidores para los usuarios en zona horaria de Mexico City, seleccionamos el nodo y aplicamos las funciones de agregación, Figura 50 y Figura 51.

Observamos en la Figura 52 que la cantidad de seguidores para el rango de 1 a 783,271 es de 3,330 usuarios, y en la Figura 53 para el rango de seguidores 783,272 a 1,566,541 tenemos 4 usuarios y en la Figura 54 para el rango de seguidores de 1,566,542 a 2,349,811 es de 2 usuarios. **Lo cual nos indica que en ese grupo hay una poca cantidad de usuarios influencers que son los que superan los millones de seguidores.**

Figura 50: Selección del supernodo “TimeZone Mexico City” y atributo “followers_count”.

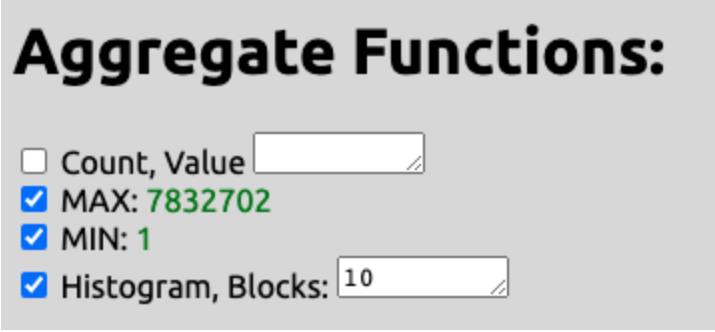

Figura 51: Funciones de agregación para supernodo “TimeZone Mexico City”.

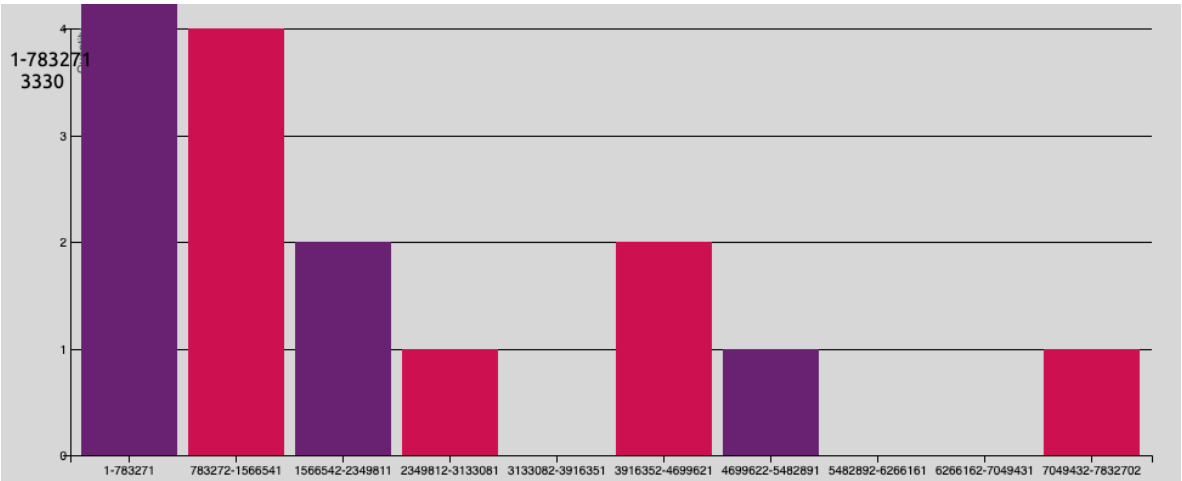

Figura 52: Distribución de seguidores bloque 1.

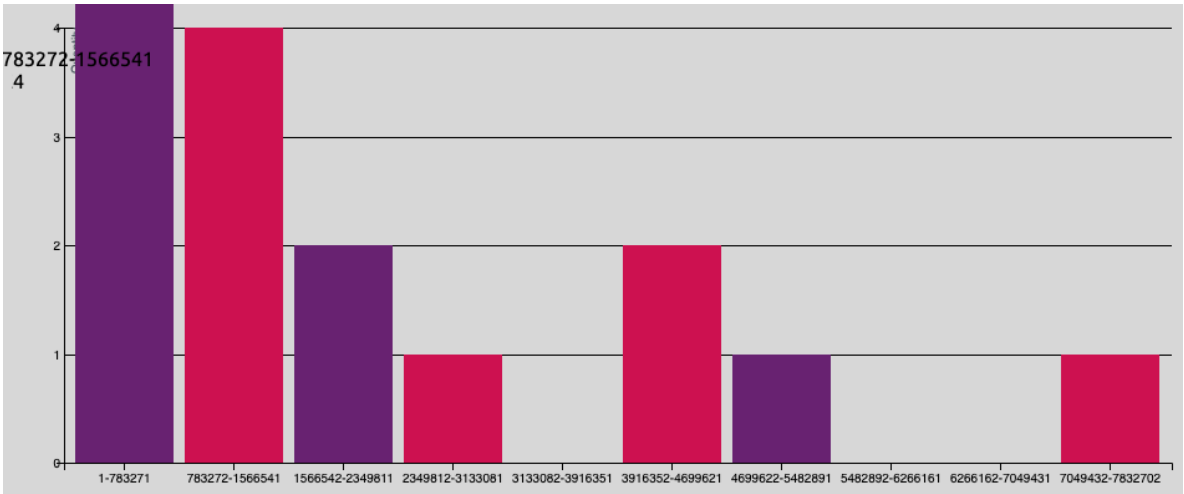

Figura 53: Distribución de seguidores bloque 2.

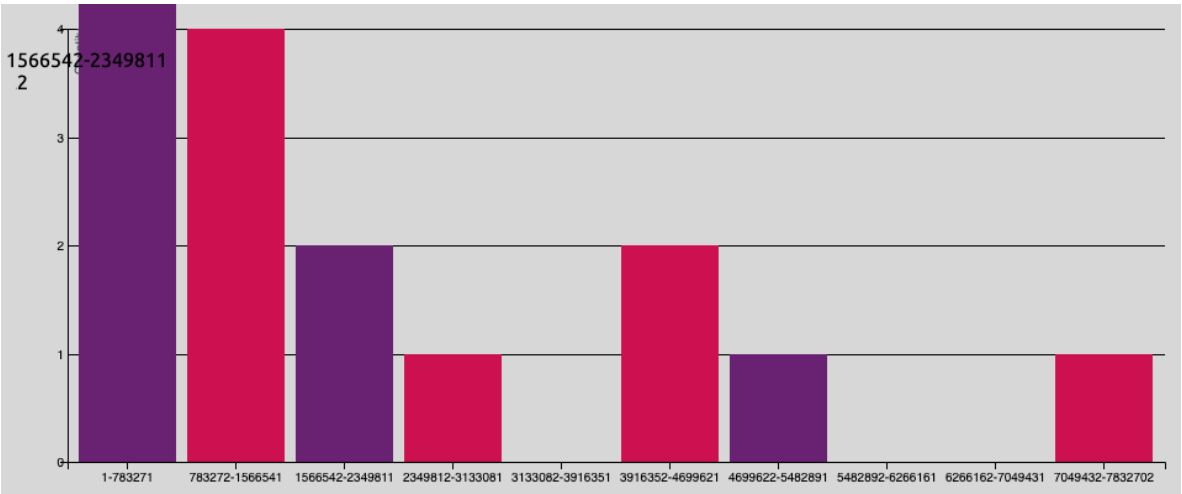

Figura 54: Distribución de seguidores bloque 3.

Para poder comprender que día fue el que se hizo la mayor cantidad de tweets con el texto "DondeEstaMancera", en este caso seleccionamos el supernodo con etiqueta TWEET DondeEstaMancera para hacer una reagrupación por el atributo de dia, ver Figura 55, como resultado obtenemos 3 nuevos supernodos con la etiqueta del día que fue creado el tweet mostrado en la Figura 56.

Observando los pesos de los supernodos reagrupados vemos que el día 28 se generaron 135 tweets, los demás días tienen un peso muy bajo, dia 29 con 7 y día 3 con 1. Figura 57, Figura 58 y Figura 59. De esta manera podemos concluir que el dia 28 fue cuando fue tendencia el tweets con texto que contenían la cadena "DondeEstaMancera".

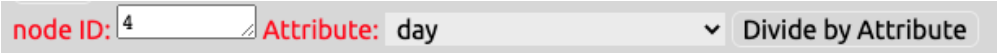

Figura 55: Operación de reagrupamiento por atributo día.

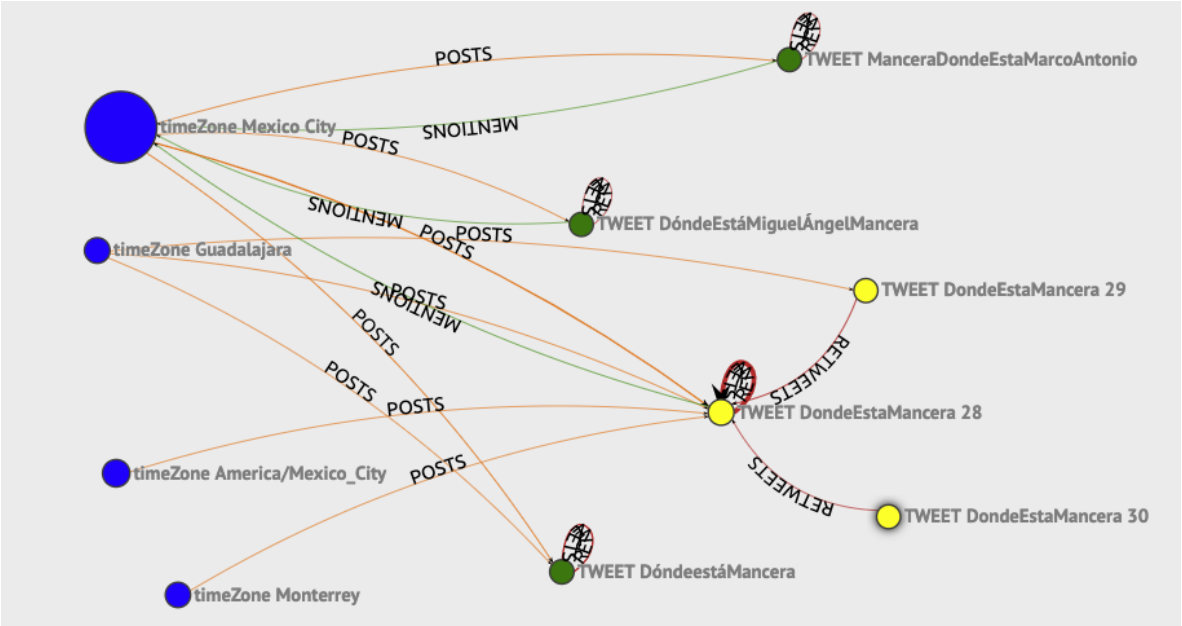

Figura 56: Resultado de reagrupamiento por atributo dia de supernodo TWEET DondeEstaMancera.

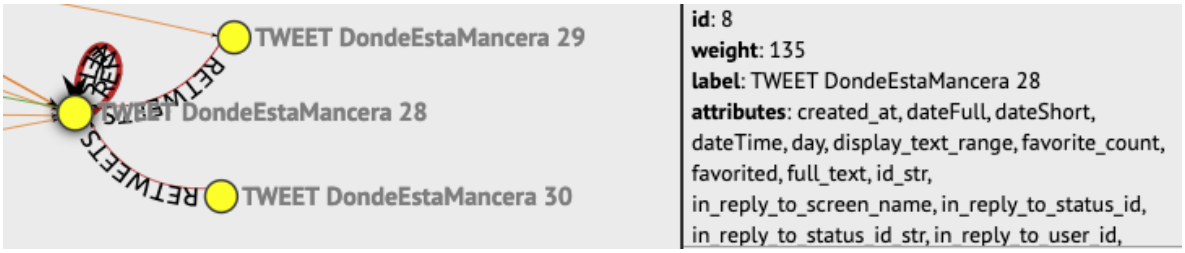

Figura 57: Peso de supernodo TWEET DondeEstaMancera 28

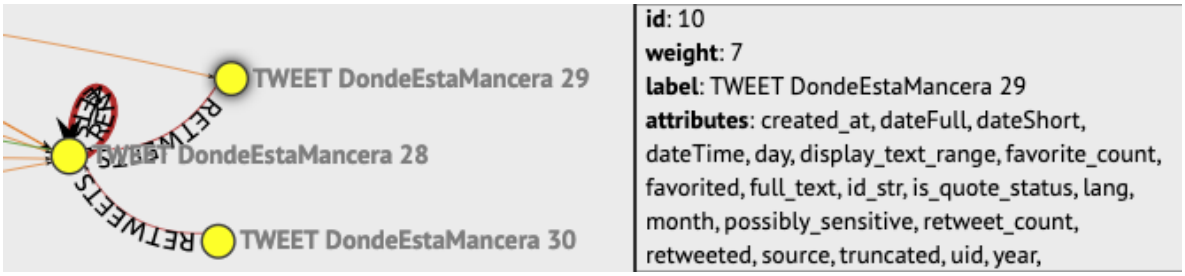

Figura 58: Peso de supernodo TWEET DondeEstaMancera 29.

Figura 59: Peso de supernodo TWEET DondeEstaMancera 30.

7. CONCLUSIONES

7.1. Conclusiones

En este trabajo se abordó el problema del análisis y visualización de grafos con miles o más nodos y relaciones. Debido a la gran cantidad de información las herramientas existentes no permiten identificar patrones entre grupos de nodos que cumplan con características similares.

Con este objetivo se propuso una herramienta que, como servicio REST, permite la sumarización de un grafo almacenado en neo4j. La sumarización obtenida está basada en las consultas definidas por el usuario. Como parte de esta propuesta se incluye la capacidad de procesar las consultas, hacer histogramas, ejecutar funciones de agregación y visualizar reagrupación por atributo. Se pudo, probar la propuesta usando 2 escenarios que analizar: uno de pequeña escala que contiene actores y películas y otro de gran escala de tweets, para ello se buscó la sumarización del grafo desde una perspectiva más genérica permitiendo al usuario definir el agrupamiento desde las consultas definidas en cypher. Se lograraron obtener grafos sumarizados donde los supernodos y super aristas representan la agrupación de la información que se quiso analizar. La herramienta nos permitió obtener información de grafos sumarizados, observando aquellos supernodos que tenían más influencia sobre otros y, de forma similar, las relaciones con más peso.

Además, una vez obtenidos los grafos sumarizados fue posible hacer un análisis exhaustivo sobre los supernodos, de manera interactiva se pudo aplicar funciones de agregación, ver un histograma, desagrupar un supernodo por atributos para visualizar información con más detalle. Esta herramienta facilitó analizar los grupos con mayor peso y ver que tan fuertemente están relacionados con otro supernodos. Haciendo uso de intersecciones pudimos observar la relación entre nodos del mismo tipo que no contienen aristas que los relacionen. **El análisis de información a gran escala ayudó ver las tendencias y patrones de comportamiento de los grupos sumarizados con gran facilidad.**

7.2. Trabajo Futuro

En una siguiente etapa se espera:

1) Agregar memoria distribuida a la solución. Esto permitirá que múltiples usuarios del servicio REST puedan utilizar la herramienta al mismo tiempo, y también permitirá a un usuario la visualización de múltiples grafos al mismo tiempo.

2) Implementar una función que permita al usuario reagrupar una selección o todos los supernodos basado en algún atributo.

BIBLIOGRAFÍA

[1] R. Pienta, F. Hohman, A. Endert, A. Tamersoy, K. Roundy, C. Gates, S. Navathe y D. H. Chau, «VIGOR: Interactive Visual Exploration of Graph Query Results,» *IEEE Transactions on Visualization and Computer Graphics,* vol. 24, nº 1, p. 215–225, 2018.

[2] [En línea]. Available: https://gephi.org/features/. [Último acceso: 31 october 2020].

[3] G. Pavlopoulos, D. Paez-Espino, N. Kyrpides y I. Iliopoulos, «Empirical Comparison of Visualization Tools for Larger-Scale Network Analysis", Advances in Bioinformatics,» 2017. [En línea]. Available: https://www.hindawi.com/journals/abi/2017/1278932/. [Último acceso: 1 noviembre 2020].

[4] D. Khokhar, Gephi Cookbook, Packt Publishing, 2015.

[5] D. Auber, D. Archambault, R. Bourqui, M. Delest, J. Dubois, A. Lambert, M. Patrick, M. Mathiaut y G. Melançon, «TULIP 5,» *Encyclopedia of Social Network Analysis and Mining,* pp. 1-28, 2017.

[6] [En línea]. Available: https://neo4j.com/labs/apoc/4.1/virtual/. [Último acceso: 8 noviembre 2020].

[7] [En línea]. Available: https://neo4j.com/labs/apoc/4.1/virtual/virtual-nodes-rels/. [Último acceso: 8 novimebre 2020].

[8] D. Gosnell y M. Broecheler, The Practitioner's Guide to Graph Data, O' Reilly Media, Inc., 2020.

[9] [En línea]. Available: https://neo4j.com/developer/graph-database/. [Último acceso: 1 noviembre 2020].

[10] [En línea]. Available: https://en.wikipedia.org/wiki/Neo4j. [Último acceso: 1 noviembre 2020].

[11] O. Panzarino, Learning Cypher, Packt Publishing, 2014.

[12] G. Ankur, Neo4j Cookbook, Packt Publishing, 2015.

[13] [En línea]. Available: https://neo4j.com/developer/cypher/guide-sql-to-cypher/. [Último acceso: 1 noviembre 2020].

[14] [En línea]. Available: https://neo4j.com/docs/cypher-manual/current/clauses/match/#related-nodes. [Último acceso: 1 noviembre 2020].

[15] K. Ullah Khan, W. Nawaz y Y. K. Lee, «Set-Based Unified Approach for Attributed Graph Summarization,» de *2014 IEEE Fourth International Conference on Big Data and Cloud Computing*, 2014.

[16] Y. LIU, A. DIGHE, T. SAFAVI y D. KOUTRA, «Graph Summarization: A Survey,» *ACM Computing Surveys*, vol. 51, nº 3, p. 34, 2018.