Facultad de Ingeniería
Escuela de Ingeniería Civil Informática

ESTRATEGIAS DE DISEÑO FÍSICO PARA UN SISTEMA GESTOR DE BASE DE DATOS ORIENTADO A GRAFOS

Por

Francisco Daniel Castillo Vera

Trabajo realizado para optar al Título de
INGENIERO EN INFORMÁTICA
Prof. Guía: Ana Aguilera Faraco
Junio 2021

Certifico que he leído este documento y que, en mi opinión, es adecuado en ámbito y calidad como trabajo para optar al título de Ingeniero Civil en Informática.

Ana Aguilera Faraco

Firmado digitalmente por Ana Aguilera Faraco
Nombre de reconocimiento (DN): cn=Ana Aguilera Faraco, o=UV, ou=Escuela de Ingeniería Informática, email=ana.aguilera@uv.cl, c=CL
Fecha: 2021.06.06 13:05:43 -04'00'

Ana Aguilera Faraco Profesor Guía

Certifico que he leído este documento y que, en mi opinión, es adecuado en ámbito y calidad como trabajo para optar al título de Ingeniero Civil en Informática.

Marlene Goncalves Da Silva Profesor Informante

Certifico que he leído este documento y que, en mi opinión, es adecuado en ámbito y calidad como trabajo para optar al título de Ingeniero Civil en Informática.

Rodrigo Olivares Órdenes Profesor Informante

Aprobado por la Escuela de Ingeniería Civil en Informática, UNIVERSIDAD DE VALPARAÍSO.

Resumen

Las bases de datos no relacionales se encuentran en constante perfeccionamiento, buscando mejorar su capacidad de procesamiento. Entre ellas, destacan las bases de datos orientada a grafos. Este tipo de bases de datos ha visto incrementado su uso por su característica principal de cómo se relacionan sus datos. Es bien sabido que para los desarrolladores de bases de datos, los aspectos ligados al diseño físico siempre han sido un punto álgido y de especial cuidado, ya que de ello depende el rendimiento de la base de datos y las aplicaciones que se desarrollen sobre ella. Desde ahí surge la necesidad de plantear y validar estrategias de diseño físico que logren optimizar la búsqueda de datos. Linked Data Benchmark Council presenta el proyecto que lleva por nombre Social Network Benchmark que busca promover que los desarrolladores planteen o propongan diferentes estrategias de diseño físico. Linked Data Benchmark Council Social Network Benchmark expone una problemática en la cual se adjuntan diferentes consultas agrupadas en dos grupos: Interactive Workload y Business Interactive Workload. En este documento se analizó cada una de las consultas logrando plantear tres estrategias de diseño físico que fueron implementadas en las 25 consultas del grupo de Business Interactive Workload. Los experimentos fueron desarrollados sobre la plataforma Neo4j con las cargas de trabajo de 1 GB, 10 GB y 100 GB. A través de esta experimentación se consiguió validar la efectividad y escalabilidad de las estrategias de diseño físico: Reescritura de Consulta, Materialización de Caminos y Creación de Índices. Según los resultados se puede precisar que se logró reducir el tiempo de ejecución. Para la primera guía de diseño se redujo en promedio un 59,21 %, para la segunda guía de diseño se consiguió una disminución en promedio de 77,02 %, y finalmente, para la tercera guía de diseño se alcanzó una reducción en promedio de 44,94 %.

Agradecimientos

En primer lugar, me gustaría agradecer el apoyo, dedicación, esfuerzo y sacrificio de mi madre durante todos estos años, gracias a ella he podido cumplir con este importante objetivo. De igual forma, quiero agradecer el respaldo que me han dado mis hermanos que sin duda este logro también es para ellos para que sigan persiguiendo sus objetivos y sueños. De igual manera, dar gracias a mis amistades que me apoyaron y contuvieron en momentos muy difíciles durante mi vida universitaria.

Destacar el enorme trabajo que efectúan los profesores de la escuela. Agradecer inmensamente la ayuda, dedicación y esfuerzo que hacen los docentes para ir formando grandes profesionales de calidad.

Quiero agradecer el apoyo brindando por Marcos Jota y Gabor Szarnyas, que sin sus recomendaciones este trabajo hubiera sido mucho más complejo de resolver, sin embargo, se consiguió un gran resultado.

Finalmente, dar gracias al estado por entregarme la posibilidad de estudiar mediante los beneficios entregados. Estos beneficios me dieron la oportunidad de poder desarrollarme como profesional en una destacada universidad estatal como lo es la Universidad de Valparaíso. Espero que con el paso de los años más personas tengan esta posibilidad para de esta forma tener más profesionales de calidad en nuestro país.

Índice general

Índice de Tablas

Índice de figuras

Índice general

Índice de figuras

Índice de tablas

Lista de Acrónimos

ACID *Atomicity, Consistency, Isolation, Durability*

BASE *Basically Available Soft state Eventual consistency*

BDOG *Base de Datos Orientada a Grafos*

BI *Business Intelligence*

BIW *Business Intelligence Workload*

CRUD *Create, Read, Update and Delete*

Datagen *LDBC-SNB Data Generator*

DB Hits *Database Hits*

IW *Interactive Workload*

LDBC *Linked Data Benchmark Council*

NoSQL *Not Only SQL*

OLTP *OnLine Transaction Processing*

RAM *Random Allocated Memory*

RDF *Resource Description Framework*

SaaS *Software as a Service*

SGBD *Sistema Gestor de Base de Datos*

SNB *Social Network Benchmark*

SQL *Structure Query Lenguage*

Capítulo 1

Introducción

Existen diversos paradigmas para bases de datos, en diversos contextos más allá de los sistemas basados en bases de datos *Structure Query Lenguage* (SQL) [3]. Ante el gran crecimiento de los tamaños de datos, se han ido evidenciando las desventajas como lo es la rigidez que poseen en su estructura [4]. Al momento de querer realizar una modificación, se hace muy complejo porque se ve repercutido el esquema de la base de datos [4].

Las bases de datos no relacionales se definen como bases de datos *Not Only SQL* (NoSQL), donde el modelo por el cual se rigen es más permisivo y se conoce como *Basically Available Soft state Eventual consistency* (BASE) [5]. De la descomposición de este acrónimo tenemos primero ”Basically Available” que garantiza la disponibilidad de los datos ante cualquier solicitud incluyendo los posibles errores. La segunda instancia ”Soft state” precisa que el estado del sistema puede variar con el tiempo si fuera requerido. Por último, ”Eventual consistency” se asegura que el sistema deberá ser consistente una vez que deje de recibir información [5]. Las bases de datos NoSQL usan una variedad de modelos de datos para acceder y administrar datos. Así mismo, se adaptan perfectamente a muchas aplicaciones modernas como dispositivos móviles, web y videojuegos que requieren ***escalabilidad***, ***flexibilidad***, ***alto rendimiento*** y que sean ***altamente funcional*** como lo precisa Amazon [6].

Existen diferentes tipos de orientaciones de los sistemas, uno de esos modelos NoSQL corresponde al de *Base de Datos Orientada a Grafos* (BDOG), como lo indica Oracle España [7]. Este paradigma se compone de un mecanismo para modelar datos dándole un mayor énfasis a las relaciones que disponen. En consecuencia, la gran diferencia con las bases de datos relaciones es que permite establecer mayor información sobre los vínculos de sus datos [8]. Recientemente, los investigadores se han interesado en las BDOG por la naturaleza de las aplicaciones emergentes, donde permite analizar con mayor precisión sus datos y cómo se relacionan consiguiendo un respaldo para la toma de

decisiones. Para esto, es necesario contar con un óptimo funcionamiento de nuestra base de datos, lo que se logra con implementar correctas estrategias de diseño físico [9]. El propósito del diseño físico de la base de datos es traducir la descripción lógica de los datos en las especificaciones técnicas para almacenar y recuperar datos [9]. El objetivo es crear un diseño para almacenar datos que proporcionen un rendimiento adecuado y aseguren la integridad, seguridad y capacidad de recuperación de la base de datos [10]. El diseño físico es específico para el motor de la base de datos, lo que significa que los objetos definidos durante el diseño físico pueden variar según el motor de la base de datos que se utilice [9]. *Linked Data Benchmark Council* (LDBC) presenta el proyecto *Social Network Benchmark* (SNB) que desafía a los desarrolladores a proponer o validar estrategias de diseño físico sobre BDOG [11]. En este trabajo de título, se propone el diseño físico de una BDOG sobre el desafío planteado por LDBC SNB con los datos proveniente del *LDBC-SNB Data Generator* (Datagen), estos datos serán almacenados en Neo4j que es a una BDOG.

Finalmente, este documento se compone de 5 capítulos. En el capítulo "Marco Referencial" se definen los conceptos necesarios para abordar el problema a resolver. Además. se presenta el estado del arte en el que se analizan los trabajos e investigaciones relacionados a las estrategias de diseño físico sobre BDOG. En el capítulo de "Definición del Problema y Análisis de Requerimientos", se presenta detalladamente el problema a resolver. Adicionalmente, se define la metodología y requerimientos esenciales para poder llevar a cabo la etapa de experimentación. En el capítulo "Guías de Diseño Físico" se expone cada guía de diseño físico acompañado de un ejemplo que nos permita entender su implementación. En el capítulo de "Estudio Experimental" se prueban las estrategias de diseño físico para visualizar su comportamiento. En el capítulo "Análisis y Comparación de Resultados" se expone los resultados obtenidos de los experimentos sobre cada carga de trabajo para ser analizados minuciosamente. El último capítulo "Conclusiones" se presenta una recapitulación del trabajo desarrollado abordando los principales puntos destacando los hallazgos más importantes dentro del presente documento.

Capítulo 2

Marco conceptual y estado del arte

En este capítulo se definen los conceptos esenciales para poder abordar el problema a resolver. Uno de ellos es el diseño de una base de datos, el cual se compone de 3 etapas: ***diseño conceptual***, ***diseño lógico*** y ***diseño físico***, en este trabajo nos concentraremos en el diseño físico. El diseño físico será implementado en una BDOG cuya base de datos definida para este trabajo es Neo4j mediante su lenguaje de consulta Cypher. Finalmente, en el estado del arte se exhiben los trabajos más valiosos en relación a los temas abordados por este trabajo de título, como lo son el diseño físico y las BDOG.

2.1. Marco conceptual

En el marco conceptual se exponen los conceptos básicos que se deben conocer para comprender desde el fundamento teórico que respalda las estrategias de diseño físico propuestas hasta el análisis de las métricas de evaluación que permiten validar los fundamentos teóricos.

2.1.1. Base de datos orientada a grafos

Primero que todo, se debe indicar qué es un grafo. Los autores del libro Graph Databases lo definen de la siguiente forma [12]:

"*Un grafo es simplemente una colección de vértices y aristas o, en un lenguaje menos intimidante, un conjunto de nodos y las relaciones que los conectan. Los grafos representan entidades como nodos y las formas en que esas entidades se relacionan con el mundo como relaciones.*"

De similar forma se tiene que mencionar la definición de estos autores sobre lo qué es una base de datos de grafos, como se aprecia a continuación [13].

"*Un sistema gestor de bases de datos de grafos (en adelante, base de datos de grafos) es un sistema gestor de bases de datos en línea con métodos de Crear, Leer, Actualizar y Eliminar (Create, Read, Update and Delete (CRUD)) que exponen un modelo de datos de grafos.*"

Otra definición sobre las BDOG es la investigación desarrollada por los docentes de Universidad Distrital Francisco José de Caldas, como se ve a continuación [14].

"*Una base de datos orientada a grafos es aquella que permite almacenar la información como nodos de un grafo y sus respectivas relaciones con otros nodos, permitiendo así aplicar la teoría de grafos para recorrer la base de datos.*"

El artículo presentado en IONOS [15] expone que la gran ventaja de las BDOG es la velocidad con la que se recorren las relaciones ya que estas no se calculan en el momento de la consulta, pero se mantienen en la base de datos. Las BDOG tienen ventajas para casos de uso como redes sociales, motores de recomendación y detección de fraude, cuando necesita crear relaciones entre datos y consultar rápidamente estas relaciones [15]. Adicionalmente, las BDOG cuentan con algoritmos especiales de búsqueda para cumplir con su función esencial que es simplificar y acelerar las consultas de mayor complejidad [15]. Finalmente, las grandes ventajas que nos entrega el trabajar con BDOG se compone de cuatro factores [15]: ***integridad***, ***rendimiento***, ***eficiencia*** y ***escalabilidad***. Sin embargo, también posee desventajas en lo que respecta a escalabilidad debido a que su arquitectura significa un gran desafío matemático que se mantiene en constantes desarrollo, así mismo, aún no existe un lenguaje de consulta uniforme en las BDOG [15].

2.1.2. Diseño físico

El diseño físico de una base de datos optimiza el rendimiento a la vez que se asegura la integridad de los datos al evitar repeticiones innecesarias [16]. Si bien el diseño lógico se puede realizar independiente de la eventual plataforma de base de datos, cabe destacar que muchos atributos físicos de ella dependen de los detalles y la semántica propia del *Sistema Gestor de Base de Datos* (SGBD) objetivo [17]. El diseño físico se realiza en dos etapas [17]. La primera, consiste en la conversión del diseño lógico a través de la implementación, definiciones del esquema, normalización y relaciones. La segunda etapa es posterior a la implementación que a menudo es realizada por el administrador de la base de datos la cual incluye mejorar el rendimiento, reducir las entradas y salidas en la consulta y optimizar las tareas de administración. Es así como lo define la Universidad de Utah [18]:

"*Proceso de producción de una descripción de la implementación de la base de*

datos en almacenamiento secundario; describe las relaciones de base, las organizaciones de archivos y los índices utilizados para lograr un acceso eficiente a los datos, así como las restricciones de integridad y las medidas de seguridad asociadas"

R. Ramakrishnan y J. Gehrke [19] proponen diferentes metodologías para conseguir un buen diseño físico, entregando conocimiento con la finalidad de que el lector aprenda a tomar buenas decisiones en las estrategias. Sin embargo, todo esto está basado en el lenguaje de consultas SQL por lo que se debe tener cuidado al momento de usar estas estrategias, aunque de todas formas es recomendable utilizarlo de referencia.

2.1.3. Linked Data Benchmark Council Social Network Benchmark

LDBC [20] se define como un programa de la Unión Europea que tiene por fin, desarrollar puntos de referencia sólidos en la industria para sistemas de gestión de datos de grafos y *Resource Description Framework* (RDF). Desde este objetivo nace el proyecto de SNB. Este proyecto plantea dos puntos de referencia en un conjunto de datos común a través de un generador de datos Datagen que simula una red social sintética con una estructura basada en la ley potencial *(power-law)* [21]. Además, los datos son obtenidos desde las conversaciones de los foros en DBpedia. El documento de especificaciones que publica LDBC [2] precisa 39 consultas que se dividen en dos puntos de referencia: *Interactive Workload* (IW) la cual posee 14 consultas y *Business Intelligence Workload* (BIW) que dispone de las 25 consultas restantes. A continuación, procederemos a explicar en que consiste y cuál es la función de cada carga de trabajo propuesta por LDBC.

- IW: Prueba el rendimiento de un sistema con consultas relativamente simples y actualizaciones simultáneas. Uno podría llamar a *IW* una carga de trabajo *OnLine Transaction Processing* (OLTP), pero si bien las consultas generalmente interactúan con una pequeña fracción de la base de datos, a menudo esto puede llegar a significar el trabajo de hasta cientos de miles de datos.

- BIW: Consta de consultas con una estructura compleja para analizar el comportamiento en línea de los usuarios con fines de marketing. Este problema está exclusivamente dedicado en la ejecución y optimización de consultas. A diferencia de *IW*, las consultas de *BIW* se relacionan con una mayor cantidad de datos a medida que crece la base de datos.

El documento especifica que un benchmark es valioso, si su carga de trabajo hace hincapié en la importancia de la funcionalidad técnica de los sistemas. El desarrollo de LDBC SNB está impulsado por el uso de *choke point*. Un *choke point* es un aspecto de la ejecución u optimización de consultas que se sabe que es problemático para la generación actual de varios SGBD (relacionales, grafos y RDF). El documento de especificaciones

clasifica los *choke points* en 7 categorías y a continuación procederemos a definir cada una de ellas [2].

1. ***Rendimiento de agregación***

   1.1. Órdenes interesantes: Se valida la facultad del motor de ejecución de mantener el orden de los datos al momento de hacer uso de índices. Este *choke point* determina si hace uso de este ordenamiento.

   1.2. Desempeño *group-by* con alta cardinalidad: Se comprueba la competencia del motor de ejecución para paralelizar *group by* por un gran número de grupos. Algunas consultas requieren realizar grandes agrupaciones. En tal caso, si una agregación produce un número significativo de grupos, se puede aprovechar la paralelización, ya que cada hilo puede hacer su propia agregación parcial. Luego, para producir el resultado, estos tienen que ser re-agregados. Este *choke point* verifica si el motor de ejecución es capaz de hacer uso de la paralelización para aumentar la velocidad de obtención de resultados.

   1.3. Obtención de los primeros k elementos: Se Verifica la idoneidad del optimizador para realizar obtener los primeros k resultados.

   1.4. *Group-by* dependiente por claves: Se evalúa la capacidad del motor de ejecución de usar efectivamente la función *group-by*, cuando se encuentra con pocos grupos producto del resultado de la consulta.

2. ***Rendimiento de Joins***

   2.1. Optimización de las opciones del *join*: Evalúa la competencia de selección del plan físico del optimizador de consultas, para encontrar órdenes óptimas del *join*, debido a que un grafo se puede recorrer de diferentes maneras. El tiempo de ejecución de estas puede diferir por órdenes de magnitud. Por lo tanto, encontrar un distribución eficiente de *join* (recorrido) es importante, lo que en general requiere la enumeración de todas las posibilidades.

   2.2. Proyección tardía: Comprueba la capacidad del optimizador de consultas para retrasar la proyección de atributos no ingresados hasta el final de la ejecución. Existen consultas en ciertas columnas donde es posible retrasar la proyección de atributos sólo hasta que sea realmente necesario.

   2.3. Selección del tipo de *Join*: Se evalúa la competencia del optimizador de consultas para seleccionar el tipo de operador de combinación adecuado, lo que implica estimaciones precisas de cardinalidades. El optimizador debe ser capaz de determinar el tipo de *join* a utilizar para cada situación.

   2.4. *Joins* por claves foráneas: Mide el rendimiento de los operadores de *join* cuando son escasos. A veces, las combinaciones implican relaciones en las que sólo se

requiere un pequeño conjunto de datos en una de las tablas para satisfacer una combinación. Cuando las tablas son más grandes, los métodos de *join* típicos pueden llegar a no ser tan óptimos. Este *choke point* mide el desempeño de la implementación de los operadores físicos del *join* en esta situación.

3. ***Accesibilidad a los datos***

   3.1. Detección de correlaciones: Pone a prueba la cualidad del optimizador de consultas para detectar correlaciones de datos y explotarlas. Si un esquema recompensa la creación de índices agrupados, la pregunta es ¿cuál de las columnas de fecha o datos se usará como clave? Una forma es a través de la creación de histogramas de atributos múltiples, después de la detección de correlación de atributos. Con los índices de MinMax, los predicados de rango en cualquier columna se pueden traducir en rangos de posición de tupla que califican. Si un valor de atributo se correlaciona con la posición de la tupla, esto evita que se realice una búsqueda de todos los datos asociados a la entidad provocando una reducción en los tiempos de ejecución. El objetivo de este choke point es evaluar la competencia del optimizador de identificar y hacer uso de estas correlaciones.

   3.2. Agrupamiento dimensional: Comprueba la idoneidad de los identificadores asignados a las entidades por el sistema de almacenamiento, para explotar mejor la localidad de datos. Dicha elección de identificador puede crear una localidad que a su vez mejora la eficiencia de la compresión o el acceso al índice.

   3.3. Patrón de acceso a índices dispersos: Confirma el rendimiento de los índices cuando se realizan accesos dispersos. La eficiencia de la búsqueda de índices es muy diferente según la localidad de las claves que llegan al acceso indexado. Técnicas como la vectorización de accesos a índices no locales pueden tener un alto impacto, perdiendo el caché en paralelo en múltiples búsquedas vectorizadas en el mismo hilo.

4. ***Cálculo de expresiones***

   4.1. Eliminación de subexpresiones comunes: Verifica la habilidad del optimizador de consultas para detectar subexpresiones comunes y reutilizar sus resultados. El objetivo es que el optimizador debe ser capaz de identificar la existencia de operaciones repetidas, para que no vuelva a calcular nuevamente el resultado que ya ha sido realizado.

   4.2. Selecciones y *joins* con expresiones lógicas complejas: Confirma la capacidad del optimizador de consultas para reordenar la ejecución de expresiones buscando mejor el rendimiento. Por ejemplo, el optimizador puede reordenar las conjunciones, para probar primero esas condiciones con mayor selectividad,

con la finalidad de que el proceso de búsqueda más complejo se resuelva en primera instancia. Este *choke point* está diseñado para probar la habilidad del optimizador de determinar el orden más eficiente para evaluar las expresiones lógicas.

4.3. Interpretación de expresiones matemáticas: Demuestra la facultad de evaluar eficientemente expresiones simples en un gran número de valores. Un ejemplo típico podría ser expresiones aritméticas simples, funciones matemáticas como funciones absolutas o fechas como la extracción de un año.

5. ***Subconsultas correlacionadas***

5.1. Aplanamiento de subconsultas: Evidenciar la competencia del optimizador de consultas para aplanar los planes de ejecución cuando hay subconsultas correlacionadas. Muchas consultas tienen subconsultas correlacionadas y sus planes de consulta se pueden aplanar, donde el optimizador debe ser capaz de identificar estas situaciones y aplicar el aplanamiento de ser necesario. Este *choke point* evalúa la capacidad del optimizador de aplanamiento del optimizador.

5.2. Superposición entre consultas internas y externas: Corrobora la cualidad del motor de ejecución para reutilizar los resultados de una consulta interna en una consulta externa, de esta manera no se debe volver a recorrer el grafo en la búsqueda de datos siempre y cuando ambos resultados sean equivalentes.

5.3. Reutilización de resultados dentro de una consulta: Pone a prueba la habilidad del motor de ejecución para reutilizar los resultados, pero esta vez entre consultas internas, a diferencia que el *choke point* anterior que era entre una consulta interna y una consulta externa.

6. ***Paralelismo y concurrencia***

6.1. Reutilización de resultados entre consultas: Comprueba la facultad del motor de ejecución para reutilizar los resultados de diferentes consultas y detectar cuando un par de sub-consultas, de consultas distintas, son similares. Se espera que el motor de ejecución reutilice los resultados, para de esta forma reducir los costo de efectuar otra búsqueda de datos que es innecesaria, reduciendo los tiempos de ejecución.

7. ***RDF o Grafos***

7.1. Reutilización de caminos: Pone a prueba la competencia del motor de ejecución para reutilizar los caminos que se recorren a través de las relaciones entre entidades. Es bien sabido que la gran cualidad de los RDF o grafos son sus datos interrelacionados y estos son utilizados para recorrer los esquemas, desde allí

nace la necesidad de reducir los tiempos de búsqueda que demora el sistema gestor en recorrer los grafos. El sistema gestor debe ser capaz de guardar en memoria los caminos más recorridos con la finalidad de reutilizarlos y no tener que recorrerlos nuevamente.

7.2. Estimación de cardinalidad de trayectorias transitivas: Verifica la capacidad del optimizador de consultas para estimar correctamente la cardinalidad de los resultados intermedios al ejecutar rutas transitivas. Esto permite al motor de ejecución definir la mejor opción desde donde iniciar su búsqueda, con esto nos referimos a la selección del nodo de inicio.

7.3. Ejecución de pasos transitivos: Comprobación de la idoneidad del motor de ejecución para expandir las relaciones que componen el camino. Las cargas de trabajo de grafos pueden tener operaciones transitivas, como por ejemplo, encontrar una ruta más corta entre vértices.

Finalmente, se debe probar las guías de diseño físico sobre las 25 consultas utilizando los *choke points* especificados en la documentación de LDBC SNB [2]. La Tabla 2.1 se compone en su eje abscisas por el acrónimo *Business Intelligence* (BI) y en el eje de ordenadas por los diferentes *choke points*. Esta figura servirá de referencia al momento de efectuar el estudio experimental, buscando validar y cumplir con el correcto funcionamiento, para de esta forma realizar los experimentos que prueban el comportamiento sobre diferentes cargas de trabajo.

	1.1	1.2	1.3	1.4	2.1	2.2	2.3	2.4	3.1	3.2	3.3	4.1	4.2	4.3	5.1	5.2	5.3	6.1	7.1	7.2	7.3	7.4
BI 1		●								●		●										
BI 2	●	●	●		●		●		●	●												
BI 3								●	●	●		●		●			●	●				
BI 4	●	●	●		●	●		●			●											
BI 5		●	●		●	●	●	●			●						●	●				
BI 6		●					●															
BI 7		●					●			●	●							●				
BI 8				●							●					●						
BI 9		●	●		●		●	●														
BI 10		●			●		●			●												
BI 11	●				●	●	●		●	●								●				
BI 12		●				●			●									●				
BI 13		●				●	●			●								●				
BI 14		●				●	●			●										●	●	●
BI 15		●					●			●	●						●	●				
BI 16		●	●				●	●			●								●	●	●	
BI 17	●						●															
BI 18	●	●		●						●			●	●								
BI 19	●		●		●		●	●			●				●						●	●
BI 20				●	●													●				
BI 21		●			●		●	●		●	●				●		●					
BI 22			●	●	●				●		●				●	●	●					
BI 23				●			●	●			●			●								
BI 24				●	●		●	●		●				●								
BI 25		●			●	●		●			●				●		●			●	●	

Tabla 2.1: Cobertura de *choke points* por consultas.

2.1.4. Datagen

LDBC [22] define un Datagen como un generador de datos que estructura una red social artificial basada en la ley potencial (power-law). Además, se debe mencionar que los datos son extraídos desde las conversaciones de los foros en DBpedia. El Datagen puede generar datos a escalas desde los 100 MB hasta los 30 TB. Estos son los factores de escala oficiales que presenta LDBC SNB, donde el volumen de datos se mide como el tamaño total sin comprimir de los datos textuales CSV.

2.1.5. Neo4j

Neo4j es un SGBD con un tipo de software *open-source* escrito en los lenguajes de programación de Java y Scala [23]. Neo4j se conoce como una base de datos de grafos nativos en virtud de que implementa de manera eficiente el modelo de grafos de propiedades, hasta el nivel de almacenamiento. Esto significa que los datos se almacenan exactamente a medida que los sitúa en la base de datos usando punteros para navegar y recorrer el grafo. En contraste con el procesamiento de grafos o las bibliotecas en memoria, también proporciona características completas de la base de datos relacionales, incluido el cumplimiento de transacciones *Atomicity, Consistency, Isolation, Durability* (ACID), el soporte de clústeres y la conmutación por error en tiempo de ejecución. Así todo lo anterior es mencionado por los autores I. Robinson, J. Webber y E. Eifrem [24].

Neo4j [25] presenta en su página el modelo de grafo de propiedad, este es uno de los pocos enfoques diferentes de los componentes clave de una base de datos grafos, donde los datos se clasifican como ***nodos***, ***relaciones*** y ***propiedades*** (características pertenecientes a los nodos o relaciones). Los nodos son usados para representar entidades, pero también pueden representar otros componentes de dominio, dependiendo del caso de uso. Los nodos pueden contener propiedades que abarcan pares de datos nombre-valor. Así mismo a los nodos se les pueden clasificar según ***tipos*** o ***roles*** usando una o más etiquetas que nos permite identificar los nodos como entidades con una identidad conceptual única. Una ***etiqueta*** es una construcción de grafos para agrupar o clasificar nodos en conjuntos, también un nodo puede no disponer de una etiqueta como también poseer un número ilimitado de ellas según disponga el usuario. Por otro lado, los nodos se conectan a través de ***relaciones***, esto nos permite identificar cómo interactúan permitiendo reconocer el nodo de origen y el nodo de destino mediante la dirección del vínculo que podemos identificar con la orientación de la flecha de la relación. Finalmente, las ***propiedades*** pueden ser asignadas a los nodos o las relaciones con el objetivo de brindar mayor información a estos elementos.

En la Figura 2.1 tenemos un ejemplo que entrega Neo4j [25] que nos permite observar los conceptos esenciales antes descritos. En la imagen podemos apreciar a Ann y

Dam, ambos en nuestra base de datos se clasifican como ***nodos*** y además se encuentran ***etiquetados*** como personas. Igualmente, sabemos que la ***relación*** que los une corresponde a que Ann está casada con Dan y a la vez vive con él como lo vemos en las relaciones *Is_Married_To* y *Lives_With*. Por último, conocemos algunas ***propiedades*** de Dan como lo son su nombre, fecha de nacimiento y su cuenta de Twitter.

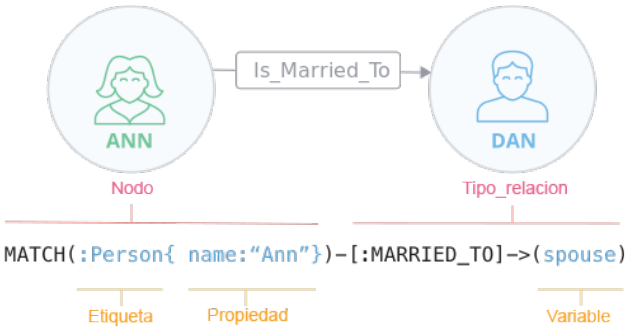


Figura 2.1: Grafo propiedad [1].

Neo4j [26] en su documentación precisa que disponen de diferentes operadores al momento de efectuar una consulta. El resultado de la ejecución de una consulta no es tan sólo la información requerida, sino que también a través de una sentencia podremos ver el plan de ejecución. El ***plan de ejecución*** posee una estructura en forma de árbol donde cada operador se representa como un nodo en el árbol y cada nodo nos proporciona noción de cuál fue el costo de ejecutar esa acción. El plan de ejecución es un componente vital que nos posibilita analizar en detalle los costos asociados a cada acción que efectúa el motor de ejecución para encontrar los registros solicitados en la consulta. La unidad de medida que define Neo4j para cuantificar las acciones ejecutadas por el motor de ejecución es el *Database Hits* (DB Hits). DB Hits es una unidad de medida abstracta exclusiva de Neo4j. Esta unidad enumera cada fila que ingresa por cada operador y cuánto necesita cada operador para interactuar con la capa de almacenamiento en la búsqueda de los datos desde el nodo inicial hasta el nodo final en el plan de ejecución. Existen dos sentencias que nos generan el plan de ejecución con el fin de analizar en detalle nuestra consulta, sin embargo, sólo usaremos el siguiente [26]:

1. ***Explain***: Si desea ver el plan de ejecución, pero no ejecutar la instrucción, anteponga su declaración Cypher con ***EXPLAIN***. La declaración siempre devolverá un resultado vacío y no realizará cambios en la base de datos.

2. ***Profile***: Si desea ejecutar la declaración y ver qué operadores están haciendo la mayor parte del trabajo, utilice ***PROFILE***. Esto ejecutará su declaración y hará un seguimiento de cuántas filas pasan a través de cada operador, y cuánto necesita cada

operador para interactuar con la capa de almacenamiento para recuperar los datos necesarios.

Neo4j especifica la información que nos entrega el plan de ejecución en el manual de Cypher [27] que pasaremos a ver a continuación.

1. ***Rows***: El número de filas que produjo el operador. Esto sólo está disponible si la consulta incluye una de las dos sentencias ya mencionadas. Debemos tener en cuenta que perfilar una consulta utiliza una mayor cantidad recursos, por lo que a veces es recomendable verificar si es necesario su uso.

2. ***EstimatedRows***: Es el número estimado de filas que se espera que produzca el operador. La estimación es un número aproximado basado en la información estadística disponible. El compilador utiliza esta estimación para elegir un plan de ejecución adecuado.

3. ***DB Hits***: Cada operador le pedirá al motor de almacenamiento Neo4j que realice trabajos como recuperar o actualizar datos. Un ***hit*** de base de datos es una unidad abstracta de este trabajo del motor de almacenamiento.

Para producir un esquema eficiente Neo4j [27] precisa que cada consulta se convierte en un plan de ejecución por el planificador de consultas Cypher. El plan de ejecución indica a Neo4j qué operaciones se deben llevar a cabo cuando se ejecuten la consulta, así como también cuándo el planificador de consultas requiere información sobre la base de datos Neo4j. El planificador de consultas utiliza esta información para determinar qué patrones de acceso producirán el mejor plan de ejecución, donde Neo4j está programado para utilizar una estrategia basada en el costo conocido como planificador de costo. El planificador de costo es el servicio de estadísticas que asigna un costo a los planes alternativos para finalmente escoger el plan más eficiente. Adicionalmente, en el plan de ejecución cada nodo del árbol es un operador físico y su posición determina el orden en el que se debe ejecutar para conseguir el resultado deseado.

Neo4j mantiene las estadísticas actualizadas de dos maneras diferentes. La primera es el recuento de etiquetas y relaciones, el número se actualiza cada vez que configura o elimina una etiqueta de un nodo. Sin embargo, para los índices se necesita escanear el índice completo para producir el número de selectividad ya que es potencialmente una operación que consume mucho tiempo.

En la documentación de Neo4j se encuentra un total de 82 operadores de los cuales sólo se pueden utilizar 62 durante la ejecución de una consulta. Igualmente, Neo4j clasifica a la totalidad de operadores en el manual de Cypher [28] de los cuales a continuación se especifican los más relevantes.

- ***Operadores de agregación***: La función de los operadores de agregación es eliminar los valores duplicados con la sentencia ***DISTINCT***.
- ***Operadores de comparación***: Los operadores de comparación su función es como su nombre lo indica comparar datos, verificando si existen igualdades, desigualdades, menores que, mayores que, entre otros. Además existen los operadores de comparación específicos de cadena, desde aquí se desprende lo siguiente:
  - ***STARTS WITH***: Efectúa una búsqueda de prefijos que distingue entre mayúsculas y minúsculas en cadenas.
  - ***ENDS WITH***: Ejecuta la búsqueda de sufijos sensibles a las mayúsculas en las cadenas.
  - ***CONTAINS***: Realiza búsquedas de inclusión de mayúsculas y minúsculas en cadena.
- ***Operadores de propiedad***: Los operadores de propiedad pertenecen a un nodo o una relación. Accede estáticamente a la propiedad de un nodo o relación utilizando un operador de punto y dinámicamente mediante un operador de subíndice. Reemplazo de propiedad para reemplazar todas las propiedades de un nodo o relación.
- ***Operadores de mapas***: Accede de forma estática al valor de un mapa por clave utilizando el operador de punto y dinámicamente usando un operador de subíndice.
- ***Operadores aritméticos***: Los operadores aritméticos permiten realizar operaciones básicas como la adición, sustracción, multiplicación, división, entre otros.

2.1.6. Versionamiento Neo4j Browser

El equipo de Neo4j busca perfeccionar presentando nuevas técnicas y herramientas para obtener el mayor rendimiento posible. En esta investigación se trabajará con la versión 3.5.18 de Neo4j Browser, Desde ahí nace la necesidad de verificar las modificaciones y actualizaciones que se han presentado en las diferentes versiones. Se especificarán las versiones bases 3.4 [29] y 3.5 [30] de Neo4j Browser, así mismo se mencionan las mejoras y mantenimientos más importantes para nuestro trabajo.

Versión 3.4

Multi-Clustering

La plataforma de grafos Multi-Clustering avanza en escala, extensión y desempeño [31]. Con las actualizaciones se puede crear y administrar múltiples agrupaciones de

bases de datos con nombre, dividiendo efectivamente el grafo en piezas independientes. El Multi-Clustering se puede usar para particionar lógicamente los grafos; crear sistemas *Software as a Service* (SaaS) multiusuario de alta y gran disponibilidad; cumplir con los requisitos de partición geográfica; o supervise implementaciones de grafos múltiples en toda la empresa.

Nuevos tipos de datos

Los nuevos tipos de datos de fecha/hora incluyen una variedad de formatos y se ajustan a un modelo similar a SQL que incluye fechas y horas locales, de tal modo que se definen con y sin zonas horarias. Además de la latitud y longitud tradicionales de los datos geoespaciales, en esta oportunidad se incluyen coordenadas cartesianas (x, y, z), distancias radiales, altitud, profundidad y pendiente, permitiendo indexar estos campos que en cuyo caso utilizarán automáticamente el nuevo índice nativo de Neo4j, que es 5 veces más rápido para las escrituras que los índices pertenecientes a Lucene.

Mejoras de rendimiento

Neo4j 3.4 es más rápido en términos de lecturas y escrituras, y estas mejoras de rendimiento general se reflejan proporcionalmente en ambas ediciones. Las escrituras ahora son hasta 5 veces más rápidas para los nodos con propiedades de cadena indexadas, gracias a los índices nativos que ahora pueden trabajar con tipos de datos.

Es relevante tener conocimiento de las versiones que aplican una mejora o modificación que tenga relación con este trabajo, por lo cual indicaremos cuáles son esas versiones.

a) Versión 3.4.7: Se corrigió un error que causaba valores de propiedad de gran volumen que impedía el óptimo desplazamiento por la base de datos. Esto provoca que los lectores fallen al leer del índice provocando el siguiente error:

   - *org.neo4j.io.pagecache.CursorException: Read unreliable key*

   Se reparó un problema de índice nativo donde la inserción de grandes valores, aunque dentro del límite de tamaño, que podría resultar en un intento de escribir fuera de límites de la página.

b) Versión 3.4.8: Soluciona un problema donde se perdía un elemento cuando se invocaba la función *IndexHits #getSingle()* después de una llamada previa a la función *IndexHits #hasNext()*.

c) Versión 3.4.9: Se incorpora la posibilidad de obtener la causa de la falla del procedimiento db.indexes. También se corrige la etiqueta que se quedaría fuera al imprimir

la descripción del índice en los registros.

d) Versión 3.4.10: Agrega la funcionalidad *IndexReader #distinctValues*.

Versión 3.5

Esta versión sigue la línea de trabajo de la versión 3.4 con las correcciones de los errores y la inserción de la nuevas funciones. Destaca la incorporación de la indexación nativa, los índices complejos, búsqueda de texto completo y la clasificación rápida con índice de respaldo *ORDER BY*. Por otro lado, se dieron de baja algunas funciones ya sea por desuso o también porque existen otras alternativas más eficientes. Los índices que fueron eliminados en esta versión son:

- lucene-1.0
- lucene+native-1.0
- lucene+native-2.0

A continuación, se especifican las actualizaciones y correcciones más significativas de la versión 3.5 de Neo4j Browser.

a) Versión 3.5.2: Se agrega un nuevo procedimiento *db.index.fulltext.awaitIndex*, que puede esperar índices especificados por nombre, en lugar del patrón de etiqueta/propiedad que ha sido soportado hasta ahora por el procedimiento *db.awaitIndex*. Lo anterior significa que ahora se puede esperar a que se ubiquen los índices de texto completo con nombre específico. Adicionalmente, se soluciona el problema en el que si un índice de esquema de texto completo y un esquema normal tienen el mismo esquema (por ejemplo, la misma etiqueta y combinación de propiedades). Cabe mencionar que no se podía descartar el índice de esquema de texto completo, pero ahora es posible incluso cuando el esquema del índice de texto completo coincide con el de un índice de esquema normal. Finalmente, el resultado del procedimiento es *db.index.fulltext.listAvailableAnalyzers*, donde se debe indicar que ahora se incluye una descripción de cada analizador. También se han añadido los analizadores *simple*, *classicy* y *stop* ha la suite de analizadores de Lucene estándar.

b) Versión 3.5.3: Esta versión incorpora la función *IndexReader #distinctValues()*, cuya cualidad es acceder a todos los valores distintos en un índice con el número de entradas indexadas para cada valor. Los valores se materializan si el usuario lo solicita y el índice admite la materialización de los valores reales.

c) Versión 3.5.4: Se efectúan tres mejoras en torno a la creación de restricciones de unicidad:

- Se pueden eliminar los índices de tipo **único** sin una restricción con la sentencia: *DROP INDEX ON: Label (key)*.

- La creación de restricciones de exclusividad no estará sujeta al tiempo de espera de *dbms.transaction.timeout*.
- La población de índice respaldada por "bloque basado en el índice de datos" podría informar falsos negativos para valores de propiedad duplicados. esto sucedió si, mientras se escanea el almacén para proporcionar actualizaciones para el índice de población, los valores de propiedad cambiaron de tal manera que el análisis vería el mismo valor dos veces aunque no existía duplicación en ningún momento. Esto se mitiga comprobando los duplicados sospechosos una vez finalizado el proceso

d) Versión 3.5.5: El rendimiento de realizar cambios en los nodos y las relaciones ya no se ve afectado por la presencia de índices y/o restricciones que no están relacionados con los cambios reales. Antes de estos cambios, un gran número de índices y/o restricciones podrían afectar negativamente el rendimiento de las transacciones de escritura, incluso si los cambios reales no estuvieran relacionados con los cambios realizados. Anteriormente, si el muestreo fallaba provocaba también la caída del índice, no obstante, con esta versión esto se podría evitar puesto que la muestra no se había limpiado después de sí misma.

e) Versión 3.5.7: Se corrigió el error en el que un índice de texto completo eventualmente consistente después de reiniciar la base de datos ya no sería consistente. También se trabajó en las consultas de distancia espacial que ahora serán resueltas por un *NodeUniqueIndexSeekByRange* en lugar de un *NodeByLabelScan* cuando exista una restricción de unicidad.

e) Versión 3.5.13: Se corrigió un error que a veces provocaba una excepción *NullPointerException* al generar comandos de actualización de índice para cambios de transacción que coincidían parcialmente con un índice de texto completo. Adicionalmente, el verificador de consistencia ahora ya no informa *NullPointerException* cuando un nodo o relación no tiene todas las propiedades declaradas en un índice de texto completo. Por último, los scripts de la utilidad Neo4j respetan *Heap_Size* sobre la configuración de neo4j.conf.

2.1.7. Cypher

El artículo Cypher: An Evolving Query Language for Property Graphs [32] estudia profundamente este lenguaje destacando la explicación del funcionamiento y sintaxis del lenguaje, sin embargo, mencionan constantemente que dicho lenguaje esta en una etapa de perfeccionamiento con la finalidad de lograr una estructura general como lo es el lenguaje de SQL. Neo4j [33] define a Cypher como un lenguaje de consultas que permite a los usuarios almacenar y recuperar datos de la BDOG. La sintaxis de Cypher proporciona una

forma visual y lógica de unir patrones de nodos, y relaciones en el grafo. Es un lenguaje declarativo inspirado en SQL para describir patrones visuales usando la sintaxis ASCII-Art haciendo que las consultas sean fáciles de leer y reconocer como un fragmento de sus datos. Adicionalmente, nos permite establecer lo que queremos seleccionar, insertar, actualizar o eliminar de nuestros datos gráficos. Neo4j también indica que Cypher es un lenguaje de consulta de código abierto promovido por el proyecto ***openCypher*** que busca establecer a Cypher como un lenguaje de consulta estándar para BDOG como lo hicieron IBM y Oracle con SQL hace unos años atrás. Al hablar del proyecto ***openCypher*** debemos citar el trabajo de J. Marton, G. Szárnyas y D. Varr [34], ya que ellos logran hacer un mapeo desde la base de ***openCypher*** al álgebra de grafos relacional. Con esto consiguen resultados experimentales en grandes grafos reales y sintéticos que demuestran que las optimizaciones específicas de grafos superan a una implementación basada en SQL por orden de magnitud. Finalmente, Cypher desempeña un papel fundamental en Neo4j ya que permite usar un lenguaje de consulta simple, pero poderoso para integrar su trabajo con todas las diversas herramientas y fases de la plataforma de grafos.

2.1.8. Índices

Neo4j [35] define que un índice de base de datos es una copia redundante de algunos de los datos con el propósito de hacer que las búsquedas sean más eficientes. Esto conlleva un costo de almacenamiento adicional y escrituras más lentas, por lo que decidir qué es lo que se debe indexar es una labor de gran importancia. Cypher permite la creación de índices en una o más propiedades para todos los nodos que tienen una etiqueta determinada. Existen dos tipos de índice, el primero es un ***índice simple*** que se crea en una sola propiedad y el segundo es un ***índice compuesto*** es el que se crea en más de una propiedad, ambos para cualquier etiqueta. Las diferencias entre ambos se orientan a las limitaciones que poseen, dado que a diferencia de índices simples, los índices compuestos sólo soportan comprobación de igualdad y verificación de pertenencia de la lista. Además, no son compatibles las consultas que contienen los tipos de predicados en las propiedades, con esto nos referimos a lo siguiente:

- ***exists(n.prop)***
- ***n.prop*** $>$ ***value***
- ***STARTS WITH***
- ***ENDS WITH***
- ***CONTAINS***

Neo4j emplea para su funcionalidad de indexación una combinación de índices nativos y Apache Lucene, donde el índice nativo es una implementación del clásico árbol B+. Neo4j en su versión 3.5 expresa que la indexación nativa ampliada acelera la inserción de datos hasta 5 veces para todos los tipos de datos. Por otro lado, los índices de esquema de texto completo están alimentados por la biblioteca de búsqueda e indexación de Apache

Lucene, igualmente se pueden usar para indexar nodos y relaciones por propiedades de cadena. Un índice de esquema de texto completo le permite escribir consultas que coincidan con el contenido de las propiedades de cadena indexada.

2.1.9. Métricas de evaluación

Las métricas de evaluación se establecen pensando en analizar cada posible estrategia a implementar desde diferentes perspectivas, procurando tomar la mejor decisión al momento de definir las estrategias para cada consulta. Para las métrica de tiempo de ejecución y DB Hits se crearán tablas de comparación por carga de trabajo y un consolidado para analizar su comportamiento al escalar los volúmenes de datos.

- Tiempo de ejecución: Esta métrica es la más importante, debido a que nuestro objetivo principal es disminuir los tiempos de ejecución por consulta manteniendo la equivalencia. El tiempo de ejecución se encuentran en milisegundos.
- DB Hits: Se usa el DB Hits como una métrica de evaluación puesto que permite evaluar el número de acciones que debe realizar el motor de ejecución para cumplir el requerimiento de la consulta. Esto producto de la comparación de los valores iniciales y finales interpretando de que a un menor número de acciones menor es el tiempo de ejecución para conseguir el resultado. Se debe indicar que esta unidad de medida fue definida en la Sección 2.1.5.
- Costo de Almacenamiento: El costo de almacenamiento es una métrica de evaluación importante al momento de llevar a cabo una de las posibles soluciones porque algunas de ellas tienen un costo de almacenamiento que se deben analizar para tomar la decisión de cuál de éstas se va a implementar.

2.2. Estado del Arte

Los diferentes trabajos relacionados directamente con el propósito de este trabajo serán ordenados por año de publicación de manera descendente, debido a que se infiere que los nuevos trabajos se basan en los estudios e investigaciones previas, en caso de que exista alguna coincidencia en el año de publicación se posicionará en primer lugar la obra que posea una mayor contribución. La Tabla 2.2 se compone de 6 columnas que pasaremos a definir a continuación.

- Autor: Presenta las personas que llevaron a cabo el trabajo.
- Título: Expone el nombre del trabajo desarrollado por los autores.

- Objetivo general: Exhibe los diferentes objetivos que se mencionan al inicio de cada trabajo con respecto a la optimización de consultas desde el diseño físico.
- Tipo de obra: Hace referencia al tipo de documento de cada trabajo.
- Resultados: Revela los resultados de los experimentos después de haber establecido sus conocimientos y principios.
- Año: Evidencia el año de publicación de la obra.

Autor - Tipo de Obra - Año	**Título**	**Objetivo General**	**Resultados**
C. Lei, R. Alotaibi, A. Quamar, V. Efthymiou y F. Özcan [36] - Artículo de Investigación - 2020	Property Graph Schema Optimization for Domain-Specific Knowledge Graphs	Demostrar que el diseño del esquema gráfico tiene un impacto significativo en el rendimiento de la consulta. Se presentan dos métodos para abordar esta problemática que se evidencian a base de diferentes experimentos.	Se establecen normas de relación desde la ontología, donde se destaca el minucioso estudio que se hace de los vértices y aristas en los esquemas de grafos. Los resultados muestran que la latencia total de la consulta en ambos esquemas optimizados es aproximadamente 2 órdenes de magnitud más rápida que la asignación directa.
M. Jota y R. Parra [37] - Trabajo de Título - 2019	Diseño Físico para la base de datos SNB sobre un gestor orientado a grafos	Exponer soluciones de diseño físico en 13 consultas establecidas por SNB, buscando así validar bajo experimentos las técnicas propuestas.	Los resultados demuestran que se logra reducir los tiempos de ejecución sobre los valores sin diseño físico, evidenciándose más aún en dos de las tres técnicas presentadas, demostrando el beneficio a medida que la carga de trabajo aumenta. Este trabajo fue citado en el libro "Data Science: Theory, Analysis and Applications" editado por Q. A. Memon y S. A. Khoja.
K. Seo, J. Ahn y D. Im [38] - Artículo de Investigación - 2019	Optimization of Shortest-Path Search on RDBMS-Based Graphs	Plantear un método de búsqueda bidireccional paralela usando multihilo. Lo anterior fundamentado en lo expuesto en las bases de datos relacionales. Así mismo, se propone otro método de optimización que utiliza indexación B-tree en lugar de funciones de agregación.	Mediante la comparación de los resultados de los diferentes métodos se consigue validar la disminución del tiempo de ejecución con respecto a los registros sin los algoritmos implementados mediante la búsqueda de la ruta más corta entre nodos. Adicionalmente, logran que el método formulado en este trabajo sea más eficiente que el algoritmo Frontier-Expand-Merge (FEM) básico y sus extensiones pertenecientes a diferentes bases de datos de grafos.
F. Rusu y Z. Huang [39] - Artículo de Investigación - 2019	In-Depth Benchmarking of Graph Database Systems with the Linked Data Benchmark Council (LDBC) Social Network Benchmark (SNB)	Comparar los tiempos de carga de las 46 consultas propuestas en el proyecto *LDBC SNB*. Las pruebas se desarrollan con diferentes cargas de trabajo en dos sistemas de bases de datos de grafos nativos: Neo4j y TigerGraph.	A través de la medición de cuatro factores de escala y tres arquitecturas informática para 46 consultas se obtienen los resultados que exponen que TigerGraph posee una mayor efectividad en la resolución de las consultas a diferencia de Neo4j, aunque no se pudieron realizar pruebas en cargas de trabajo de mayor volumen, dado que TigerGraph tiene una capacidad procesamiento límite de 1000 GB, no así Neo4j que se destaca por su escalabilidad.

Autor - Tipo de Obra - Año	**Título**	**Objetivo General**	**Resultados**
A. Ben Ammar [40] - Artículo de Investigación - 2016	Query optimization techniques in graph databases	Optimizar las consultas sobre las BDOG con estrategias dirigidas hacia el diseño físico.	Presenta métodos de optimización muy similares al documento "*Diseño Físico para la base de datos SNB sobre un gestor orientado a grafos*" de M. Jota y R. Parra, pero no desarrolla experimentos para verificar su efectividad. Se desprende de este trabajo una vez más la conclusión que hace referencia a que no existe un lenguaje estándar de consultas, en consecuencia las técnicas operan de forma distinta dependiente del lenguaje en que se trabaje entregando resultados variados.
J. Maginecz G. Szárnyas y D. Varró [41] - Artículo de Investigación - 2016	Evaluation of optimization strategies for incremental graph queries	Evaluar el desempeño de diferentes tácticas de optimización de consultas de bases de datos relacionales sobre bases de datos orientadas a grafos.	Se plantean técnicas de optimización desde las bases de datos relacionales observados de diferentes puntos de vista, nos referimos al flujo de trabajo y optimización basada en costos de función, las cuales están relacionadas con nuestro trabajo. Los resultados sugieren que, de manera similar a los motores de optimización de consultas relacionales, los motores de optimización de consultas de grafos debieran centrarse en evitar los productos cartesianos en el plan de consultas.
A. Gubichev [42] - Trabajo de Título - 2015	Query processing and optimization in graph databases	Efectuar diferentes experimentos de optimización de consultas para dos lenguajes de consultas orientadas a grafos, como lo son SPARQL y Cypher. Además de definir una guía para la selección de parámetros en las consultas de grafos sobre los datos entregados por SNB.	Exponen los resultados de los experimentos buscando a través de la optimización de consultas reducir los tiempos de respuesta. Lo anterior entre dos versiones de Neo4j. Se logra demostrar la significativa diferencia a favor de la nueva versión en cuanto a tiempos de compilación y ejecución.
J. Cheng, Y. Ke y W. Ng [43] - Artículo de Investigación - 2009	Efficient Query Processing on Graph Databases	Proponer un índice eficiente para resolver el problema de procesamiento de consultas de subgrafos. Utilizando el índice de características para reducir el costo de sondeo y el índice de preguntas frecuentes, que se construye dinámicamente a partir del conjunto de consultas no FG de preguntas frecuentes.	Si bien el objetivo no se logra completar del todo, dado que los resultados no logran dar garantía para utilizar está metodología, los autores demuestran que se obtiene una mejora en los tiempos de ejecución, es por este motivo que se toma la investigación para ser analizada y tomar posibles técnicas que puedan llegar a ser implementadas.
H. He y A. K. Singh [44] - Artículo de Investigación - 2008	Graphs-at-a-time: Query Language and Access Methods for Graph Databases	Presenta un álgebra de grafos extendida desde el álgebra relacional en la que el operador de selección se generaliza a la coincidencia de patrones de grafos. e introduce un operador de composición para reescribir los grafos de coincidencias. Se busca llegar a los resultados esperados mediante una combinación de técnicas: uso de subgrafos y perfiles de vecindario, reducción conjunta del espacio de búsqueda y optimización del orden de búsqueda.	Se expone la optimización del lenguaje de consulta, como también los métodos de accesos a las bases de datos relacionales y las BDOG. Estas últimas se distinguen por su alta escalabilidad y principalmente sus planes de accesos para recorrer la base de datos. El artículo menciona que si bien se pueden efectuar pequeñas mejoras con ajustes cuidadosos, los resultados demuestran que el procesamiento de consultas en BDOG poseen claras ventajas.

Tabla 2.2: Estado del arte.

2.2.1. Comparación entre trabajos relacionados

La comparación entre los trabajos e investigaciones presentadas en la sección anterior tiene por objetivo definir las fortalezas y debilidades enfocado en los aportes y contribuciones de cada obra hacia la elaboración de la solución de nuestro diseño físico.

Autor	Título	Fortalezas	Debilidades
C. Lei, R. Alotaibi, A. Quamar, V. Efthymiou y F. Özcan [36]	Property Graph Schema Optimization for Domain-Specific Knowledge Graphs	Destaca la importancia y establece buenas prácticas en el diseño del modelo de grafos permitiendo trabajar con un esquema optimizado.	No se aborda la etapa del diseño lógico de una BDOG, por lo que no nos provee un fundamento esencial para establecer nuestras estrategias de diseño físico.
M. Jota y R. Parra [37]	Diseño Físico para la base de datos SNB sobre un gestor orientado a grafos	Nos entrega un diseño físico el cual considera un avance significativo cada vez que se aumentan las cargas de trabajo. Los porcentajes de reducción de los tiempos de respuestas destacan considerablemente en algunas de las consultas.	El trabajo está desarrollado sobre 13 de las 25 consultas presentadas en SNB pertenecientes a BIW, por lo que no posibilita comprobar la aplicación transversal de sus estrategias en consultas con diferentes estructuras, pero si con algunas de ellas.
K. Seo, J. Ahn y D. Im [38]	Optimization of Shortest-Path Search on RDBMS-Based Graphs	Las estrategias se consideran óptimas para su uso en Neo4j previa adaptación en cada consulta del proyecto LDBC SNB. Cabe mencionar que Neo4j nos permite hacer recorridos bidireccionales y además incorpora el uso de la indexación B-tree.	Posee un punto de vista teórico y no profundiza en el aspecto práctico de la implementación de estas estrategias faltando recomendaciones para uso.
F.Rusu y Z. Huang [39]	In-Depth Benchmarking of Graph Database Systems with the Linked Data Benchmark Council (LDBC) Social Network Benchmark (SNB)	Entrega una referencia en cuanto los tiempos de carga por consulta de la problemática que aborda este trabajo de título. Se recalca que las pruebas que se hicieron fue en diferentes cargas de trabajo llegando inclusive a 1000 GB.	No aborda técnicas de optimización como lo plantea el proyecto de LDBC SNB que procura establecer nuevos métodos de optimización en las bases de datos de grafos fundamentado en el diseño físico.
A. Ben Ammar [40]	Query optimization techniques in graph databases	Presenta técnicas de optimización en BDOG citando diferentes obras que entregan referencias al momento de investigar estrategias de diseño físico.	No presenta resultados de experimentos que demuestren su efectividad. Sin embargo, se debe indicar que es un buen punto de partida para indagar diferentes trabajos de investigación en la materia.

Autor	Título	Fortalezas	Debilidades
J. Maginecz G. Szárnyas and D. Varró [41]	Evaluation of optimization strategies for incremental graph queries	Destaca en el documento el procesamiento de consultas relacionales que nos introduce al álgebra relacional extendida. Concepto fundamental para entender los patrones de grafos.	A través del algoritmo Rete que es un algoritmo de reconocimiento de patrones busca evaluar el rendimiento de diferentes planes de consulta lo cual dista de nuestra área de trabajo.
A. Gubichev [42]	Query processing and optimization in graph databases	Presenta trabajos sobre estrategias de optimización donde dos capítulos nos aportan conocimiento en la optimización de consultas en Cypher. Expone recomendaciones y técnicas para la selección de parámetros sobre el desafío presentado por LDBC.	El capítulo de optimización de consultas está enfocado en la comparación de dos versiones de Neo4j para verificar el mejoramiento de la nueva versión. Otra desventaja es el primer capítulo donde se aborda la optimización de consultas complejas de grafos sobre RDF.
J. Cheng, Y. Ke and W. Ng [43]	Efficient Query Processing on Graph Databases	Aporta amplio conocimiento mediante la exposición de diferentes tácticas de planteamiento y combinación de índices. Exhibe recomendaciones para que no se vea afectado el rendimiento y la memora de la base de datos con la modificación de índices.	Los algoritmos se encuentran en SQL provocando efectuar un exhaustivo análisis al lenguaje de consultas y su base de datos para validar su posible migración a Neo4j.
H. He and A. K. Singh [44]	Graphs-at-a-time: Query Language and Access Methods for Graph Databases	Presenta un nuevo lenguaje declarativo en BDOG llamado GraphQL si bien no tiene relación con el lenguaje con el que vamos a trabajar, nos ayuda entender la lógica de procesamiento de las consultas y las técnicas de optimización que expone.	Aborda diferentes lenguajes de consultas tanto de bases de datos SQL como bases de datos NoSQL. No se incluye Cypher, pero si SPARQL la principal competencia de Cypher.

Tabla 2.3: Cuadro comparativo de trabajos relacionados.

Habiendo analizados todos estos documentos e investigaciones debemos hacer una análisis de los diferentes aportes de cada trabajo a la presente investigación. En primer lugar, tenemos que indicar que esta investigación es una extensión del documento "Diseño Físico para la base de datos SNB sobre un gestor orientado a grafos" elaborado por M. Jota y R. Parra [37]. Ambos autores en su trabajo definen trabajar con 13 consultas escogidas al azar, las cuales pertenecen al grupo de consultas BIW de LDBC SNB. De este trabajo cabe destacar que establecen con sus guías de diseño físico un punto de partida puesto que ya han sido validadas mediante variados experimentos. Sin embargo, no queda del todo claro la implementación de cada guía de diseño en sus ejemplos. Este trabajo busca complementar lo ya desarrollado por ambos autores, validando estas guías de diseño físico sobre una versión actualizada de Neo4j, donde adicionalmente se trabaja con las 25

consultas exhibidas en el documento de especificación de LDBC SNB [2]. Igualmente, se exponen nuevas estrategias y se modifican guías de diseño físico con el objetivo de ser aplicadas a un menor costo de almacenamiento. Por otro lado, existen diversos trabajos que hacen alusión a estrategias de diseño físico en BDOG, pero no son puestas a pruebas en el proyecto de LDBC SNB. También nos encontramos con investigaciones que si trabajan con las consultas de LDBC SNB, pero basado en diferentes objetivos como comparar el rendimiento de distintos SGBD o abordar el problema con variados lenguajes de consultas. Incluso existen trabajos que sólo plantean teóricamente sus estrategias sin presentar experimentos que las respalden.

Posterior a todo lo expuesto en el párrafo previo, es necesario indicar que el trabajo que se va a desarrollar no tiene un símil ya presentado.

Capítulo 3

Definición del problema y análisis de requerimientos

En este capítulo se formula el problema que abordaremos, especificando los requerimientos y definiendo el esquema de datos. Posteriormente se analizarán los requerimientos expuestos por LDBC SNB para un grupo de consultas seleccionadas. De igual forma se establecen los objetivos generales y específicos para luego dar paso a presentar la metodología de trabajo. Por último, se precisan los requerimientos funcionales y no funcionales del proyecto.

3.1. Formulación del problema

En el Capítulo 2 se hace referencia al problema que plantea LDBC SNB y sus respectivos desafíos. Esos desafíos correspondían a establecer estrategias de diseño físico sobre dos grupos de consultas: IW que son consultas específicas y BIW que por el contrario son consultas de carácter genérico. Habiendo mencionado lo anterior, debemos indicar que para probar que las estrategias de diseño físico son válidas, las consultas deben recorrer el grafo en su totalidad o abarcar la mayor área posible. LDBC SNB define que las consultas IW abarcan un cantidad significativa de datos pero con la proximidad cercana a sólo un nodo, en cambio las consultas de BIW recorren la totalidad del grafo en la búsqueda de la información. Bajo esta razón las consultas que cumplen esta condición son las consultas de BIW. Por lo tanto, el problema a resolver por esta investigación es comprobar las estrategias de diseño físico sobre las consultas de BIW, esto se efectuará sobre cargas de trabajo de 1 GB, 10 GB y 100 GB con el objetivo adicional de corroborar la escalabilidad de estas estrategias.

El esquema de datos que se muestra en la Figura 3.1 es presentado por LDBC [2]. Define la estructura de los datos utilizados en el benchmark en términos de entidades y

sus relaciones. Los datos representan la actividad de una red social durante un período de tiempo definido. A continuación, pasaremos a definir las entidades y relaciones presentes en el esquema de datos.

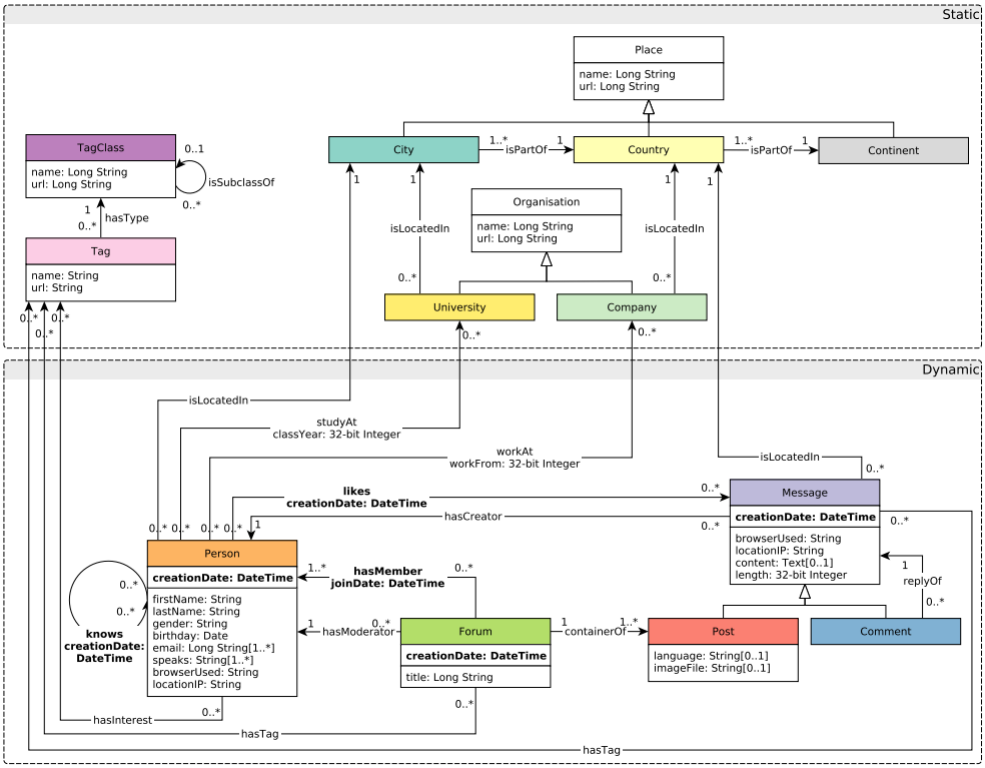


Figura 3.1: Esquema de datos LDBC SNB [2].

3.1.1. Entidades

- City: Es una subclase de la entidad Place. Las entidades City se utilizan para especificar dónde viven las personas, así como dónde operan las universidades.
- Comment: Es una subclase de la entidad Message, y representa un comentario hecho por una persona a un mensaje existente ya sea una publicación o un comentario.
- Company: Es una subclase de la entidad Organisation, y representa una empresa en la que trabajan personas.
- Country: Es una subclase de la entidad Place, y representa un continente del mundo.
- Forum: Es un punto de encuentro donde la gente publica mensajes. Los foros se caracterizan por los temas (representados como etiquetas) de los que habla la gente

en el foro. Existen tres tipos diferentes de foros: perfiles personales de personas, álbumes de imágenes y grupos. Se distinguen por sus identificador, sus títulos y fecha de creación.

- Message: Es una entidad abstracta que representa un mensaje creado por una persona. Esta cuenta con un identificador, con el navegador que fue usado por la persona para crear el mensaje, con una fecha de creación, con la ubicación (referenciado con IP) desde la que se creó el mensaje, su contenido y su respectivo longitud.
- Organisation: Representa un organización en el mundo, la que se identifica con identificador, un nombre y una url.
- Person: Es el perfil creado por una persona. Este perfil debe contar con un identificador, un nombre y apellido, el género de la persona, su fecha de nacimiento, su correo electrónico, los lenguajes que habla la persona, el navegador usado, la fecha de creación y la ubicación (referenciado con IP) para la creación de la cuenta en la red social.
- Place: Representa un lugar en el mundo. Esta entidad debe contar con un identificador, un nombre y una url.
- Post: Es una subclase de la entidad Message, que se publica en un foro. Las publicaciones son creadas por personas en los foros a los que pertenecen. Las publicaciones contienen contenido o archivo de imagen, y se debe especificar el lenguaje de la publicación.
- Tag: Representa un tema o un concepto. Las etiquetas se utilizan para especificar los temas de foros y publicaciones, así como los temas que le interesan a una persona. Además, las etiquetas cuentan con un identificador, un nombre y una url.
- TagClass: Es una clase utilizada para construir una jerarquía de etiquetas. Esta cuenta con un identificador, un nombre y una url.
- University: Es una subclase de la entidad Organisation, y representa una institución donde las personas estudian.

3.1.2. Relaciones

La Tabla 3.1 presenta el detalle de las conexiones entre las entidades descritas en el punto previo. Esta tabla se compone del nombre de la relación, la cardinalidad que se divide entre los puntos de origen y destino, y su descripción.

Nombre	**Cardinalidad**		**Descripción**
conteinerOf	Forum [1]	Post [1..*]	Un foro que contiene una publicación que contiene.
hasCreator	Message [0..*]	Person [1]	Un mensaje y su creador (Person).
hasInterest	Person [0..*]	Tag [0..*]	Una persona y una etiqueta representando un tópico que una persona está interesada.
hasMember	Forum [0..*]	Person [1..*]	Un foro y un miembro de un foro (Person) el que debe contar con la fecha de ingreso.
hasModerator	Forum [0..*]	Person [1]	Un foro y su moderador (Person).
hasTag	Message [0..*]	Tag [0..*]	Un mensaje y una etiqueta representando el tópico de ese mensaje.
hasTag	Forum [0..*]	Tag [0..*]	Un foro y una etiqueta representando el tópico de ese foro.
hasType	Tag [0..*]	TagClass [1]	Una etiqueta y una clase de etiqueta que pertenezca a ella.
isLocatedIn	Company [0..*]	Country [1]	Una empresa y su país de origen.
isLocatedIn	Message [0..*]	Country [1]	Un mensaje y el país desde el que se emitió.
isLocatedIn	Person [0..*]	City [1]	Una persona y su país de origen.
isLocatedIn	University [0..*]	City [1]	Una universidad y la ciudad en que la se ubica.
isPartOf	City [1..*]	Country [1]	Una ciudad que es parte de un país.
isPartOf	Country [1..*]	Continent [1]	Un país que es parte de un continente.
isSubclassOf	TagClass [0..*]	TagClass [0..1]	Una clase de etiqueta y su clase de etiqueta padre.
knows	Person [0..*]	Person [0..*]	Dos personas que se conocen la que debe precisar la fecha en la que se conocieron.

Nombre	**Cardinalidad**		**Descripción**
likes	Person [0..*]	Message [0..*]	Una persona que le entrega un like a un mensaje, se debe indicar la fecha de emisión del like.
replyOf	Comment [0..*]	Message [1]	Un comentario y el mensaje que responde.
studyAt	Person [0..*]	University [0..*]	Una persona y una universidad en la que ha estudiado y el año en que se graduó .
workAt	Person [0..*]	Company [0..*]	Una persona y una compañía en la que trabaja y el año en que la persona comenzó a trabajar en la compañía.

Tabla 3.1: Descripción de las relaciones entre entidades.

3.2. Análisis de consultas

En la presente sección se analizan los requerimientos de cinco consultas con el fin de identificar los diferentes inconvenientes que puedan presentar. Para esto, se utilizan las consultas BI1, BI2, BI7, BI17 y BI21. Estas consultas son seleccionadas bajo el argumento de la estructura que presentan y la cantidad propiedades y operaciones con las que se trabajan lo que permite evaluar de mejor forma los diferentes factores y casos que se puedan dar al momento de la aplicación de cada estrategia.

3.2.1. Consulta BI1

El documento de especificaciones de LDBC SNB precisa los requerimientos para la consulta BI1 [2]. El objetivo es retornar todos los mensajes creados antes de una fecha indicada y agruparlos en 3 niveles: el año de creación, por cada año se debe agrupar los mensajes que sean de tipo comentario y los que no lo sean. Por último, por cada año se deben agrupar los mensajes en función de su longitud. Al analizar los requerimientos de la consulta nos damos cuenta que se evidencia un gran número de operaciones para cumplir con las condiciones, es ahí donde tenemos que indicar que las operaciones de agregación son una de las acciones más costosas para el motor de ejecución. Otra cosa que llama la atención es que se fije la consulta sobre el patrón de un tipo de nodo. Todo lo anterior se puede apreciar en la Tabla 3.2.

BI / read / 1

query	BI / read / 1
title	Posting summary
pattern	message: Message *creationDate < $date* length year(creationDate)
desc.	Given a date, find all Messages created before that date. Group them by a 3-level grouping: 1. by year of creation 2. for each year, group into Message types: is Comment or not 3. for each year-type group, split into four groups based on length of their content • `0`: 0 <= length < 40 (short) • `1`: 40 <= length < 80 (one liner) • `2`: 80 <= length < 160 (tweet) • `3`: 160 <= length (long)

Tabla 3.2: Definición Requerimientos Consulta BI1

3.2.2. Consulta BI2

La solicitud plantea que para la consulta BI2 se deben retornar todos los mensajes creados dada una fecha de inicio y una fecha final por las personas ubicadas en los países entregados. De igual forma, debemos seleccionar las personas que hayan creado los mensajes y sus etiquetas, allí se dividen a las personas en grupos según los requerimientos especificados en la sección *desc.* que se evidencia en la Tabla 3.3 . Esto se agrupa en cinco niveles: el nombre del país de la persona, el mes en que fue creado el mensaje, el género y el grupo de edad de la persona y el nombre de la etiqueta adjunta al mensaje, considerando sólo los grupos que posean más de 100 mensajes. A simple vista revisando la sección *pattern* podríamos encontrarnos con complicaciones en la relación *hasCreator* que vincula los nodos *Message* y *Person*, puesto que ambas entidades son las más complejas al momento de trabajar. Lo anterior también se repite en la sección *desc.* donde cuatro de los cinco niveles de agrupamiento hacen referencia a las entidades antes mencionadas, lo que indica un alto trabajo con sus propiedades.

3.2.3. Consulta BI7

El objetivo de la consulta BI7 se aprecia en la Tabla 3.4 el cual consiste en encontrar a todas las personas que hayan creado un mensaje con una etiqueta dada. Además, para los mensajes de la segunda persona enumera la suma de total de *likes* recibidos por la tercera persona. Al igual que en la consulta anterior podemos ver que se repiten algunas condiciones como lo es la relación *hasCreator* que une los nodos *Message* y *Person*. Sin embargo, en este caso la mayor complicación que podríamos tener es en la búsqueda de los nodos ya que este proceso se presenta en tres oportunidades. Si bien el documento indica sólo el acceso a una propiedad nos damos cuenta que existen dos parámetros calculados:

BI / read / 2

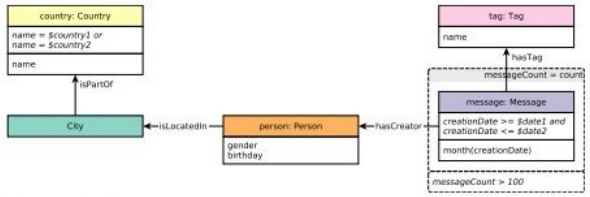
query	BI / read / 2
title	Top tags for country, age, gender, time
pattern	
desc.	Select all Messages created in the range of `[startDate, endDate]` by Persons located in `country1` or `country2`. Select the creator Persons and the Tags of these Messages. Split these Persons, Tags and Messages into a 5-level grouping: 1. name of country of Person, 2. month the Message was created, 3. gender of Person, 4. age group of Person, defined as years between person's birthday and end of simulation (2013-01-01), divided by 5, rounded down (partial years do not count), 5. name of tag attached to Message. Consider only those groups where number of Messages is greater than 100.

Tabla 3.3: Definición Requerimientos Consulta BI2.

authority score y *popularity score*. Estos se basan en el cálculo de diferentes parámetros, donde nos encontramos con una restricción puesto que el primer parámetro es dependiente del segundo imposibilitando su modificación.

BI / read / 7

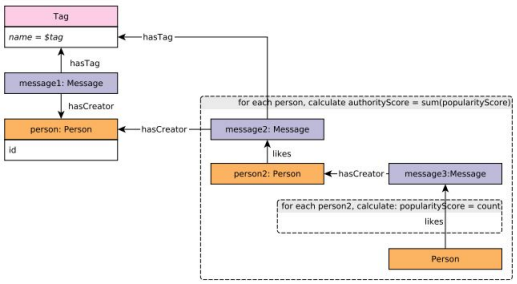
query	BI / read / 7
title	Most authoritative users on a given topic
pattern	
desc.	Given a Tag, find all Persons (`person`) that ever created a Message (`message1`) with the given Tag. For each of these Persons (`person`) compute their "authority score" as follows: • The "authority score" is the sum of "popularity scores" of the Persons (`person2`) that liked any of that Person's Messages (`message2`) with the given Tag. • A Person's (`person2`) "popularity score" is defined as the total number of likes on all of their Messages (`message3`).

Tabla 3.4: Definición Requerimientos Consulta BI7

3.2.4. Consulta BI17

Para la consulta BI17 que se observa en la Tabla 3.5 el documento de requerimientos de LDBC SNB plantea que dado un país determinado, contar la cantidad de nodos que cumplan con poseer tres amistades diferentes de tal manera que:

- a sea amigo de b.
- b sea amigo de c.
- c sea amigo de a.

El esquema nos muestra que el único inconveniente se halla en la relación *knows* que une a los diferentes nodos *Person* generando condiciones que se deben cumplir, desencadenando un mayor costo de procesamiento reflejados en DB Hits para el motor de ejecución.

BI / read / 17

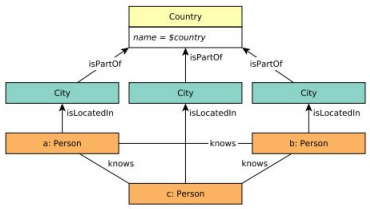
query	BI / read / 17
title	Friend triangles
pattern	
desc.	For a given `country`, count all the distinct triples of Persons such that: • `a` is friend of `b`, • `b` is friend of `c`, • `c` is friend of `a`. Distinct means that given a triple t_1 in the result set R of all qualified triples, there is no triple t_2 in R such that t_1 and t_2 have the same set of elements.

Tabla 3.5: Definición Requerimientos Consulta BI17

3.2.5. Consulta BI21

Para la consulta BI21 se debe encontrar a los *zombies* dado un país y efectuar una operación para calcular su puntuación. Un *zombie* es una entidad *Person* creado antes de una fecha. Su puntuación corresponde a la división entre el número de *likes* recibido por otros *zombies* y el número total de *likes*. Nuevamente, nos encontramos con las entidades *Message* y *Person*, no obstante, en este caso es diferente puesto que para obtener la puntuación se deben efectuar búsquedas independientes con ambas entidades presentes en cada caso lo que implica un alto costo de procesamiento. Lo antes descrito aparece con un mayor detalle en la Tabla 3.6.

BI / read / 21

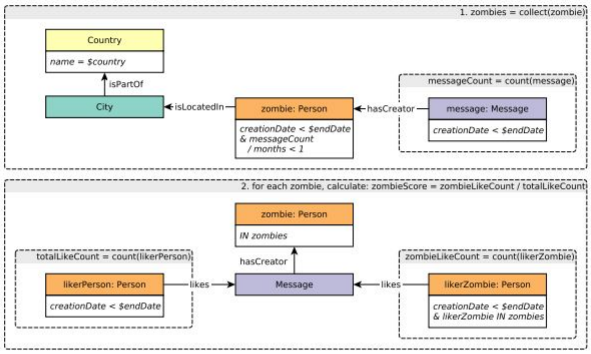
query	BI / read / 21
title	Zombies in a country
pattern	
desc.	Find zombies within the given `country`, and return their zombie scores. A `zombie` is a Person created before the given `endDate`, which has created an average of `[0, 1)` Messages per month, during the time range between profile's `creationDate` and the given `endDate`. The number of months spans the time range from the `creationDate` of the profile to the `endDate` with partial months on both end counting as one month (e.g. a `creationDate` of Jan 31 and an `endDate` of Mar 1 result in 3 months). For each `zombie`, calculate the following: • `zombieLikeCount`: the number of likes received from other zombies. • `totalLikeCount`: the total number of likes received. • `zombieScore`: `zombieLikeCount` / `totalLikeCount`. If the value of `totalLikeCount` is 0, the `zombieScore` of the `zombie` should be 0.0. For both `zombieLikeCount` and `totalLikeCount`, only consider likes received from profiles that were created before the given `endDate`.

Tabla 3.6: Definición Requerimientos Consulta BI21

3.3. Solución propuesta

La Figura 3.2 nos presenta el esquema de la solución propuesta, en el cual deberemos inicialmente evaluar el hardware y software del ambiente experimental para identificar los potenciales inconvenientes o restricciones que puedan surgir. Posteriormente, es indispensable resolver los problemas identificados certificando el correcto funcionamiento del ambiente experimental para efectuar las pruebas de la mejor forma posible. Seguidamente, es necesario analizar y estudiar las diferentes opciones de estrategias con el fin de establecer las estrategias de diseño físico que se expondrán en este trabajo. En el estudio experimental es inevitable determinar la metodología con la que se van a capturar las estadísticas de cada carga de trabajo. Igualmente, es imperioso realizar las operaciones matemáticas con el objetivo de efectuar las validaciones posteriores. En el siguiente paso se analizan y validan las estadísticas conseguidas buscando identificar patrones y explicar el comportamiento que presenta cada consulta. Habiendo validado los resultados obtenidos se procede a establecer las diferentes conclusiones extraídas del análisis. De la misma forma, se establecen recomendaciones de uso según la prioridad y conveniencia de cada estrategia de diseño físico.

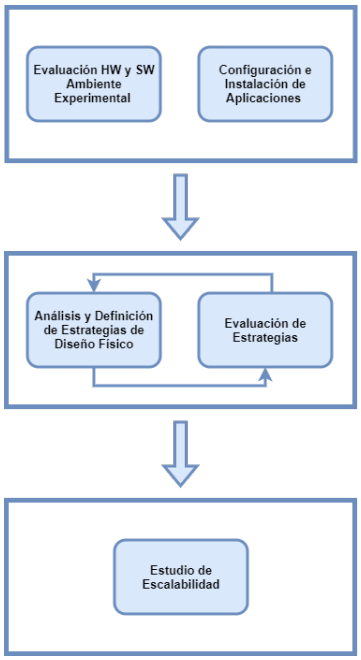


Figura 3.2: Esquema Solución Propuesta

3.4. Objetivos

A continuación, se exponen los objetivos generales y específicos. Los objetivos generales están orientados a definir lo que se busca lograr o demostrar. Los objetivos específicos apuntan a precisar cuáles serán los parámetros a evaluar para demostrar el cumplimiento de los objetivos generales.

3.4.1. Objetivo general

El objetivo general de este trabajo es proponer y validar estrategias de diseño físico que impacten positivamente el tiempo de respuesta de una *BDOG* mediante el análisis e implementación en cada consulta de *BIW*. Como ya lo hemos mencionado se busca satisfacer este fin efectuando una implementación de diferentes cargas de trabajo sobre Neo4j Browser. El propósito de llevar a cabo este trabajo es proponer y validar estrategias de diseño físico que cumplan con el objetivo de los usuarios de reducir los tiempos de respuestas a sus consultas en Neo4j.

3.4.2. Objetivos específicos

Desde el punto anterior se pueden desprender los siguientes objetivos específicos que buscan validar las estrategias propuestas en este trabajo.

- Evaluar las estrategias de diseño físico propuesto con el fin de verificar que las estrategias propuestas cumplan su propósito.
- Validar mediante un estudio de escalabilidad que las técnicas propuestas se desempeñen de la misma forma independiente del volumen de datos en el que sea implementado.

3.5. Metodología

El procedimiento usado en este experimento es el Método Científico presentado por J. Cueva [45]. En este trabajo se exponen principios fundamentales que debemos seguir como es el garantizar la reproducibilidad de los experimentos describiendo de forma cuidadosa y detallada, añadiendo la posibilidad de descargar los datos con los que se trabajan. Otro principio que seguimos en esta investigación es la probar que una hipótesis es verídica o no mediante un respaldo de datos. El origen de este trabajo surge desde la investigación desarrollada por M. Jota y R. Parra [37], sin embargo, en la presente investigación se realizan diferentes modificaciones y estas son probadas sobre una versión actualizada de Neo4j Browser. Debemos averiguar y analizar los nuevos trabajos relacionados con las guías diseño que se hayan publicado en este período de tiempo. Luego tenemos que plantear el problema a resolver con la especificación de los alcances, límites y condiciones, teniendo en cuenta la construcción de la hipótesis basada en que los tiempos de ejecución en una base de datos sin diseño físico es mayor que los tiempos de ejecución en una base de datos con diseño físico. Para esto tenemos que validar la hipótesis mediante pruebas estadísticas que garanticen matemáticamente el cumplimiento del objetivo. El diseño que se establece para esta investigación es el diseño experimental donde se definen variables a estudiar de los resultados recordando que nuestras variables a examinar son el tiempo de ejecución y los DB Hits. Luego, se ejecutarán las pruebas experimentales sobre las diferentes cargas de trabajo probando así la escalabilidad de los estrategias propuestas. Al analizar los resultados se debe comprobar la hipótesis mediante la presentación de las pruebas estadísticas basadas en los resultados obtenidos exponiendo así las conclusiones al trabajo realizado. Lo recientemente explicado se puede apreciar en el esquema de la figura 3.3.

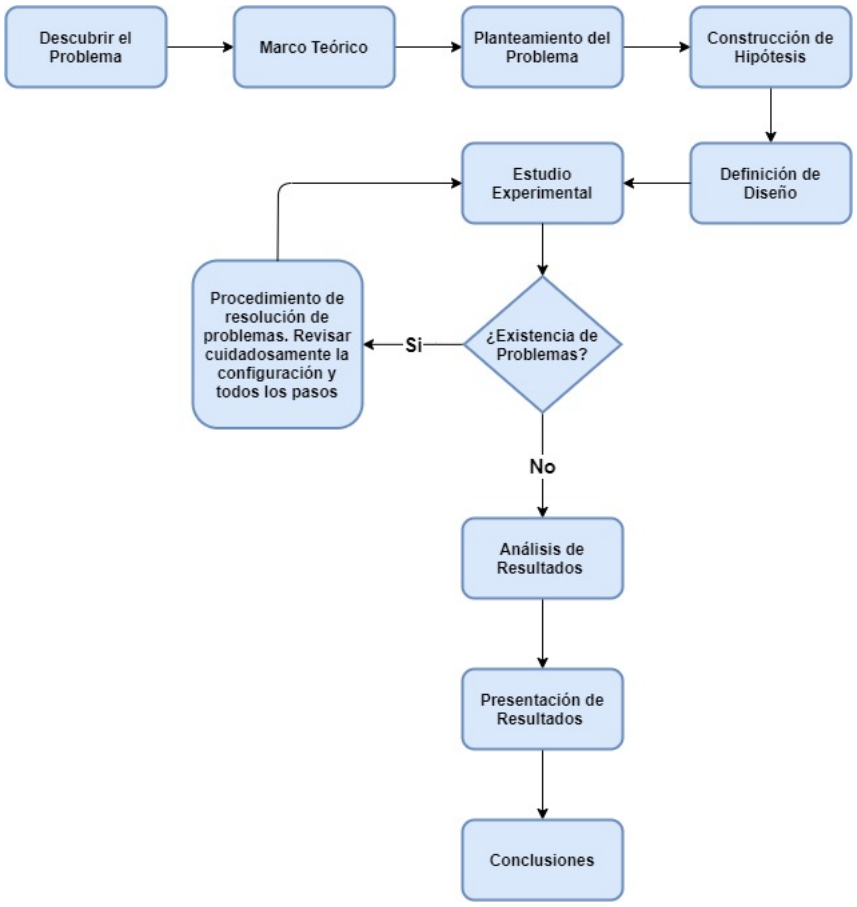


Figura 3.3: Esquema Metodología de Trabajo

3.6. Especificación de requerimientos

3.6.1. Requerimientos funcionales

A continuación, definiremos los requerimientos funcionales del proyecto.

- El SGBD debe ser capaz de cargar y trabajar con los diferentes volúmenes de datos.
- Clasificar estados de error ante la existencia de algún inconveniente de procesamiento en las pruebas de experimentación y la acción a ejecutar en caso de presentarse.
- Cada prueba de experimentación debe ser registrada. En caso de existir diferentes opciones en cuanto a estrategias, se debe efectuar una comparación entre ellas.

- Se debe tener registro del costo de almacenamiento al momento de implementar una guía de diseño.
- Las pruebas de experimentación para cada carga de trabajo se deben realizar bajo las mismas circunstancias para que los registros no se vean afectados.

3.6.2. Requerimientos no funcionales

Como ya sabemos los requerimientos no funcionales son de gran importancia en cada proyecto por lo que pasaremos a definirlos.

- Alta disponibilidad de la base de datos: La disponibilidad del SGBD debe ser continuo con un nivel de servicio por sobre el 99 % para las diferentes etapas de pruebas del experimento [46]. El porcentaje de disponibilidad se determina con el fin de garantizar que los resultados del estudio experimental no se vean afectados.
- Integridad de los datos: El término integridad de datos se refiere a que la información se encuentre correcta y completa en un SGBD. En este caso es fundamental ya que si no contamos con esto nos impide levantar conclusiones basados en las estadísticas recopiladas.
- Alta capacidad operacional: Es necesario que el ambiente de pruebas tenga la capacidad de procesar altos volúmenes de datos debido a las diferentes consultas con las que se trabajan en esta investigación.

Capítulo 4

Guías de diseño físico

En este capítulo se presenta la definición de las guías de diseño físico indicando sus ventajas y desventajas, y en qué casos es conveniente aplicar cada una de ellas. En la segunda parte de este capítulo expondremos la implementación sobre las consultas que se mencionaron en la Sección 3.2 haciendo énfasis en el análisis del plan de ejecución, el código de la consulta y la implementación de las estrategias. Cabe destacar que antes de empezar debemos definir un concepto primordial para este capítulo que corresponde a la equivalencia de resultados. Cuando hablamos de equivalencia de resultados nos referimos a que los resultados de la consulta con diseño físico deben ser iguales a los resultados de la consulta sin diseño físico ya que nosotros planteamos guías de diseño físico que reduzcan el tiempo de ejecución sin modificar el resultado de la consulta.

4.1. Consideraciones de hardware y software

Nuestro primer paso es analizar las especificaciones de hardware del ambiente experimental donde vamos a trabajar haciendo hincapié en la capacidad del procesador, memoria *Random Allocated Memory* (RAM) y almacenamiento disponible. La memoria RAM es un factor fundamental al momento de implementar nuestra solución debido a lo cual se estudian opciones viables dada las limitaciones del servidor.

4.2. Guía de aplicación de estrategias de diseño físico

Es necesario hacer un exhaustivo análisis de los planes de ejecución obtenidos de las consultas sin diseño físico. Para esto explicaremos los patrones identificados que justifican cada guía de diseño propuesta. Esta solución se compone por tres guías de diseño físico: Reescritura de Consulta, Materialización de Caminos y Creación de Índices. La primera guía de diseño se descompone en tres estrategias: Consulta Mínima, Descomposición de

Consulta y Propiedades Limitadas. Se debe indicar que las estrategias Consulta Mínima, Materialización de Caminos y Creación de índice fueron expuestas por M. Jota y R. Parra [37]. En este trabajo de investigación se proponen las estrategias Descomposición de Consulta y Propiedades Limitadas. Por otro lado, la estrategia Materialización de Caminos fue expuesta por los autores antes mencionados, no obstante, se efectúa una modificación que permite optimizar la idea original. Para finalizar, debemos definir un concepto primordial que es la equivalencia de datos. Este concepto precisa la igualdad en los resultados, sin embargo, no hace énfasis en cómo se obtiene lo que permite conseguir un resultado en un menor tiempo, siempre y cuando se mantenga la equivalencia con respecto a los datos adquiridos desde la ejecución sin diseño físico.

1. Reescritura de Consulta: La principal virtud de las BDOG es la interrelación de los datos, los cuales forman caminos que el motor de ejecución debe recorre para obtener los datos requeridos por el usuario. Considerando lo anterior, se puede llegar a disminuir los tiempos de ejecución replanteando la consulta y manteniendo la equivalencia de los resultados. Esta estrategia no tiene costo de almacenamiento al momento de ser implementada.

    a) Consulta Mínima: Cuando se elabora una consulta se debe construir teniendo en cuenta su eficiencia. Para esto, es necesario analizar las estadísticas de procesamiento para validar que no exista redundancia. La estrategia Consulta Mínima aborda lo antes mencionado. Esta estrategia tiene el objetivo de eliminar rutas duplicadas, donde la única diferencia que puede existir son los nombres de las variables pero siempre y cuando los nodos tengan la misma etiqueta y relación. Para la implementación de esta estrategia se requiere hacer un profundo análisis y en caso que se encuentren posibles caminos duplicados validar que sean equivalentes en sus resultados entre la consulta inicial y la consulta optimizada. Para eliminar esta duplicidad es necesario reutilizar las variables asignadas según sea el caso, y así tener la posibilidad de eliminar el camino duplicado manteniendo la equivalencia en el código. Al cumplir con lo anterior logramos que el motor de ejecución no deba recorrer caminos adicionales que deriven en un costo agregado con respecto al tiempo de ejecución y *DB Hits*.

    b) Descomposición de Consulta: Una consulta se inicia con un conjunto de datos que van siendo modificado según los requerimientos mediante condiciones que nos entregan por resultado un subconjunto de datos. En base de datos una de las buenas prácticas es trabajar con consultas lo más simple y eficiente posible. Nosotros planteamos la estrategia Descomposición de Consulta, el modo de aplicar es reubicar las operaciones de agregación o condiciones donde se ubiquen los nodos o relaciones que se encuentren vinculados. Esto significa introducir condiciones entre cada sentencia *MATCH* u *OPTIONAL MATCH*, para reducir el conjunto de datos inicial a un subconjunto que con el avance de la consulta

se vaya transformando en otro subconjunto y así sucesivamente. Lo anterior se justifica siempre y cuando existan operaciones de agregación o condiciones que se puedan reubicar.

c) Propiedades Limitadas: Observando la documentación de Neo4j y habiendo realizado diferentes experimentos, nos pudimos dar cuenta que una de las tareas más costosas es es el acceso a las propiedades. LDBC SNB define en la mayoría de sus consultas que se deben limitar la cantidad de registros como resultado pero en primera instancia accede a las propiedades y luego restringe el número de datos mediante las sentencias *ORDER BY* y *LIMIT*. M. Hunger [47] propone restringir en primera instancia el número de registro y después acceder a las propiedades. Por lo que nosotros tomamos esa técnica, sin embargo, esto sólo es aplicado sobre las consultas que utilicen una operación de agregación en una sentencia *RETURN*. Para esto proponemos reubicar las operaciones de agregación en una sentencia *WITH* previo a las sentencias *ORDERY BY* y *LIMIT*, dejando en última instancia la entrega de resultados mediante la sentencia *RETURN*. Otra forma de ser aplicada esta estrategia es la remoción de variables precalculadas y establecer el cálculo directamente donde se requiere logrando eliminar el costo generado por esa operación.

2. Materialización de Caminos: Las bases de datos relacionales poseen una estrategia conocida por muchos profesionales llamada vista materializada, esta se define como una vista común, pero en lugar de almacenar la definición de la vista, almacena el resultado de la consulta [48]. Por esta razón se estudiaron los documentos técnicos de la plataforma y nos dimos cuenta que Neo4j no implementa esta estrategia, producto de lo anterior nos vimos en la necesidad de estudiar una forma de replicar esta estrategia en una BDOG. M. Jota y R. Parra [37] en su documento presentan una alternativa denominada "Materialización de Caminos" que definen su implementación de la siguiente forma:

   "*Esta guía consiste en generar una relación que represente un camino precalculado, es decir, se precalcula el camino y se almacena en una relación. Al ocasionar una alternativa más corta que la original se rebaja el costo generado por la expansión de caminos*"

   Según lo comprendido de esta definición la Materialización de Caminos se explica como la creación de relaciones entre dos nodos buscando establecer caminos directos hacia el nodo objetivo obteniendo una disminución en los tiempos de ejecución y sobre todo en los *DB Hits*. Al estudiar lo que proponen ambos autores percibimos que su implementación es sobre todos los nodos que posean la relación a materializar. Nosotros consideramos que no es necesario aplicar esta estrategia de esa forma

ya que genera un mayor costo de almacenamiento, por lo que proponemos limitar lo anterior a sólo a los tipos de nodos que se vean implicados en el punto crítico de la consulta sujeto a las condiciones que presente la consulta en la que se implementará esta solución.
La primera desventaja de aplicar esta estrategia corresponde a que Neo4j no la implementa de manera que no existe una actualización automática, en consecuencia, debemos encargarnos de actualizar este camino materializado a medida que se ingresen nuevos registros.
La segunda desventaja corresponde a que la Materialización de Caminos como sabemos es una estrategia de alta complejidad donde Neo4j utiliza memoria para establecer las nuevas relaciones. Al no poder cubrir la totalidad de relaciones por las posibles restricciones de memoria RAM, la solución sería el uso de las sentencias *LIMIT* y *SKIP*. Estas funciones nos permiten generar dos grupos de nodos para ser trabajados de forma dedicada. Un ejemplo de lo anterior es aplicar ambas funciones sobre un grupo de 100 nodos, empezando por la sentencia *LIMIT* donde definiremos 50 nodos y en consecuencia la diferencia la abordaremos con la función *SKIP* 50. Con esto podemos dividir la tarea de materializar una relación al momento de implementar esta estrategia.
La tercera desventaja corresponde al impacto en el rendimiento que puede ser positivo o negativo en otras consultas. En el primer caso puede darse la situación de que el motor de ejecución reutilice los caminos materializados dado que puede existir una coincidencia en los caminos que deba recorrer otra consulta. En el caso de un impacto negativo este se debe al existir una gran cantidad de relaciones provocando que el motor de ejecución deba navegar por una mayor cantidad de caminos.
La cuarta desventaja es el costo de almacenamiento que tiene la aplicación de esta estrategia. Dado que la forma en que aplicamos la estrategia es diferente a lo propuesto por M. Jota y R. Parra [37], el costo de su implementación estará sujeto exclusivamente a la cantidad de relaciones que se deben materializar por consulta.

3. Creación de índice: Los índices provienen de las bases de datos relacionales donde cuyo objetivo es apuntar a un valor de una determinada columna [49]. En Neo4j se replica el concepto con la única diferencia que en vez de apuntar hacia una columna apuntamos hacia un tipo nodo y su propiedad. Cabe destacar además que al igual que las bases de datos relacionales los índices poseen un costo de almacenamiento que variará según la cantidad de datos que se deben referenciar. Esto crea el deber de analizar qué tan conveniente es implementar esta estrategia y sobre todo mantener un equilibrio entre costo y beneficio.

4.3. Implementación de estrategias de diseño físico sobre el Social Network Benchmark

Con el objetivo de analizar, diseñar e implementar una solución para mostrar la implementación de cada guía de diseño y en consecuencias sus respectivas estrategias, se han seleccionado un subconjunto de consultas que ya fueron presentadas en la Sección 3.1. Se expone la implementación del subconjunto de consultas restantes en el Anexo A con el análisis, diseño e implementación en detalle.

4.3.1. Reescritura de consulta

La principal virtud de las BDOG es la interrelación de sus datos, los cuales forman caminos que el motor de ejecución debe recorrer para obtener los datos requeridos. Considerando lo anterior, se puede llegar a reducir los tiempos de ejecución replanteando la consulta, siempre y cuando se mantenga la equivalencia. Esta estrategia no posee costo de almacenamiento al momento de ser implementada.

Consulta mínima

En la consulta BI7 se implementa la estrategia Consulta mínima. En la Figura 4.1 podemos observar que en el primer recuadro rojo se destacan la duplicidad de dos caminos, adicionalmente en el segundo recuadro rojo se aprecia un alto costo de procesamiento producto de diferentes operaciones sobre los registros. A continuación, se presentan los operadores que se encuentran en el segmento del plan de ejecución.

1. Dos *scan* para los nodos *Tag*.
2. Ocho operadores *Filter* para los nodos *Tag*, *Message* y *Person*.
3. Seis operadores *Expand* perteneciente a las relaciones *Tag* y *Message* y en segunda instancia *Message* y *Person*.
4. Un operador *NodeHasJoin* para unir los resultados de la búsqueda de personas que cumplan con los requerimientos.
5. Un operador *OptionalExpand* que valida el *like* que le ha dado la segunda persona al mensaje creado por la primera persona.
6. Un operador *Argument* para capturar los resultados del punto previo.
7. Un operador *Optional* que identifica la tercera persona que le haya dado *like* a los mensajes creados por la segunda persona.

8. Un operador *Apply* para identificar las personas que le han dado *like* a los respectivos mensajes.

9. Un operador *EagerAggregation* que ordena los resultados accediendo a las propiedades de los nodos *authorityScore* y *person1*

10. Un operador *Top* que limita la cantidad de registros.

Al momento de revisar el código de la consulta encontramos redundancia en dos nodos *Message* cuya única diferencia es el nombre de las variables, esto lo podemos apreciar en la Figura 4.2. Nuestra solución es eliminar la segunda línea de código y reutilizar la variable *message1*. Adicionalmente, notamos que al aplicar esta estrategia rebajamos el conjunto de datos causando un impacto positivo en el tercer camino delimitado en el cuadro azul. Cabe destacar que se podría haber implementado la Materialización de Caminos sobre el punto crítico de la consulta, el cual corresponde a la relación *LIKES* que vincula a los nodos *Person* y *Message*. Esto no se realiza debido a que se consigue un significativo porcentaje de mejora sin un costo de almacenamiento asociado.

Al revisar nuevamente el código nos damos cuenta que en el segundo recuadro de color rojo se puede implementar otra estrategia que corresponde a la de Propiedades Limitadas. La Figura 4.3a demuestra que se puede modificar la estructura del código para primero limitar el número de registros y luego acceder a las diferentes propiedades y operación de agregación. Una vez aplicada la tercera técnica de la primera guía de diseño se puede visualizar el código final en la Figura 4.3b.

```
RETURN
  person1.id,
  count(DISTINCT like) AS authorityScore
ORDER BY
  authorityScore DESC,
  person1.id ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  person1,
  count(DISTINCT like) AS authorityScore
ORDER BY
  authorityScore DESC,
  person1.id ASC
LIMIT 100
RETURN
  person1.id,
  authorityScore
```

(b) Segmento Código Optimizado

Figura 4.3: Solución Propuesta “Propiedades Limitadas” Consulta BI7.

Finalmente, podemos observar en la Figura 4.4 el resultado de la implementación y el impacto que produjo estas dos estrategias en el plan de ejecución. Allí sobresale el impacto positivo de la eliminación de la duplicidad de los caminos que se enmarca en el recuadro azul y también la disminución del costo de procesamiento del segundo recuadro rojo. La combinación de estas dos estrategias consigue reducir el tiempo de ejecución y los DB Hits de la consulta sin diseño físico en un 91,68 % y 92,73 %, respectivamente.

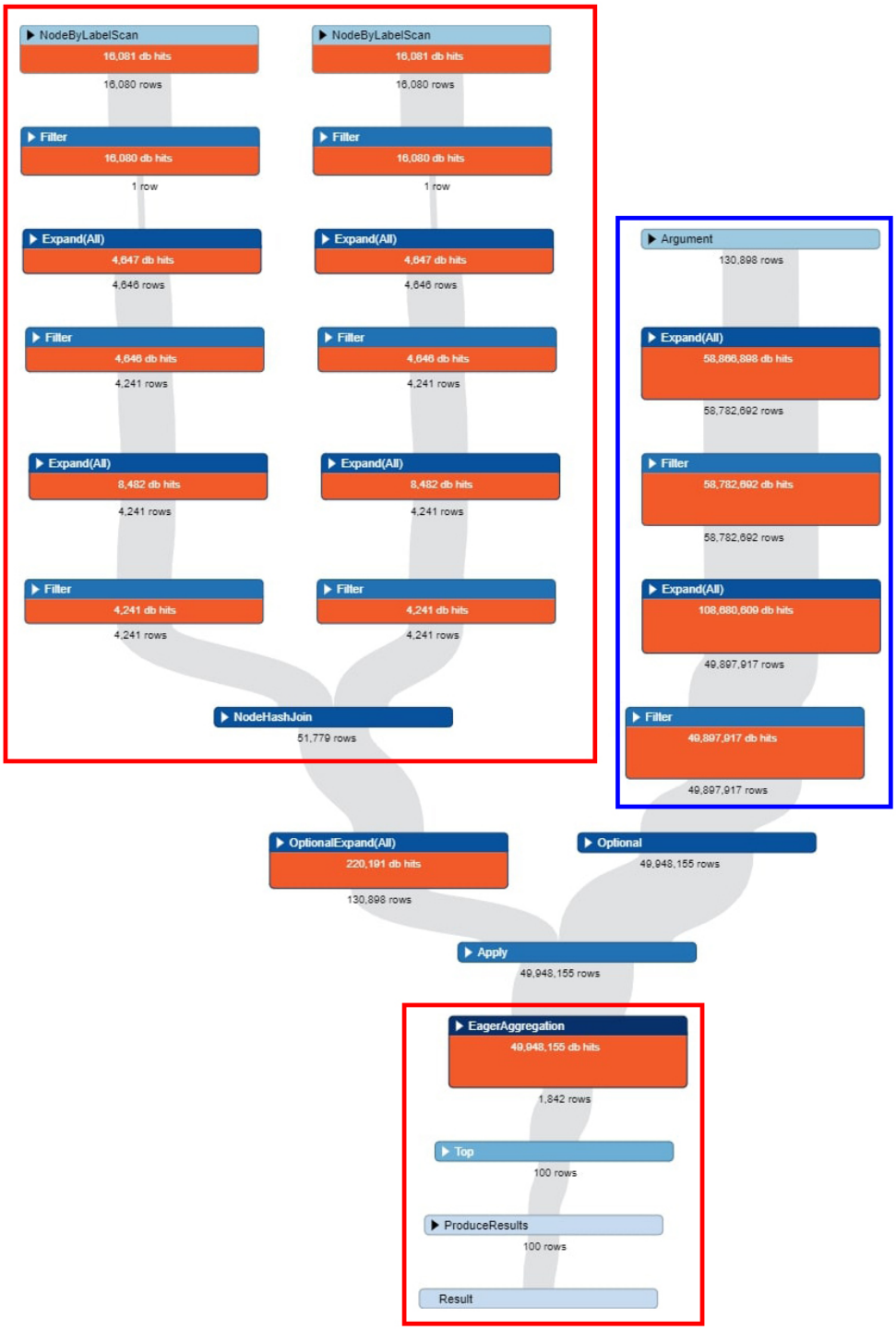


Figura 4.1: Plan de Ejecución Consulta BI7.

```
MATCH (tag)<-[:HAS_TAG]-(message1:Message)-[:HAS_CREATOR]->(person1:Person)
MATCH (tag)<-[:HAS_TAG]-(message2:Message)-[:HAS_CREATOR]->(person1)
```

Figura 4.2: Solución Propuesta “Consulta Mínima” Consulta BI7.

Descomposición de consulta

En la Figura 4.5 se puede observar el recuadro que resalta el operador *Projection* que corresponde a uno de los puntos críticos de la consulta BI2. El gran causante de esto es el requerimiento de la consulta que se concentra en sólo una sentencia *MATCH*. En razón de esto existen dos soluciones la Materialización de Caminos en la relación *HAS_CREATOR* y la Descomposición de Consulta. Nosotros nos inclinamos por la segunda opción ya que para su aplicación no posee un costo de almacenamiento.Adicionalmente, esta consulta no es costosa en comparación al resto. La segunda opción propone descomponer la estructura de la consulta para sólo realizar las operaciones sobre los nodos y relaciones que se encuentren vinculados mermando el número de registros que se deben trabajar. A continuación, se presenta un segmento del plan de ejecución con sus respectivos operadores.

1. Tres operadores *Filter* sobre las propiedades de los nodos *Person*, *Message*, *Country*, *Tag*.
2. Dos operadores *Expand* que se aplican en las relaciones *Person* y *Message*, *Message* y *Tag*.
3. Un operador *Projection* donde se efectúan las operaciones de agregación y sus cálculos respectivos.
4. Un operador *EagerAggregation* que efectúa el conteo de la cantidad de mensajes.

En la Figura 4.6 se encuentra el segmento original de la consulta BI2, nos podemos dar cuenta que todas las condiciones se establecen en una sentencia *WHERE*. Según lo presentado anteriormente, debemos descomponer la consulta e introducir las condiciones que se relacionan con sus nodos respectivos. El primero corresponde a la propiedad *creationDate* del nodo *Message* y el segundo se aplica sobre la propiedad *name* del nodo *Country*.

```
MATCH
  (country:Country)<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(person:Person)
  <-[:HAS_CREATOR]-(message:Message)-[:HAS_TAG]->(tag:Tag)
WHERE message.creationDate >= 20091231230000000
  AND message.creationDate <= 20101107230000000
  AND (country.name = 'Ethiopia' OR country.name = 'Belarus')
```

Figura 4.6: Segmento Código Original Consulta BI2

Figura 4.4: Plan de Ejecución Solución Propuesta Consulta BI7.

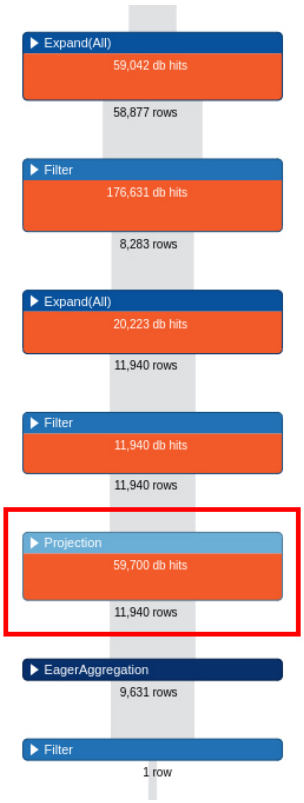


Figura 4.5: Segmento Plan de Ejecución Consulta BI2.

La implementación de nuestra solución propuesta se puede observar en la Figura 4.7. El uso de esta estrategia está sujeto al requisito de que existan sentencias *WHERE* donde se agrupen todas las condiciones. Cabe destacar que el efecto que provoca su implementación es la disminución del conjunto inicial repercutiendo directamente las operaciones de agregación o aritméticas. Por ejemplo, el costo de procesamiento de las operaciones de agregación sobre un conjunto de 1000 registros no es el mismo que el de un conjunto de 400 registros.

```
MATCH
  (country:Country)<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(person:Person)
WHERE
  country.name = 'Ethiopia' OR country.name = 'Belarus'
MATCH
  (person)<-[:HAS_CREATOR]-(message:Message)-[:HAS_TAG]->(tag:Tag)
WHERE
  message.creationDate >= 20091231230000000
  AND message.creationDate <= 20101107230000000
```

Figura 4.7: Solución Propuesta “Descomposición de Consulta” Consulta BI2.

Se presenta el segmento del plan de ejecución de la solución propuesta en la Figura 4.8, allí se enmarca un recuadro donde indicamos los operadores que sufrieron modificaciones. Una de esas modificaciones corresponde al operador *Projection* ya que sufrió una baja quedando en 16.566 DB Hits. Lo anterior implica que los resultados del tiempo de ejecución y DB Hits van a mejorar a medida que se vaya escalando la carga de trabajo.

Figura 4.8: Segmento Plan de Ejecución Solución Propuesta Consulta BI2.

Propiedades limitadas

La estrategia de Propiedades Limitadas puede ser combinada con cualquier otro tipo de estrategia puesto que nos permite incrementar la optimización de una consulta y su uso es altamente recomendado. El plan de ejecución de la consulta BI21 expuesto en la Figura 4.9 nos muestra el punto crítico destacado en el recuadro rojo. El punto crítico hace referencia al operador *EagerAggregation*, en este operador se realizan operaciones aritméticas

cuyo resultado se asigna a nuevas variables, esto genera un alto costo de procesamiento que se puede optimizar mediante la aplicación de la estrategia Propiedades Limitadas. A continuación, se especifican los operadores que presentan costo de procesamiento en el plan de ejecución.

1. Un operador *scan* que se encarga de la búsqueda de los nodos dado un país.
2. Un operador *Projection* que asigna el valor a las variables *endDateYear* y *endDateMonth*.
3. Tres operadores *Filter* aplicados sobre las propiedades *country.name* y *message.creationDate*, además de las dos variables mencionadas en el punto anterior.
4. Un operador *EagerAggregation* que realiza las diferentes operaciones de agregación.

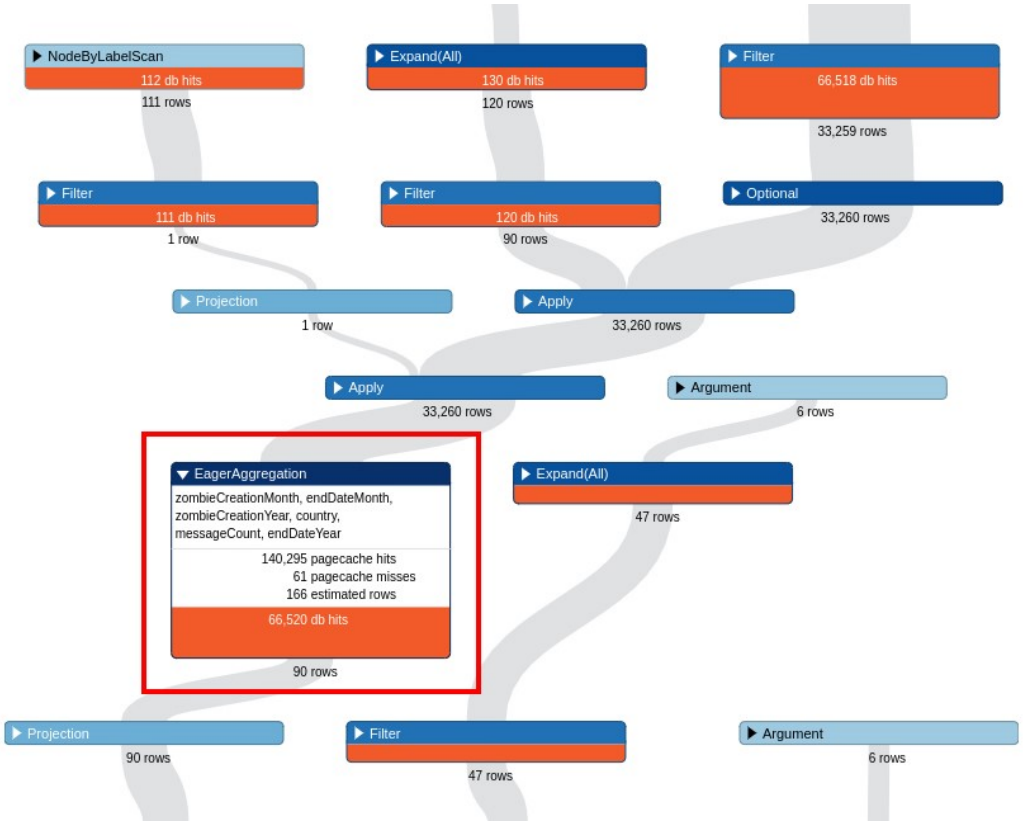


Figura 4.9: Segmento Plan de Ejecución Consulta BI21.

El segmento de código original que se encuentra en la figura 4.10a evidencia la

asignación del resultado de operaciones aritméticas a las variables *endDateYear*, *endDateMonth*, *zombieCreationYear* y *zombieCreationMonth*. Esto genera un costo de procesamiento adicional al motor de ejecución, por lo tanto, nosotros proponemos no asignarlo a variables y utilizarlo directamente en las funciones de agregación u operaciones matemáticas que se estimen conveniente como se ve en la figura 4.10b. En consecuencia, al aplicar esta recomendación disminuiremos un buen margen de tiempo de ejecución y DB Hits.

```
20130101000000000/10000000000000   AS endDateYear,
20130101000000000/100000000000%100 AS endDateMonth

zombie.creationDate/10000000000000   AS zombieCreationYear,
zombie.creationDate/100000000000%100 AS zombieCreationMonth,

WITH
  country,
  zombie,
  12 * (endDateYear  - zombieCreationYear )
     + (endDateMonth - zombieCreationMonth)
     + 1 AS months,
  messageCount
```

(a) Segmento Código Original Consulta BI21

```
WITH
  country,
  zombie,
  12 * (20130101000000000/10000000000000   - zombie.creationDate/10000000000000 )
     + (20130101000000000/100000000000%100 - zombie.creationDate/100000000000%100)
     + 1 AS months,
  messageCount
```

(b) Segmento Código Optimizado Consulta BI21

Figura 4.10: Solución Propuesta “Propiedades Limitadas” Consulta BI21.

El plan de ejecución obtenido luego de aplicar la estrategia se puede ver en la Figura 4.11. Allí el punto crítico del plan de ejecución del conjunto original se redujo a cero, garantizando el correcto funcionamiento de la estrategia Propiedades Limitadas.

Como ya se había indicado previamente esta estrategia puede ser usada de forma complementaria a cualquier otra. La estrategia con la que más se combinó es la Materialización de Caminos en las consultas BI4, BI5, BI8, BI10, BI11, BI14, BI16, BI19, BI20, BI23 y BI24. Así mismo, complementamos la estrategia Consulta Mínima en la consulta BI7.

4.3.2. Materialización de caminos

La Materialización de Caminos se aplicó en la estrategia BI17 sobre los puntos críticos identificados. En la Figura 4.12 podemos ver un segmento del plan de ejecución de la consulta BI17 que se compone de los siguientes operadores:

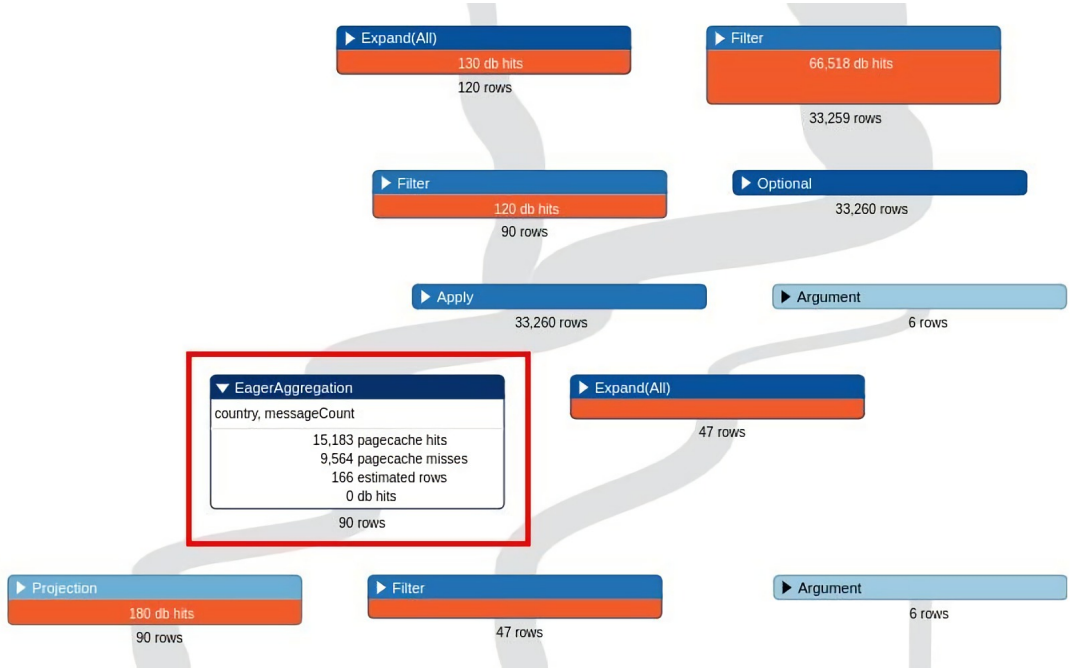


Figura 4.11: Plan de Ejecución Solución Propuesta Consulta BI21.

1. Un operador *scan* que realiza la búsqueda sobre la variable *b* para encontrar los nodos que vivan en el país asignado.

2. Tres operadores *Filter* en los identificadores de los nodos *a*, *b* y *c* para validar que se cumpla que ninguno de ellos se conoce.

3. Tres operadores *Expand* en la relación *KNOWS* que involucra a los nodos *Person*.

4. Un operador *NodeHashJoin* para la unión del resultado de las personas que vivan en el país indicado.

Los puntos críticos de la consulta se encuentran enmarcados en el plan de ejecución. Estos puntos críticos corresponden a la relación *KNOWS* que conecta a los nodos *a*, *b* y *c*. Hacemos esta distinción porque la consulta BI17 tiene un filtro inicial en el nodo *Country* seleccionando únicamente los nodos que se hallen en el país requerido. Esto es fundamental puesto que aquí se aprecia la diferencia de lo propuesto por M. Jota y R. Parra [37], en razón de que materializamos la relación *KNOWS* exclusivamente por encima de los nodos que cumplan con el requerimiento aminorando el conjunto de datos. Como mencionábamos existen dos formas de implementar esta estrategia. La primera es materializar una relación completa en la base de datos independiente de la consulta lo que nos permite reutilizar la relación creada pero asociado a un gran costo de almacenamiento.

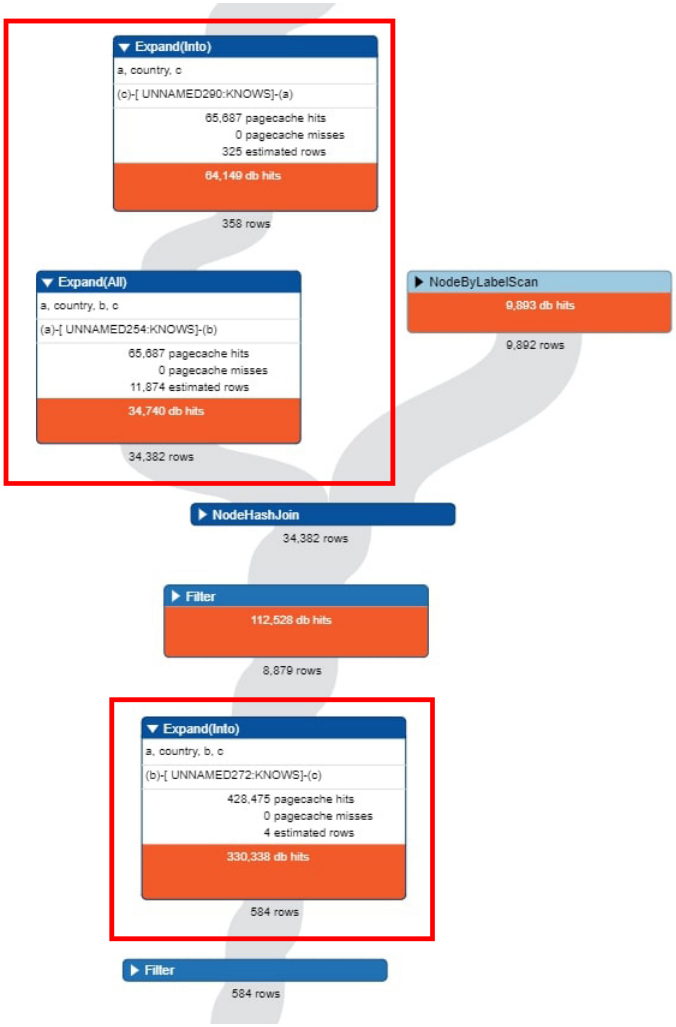


Figura 4.12: Segmento Plan de Ejecución Consulta BI17.

La segunda opción es la indicada en el párrafo anterior, cuya implementación es de forma personalizada con respecto a cada consulta obteniendo como resultado disminuir en un alto porcentaje los costos de almacenamiento ya que se aplican diferentes filtros si la consulta lo posee. Se recomienda esta forma de trabajar cuando existen restricciones de almacenamiento.

Analizando el plan de ejecución y la consulta podríamos haber aplicado la indexación en la propiedad *name* del país requerido, sin embargo, no es conveniente ya que el costo de la búsqueda de esos nodos es ínfimo. De igual forma, tenemos los nodos *Person* que al aplicar la indexación produce un enorme costo en contraste al beneficio que nos

puede entregar. Por otra parte, los nodos *City* son nodos intermedios donde su recorrido no representa un esfuerzo para el motor de ejecución. Habiendo fundamentado la decisión de inclinarnos por la segunda guía de diseño es necesario implementarla, esto se aprecia en la figura 4.13.

```
MATCH
  (a:Person)-[:KNOWS]->(b:Person)-[:IS_LOCATED_IN]->(:City)-[:IS_PART_OF]->(country:Country)
WHERE country.name = 'Spain'
CREATE (a)-[:BI17_person]->(b);
```

Figura 4.13: Solución Propuesta “Materialización de Caminos” Consulta BI17.

En la Figura 4.14 existen dos recuadros enmarcados en el segmento del plan de ejecución resultante de la aplicación la segunda guía de diseño. Allí se destacan los considerables descensos en los DB Hits del punto crítico e incluso disminuir el número de caminos de dos a uno. Adicionalmente, se decide reutilizar la relación materializada en los dos casos restantes consiguiendo de igual forma una reducción en las acciones de cada operador. Esto respalda una vez más la segunda guía de diseño, aunque es necesario recordar que esta estrategia no es implementada por Neo4j y en consecuencia debe ser la persona encargada la que deba actualizar la relación materializada.

Finalmente, la segunda guía de diseño se aplicó exclusivamente en las consultas BI17, BI18, BI22 y BI25. De igual forma, la Materialización de Caminos se puede combinar con cualquiera de las alternativas de la primera guía de diseño como se hizo con las consultas BI4, BI5, BI8, BI10, BI11, BI14, BI16, BI19, BI20, BI23 y BI24.

4.3.3. Creación de índice

La última estrategia aborda la consulta BI1 que cumple con los requisitos que fundamentan la decisión de su implementación. La Figura 4.15 posee los siguientes operadores que hace referencia a un segmento del plan de ejecución de la consulta BI1.

1. Dos *scans* sobre los nodos Message.
2. Dos operadores *Filter* sobre los nodos *Message* y su propiedad *creationDate*.
3. Dos operadores *EagerAggregation*, uno para convertir la variable *totalMessageCountInt* en un punto flotante y el segundo es para obtener el promedio, el número de punto flotante y la suma de longitudes pertenecientes al nodo *Message*.
4. Un operador *Apply* que realiza un *join* con el segundo *scan*.
5. Dos operadores *Projection*.

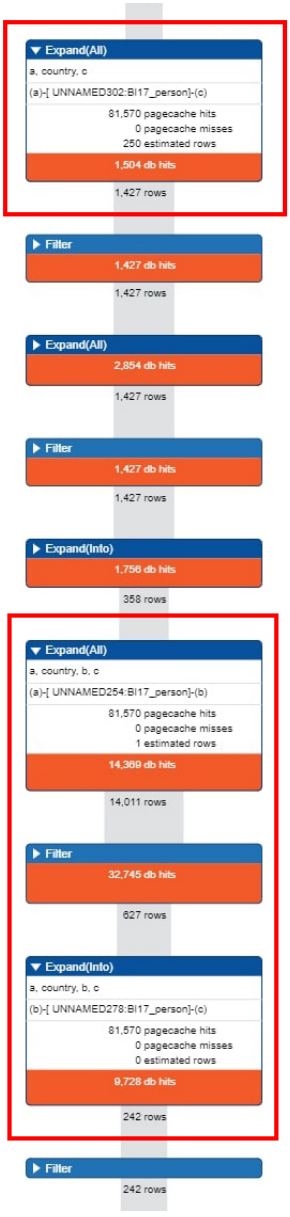


Figura 4.14: Segmento Plan de Ejecución Solución Propuesta Consulta BI17.

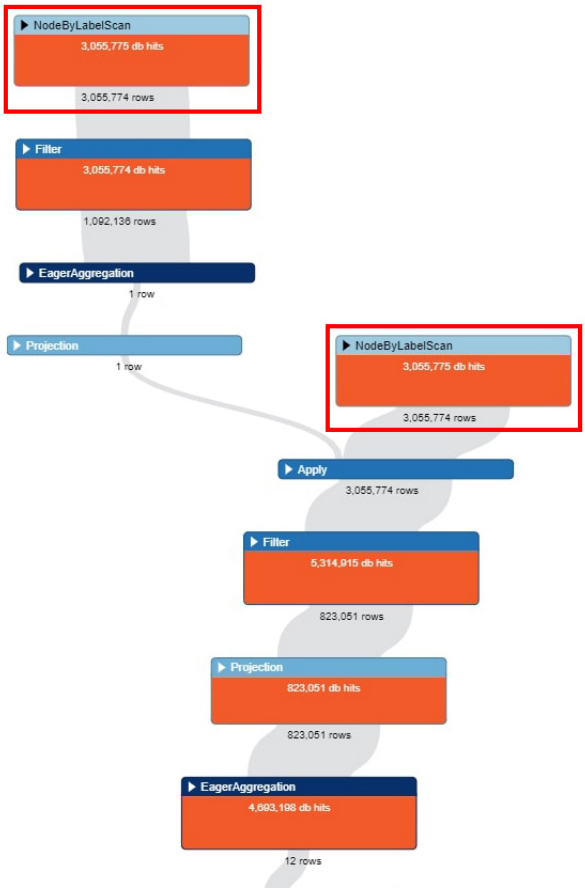


Figura 4.15: Segmento Plan de Ejecución Consulta BI1.

Esta estrategia fue aplicada sobre el identificador del nodo *Message* definido por LDBC SNB ya que la estructura de la consulta no permite la aplicación de otra guía de diseño. Se analizó la idea de aplicar alguna de las estrategias de Reescritura de Consulta pero en la consulta sólo se trabajan funciones de operaciones de agregación. En segunda instancia, se investigó la posibilidad de implementar la estrategia de Materialización de Caminos, sin embargo, no existe alguna relación en la cual aplicar dicha estrategia. Por lo tanto, en la Figura 4.16 podemos apreciar la consecuencia del uso de la tercera guía de diseño donde no sólo redujo el costo de procesamiento en un tercio sino que también eliminó ambos filtros. Producto de lo anterior se pudo cumplir con el objetivo logrando además una disminución de DB Hits indirectamente.

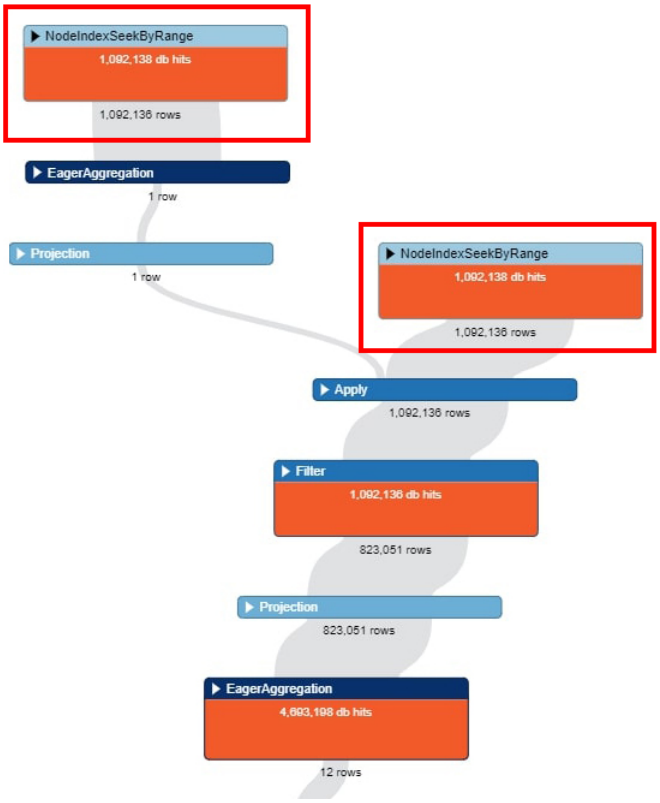


Figura 4.16: Segmento Plan de Ejecución Solución Propuesta Consulta BI1.

Capítulo 5

Estudio experimental

En este capítulo se presenta el plan de pruebas y su ejecución. Esto contempla la ejecución de las consultas optimizadas con diferentes volúmenes de datos con el fin de estudiar el impacto de la optimización a medida que se van escalando las cargas de trabajo. Una vez definidas las estrategias que se aplican por consulta se procede a hacer la toma de datos por cada carga de trabajo correspondientes a 1 GB, 10 GB y 100 GB, con el fin de medir la escalabilidad mediante una diferencia equivalente entre las cargas de trabajo. Es necesario definir dos conceptos que se pueden presentar en los experimentos puesto que pueden surgir inconvenientes producto de las restricciones de memoria con las que estamos trabajando en el servidor. El primer concepto es el *buffer overflow* o *desbordamiento de búfer* que aparece cuando existe una reescritura de espacios adyacentes en la memoria [50]. El segundo concepto corresponde *timeout*, este consiste en definir un tiempo límite de ejecución para cada consulta como también para cada estrategia implementada.

5.1. Configuración de experimentos

En este punto describiremos las configuraciones que se debieron hacer al servidor para poder montar el ambiente de pruebas. El plan de pruebas se establece debido a la necesidad de corroborar la escalabilidad de las guías de diseño y sus respectivas estrategias. De esta forma podemos aconsejar el uso de estas estrategias en diferentes volúmenes de datos. Para finalizar, esta sección se divide en la metodología que se usó, la definición de las cargas de trabajo y el ambiente experimental donde se desarrolló la implantación.

5.1.1. Metodología

En primera instancia procedemos a configurar el ambiente pruebas, el sistema operativo definido para esto es Microsoft Windows 10 Pro. Con el fin de evitar caer en la problemática del *buffer overflow* [50] cogemos la opción de memoria virtual asignándole

un parámetro de 100 GB. Una vez configurado el ambiente de pruebas debemos generar las cargas de trabajo. Para ello, se debe tener en cuenta que el Datagen está desarrollado en un ambiente basado en Linux, por esta razón, montamos el sistema operativo Ubuntu 18.04 LTS con el único fin de generar y convertir las cargas de trabajo a Cypher como lo indica LDBC SNB [22]. Una vez obtenidas las cargas de trabajo procedemos a instalarlas en los contenedores generados en Neo4j Desktop para luego iniciar cada contenedor validando así que se haya cargado correctamente a la aplicación. Finalmente, se procede a efectuar las pruebas con las diferentes cargas de trabajo. Inicialmente deberemos ejecutar la consulta respectiva con la memoria caché en frío, lo que quiere decir que se ejecuta por primera vez puesto que el motor de ejecución nos genera un costo adicional al producir el plan de ejecución. Desde la segunda iteración en adelante este costo ya no se considera. Desde ahí, debemos capturar las estadísticas de cada consulta con las iteraciones definidas por cada carga de trabajo. Recordar que las iteraciones definidas para las cargas de trabajo son 25, 10 y 5 iteraciones, respectivamente. Esta decisión se justifica dado el número de consultas a correr y sus relativos tiempo de ejecución, considerando que algunas de ellas presentan un tiempo de ejecución excesivamente más alto que el tiempo promedio. Esto último producto de que la ejecución de algunas consultas exceden el día de ejecución con cargas de trabajo de 10 y 100 GB, sumándole además sus respectivas iteraciones. Una vez registrados los tiempos de ejecución y DB Hits de la consulta sin diseño físico, llevaremos a cabo la aplicación de la estrategia de diseño capturando las estadísticas de almacenamiento previo y posterior para conseguir el costo de almacenamiento de su implementación. Cuando ya hayamos implementado nuestra estrategia deberemos reiniciar la base de datos con el objetivo de limpiar la memoria caché y así poder volver a capturar los datos de la consulta optimizada. Una vez apuntados los datos de la consulta optimizada tendremos que eliminar la estrategia implementada y nuevamente reiniciar la base de datos. Ya habiendo terminado el proceso de captura de datos de la consulta correspondiente deberemos repetir dicho proceso con la consulta siguiente y así sucesivamente hasta terminar el registro de datos. Debemos indicar que el tiempo límite definido es de 86,400,000 ms (24 horas) puesto que se utiliza el mismo fundamento para definir el número de iteraciones. Además, se debe considerar que en el peor de los casos el tiempo de ejecución de una consulta por carga de trabajo puede llegar a ser de 24 horas y dependiendo de la cantidad de iteraciones puede llegar a ser una restricción en cuanto a los tiempos establecidos para efectuar el registro total de datos. Todo lo antes descrito se puede observar de mejor forma con las etapas definidas en la Figura 5.1.

Figura 5.1: Esquema Metodología Estudio de Escalabilidad.

5.1.2. Conjunto de datos y consultas

La Tabla 5.1 cuya primera columna hace referencia al conjunto de datos generados en Datagen, después contamos en la segunda columna el tamaño inicial de la base de datos en Neo4j, seguidamente disponemos de la tercera columna que contiene los costos asociados de implementar las estrategias de diseño físico, por último, en la cuarta columna se puede apreciar el tamaño final de la base de datos.

Conjunto	Estado inicial (GB)	Implementación Diseño Físico (GB)	Total (GB)
1 GB	1.3	0.79	2.09
10 GB	13.09	9.82	22.91
100 GB	130.66	4.7	135.36

Tabla 5.1: Espacio ocupado por la base de datos para cada tamaño del conjunto de datos generado por DATAGEN.

Al momento de ser incorporado el conjunto de datos de 1 GB a Neo4j Browser, este escala a 1.3 GB como se indica en la columna Estado Inicial de la Tabla 5.1. Esto se produce debido a que existe un costo adicional al momento de cargar el volumen de datos al contenedor de Neo4j Desktop. Este costo depende exclusivamente del volumen de datos que se desea incorporar al SGBD. Al implementar las diferentes estrategias de diseño físico en la base de datos nos genera un costo de almacenamiento de 0.79 GB entregándonos un tamaño total de la base de datos de 2.09 GB. De igual forma, se realiza este procedimiento en el conjunto de datos de 10 GB que al ser incorporado a Neo4j Browser asciende su dimensión a 13.09 GB. La implementación de las estrategias de diseño físico nos originan un costo de almacenamiento de 9.82 GB dejando así una medida total de la base de datos de 22.91 GB. Finalmente, al momento de posicionar la carga de trabajo de 100 GB en la plataforma nos arroja un valor total de 130,66 GB. Una vez cargados los registros se procede a implementar las estrategias teniendo un costo de almacenamiento de 4.7 GB entregándonos es esta forma un total de 135.36 GB.

Llama la atención el bajo costo de implementación de estrategias de diseño físico en la carga de trabajo de 100 GB. Esto se explica ante la falta de 9 consultas que poseen el mayor costo de implementación, debido a que excedieron el límite de tiempo de ejecución cayendo en el antes mencionado *timeout*.

5.1.3. Ambiente experimental

El estudio experimental se desarrolló en un servidor de la universidad que cuenta con un procesador Intel(R) Core(TM) i7-8700 CPU que se compone por 6 procesadores principales y 12 procesadores lógicos, además posee 32 GB de memoria RAM, un disco

duro principal de 256 GB SSD y un disco duro secundario de 4 TB HDD. Ante la restricción de memoria RAM se decide implementar memoria virtual asignándole un valor de 100 GB para de esta forma poder abarcar la mayor cantidad de consultas.

El sistema operativo utilizado es Microsoft Windows 10 Pro donde se instaló Neo4j Desktop 1.2.9 y a su vez se configuró Neo4j Browser 3.5.18 en el disco secundario. En consecuencia, las bases de datos fueron almacenadas en las carpetas de datos de cada contenedor de Neo4j Desktop en el disco secundario.

Cada base de datos establecida en la plataforma fue configurada con las recomendaciones que indica Neo4j. Allí se nos presentan tres variables de configuración *initial heap* y *maximum heap size* que corresponde a la pila de almacenamiento dinámico, por otro lado, tenemos la variable *pagecache* que se usa para almacenar en caché los datos de Neo4j y los índices nativos. Según las recomendaciones de Neo4j indica que estas dos primeras se deben configurar con el mismo valor que corresponde a 44,4 GB. Debido a la restricción de memoria se asigna el valor antes mencionado a la tercera variable.

Capítulo 6

Análisis y comparación de resultados

En este capítulo haremos un minucioso análisis a los resultados obtenidos por cada carga de trabajo para finalmente comparar los resultados obtenidos de los experimentos descritos en el capítulo anterior. De igual forma, validaremos mediante pruebas estadísticas la exactitud de las estrategias de diseño físico planteadas en las cargas de trabajo de 1 GB, 10 GB y 100 GB. Cabe mencionar que en este capítulo nos encontramos con dos tipos de gráficos: el primero expone los porcentajes de mejora, cuyo cálculo se basa en la diferencia relativa porcentual entre el valor sin diseño físico y el valor con diseño físico. Por otro lado, el segundo gráfico presenta la comparación los valores sin diseño físico y los valores con diseño físico en una escala logarítmica permitiéndonos visualizar de mejor forma los resultados.

6.1. Impacto del diseño físico para una base de datos de 1 GB

En la Figura 6.1 podemos apreciar los resultados de la primera métrica definida que corresponde al tiempo de ejecución. Esta gráfico está compuesto en el eje de las abscisas con la especificación de las consultas que llevan por nombre el acrónimo BI acompañado del número de la consulta y en el eje de las ordenadas encontramos el porcentaje de mejora en relación al tiempo de ejecución. Recordemos que en la carga de trabajo de 1 GB se emplean 25 iteraciones en la captura de datos. A primera vista se puede notar que las consultas que más sobresalen son las consultas BI10, BI11, BI12, BI22 y BI25, todas estas están sobre el 95 % de optimización. En contraste las consultas que presentan un menor porcentaje de mejora son las consultas BI1, BI2, BI6, BI13 y BI24.

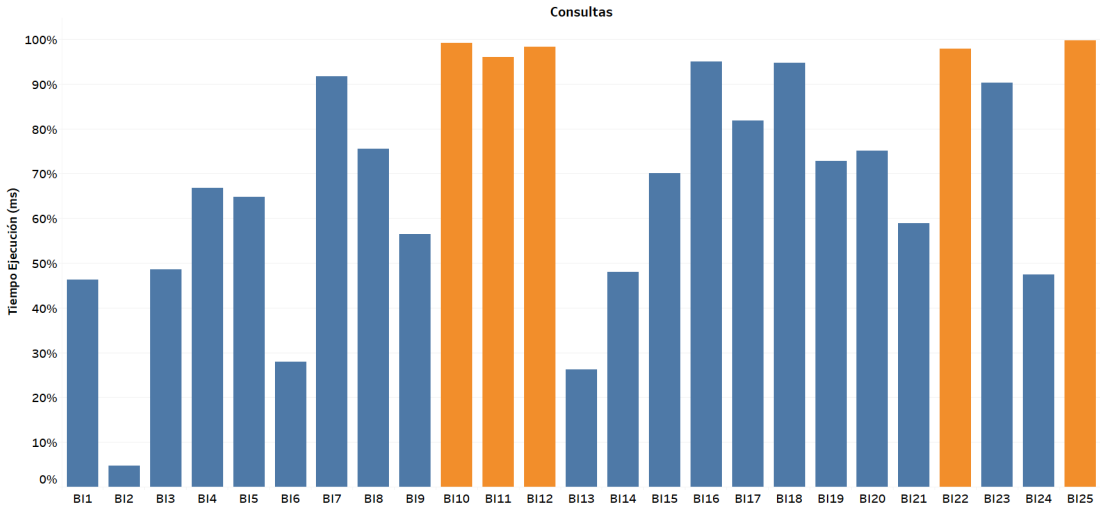


Figura 6.1: Tiempo de ejecución por consulta de una base de datos de 1 GB.

Consulta	BI1	BI2	BI3	BI4	BI5	BI6	BI7	BI8	BI9	BI10	BI11	BI12	BI13
Tiempo Ejecución (%)	46,34	4,76	48,64	66,78	64,82	27,89	91,68	75,54	56,42	99,16	95,96	98,34	26,23

Tabla 6.1: Porcentaje de mejora sobre valores sin diseño físico en el tiempo de ejecución para las consultas BI1 - BI13 de una base de datos de 1 GB.

Consulta	BI14	BI15	BI16	BI17	BI18	BI19	BI20	BI21	BI22	BI23	BI24	BI25
Tiempo Ejecución (%)	48,00	70,08	94,99	81,86	94,68	72,80	75,05	58,86	97,80	90,33	47,49	99,76

Tabla 6.2: Porcentaje de mejora sobre valores sin diseño físico en el tiempo de ejecución para las consultas BI14 - BI25 de una base de datos de 1 GB.

En las Tablas 6.1 y 6.2 tenemos el detalle de los porcentajes de mejora sobre los valores sin diseño físico con respecto al tiempo de ejecución. A partir de esto podemos hacer un análisis a cada consulta para entender su conducta y por ende los resultados obtenidos. Como mencionábamos anteriormente, las consultas que más se destacan son BI10, BI11, BI12, BI22 y BI25, estas 5 consultas implementan la estrategia de Materialización de Caminos con una salvedad, puesto que en la consulta BI10 y BI11 se complementa la estrategia con otra perteneciente a la primera guía de diseño. Estas consultas poseen una combinación de estrategias entre la Materialización de Caminos y la subtécnica de Propiedades Limitadas. Así mismo se evidenció este similar comportamiento entregando un rango entre el 66,78 % y 94,68 % en las consultas BI4, BI8, BI15, BI16, BI17, BI18, BI19, BI20 y BI23. Estas consultas también trabajan en su mayoría con la combinación de las estrategias referida recientemente, exceptuando las consultas BI15, BI17 y BI18. En

estas consultas surgen dos casos bien particulares. En la consulta BI15 y BI17 se implementa la materialización sobre una relación que se vuelve a utilizar en dos oportunidades, por lo tanto, con sólo materializar un vínculo podemos reutilizar la nueva relación creada consiguiendo un porcentaje de mejora sobre valores sin diseño físico de un 70,08 % y 81,86 %. En la consulta BI18 encontramos que la implementación de la Materialización de Caminos nos permite eliminar condiciones las cuales acceden en su mayoría a las propiedades del nodo *Message*. Este nodo corresponde a uno de los más complejos de manipular, sin embargo, tenemos la opción mediante esta estrategia de reducir el número de relaciones de dos a tan sólo una obteniendo un porcentaje de optimización de 94,68 %. En la consulta BI19 encontramos la misma situación de la consulta previa, con esto nos referimos a la posibilidad de eliminar condiciones respecto a las propiedades del nodo *Person*, igualmente se establecen dos consultas en dos condiciones cuyo objetivo es la búsqueda de los nodos *Person* y *Message*. Por último, se exponen las consultas que no entregaron un buen retorno haciendo exclusiva alusión a las consultas BI3, BI5, BI6, BI13, BI14 y BI24, el rango de mejora sobre valores sin diseño físico de estas consultas fluctúa entre 26,23 % y 66,78 % teniendo una tendencia hacia el extremo inferior como se aprecia en las Tablas 6.1 y 6.2. Se justifica la conducta dado que se aplica la Materialización de Caminos sobre la sentencia *OPTIONAL MATCH*, que cumple con el mismo objetivo que la función *MATCH* con la gran diferencia que si no se encuentran coincidencias se asignará el valor *null*. Es por esto que el conjunto de datos a encontrar es mucho mayor que si se implementara únicamente encima de las sentencias *MATCH*. Lo anterior obliga a que el motor de ejecución deba buscar los nodos y propiedades restantes generando un costo de ejecución extra, no obstante, de todos modos se consigue un buen porcentaje de mejora. Con las consultas ya analizadas se puede apreciar que esta estrategia funciona adecuadamente, añadiendo que la combinación con la segunda técnica de la primera guía de diseño nos entrega un porcentaje adicional de mejora. Finalmente, en la consulta BI9 tenemos una condición que se basa en contar la cantidad de nodos *Person* y validar, provocando un mayor costo de procesamiento para el motor de ejecución, sin embargo, mediante la implementación de esta estrategia conseguimos eliminar este paso accediendo directamente a los nodos ya filtrados.

La consulta BI7 es un caso a destacar dado que se obtuvo un 92,73 % de optimización usando sólo las estrategias de Reescritura de Consulta, estas son la Consulta Mínima y la de Propiedades Limitadas. Lo anterior se desprende del beneficio de reutilizar una variable, incluso en esas dos líneas de código se manipulan los nodos con mayor presencia en la base de datos que corresponden a los nodos *Person* y *Message*. Por otra parte, el uso de la estrategia Propiedades Limitadas ha sido de gran aporte como ya hemos analizado, pero la consulta BI21 usa únicamente esta técnica en la cual se eliminan los campos cálculos que posteriormente son utilizados en variables por establecer el cálculo directamente donde se le requiera consiguiendo de esta forma un porcentaje de mejora sobre los valores

sin diseño físico de un 58,86 %. Finalmente, la consulta BI2 nos da como resultado un porcentaje de optimización de 4,76 % siendo el menor. En esta consulta se implementa la estrategia de Descomposición de Consultas, debemos recordar que la primera estrategia reduce el conjunto de datos disminuyendo el número de operaciones de agregación que se deben efectuar.

Por último, en la consulta BI1 se aplica la conocida estrategia de indexación sobre el nodo *Message* y su propiedad *creationDate* con la finalidad de hacer un filtro sobre la fecha de creación de los mensajes. Al hacer uso de esta estrategia reducimos a un tercio el costo de búsqueda de estos nodos y así mismo la operación de depurar los datos obteniendo un porcentaje de 46,34 %.

Observando la Figura 6.2 nos surge una pregunta ¿Por qué en algunas consultas se aprecia una mayor diferencia? Después de un profundo análisis nos damos cuenta que este factor está sujeto a la estrategia que se utiliza. Esto quiere decir que vemos un mayor beneficio en las estrategias que implementan la segunda guía de diseño, ya que al eliminar nodos y relaciones en la consulta beneficia al motor de ejecución puesto que debe efectuar menos operaciones para encontrar el resultado. Una excepción a lo antes indicado es la consulta BI7 que utiliza la estrategia Consulta Mínima donde al reutilizar una variable nos permite eliminar una linea de código completa lo que produce el mismo efecto mencionado previamente justificando los resultados de esa consulta. Salvo el caso anterior, la primera y tercera guía de diseño no debieran presentar resultados con una gran diferencia debido a que la tercera guía de diseño propone establecer una copia de datos con el fin de hacer más eficiente la búsqueda lo que significa que los operadores reducen considerablemente el alto costo de DB Hits que presentaban en el plan de ejecución inicial. Con respecto a la optimización de primera guía de diseño siempre dependerá con la cantidad de registros con los que se trabaje, sin embargo, en la mayoría de las consultas entregadas por LDBC SNB se trabaja con un bajo número de registros.

Una vez analizados los resultados es necesario validar estadísticamente que los tiempos de ejecución sobre nuestra base de datos con diseño físico representen una mejora con respecto a los tiempos de ejecución pertenecientes a la base de datos que no posee las estrategias de diseño físico. En primera instancia debemos ratificar que ambos tiempos de ejecución siguen una distribución normal, por lo tanto, llevaremos a cabo la prueba de *Shapiro-Wilk* [51]. Ambos autores consideran que la muestra sigue una distribución normal cuando se cumple que el *p-valor* sea superior a 0.05. Según los resultados de la aplicación de esta prueba que se encuentran en la Tabla 6.3, se puede concluir que según las muestras hay resultados en los que no se puede validar la conjetura de una distribución normal debido a valores inferiores 0.05.

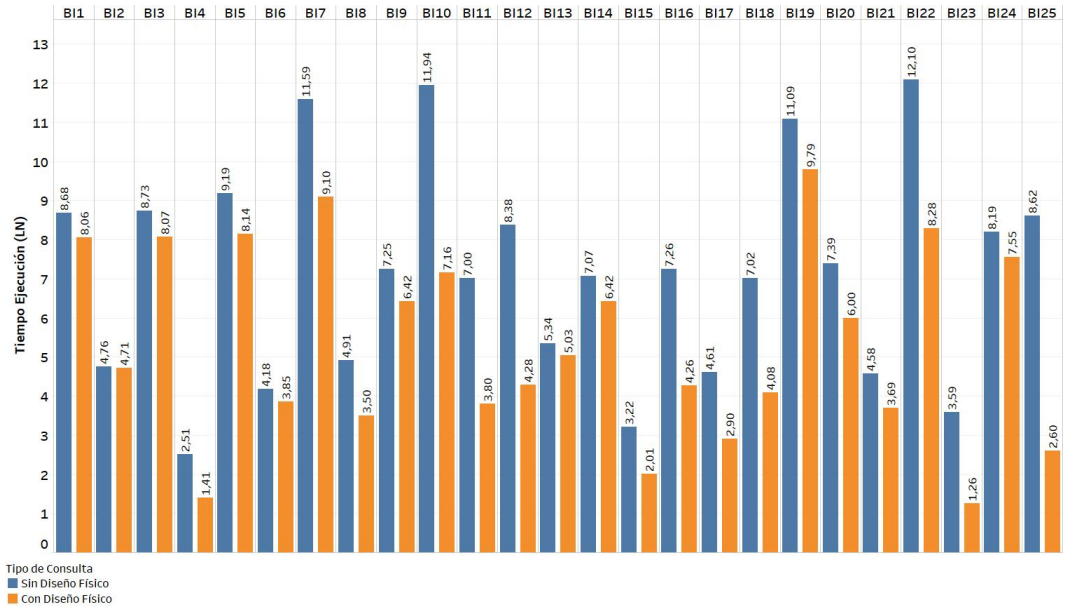


Figura 6.2: Tiempo de ejecución en escala logarítmica por cada consulta de una base de datos de 1 GB.

Consulta	p-valor (original)	p-valor (diseño físico)
BI1	0.218	0.6483
BI2	**3.78x10^{-6}**	**1.25x10^{-5}**
BI3	0.5586	**4.58x10^{-3}**
BI4	**1.87x10^{-4}**	**4.47x10^{-4}**
BI5	0.7952	**5.13x10^{-3}**
BI6	**2.85x10^{-7}**	**9.92x10^{-3}**
BI7	**0.03439**	0.09587
BI8	**2.36x10^{-3}**	**0.01625**
BI9	0.09111	**4.49.x10^{-7}**
BI10	0.6551	0.07871
BI11	0.05071	**9.62x10^{-5}**
BI12	0.0971	**4.63x10^{-4}**
BI13	**6.80x10^{-4}**	**1.92x10^{-4}**
BI14	0.09982	**1.77x10^{-3}**
BI15	**6.36x10^{-3}**	**4.05x10^{-6}**
BI16	**0.00143**	**0.02321**
BI17	**2.14x10^{-4}**	**1.06x10^{-3}**
BI18	0.5242	**3.33x10^{-4}**
BI19	**0.01486**	**0.01307**

Consulta	p-valor (original)	p-valor (diseño físico)
BI20	0.9565	0.3918
BI21	0.1157	**4.55x10^{-5}**
BI22	**1.61x10^{-3}**	**0.04222**
BI23	**3.96x10^{-7}**	**2.85x10^{-4}**
BI24	**3.21x10^{-4}**	**3.20x10^{-4}**
BI25	0.5799	**2.72x10^{-7}**

Tabla 6.3: *p-valor* obtenido de la prueba Shapiro - Wilk

Producto de tener resultados que no nos permiten verificar la hipótesis de una distribución normal nos vemos en la necesidad de realizar otra prueba que no requiera esta condición. La prueba *Wilcoxon signed-rank paired* [52] no requiere esa condición puesto que es una prueba no paramétrica que permite comparar los promedios de los tiempos de ejecución de todas las consultas correspondientes a la base de datos sin diseño físico (μ_0) y la base de datos con diseño físico (μ_1). El supuesto que se desea probar es $\mu_0 > \mu_1$. Luego de haber aplicado esta prueba se obtiene un *p-valor* de 5,96x10^{-8} con un nivel de significancia de 0.05, lo que nos permite aseverar que $\mu_0 > \mu_1$.

Finalmente, se puede confirmar en base a pruebas estadísticas que los tiempos obtenidos sobre la base de datos sin diseño físico son considerablemente mayores que los tiempos conseguidos de la base de datos con diseño físico sobre un conjunto de datos de 1 GB.

De la misma forma en que se exponen los resultados según el porcentaje de mejora en el tiempo de ejecución se hace para los resultados de DB Hits. La Figura 6.3 se compone en el eje de las abscisas por las consultas puestas a prueba que llevan por nombre el acrónimo BI acompañado del número de la consulta. Por otro lado, en el eje de las ordenadas encontramos el porcentaje de mejora con referencia al DB Hits. En la figura 6.3 podemos percibir que todas las consultas consiguieron un buen registro lo que pasaremos a analizar a continuación.

En las Tablas 6.4 y 6.5 se pueden ver los porcentajes de mejora en los DB Hits, ahí se destaca el porcentaje de la consulta BI6 con un 37,58 %, se fundamenta producto de una gran cantidad de operaciones de agregación incluyendo una fórmula que multiplica cada operación de agregación para finalmente sumarlas. Esto último también se aprecia en las consultas BI2, BI9, BI13 y BI21, destacando las consultas BI13 y BI21 con un gran número de operaciones de agregación y operaciones aritméticas generando un mayor costo de procesamiento. De igual forma, se comporta la consulta BI2 que implementa la estrategia de Descomposición de Consulta. Su comportamiento se explica en el conjunto de datos con el que trabaja ya que su fundamento es reducir el conjunto de datos inicial

a medida que avance la consulta. Por lo anterior, se entiende al contar con un menor conjunto de datos es menor el impacto que puede generar el uso de esta estrategia. El resto de las consultas posee un porcentaje de mejora promedio sobre los valores sin diseño físico de un 78,91 % esto se explica dado que las estrategias de diseño físico que fueron aplicadas logran eliminar operaciones de agregación, operaciones aritméticas y condiciones reduciendo considerablemente las acciones que debe efectuar el motor de ejecución. Adicionalmente, destacan las consultas BI10 y BI25 por su gran porcentaje de mejora sobre los valores sin diseño físico correspondientes a 99,58 % y 99,82 %, respectivamente. En la primera consulta se aplica la estrategia de Materialización de Caminos sobre una función *length*. Esta obtiene los saltos de distancia entre dos nodos, de manera que al aplicar esta estrategia conseguimos reducir las acciones que debe efectuar el motor de ejecución para obtener los saltos de distancia pertenecientes a cada nodo. Así mismo, en la consulta BI25 se aplica esta estrategia encima de cuatro operadores *Joins* consiguiendo establecer una relación directa entre ambos extremos de esas consultas logrando eliminar los operadores *Joins* y de esta forma disminuir esas dos consultas a tan sólo una.

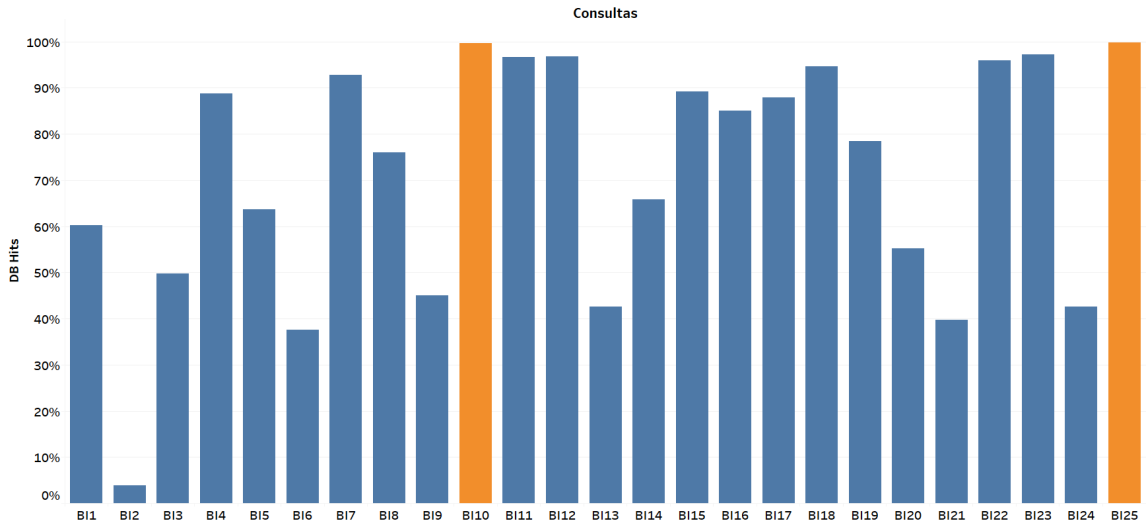


Figura 6.3: DB Hits por cada consulta en una base de datos de 1 GB.

Consulta	BI1	BI2	BI3	BI4	BI5	BI6	BI7	BI8	BI9	BI10	BI11	BI12	BI13
DB Hits (%)	60,15	3,89	49,81	88,75	63,66	37,58	92,73	75,95	44,98	99,58	96,69	96,84	42,51

Tabla 6.4: Porcentaje de mejora sobre valores sin diseño físico en los DB Hits para las consultas BI1 - BI13 de una base de datos de 1 GB.

Consulta	BI14	BI15	BI16	BI17	BI18	BI19	BI20	BI21	BI22	BI23	BI24	BI25
DB Hits (%)	65,87	89,24	84,95	87,92	94,56	78,42	55,15	39,66	95,95	97,16	42,60	99,82

Tabla 6.5: Porcentaje de mejora sobre valores sin diseño físico en los DB Hits para las consultas BI14 - BI25 de una base de datos de 1 GB.

Podemos desprender de la Figura 6.4 que no existe una diferencia notoria entre las consultas que su punto crítico se encuentra en una relación o nodo y las consultas cuyo punto crítico se ubica en las operaciones de agregación y operaciones aritméticas. Desde ese punto de vista se fundamenta, ya que ambas tienen un punto en común que corresponde al acceso a las propiedades tanto de los nodos como las relaciones. Sin embargo, la diferencia de la segunda guía de diseño es la eliminación del acceso a las propiedades producto de establecer relaciones directas entre los nodos, no así las otras guías de diseño que establecen una consulta eficiente pero manteniendo el acceso a las propiedades. También se aprecia una tendencia que al disminuir el número de DB Hits se reduce el tiempo de ejecución, no obstante, debemos verificar que esto se replique en las cargas de trabajo restante.

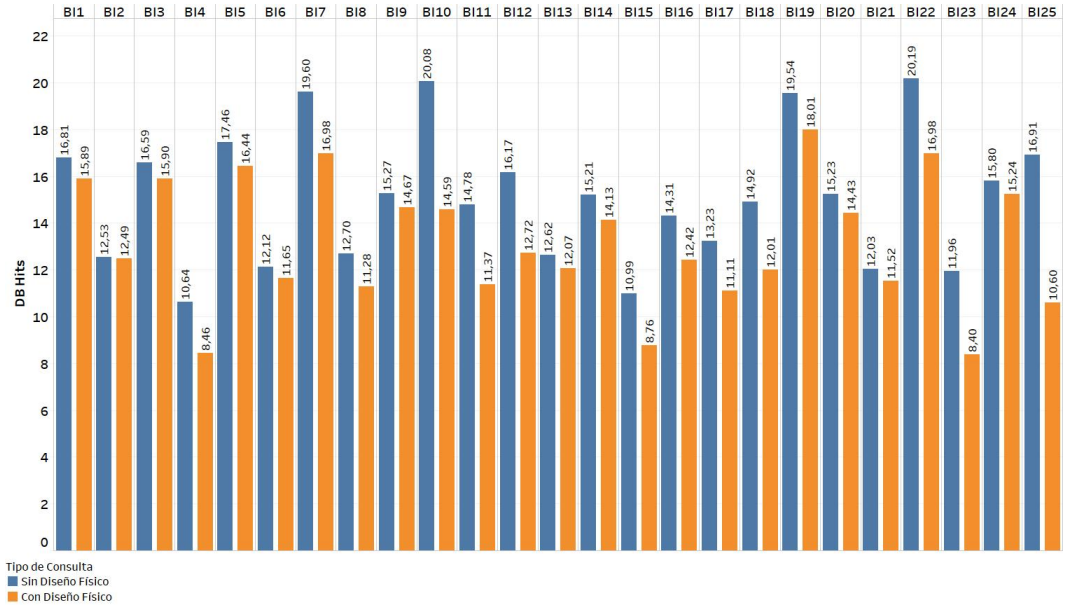


Figura 6.4: DB Hits en escala logarítmica por cada consulta de una base de datos de 1 GB.

6.2. Impacto del diseño físico para una base de datos de 10 GB

En la Figura 6.5 podemos apreciar los resultados de los experimentos con respecto al tiempo de ejecución sobre la carga de trabajo de 10 GB, cabe mencionar que para este

experimento se efectuaron 10 iteraciones para la captura de datos. La Figura 6.5 se compone en el eje de las abscisas por las consultas puestas a prueba que llevan por nombre el acrónimo BI acompañado del número de la consulta. Por otro lado, en el eje de las ordenadas encontramos el porcentaje de mejora con referencia al tiempo de ejecución. Analizando la Figura 6.5 nos podemos dar cuenta en primera instancia que las consultas que más destacan son las BI10, BI12, BI22, BI23 y BI25, siendo nuevamente esta última la que cuenta con el porcentaje de mejora más alto. En cambio, las consultas en las que se aprecia un menor porcentaje de optimización son BI1, BI2, BI6, BI13 y BI24, aunque nos podemos dar cuenta que con la excepción de estas 4 consultas el resto se encuentra por sobre el 40 % de mejora sobre valores sin diseño físico.

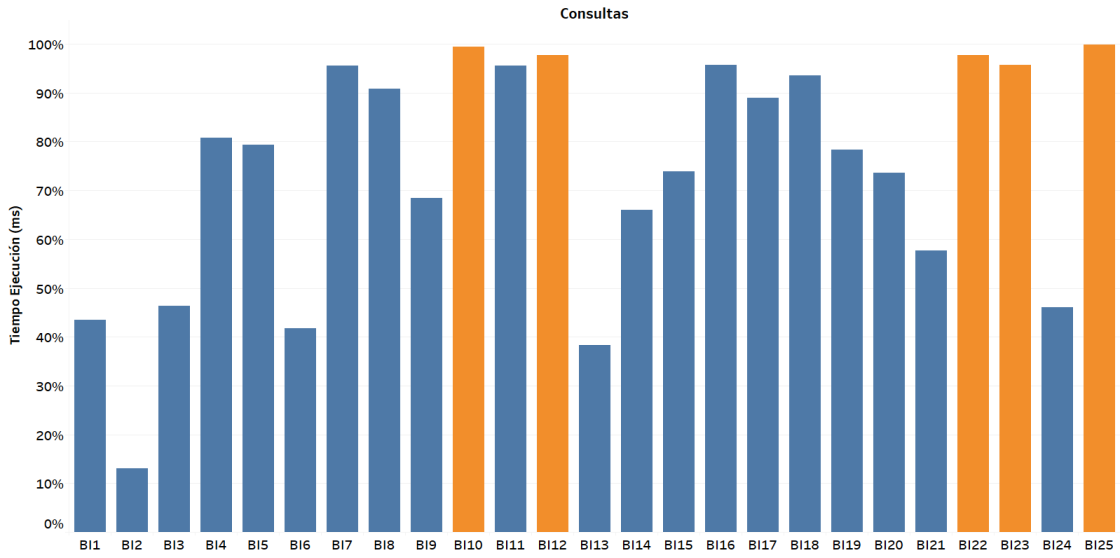


Figura 6.5: Tiempo de ejecución por cada consulta de una base de datos de 10 GB.

Consulta	BI1	BI2	BI3	BI4	BI5	BI6	BI7	BI8	BI9	BI10	BI11	BI12	BI13
Tiempo Ejecución (%)	43,53	13,03	46,39	80,85	79,32	41,79	95,53	90,87	68,39	99,44	95,53	97,64	38,31

Tabla 6.6: Porcentaje de mejora sobre valores sin diseño físico en el tiempo de ejecución para las consultas BI1 - BI13 de una base de datos de 10 GB.

Consulta	BI14	BI15	BI16	BI17	BI18	BI19	BI20	BI21	BI22	BI23	BI24	BI25
Tiempo Ejecución (%)	65,97	73,84	95,67	89,03	93,49	78,40	73,55	57,65	97,75	95,71	35,63	99,89

Tabla 6.7: Porcentaje de mejora sobre valores sin diseño físico en el tiempo de ejecución para las consultas BI14 - BI25 de una base de datos de 10 GB.

En las Tablas 6.6 y 6.7 podemos ver con mayor detalle los diferentes porcentajes de mejora sobre los registros sin diseño físico de cada consulta. En primer lugar analizaremos las consultas que usan exclusivamente la estrategia Materialización de Caminos y también la combinación con la técnica Propiedades Limitadas. Allí sobresalen las consultas BI10, BI12, BI22, BI23 y BI25, al comparar estos resultados con los obtenidos en la carga de trabajo de 1 GB nos damos cuenta que no sufrieron un cambio significativo, incluso las consultas variaron en un margen de tan sólo 1 % con excepción de la consulta BI23 que incrementó su porcentaje de mejora sobre los valores sin diseño físico en un 5,38 %. De igual forma debemos destacar el aumento de las estadísticas de las consultas BI4, BI5, BI8, BI9, BI14, BI16, BI17 y BI19. En estas consultas existen variados porcentajes que se distinguen como lo son las consultas BI4, BI5, BI8 y BI14. Esto debido a que consiguen un incremento sobre los valores sin diseño físico de un 14,07 %, 14,5 %, 15,33 % y 17,97 %, respectivamente. Lo anterior se explica mediante el aumento de la cantidad de registros generando un mayor impacto de la estrategia Propiedades Limitadas. Adicionalmente, las consultas restantes de este grupo tienen un variado proceder como la consulta BI9 que entrega un porcentaje de mejora sobre el valor sin diseño físico de un 11,97 %, en cambio la consulta BI16 arroja un incremento sobre el registro sin diseño físico de un 0,68 %. Sin embargo, debemos recordar que esta última consulta es una de las que posee un mayor porcentaje de mejora en el tiempo de ejecución en la carga de trabajo anterior. Igualmente, las consultas BI3, BI11, BI18 y BI20 presentan un pequeña disminución en sus porcentajes, pero tenemos que hacer hincapié en que esta diferencia es mínima puesto que corresponden a un 2,25 %, 0,43 %, 1,19 % y 1,5 %, manteniendo sus porcentajes de optimización por sobre un 73,55 % con excepción de la consulta BI3 que se encuentra en un 46,39 % sobre el resultado sin diseño físico. Para finalizar tenemos las consultas que exhibieron menor porcentaje de optimización, estas son BI6, BI13 y BI24. Si bien siguen siendo las consultas con menor porcentaje de mejora tenemos que enfatizar que todas aquellas mostraron un incremento considerable. Desde otro punto de vista las consultas BI6 y BI13 fueron las que tuvieron un gran incremento en sus valores obteniendo un 13,9 % y 12,08 %, respectivamente. La justificación para este comportamiento es que ambas consultas tienen un gran número de operaciones de agregación y operaciones aritméticas en su estructura. De estas dos consultas se destaca la consulta BI6 debido a que esas operaciones se encontraban situadas en la sentencia *RETURN* provocando un alto costo de procesamiento.

Las estrategias relacionadas con la Reescritura de Consulta tuvieron un buen desempeño manteniendo o incluso incrementando sus porcentajes de mejora. La primera consulta que abordaremos trabaja con la estrategia Consulta Mínima, esta corresponde a la consulta BI7 que aumento su porcentaje de mejora en un 3,85 % llegando de esta forma a una cifra de un 95,53 %. De la misma manera se comporta la consulta BI2 que incrementa su porcentaje con respecto al experimento previo desde un 4,76 % a un 13,03 %

consiguiendo de esta forma un aumento de un 8,27 %. Por último, tenemos la consulta BI21 que obtuvo una pequeña disminución en su porcentaje alcanzando una diferencia de un 1,21 % dejándonos así un valor referencial de 57,65 %.

Finalmente, la consulta BI1 que trabaja con la estrategia de indexación arrojó un resultado ligeramente negativo dado que disminuyó su porcentaje de mejora en un 2,81 %, de todas formas no es un valor significativo debido a que obtuvo un porcentaje de optimización de un 43,53 %.

Al revisar los resultados de la Figura 6.6 apreciamos que se mantiene el comportamiento analizado para la carga de trabajo anterior. Respaldando nuevamente la tesis expuesta donde las consultas que implementan la segunda guía de diseño tendrán un mayor impacto debido a que se eliminan tareas que debe hacer el motor de ejecución. En referencia a los datos estadísticos se apoya la idea que las variaciones que sufrieron los resultados en comparación a la carga de trabajo previa son mínimos. Esto nos permite mantener en pie las estrategias de diseño físico planteadas en este trabajo.

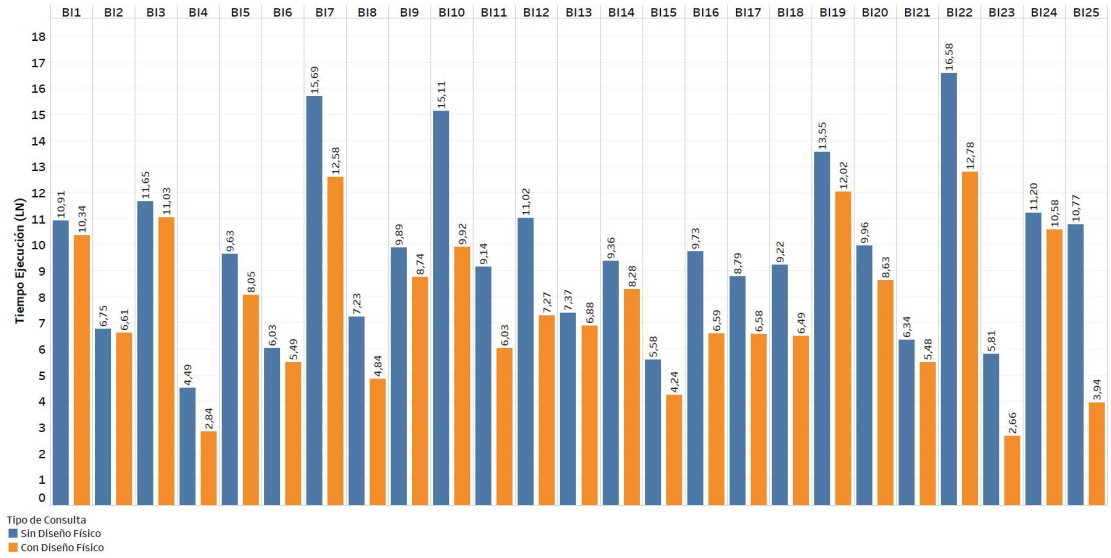


Figura 6.6: Tiempo de ejecución en escala logarítmica por cada consulta de una base de datos de 10 GB.

Según lo analizado en la Sección 6.1 se infiere que la muestra de datos de la actual carga de trabajo no posee una distribución normal a partir de las pruebas estadísticas de *Shapiro-Wilk*. Sin poder comprobar que los tiempos de ejecución de la muestra de la base de datos con diseño físico son menor que la muestra de tiempos de ejecución de la base de datos sin diseño físico, se procedió a ejecutar el test de *Wilcoxon signed-rank paired*.

La prueba permite comparar si el promedio de los tiempos de ejecución de las consultas sobre la base de datos sin diseño físico μ_0 son menor que el promedio de los tiempos de ejecución de las consulta sobre la base de datos con diseño físico μ_1. Los resultados de esta prueba no entregan un *p-valor* de 5,96x10^{-8}, con este número se puede afirmar con un nivel de significancia de 0.05, que $\mu_0 > \mu_1$. En definitiva en visto de los resultados conseguimos probar que en base a pruebas estadísticas los tiempos de ejecución obtenidos al ejecutar las consultas sobre la base de datos sin diseño físico son notablemente mayores en comparación a los tiempos de ejecución conseguidos al efectuar las consultas sobre una base de datos con diseño físico en un conjunto de datos de 10 GB.

Una vez analizados los porcentajes de mejora en los tiempos de ejecución debemos analizar los resultados obtenidos en los DB Hits. La Figura 6.7 se compone en el eje de las abscisas por las consultas puestas a prueba que llevan por nombre el acrónimo BI acompañado del número de la consulta. En el eje de las ordenadas encontramos el porcentaje de mejora con referencia al DB Hits. A simple vista en la Figura 6.7 observamos que existe un alza en los porcentajes de las consultas BI11, BI12, BI18, BI23 y BI25. En cambio, tan sólo una consulta se aprecia con una significativa diferencia con respecto al resto, esta es la consulta BI2. Exceptuando la consulta anterior, las 24 consultas restantes se encuentran por sobre el 39,76 % de mejora lo que es bien importante. Debemos considerar en este último punto que si bien el reducir el valor de DB Hits no significa una disminución de los tiempos de ejecución. Adicionalmente, no se aprecian las estadísticas de las consultas BI7, BI10, BI19 y BI22, esto ocurre en vista a que se manifiesta un concepto que indicamos previamente en el capítulo 5, este corresponde a *buffer overflow*. Este se presenta en un recuadro en cada plan de ejecución con una cifra cuantiosamente negativa por lo tanto no se pueden tener en cuenta estas cifras para comparar con los resultados del experimento en los DB Hits de la carga de trabajo de 1 GB.

Con la excepción de la consultas ya antes mencionadas se observa en las tablas 6.8 y 6.9 que las nuevas consultas que poseen un mayor porcentaje de mejora son BI11, BI12, BI18, BI23 y BI25, todas estas están por sobre un 93,28 %. En cambio la consulta BI2 sufrió un pequeño descenso de 0,34 % que no genera ningún inconveniente dado que el tiempo de ejecución subió más de un 8 %. De distinta forma se comportó la consulta BI21 que obtuvo un pequeño crecimiento de un 0,1 % de mejora. Por otro lado, tenemos la consulta BI1 que al igual que la consulta anterior demuestra una pequeña mejora de un 1,15 %.

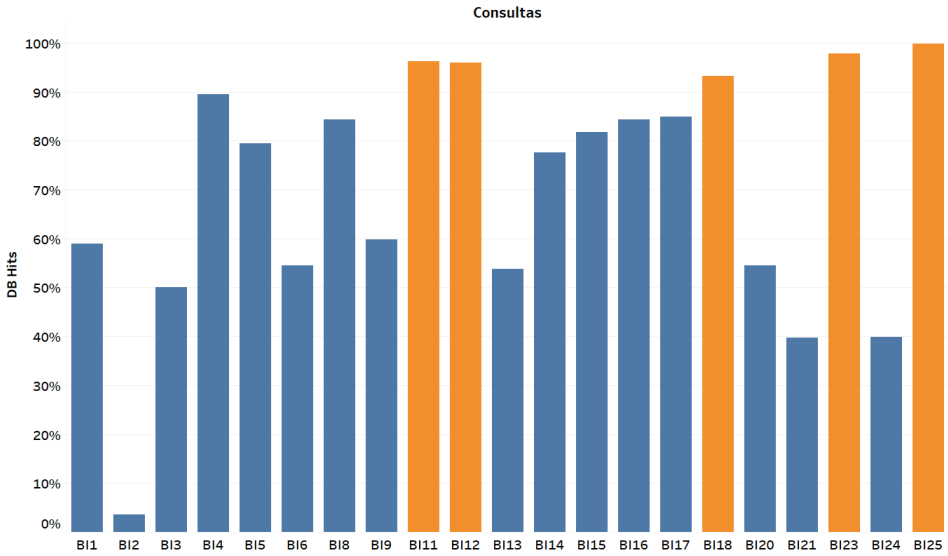


Figura 6.7: DB Hits por cada consulta en una base de datos de 10 GB.

Consulta	BI1	BI2	BI3	BI4	BI5	BI6	BI8	BI9	BI11	BI12	BI13
DB Hits (%)	59,00	3,55	50,11	89,47	79,53	54,50	84,41	59,75	96,24	95,92	53,77

Tabla 6.8: Porcentaje de mejora sobre valores sin diseño físico en los DB Hits para las consultas BI1 - BI15 de una base de datos de 10 GB.

Consulta	BI14	BI15	BI16	BI17	BI18	BI20	BI21	BI23	BI24	BI25
DB Hits (%)	77,67	81,77	84,31	84,98	93,28	54,54	39,76	97,77	54,90	99,86

Tabla 6.9: Porcentaje de mejora sobre valores sin diseño físico en los DB Hits para las consultas BI14 - BI25 de una base de datos de 10 GB.

Se han generado dos grupos de consultas que usan la estrategia de Materialización de Caminos, el primero se conforma por un total de 11 consultas que obtienen un porcentaje de mejora sobre los valores sin diseño físico entre un 0,05 % y 16,92 %. Esto se puede explicar en razón de que 6 de las 11 consultas poseen la estrategia de Propiedades Limitadas, esto significa que se consigue reubicar las operaciones de agregación posicionadas en la sentencia *WITH* o la eliminación de variables intermedias que almacenan el resultado de un cálculo previo. Lo que se evidencia en la consulta BI6 que tiene un porcentaje de optimización de un 16,92 % y se fundamenta exclusivamente con el aplazamiento de la manipulación de las propiedades en razón de que apoya la mejora que aporta la técnica de Materialización de Caminos. En cambio el segundo grupo posee 7 consultas que presentan una pequeña caída en el porcentaje de mejora sobre los registros sin diseño físico entre

un 0,45 % y 2,94 %, esta pequeña caída se puede deber a dos razones puntualmente. La primera corresponde a que las consultas exhiben la sentencia *OPTIONAL MATCH* en su estructura, afectando directamente la tarea del motor de ejecución cuando se efectúa la búsqueda de información puesto que al incrementar el volumen de datos existe una mayor cantidad de valores que se establecen como *null*, esto denota que según la definición de la sentencia *OPTIONAL MATCH* debe capturar estos registros incrementando su labor y en consecuencia reduciendo el porcentaje de optimización. Así mismo, el tener condiciones propias de la función *WHERE* genera un impacto negativo considerando que se acceden a las propiedades de los nodos, sabemos que esta operación es una de las más costosas dentro Neo4j por lo que es otra causa del decrecimiento de estos porcentajes de mejora.

En la Figura 6.8 se identifica un alza considerable de los registros manteniendo un promedio cercano a los 15,5 puntos. Esto se fundamenta con el *buffer overflow* que entregaron las consultas BI7, BI10, BI19 y BI22. Igualmente, vemos que la diferencia entre el ambos valores es menor lo que es derivado del aumento de operaciones de agregación, operaciones aritméticas y el acceso a las propiedades en áreas del código que no se pudieron abordar con las estrategias de diseño físico. Esto último es debido a que las estrategias se implementan en su mayoría en el punto crítico identificado en cada consulta. Finalmente, se mantiene la tendencia donde el valor sin diseño físico es mayor que el valor con diseño físico lo que sostiene nuestra tesis de la relación entre el tiempo de ejecución y los DB Hits con excepción de las consultas antes indicadas.

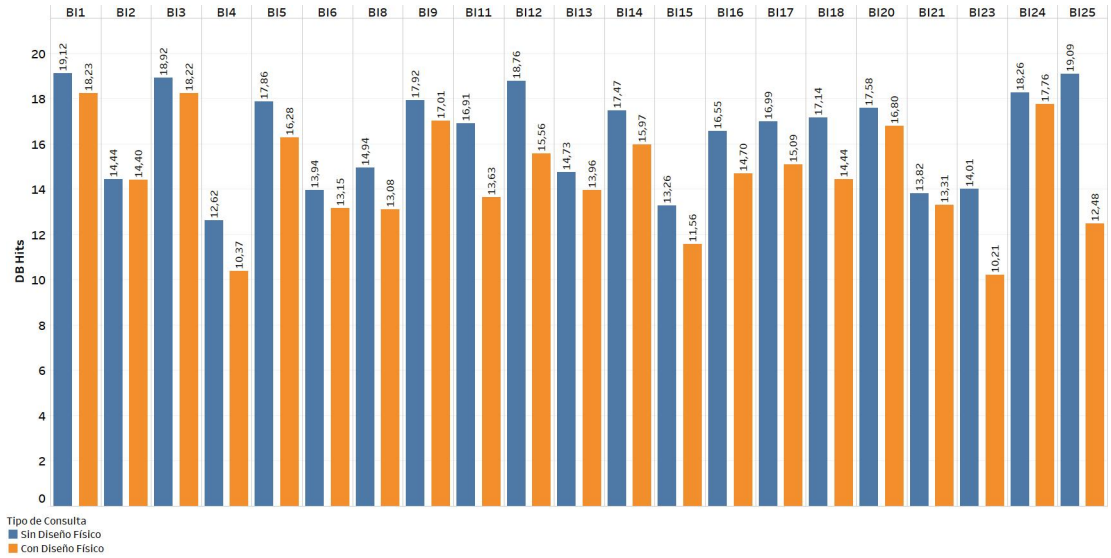


Figura 6.8: DB Hits en escala logarítmica por cada consulta de una base de datos de 10 GB.

6.3. Impacto del diseño físico para una base de datos de 100 GB

La carga de trabajo de 100 GB es nuestra última prueba, cuyos resultados se encuentran en la Figura 6.9. Debemos recordar que para experimentación se definieron 5 iteraciones para la captura de datos. La Figura 6.9 se compone en el eje de las abscisas por las consultas puestas a prueba que llevan por nombre el acrónimo BI acompañado del número de la consulta. Por otro lado, en el eje de las ordenadas encontramos el porcentaje de mejora con referencia al tiempo de ejecución. A primera vista nos damos cuenta que existe un menor número de consultas, esto es producto de que se presentaron algunos *timeout* al momento de capturar las estadísticas. Las consultas que cruzaron el limite de tiempo definido en la Sección 5.1.1 son BI1, BI3, BI7, BI10, BI12, BI19, BI20, BI22 y BI24. En la figura 6.9 podemos ver que las consultas que más destacan son las consultas BI11, BI16, BI18, BI23 y BI25. Estas consultas revelan un margen de mejora sobre los valores sin diseño físico entre un 97,38 % hasta un 99,52 % resaltando la consulta BI25 como lo ha sido en cada carga de trabajo. Desde otro punto de vista, las consultas que exhiben un menor porcentaje de mejora son BI6, BI9, BI14, BI15 y BI17, siendo la consulta BI13 la que posee el porcentaje más bajo con un 33,54 %.

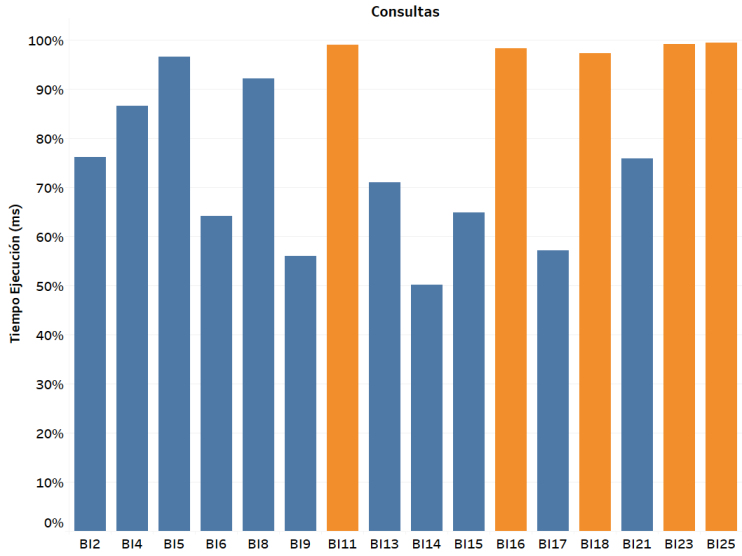


Figura 6.9: Tiempo de ejecución por cada consulta de una base de datos de 100 GB.

Consulta	BI2	BI4	BI5	BI6	BI8	BI9	BI11	BI13
Tiempo Ejecución (%)	76,26	86,60	96,69	64,23	92,20	56,03	99,04	71,03

Tabla 6.10: Porcentaje de mejora sobre valores sin diseño físico en el tiempo de ejecución para las consulta BI2 - BI13 de una base de datos de 100 GB.

Consulta	BI14	BI15	BI16	BI17	BI18	BI21	BI23	BI25
Tiempo Ejecución (%)	50,16	64,90	98,40	57,21	97,38	75,92	99,18	99,52

Tabla 6.11: Porcentaje de mejora sobre valores sin diseño físico en el tiempo de ejecución para las consulta BI14 - BI25 de una base de datos de 100 GB.

En las Tablas 6.10 y 6.11 vemos el detalle de los porcentajes de mejora sobre los registros sin diseño físico expuestos en la Figura 6.9. En primera instancia, las consultas que sólo utilizan la estrategia de Materialización de Caminos son BI6, BI9, BI17, BI13, BI15, BI18 y BI25. De las últimas dos consultas vemos que se mantienen cercano o incluso superando los porcentajes de mejora sobre valores sin diseño físico de las cargas de trabajo de 1 GB y 10 GB. La consulta BI17 expone una baja considerable en su porcentaje de mejora con respecto al registro sin diseño físico ya que presenta un valor de 57,21 % siendo que sus porcentajes previos fueron de un 81,86 % y 89,03 %, respectivamente. Lo anterior se puede explicar con el aumento del conjunto de datos, debido a que existen dos condiciones de filtro en las que se accede al identificador del nodo *Person* que corresponde a una propiedad. Sabemos que esta es una de las acciones más costosas, sumándole que no existe una limitación de los registros a procesar mediante la sentencia *LIMIT*. Igualmente, como indicábamos anteriormente la consulta BI13 es la que evidencia el menor porcentaje de mejora producto de que posee una sentencia *OPTIONAL MATCH*. Esta sentencia al aumentar el conjunto de datos puede llegar a generar un inconveniente puesto que aumentan los registros *null* provocando que el motor de ejecución deba rastrear la diferencia de registros con respecto a los ya materializados. Las consultas que presentan una combinación con alguna de las estrategias de Reescritura de Consulta son BI4, BI5, BI8, BI11, BI14, BI16 y BI23. De igual forma que la consulta BI13 apreciamos este proceder en las consultas BI9 y BI14 pasando su porcentaje de mejora sobre los valores sin diseño físico de un 68,39 % y 65,97 % a un 56,03 % y 57,21 %, respectivamente. En la primera consulta nos encontramos con el no haber materializado la segunda sentencia *MATCH* provocándonos una disminución en el porcentaje de mejora para esta carga de trabajo porque debe recorrer una mayor cantidad de registros en comparación a la primera sentencia *MATCH* en el que si se materializó su punto crítico. Esta decisión fue tomada ante las condiciones de experimentación con las que se trabajaron, sin embargo, en el caso de que no se encuentre en esta situación se puede implementar más de una vez. Por otro lado, en la segunda consulta tenemos la situación en que no se incorporaron todas las

condiciones de la función *WHERE* ya que no eran parte de la relación que se materializó provocando de esta manera el acceso a las propiedades. Es por este motivo que al aumentar el conjunto de datos provocó una disminución en el porcentaje de mejora.

Las consultas de la estrategia Reescritura de Consulta que no tuvieron inconvenientes de *timeout* fueron las consultas BI2 y BI21. Ambas exhibieron un destacable desempeño sobre todo la primera de estas logrando ascender un 63,23 % con respecto a su porcentaje de mejora sobre el registro sin diseño físico de la carga de trabajo de 10 GB quedando de esta forma en un 76,26 %. La consulta BI21 consiguió escalar aproximadamente 18 puntos porcentuales logrando un resultado de 75,92 %. Todo lo anterior es producto de plantear de manera eficiente la estructura de una consulta consiguiendo reestructurar el segmento de código original y modificar la posición de las operaciones de agregación.

Finalmente, la estrategia de Creación de Índices no pudo ser probada ya que traspasó el límite de tiempo cayendo en el estado de *timeout*. De todas formas este proceder se puede justificar por la numerosa cantidad de operaciones de agregación y operaciones aritméticas que se encuentran en dicha consulta. Así mismo, tampoco está la posibilidad de limitar los registros mediante la sentencia *LIMIT* por lo que también podría ser considera una razón que explique esta conducta.

En comparación a la carga de trabajo anterior existen diferentes conductas en las consultas que se visualizan en la Figura 6.10. Las mayoría de las consultas lograron un significativo aumento en la diferencia de los tiempos de ejecución sin diseño y con diseño físico, sin embargo, la consulta BI15 y BI17 tuvo una diferencia menor que se explica por las sentencias *WHERE* que poseen. Estas condiciones son imprescindibles sabiendo que significan un acceso a las propiedades de los nodos y relaciones lo que provoca esta conducta. Cabe mencionar que las consultas que no aprecian en la Figura 6.10 es debido a que se encuentran en la condición de *buffer overflow* o *timeout*.

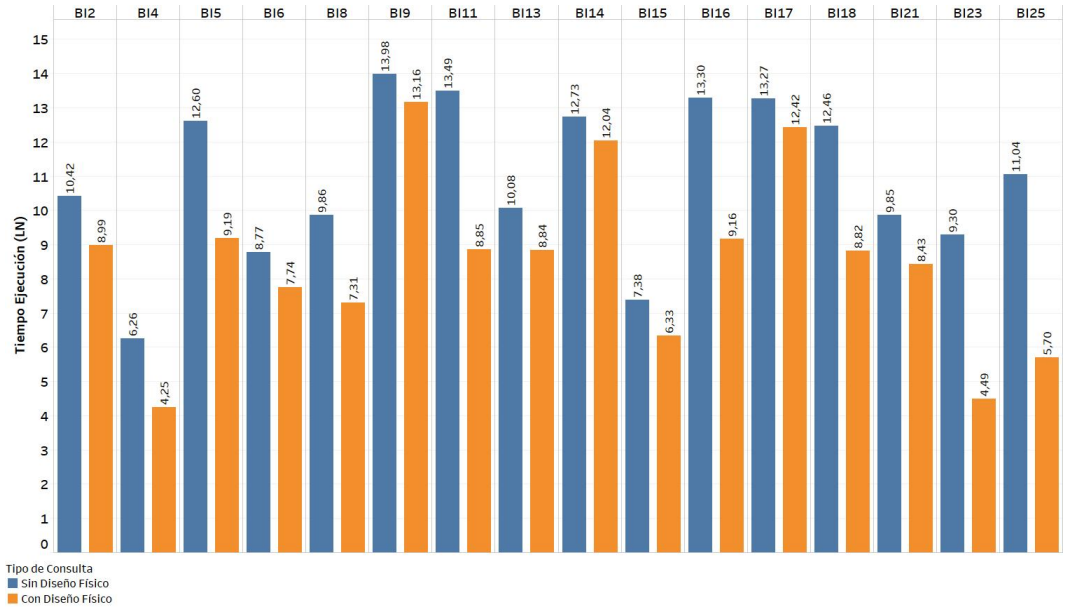


Figura 6.10: Tiempo de ejecución en escala logarítmica por cada consulta de una base de datos de 100 GB.

Según lo analizado en la Sección 6.1 se infiere que la muestra de datos de la actual carga de trabajo no posee una distribución normal a partir de las pruebas estadísticas de *Shapiro-Wilk*. Sin poder comprobar que los tiempos de ejecución de la muestra de la base de datos con diseño físico son menor que la muestra de tiempos de ejecución de la base de datos sin diseño físico, se procedió a ejecutar el test de *Wilcoxon signed-rank paired*. La prueba permite comparar si el promedio de los tiempos de ejecución de las consultas sobre la base de datos sin diseño físico μ_0 son menor que el promedio de los tiempos de ejecución de las consulta sobre la base de datos con diseño físico μ_1. Los resultados de esta prueba no entregan un *p-valor* de 3,052x10^{-5}, con este número podemos afirmar con un nivel de significancia de 0.05, que $\mu_0 > \mu_1$. En definitiva en vista de los resultados podemos afirmar que en base a pruebas estadísticas los tiempos de ejecución obtenidos al ejecutar las consultas sobre la base de datos sin diseño físico son notablemente mayores en comparación a los tiempos de ejecución conseguidos al efectuar las consultas sobre una base de datos con diseño físico en un conjunto de datos de 100 GB.

Habiendo analizado la primera métrica de evaluación consideraremos los resultados de obtenidos con respecto a los DB Hits. La Figura 6.11 se compone en el eje de las abscisas por las consultas puestas a prueba que llevan por nombre el acrónimo BI acompañado del número de la consulta. Por otro lado, en el eje de las ordenadas encontramos el porcentaje de mejora con referencia al DB Hits. En la Figura 6.11 vemos la falta de algunas consultas al igual que en la figura relacionada a los resultados conseguidos en el

tiempo de ejecución. La única diferencia es que en la Figura 6.11 exhibe una consulta menos. Nos referimos a la consulta BI17 la que presenta un *buffer overflow* en su punto crítico correspondiente a la relación *KNOWS* entre las variables *a* y *b*. A primera vista podemos ver que las consultas que poseen un mayor porcentaje de mejora son las consultas BI4, BI11, BI18, BI23 y BI25. La consulta BI18 y BI25 implementan exclusivamente la estrategia de Materialización de Caminos, en cambio, las consultas restantes incorporan la estrategia Propiedades Limitadas. Al contrario de las consultas mencionadas, tenemos que las consultas que presentan un menor porcentaje de mejora son las consultas BI2 y BI21. Es aquí cuando nos damos cuenta que si bien se podría inferir que existe una relación entre los DB Hits y el tiempo de ejecución, no es del todo cierto, puesto que las consulta BI2 y BI21 demuestran un elevado ascenso en sus porcentajes de mejora en relación a su tiempo de ejecución, sin embargo, no es proporcional s sus porcentajes de mejora en los DB Hits.

Hemos hablado del rendimiento de las consultas que implementan la estrategia de Materialización de Caminos con su combinación antes mencionada que presentan los porcentajes más altos de mejora. Desde ahí debemos retomar las consultas BI4, BI5, BI8, BI14 y BI16. Estas consultas trabajan con la combinación de las estrategias Materialización de Caminos y Propiedades Limitadas, obteniendo un margen de mejora desde un 75,65 % hasta un 89,59 % entregando un promedio de mejora de un 85,08 %. Existe una consulta que presenta el menor porcentaje de mejora, esa es la consulta BI6 con un 52,94 %, sin embargo, es necesario tratar de entender este comportamiento. Este se explica desde la presencia de dos sentencias *OPTIONAL MATCH*, sumándole además que esta consulta cuenta con una alta cantidad de operaciones aritméticas aplicadas sobre operaciones de agregación que aumentan el costo de procesamiento del motor de ejecución, y en consecuencia reduce el porcentaje de mejora. De todos modos el porcentaje de mejora entregado por esta consulta es buen resultado ya que esta por sobre el 50 %.

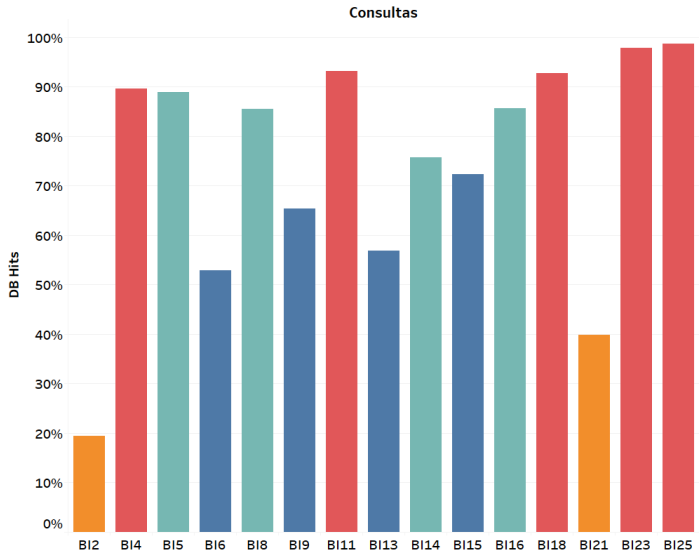


Figura 6.11: DB Hits por cada consulta en una base de datos de 100 GB.

Consulta	BI2	BI4	BI5	BI6	BI8	BI9	BI11	BI13
DB Hits (%)	19,48	89,59	88,94	52,94	85,53	65,34	93,19	56,81

Tabla 6.12: Porcentaje de mejora sobre valores sin diseño físico en los DB Hits para las consultas BI2 - BI13 de una base de datos de 100 GB.

Consulta	BI14	BI15	BI16	BI18	BI21	BI23	BI25
DB Hits (%)	75,65	72,36	85,70	92,67	39,83	97,83	98,72

Tabla 6.13: Porcentaje de mejora sobre valores sin diseño físico en los DB Hits para las consultas BI14 - BI25 de una base de datos de 100 GB.

Con respecto a las consultas BI2 y BI21 se justifican sus resultados ya que son cruciales en su búsqueda las diferentes operaciones de manipulación de datos, no obstante, podemos ver que de todas formas con sólo descomponer la consulta y a su vez reubicar las operaciones de agregación o establecer directamente el cálculo de una variable se obtiene un porcentaje de mejora de un 19,48 % y 39,83 % sobre los registros sin diseño físico.

En la Figura 6.12 podemos ver un caso particular con la consulta BI2 donde sus registros se mantienen por cada carga de trabajo. Esto nos lleva a cuestionar nuestra tesis de la supuesta relación entre el tiempo de ejecución y los DB Hits. Sin, embargo, como no se puede aseverar tampoco se puede desechar esta relación, por lo tanto, sólo nos queda

indicar que podría existir tal relación pero no se afirmar esa hipótesis. Con respecto al resto de las consultas, estas mantienen su comportamiento mostrando una posible estabilidad en los resultados a medida que se aumenta el volumen de datos.

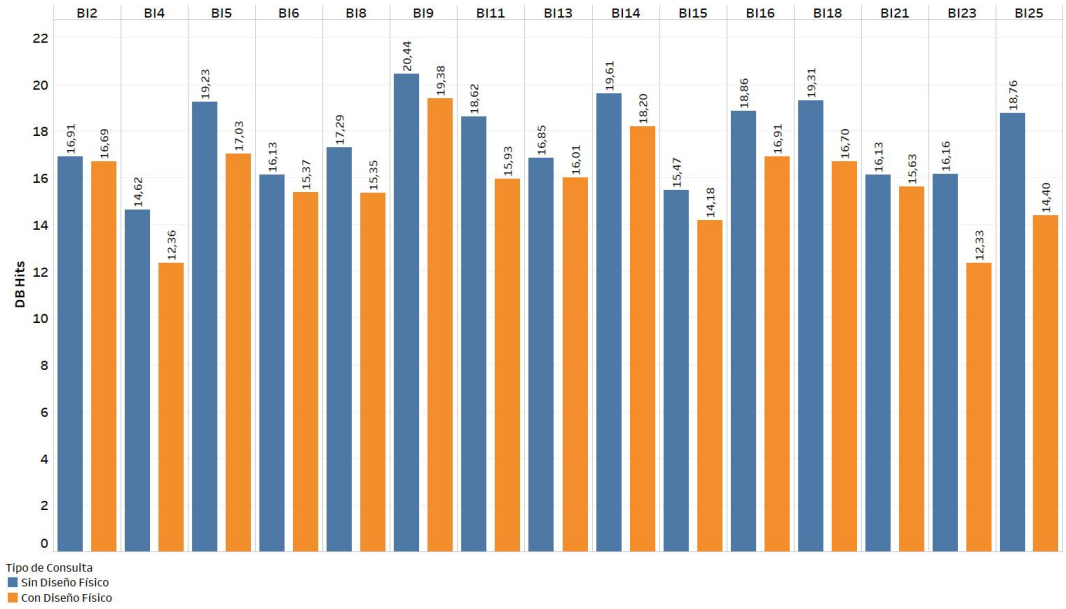


Figura 6.12: DB Hits en escala logarítmica por cada consulta de una base de datos de 100 GB.

6.4. Impacto de la carga de trabajo en el desempeño de las consultas

En esta sección comparemos los resultados de cada carga de trabajo por consulta para darnos así una mirada abstracta de la actuación de las estrategias implementadas para cada métrica de evaluación. En primer lugar abordaremos la comparación de resultados del tiempo de ejecución para después analizar los resultados conseguidos para los DB Hits.

En la Figura 6.13 observamos los resultados de todas las cargas de trabajo agrupados por consulta. Debemos puntualizar que los campos que se encuentran sin información se debe a que las consultas presentaron *timeout* en sus respectivas cargas de trabajo. Lo primero que vamos a analizar son los resultados con un mayor estabilidad pero enfocándonos en las consultas que presentan todas sus estadísticas. Las consultas BI6, BI13, BI18 y BI25 trabajan únicamente con la técnica Materialización de Caminos, en cambio, las consultas BI11, BI16 y BI23 incorporan la estrategia de Propiedades Limitadas. Estas consultas demuestran una estabilidad en sus tres cargas de trabajo de manera que no se evidencian

grandes variaciones, inclusive la mayor diferencia entre sus porcentajes de mejora es de tan sólo un 10 % perteneciente a la consulta BI23. Por otro lado tenemos las consultas que sufrieron un alto impacto a medida que iba aumentando la carga de trabajo, con esto nos referimos a las consultas BI2, BI4, BI5, BI6, BI8, BI13 y BI21. De este grupo sorprende la primera consulta que implementa la técnica Descomposición de Consulta, puesto que había exhibido bajos porcentajes de mejora en sus resultados de 1 GB y 10 GB, sin embargo, es la que revela una mayor variación con un alza de 72 % aproximadamente. La consulta BI21 que utiliza exclusivamente la estrategia de Propiedades Limitadas consigue un ascenso de 17 puntos porcentuales dejando en claro que ambas estrategias incrementan su rendimiento a medida que crece el conjunto de datos. Las consultas restantes de este grupo hacen uso de la combinación de las técnicas de Materialización de Caminos y Propiedades Limitadas que consiguen un alza promedio de 30 puntos. El otro patrón que se identifica en esta figura son las consultas que tuvieron inconvenientes, estas son las consultas BI1, BI3, BI7, BI10, BI12, BI19, BI20, BI22 y BI24. Se puede observar que en estas consultas existe un cierto equilibrio en sus porcentajes resaltando la consulta BI1 que usa la estrategia Creación de Índices. Esta consulta presenta el mayor descenso de porcentajes de mejora dentro de este grupo, no obstante, es de tan sólo un 2,81 %. Finalmente, analizaremos el último grupo que corresponde a las consultas que presentaron un descenso en sus estadísticas, las consultas que componen este grupo son BI9, BI14, BI15 y BI17. Dentro de estas cuatro consultas identificamos diferentes factores que nos pueden ayudar a comprender este proceder. El primer factor corresponde a que las consultas poseen un alta cantidad de operaciones, por otro lado, existe otro factor que se evidencia en la consulta BI17 el cual hace referencia al uso de condiciones mediante la sentencia *WHERE* y en consecuencia accediendo a las propiedades de los nodos. De igual forma, esta consulta no posee una sentencia que limite los registros para reducir el costo de procesamiento al manipular tal cantidad de registros. Anteriormente en la sección 3.3 indicamos que ante las limitaciones existentes en memoria principal debimos configurar la memoria virtual. Lo anterior explica el desempeño de estas cuatro consultas ya que al trabajar con un mayor conjunto de datos el costo en memoria es enorme provocando un mayor costo de procesamiento para el motor de ejecución.

La Figura 6.14 tiene por objetivo exhibir el comportamiento en relación al tiempo de ejecución de las consultas que fueron seleccionadas para representar a cada estrategia en el capítulo 4. Hay que recordar que las consultas que trabajan con las estrategias que se definen en la Figura 6.14 son la BI7, BI2, BI21, BI17 y BI1, respectivamente. Al analizar las técnicas de la primera guía de diseño vemos que las tres cumplen con el fin de disminuir el tiempo de ejecución exceptuando los 100 GB de la primera estrategia por *timeout*. La consulta que tuvo un desempeño efectivo y estable es la estrategia de Propiedades Limitadas, no obstante, también se debe mencionar la significativa reducción en la última carga de trabajo de la técnica de Descomposición de Consulta. La segunda guía de diseño presenta resultados estables con una clara tendencia a la reducción de los tiempos de ejecución

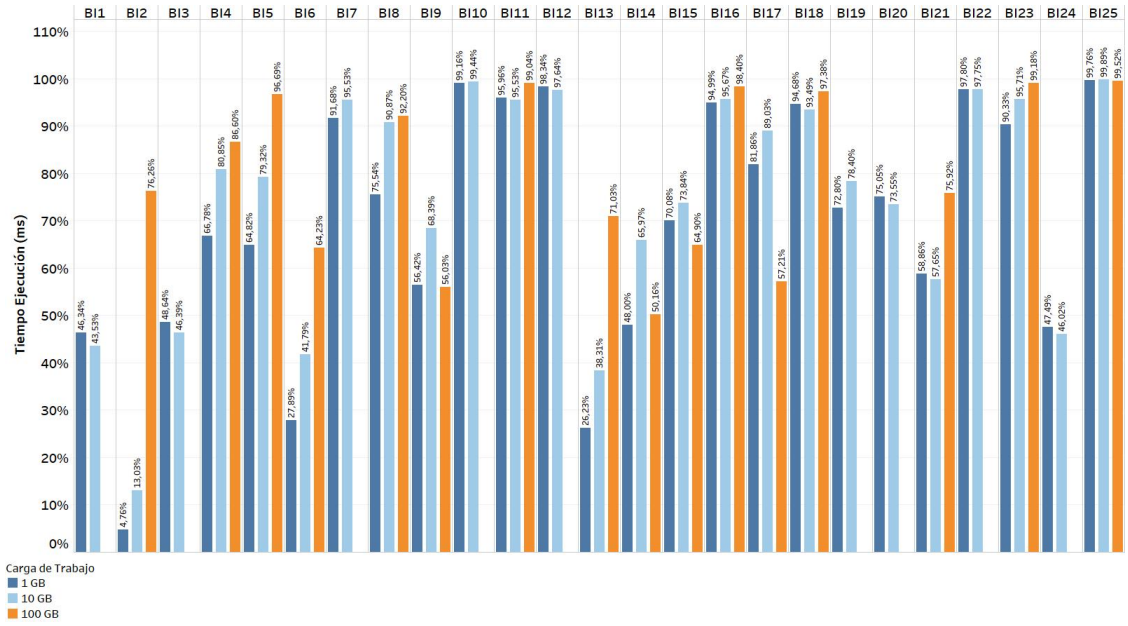


Figura 6.13: Comparación de porcentajes de mejora del tiempo de ejecución por carga de trabajo para cada consulta.

a medida que crece el volumen de datos. Aunque, se visualiza que en la última carga de trabajo se redujo su diferencia entre los tiempos de ejecución sin diseño físico y con diseño físico. Finalmente, la tercera guía de diseño demostró el mismo inconveniente que la estrategia de Consulta Mínima en la carga de trabajo de 100 GB, no obstante, se evidencia una reducción estable en ambas cargas de trabajo restante.

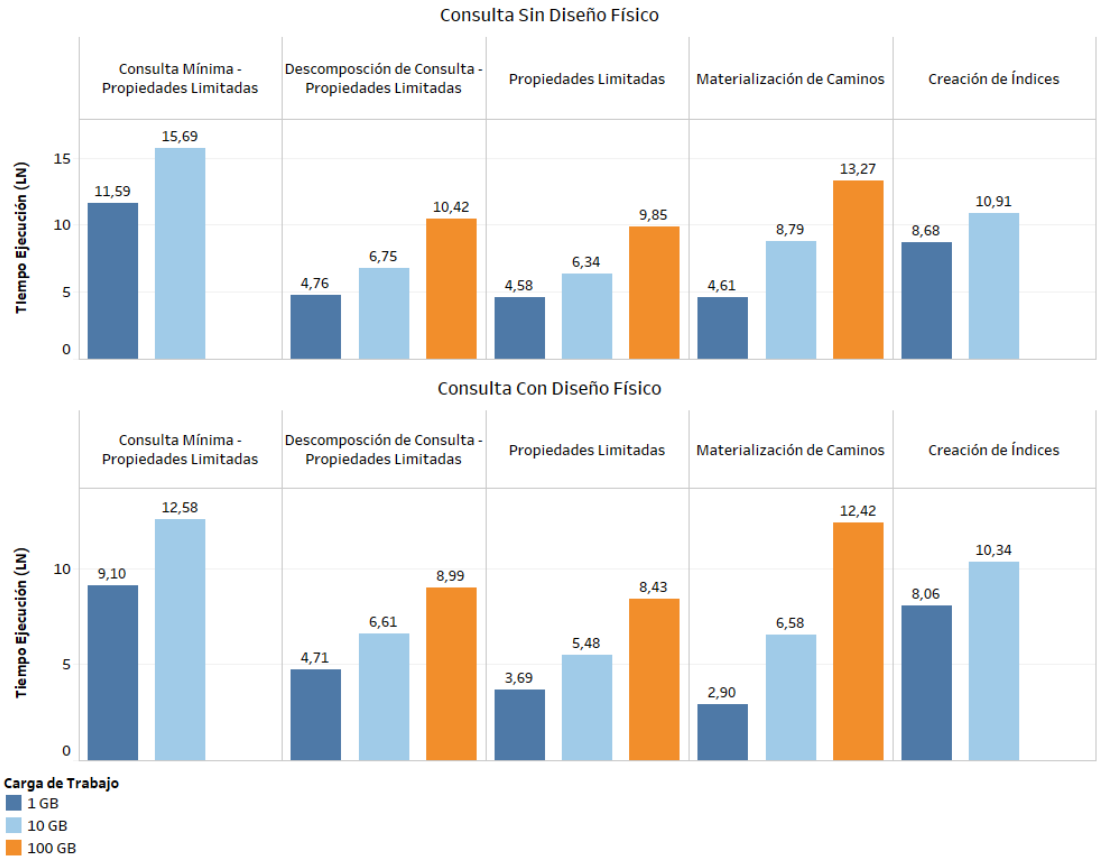


Figura 6.14: Comparación de tiempo de ejecución en escala logarítmica por carga de trabajo para cada estrategia.

En la Figura 6.15 podemos observar los porcentajes de mejora sobre los registros sin diseño físico de los DB Hits por consulta para cada carga de trabajo. En primer lugar nos damos cuenta que existen aún menos registros que la Figura 6.13, esto es producto de las restricciones de memoria que provocaron que las consultas arrojaran *buffer overflow* es por este motivo que no se consideraron. Como se puede observar la mayoría de las consultas evidencian una estabilidad aún mayor que los resultados obtenidos en el punto anterior. Sin embargo, de igual forma existen consultas que sufrieron algunas variaciones tanto positiva como negativamente. En el primer grupo se encuentran todas las consultas con excepción de la consulta BI15. Allí se destacan las consultas BI5 y BI9 con un alza de 25 % y 20 % aproximadamente. En cambio, la consulta BI15 es la única que presenta una baja sostenida en su porcentaje de mejora iniciando en un 89,24 % a un 72,36 %. Esto se fundamenta por el crecimiento del conjunto de datos ya que la estructura de la consulta originó una mayor complejidad al momento de manipular los datos y hacer los respectivos cálculos.

Por último, queremos destacar la consulta BI2 que en las primeras dos cargas de trabajo mantuvo un promedio de 3,72 %, no obstante, exhibió un gran incremento en el conjunto de datos de 100 GB consiguiendo llegar al 19,48 % aumentando casi 16 puntos porcentuales.

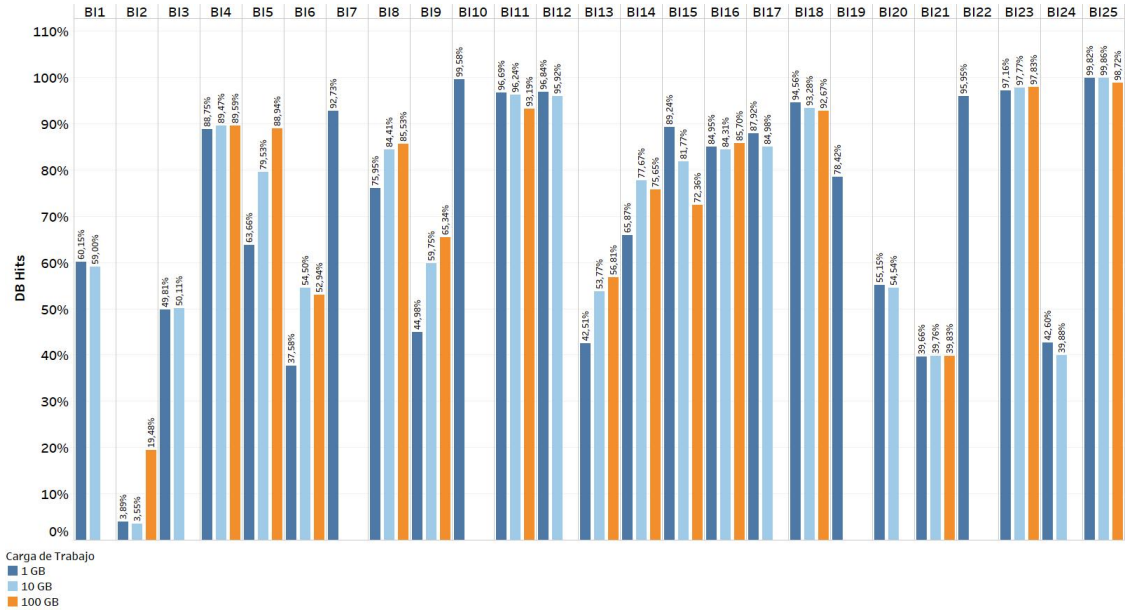


Figura 6.15: Comparación de porcentajes de mejora sobre valores sin diseño físico de los DB Hits por carga de trabajo para cada consulta.

Al haber analizado y comparado los impactos por cada conjunto de datos nos damos cuenta que las estrategias implementadas cumplieron su objetivo reduciendo tanto el tiempo de ejecución como también los DB Hits. De igual forma, podemos suponer que ante una reducción de los DB Hits disminuirá el tiempo de ejecución, sin embargo, no existe una relación directa en cuanto a las cantidades. Finalmente, cabe mencionar que si en el ambiente de desarrollo se cuenta con las condiciones óptimas estas estrategias pueden incluso demostrar un mejor desempeño especialmente las consultas que trabajen con operaciones de agregación y operaciones aritméticas.

En la Figura 6.16 se evalúa con la métrica de tiempo de ejecución cada estrategia exhibida en el capítulo 4. A primera vista nos damos cuenta que faltan resultados de la primera y cuarta columna, con respecto a la primera se presenta *buffer overflow* en la carga de trabajo de 10 GB y en la de 100 GB se obtuvo *timeout*. La segunda guía de diseño generó *buffer overflow* para la carga de trabajo de 100 GB en la consulta sin diseño físico. La primera guía de diseño demuestra tener una estabilidad en la disminución de los DB Hits, sin embargo, llaman la atención la estrecha diferencia que se observa en la estrategia Descomposición de Consulta en combinación con la de Propiedades Limitadas. Como ya

hemos analizado anteriormente esto no impacta negativamente a los resultados de los tiempos de ejecución de esta estrategia. La segunda guía de diseño nos muestra una reducción de casi dos puntos para las dos primeras cargas de trabajo, no obstante, sabemos que hubo una rebaja en la diferencia de los tiempos de ejecución sin diseño físico y con diseño físico, pero no se pudo estudiar desde los DB Hits este comportamiento por las razones antes mencionadas. Finalmente, la tercera guía de diseño evidencia una diferencia cercana a un punto para las dos primeras cargas de trabajo, recordando además que la última carga de trabajo presentó *timeout*.

Figura 6.16: Comparación de DB Hits en escala logarítmica por carga de trabajo para cada estrategia.

Otro punto a analizar de los resultados obtenidos corresponde a los costos de almacenamiento asociados a la implementación de las guías de diseño en cada consulta. Esta información la podemos observar en la Tabla 6.14 donde no se visualizan las consultas BI2, BI7 y BI21. Esto se debe a que las consultas utilizan estrategias de la primera guía de diseño lo que conlleva que no posean un costo almacenamiento en su aplicación. Otro

caso a mencionar es el de la consulta BI1 que utiliza la tercera guía de diseño, por lo tanto, el resto de consultas que observamos en la figura usan la segunda guía de diseño en complemento de la estrategia Propiedades Limitadas de primera guía de diseño según sea el caso. Por último, se debe mencionar que las columnas vinculadas a las cargas de trabajo por cada consulta que se encuentran sin información es debido a que cayeron en *timeout* en las pruebas de experimentación. Habiendo aclarado estos diferentes puntos procedemos a analizar los resultados. Destaca en el primer gráfico la consulta BI1 ya que presenta un alza considerable en el costo de almacenamiento, no obstante, esto se justifica al tener que abarcar un mayor número de propiedades a indexar. En las consultas BI3 y BI13 se evidencia un comportamiento similar puesto que la aplicación de la segunda guía de diseño fue sobre la entidad *Message* sabiendo que es una de las más costosas junto a las entidades *Forum* y *Person*. Esto implica que al aplicar alguna de estas guía de diseño y al aumentar el número de registros ocasionará un crecimiento de los costos de almacenamiento. En las consultas BI9 y BI11 se presenta lo antes mencionado en vista de que la implementación de la segunda guía de diseño es sobre las entidades *Forum* y *Person*, respectivamente. En referencia al segundo gráfico de la Figura 6.14 vemos que sobresale la consulta BI22. Allí sucede algo anecdótico a causa de que ambas materializaciones aplicadas se efectúan sobre entidades *Person*, por esta razón se comprende este abrupto incremento en los costos de almacenamiento. En la consulta BI18 sucede el peor de los casos en razón de que la materialización se establece en una relación directa entre las entidades *Message* y *Person* explicando esta alza considerable. En la consulta BI20 sucede algo parecido en las consultas BI3 y BI13, sin embargo, la diferencia es que no se buscan los mensaje sujeto a un tema puntual sino que están enlazados a tres clases de etiqueta pertenecientes a la entidad *TagClass*. Finalmente, tenemos un comportamiento anormal a la lógica en la consulta BI25 debido a que se presume que al incrementar el volumen de datos debiera aumentar el costo de almacenamiento. No obstante, en esta consulta no es así pero se justifica este comportamiento, puesto que en la mayoría de las consultas se incorporan condiciones a través de sentencias *WHERE* y se puede dar el caso de que los registros de la carga de trabajo no cumplan con la condición, dejándolos fuera de la materialización sosteniendo de este modo el descenso entre las dos últimas cargas de trabajo de la consulta BI25.

Consulta	**1 GB**	**10 GB**	**100 GB**
BI1	0,09100	0,84000	TimeOut
BI2	0,00000	0,00000	0,00000
BI3	0,01907	0,16500	TimeOut
BI4	0,00017	0,00200	0,01616
BI5	0,02244	0,02000	0,05173
BI6	0,00003	0,00010	0,00127
BI7	0,00000	0,00000	TimeOut
BI8	0,00364	0,02867	0,03500

Consulta	**1 GB**	**10 GB**	**100 GB**
BI9	0,00004	0,06800	0,11220
BI10	0,00184	0,02000	TimeOut
BI11	0,00119	0,01000	0,09760
BI12	0,00015	0,00200	TimeOut
BI13	0,00568	0,04000	0,35853
BI14	0,01606	0,12000	0,93272
BI15	0,00004	0,00318	0,02848
BI16	0,00348	0,02700	0,25043
BI17	0,00026	0,00294	0,02210
BI18	0,02104	0,24500	2,32000
BI19	0,01217	0,13200	TimeOut
BI20	0,19472	2,00000	TimeOut
BI21	0,00000	0,00000	0,00000
BI22	0,31000	4,40000	TimeOut
BI23	0,00100	0,00234	0,02043
BI24	0,08463	0,87000	TimeOut
BI25	0,00762	0,82350	0,40683

Tabla 6.14: Comparación de costos de almacenamiento (GB) en referencia a la implementación de cada estrategia.

Capítulo 7

Conclusiones

En este trabajo de título se evaluó y experimentó la implementación de un diseño físico sobre una base de datos orientada a grafos. Neo4j fue seleccionada para realizar los diferentes experimentos que nos permitieron evaluar el rendimiento de cada estrategia. Estos experimentos efectuados provienen del problema planteado por SNB, el cual dispone de un generador de datos y 25 consultas a resolver. Nuestro trabajo aborda las 25 consultas y se opta por trabajar con los conjuntos de datos de 1 GB, 10 GB y 100 GB. El diseño físico propuesto se compone por tres estrategias que consisten en la Reescritura de Consulta, Materialización de Caminos y Creación de Índices.

A lo largo de este trabajo nos pudimos dar cuenta de diferentes factores que crea una serie de recomendaciones para cualquier usuario que decida ingresar a las BDOG. Primero que todo, siempre se debe priorizar el ambiente de trabajo haciendo énfasis en el hardware de la máquina puesto que es necesario contar con una memoria principal de buena capacidad según el conjunto de datos con el que trabajemos. De igual forma se recomienda contar con un almacenamiento adicional entre un 20 % a 30 % al tamaño de la base de datos dependiendo de la tasa de crecimiento. Una vez instalado el hardware es vital efectuar un correcta configuración de la base de datos la cual estará sujeta a la disponibilidad que exista en memoria principal. Para esto es necesario seguir todas las recomendaciones sugeridas por la marca de la base de datos en su documento de especificación técnica. Con respecto a las recomendaciones de las estrategias de diseño físico podemos aconsejar que se debe priorizar la Reescritura de Consulta ya que no posee costo de almacenamiento y se pueden obtener grandes beneficios con su implementación. En segunda instancia se recomienda el uso de la estrategia Creación de Índices a causa de que las BDOG incorporan esta técnicas en sus plataformas mediante algoritmos optimizados. Finalmente, se recomienda el uso de la Materialización de Caminos en razón de que no se encuentra implementada por las BDOG por lo que puede ser un trabajo complejo y costoso en su constante implementación. A continuación, pasaremos a

precisar las ventajas y desventajas de la implementación de cada estrategia de diseño físico.

La estrategia Reescritura de Consulta es la que más se recomienda usar ya que no posee un costo de almacenamiento, sin embargo, debemos tener especial cuidado al momento de su implementación puesto que si no se aplica de manera correcta, se puede llegar a perder la equivalencia de los datos. Esto nos llevaría a entregar registros incorrectos a los que se solicitan. Por esta razón, es de vital importancia estar validando regularmente la equivalencia de los resultados. Adicionalmente, se sugiere la combinación de esta estrategia con las restantes ya que aumenta el rendimiento de la consulta sin ocasionar algún inconveniente.

La estrategia Materialización de Caminos es la que presenta los mejores resultados, también se puede destacar que la forma en que se aplicó esta estrategia consiguió reducir el costo de su implementación, no obstante, tenemos el gran problema de que las BDOG aún no implementan algoritmos que se encarguen de actualizar automáticamente las relaciones materializadas a medida que ingresan nuevos registros. Lo anterior crea una constante preocupación para el administrador de bases de datos por tener que actualizar manualmente estas relaciones. Al igual que la estrategia anterior se debe tener un cuidado especial en la equivalencia de los resultados debido a que si al momento de implementar no indicamos de forma correcta la dirección de la relación podremos llegar a perder la equivalencia.

La estrategia Creación de Índices se recomienda usar cuando existan condiciones de alta selectividad, pero se debe hacer analizar previamente la cantidad de registros a indexar frente al costo de almacenamiento que nos pueda generar ya que a veces existen restricciones de memoria que nos obligan a prescindir de esta estrategia. Neo4j implementa esta estrategia en su lógica por lo que no existe el impedimento que se presenta en la estrategia anterior.

Con los resultados obtenidos del estudio experimental que se efectuó en tres diferentes cargas de trabajo nos pudimos dar cuenta del rendimiento de cada una de ella lo que pasaremos a ver en detalle a continuación.

La primera guía de diseño físico se compone de tres técnicas que se definieron como Consulta Mínima, Descomposición de Consulta y Propiedades Limitadas. En la primera de estas obtuvimos un buen desempeño manteniéndose en un promedio 93,6 % con la salvedad de que no pudo ser examinada con un conjunto de datos de 100 GB. La segunda estrategia cuenta con un desempeño dispar puesto que en los primeros dos conjunto de datos sólo conseguimos un 4,76 % y 13,03 %, no obstante, en el último conjunto de datos logramos un porcentaje de mejora de 76,26 % demostrándonos que esta estrategia

se encuentra sujeta al número de registros con los que se trabajan. La última estrategia posee un buena actuación ya que se mueve en un rango entre 57,65 % y 75,92 % sólo limitando el acceso a las propiedades de los nodos. Debemos recordar que estas tres técnicas pertenecientes a la estrategia de Reescritura de Consulta no tiene costo alguno al momento de ser utilizadas en Neo4j.

La segunda guía de diseño se aplica en 21 de las 25 consultas a causa de la estructura de cada una de ellas. De esas consultas cuatro utilizan sólo esta técnica, no obstante, el resto incorpora la estrategia de Propiedades Limitadas. Esta dos formas de implementar la segunda guía son las que poseen el mejor desempeño, de esta forma lo certifican los promedios obtenidos por cada carga de trabajo que corresponden a un 73,19 %, 78,45 % y 80,56 %, respectivamente. Si bien existen consultas que no se pudieron validar en la carga de trabajo de 100 GB, esta corresponde a una cantidad menor que nos permite de todas formas comprobar esta guía de diseño.

La tercera guía de diseño sólo se pudo implementar en una consulta, adicionalmente no se pudo probar el desempeño de esta estrategia en el conjunto de datos de 100 GB. El rendimiento de esta técnica alcanzó un 46,34 %. A pesar de lo anterior registró un estable y alto porcentaje mejora cercano al 60 %.

Desde otro punto de vista necesitamos indicar cuáles fueron las limitaciones que tuvo este trabajo que impidieron un ambiente experimental óptimo. La primera limitación corresponde a la restricción de memoria con la que se trabajó ya que se tuvo que hacer uso de la configuración de memoria virtual para experimentar las estrategias de diseño físico en los conjuntos de datos de 10 GB y 100 GB. Si bien se pudo solucionar este inconveniente no es recomendable hacer uso de esta herramienta puesto que se deben cumplir ciertas condiciones que terminan afectando el rendimiento de la máquina. La segunda limitación es el haber trabajado en Neo4j y por ende con su lenguaje de consulta Cypher. Sabemos que el paradigma de las BDOG aún no se encuentra estandarizado de modo que no tenemos la certeza de que estas estrategias puedan ser implementadas en otro SGBD, y de ser así tampoco podemos asegurar que entreguen los mismos resultados. El último punto es producto de que los diferentes SGBD poseen su propio lenguaje de consulta debido a lo cual no podemos asegurar que estas estrategias retornen los mismos resultados de este documento en el caso de que puedan ser implementadas. La tercera limitación corresponde al proyecto de LDBC SNB, ya que está en constante perfeccionamiento, actualizando la lista de consultas o incluso eliminando consultas que no fueron bien planteadas en su lógica, de esta forma lo define LDBC en su documento de especificaciones [53]. Por lo anterior, es necesario validar las estrategias de diseño físico propuestas en la nueva versión que entrega LDBC SNB.

Nuestros trabajos futuros se ven ligados en resolver la nueva versión del problema presentado por LDBC que tiene por nombre "The LDBC Social Network Benchmark (version 0.4.0-SNAPSHOT)" [53] para seguir evaluando el desempeño de nuestras estrategias de diseño físico. Adicionalmente, también está previsto evaluar nuevas estrategias de diseño físico que sean aptas para ser puestas a prueba en Neo4j con Cypher. Debido a que LDBC SNB plantea tres lenguajes de consulta para resolver el problema como lo son: Cypher, PostgreSQL y SPARQL. Se tiene considerado aplicar estas estrategias de diseño físico desde el lenguaje de consulta SPARQL, generando así un análisis de la compatibilidad y posterior adaptación de las estrategias de diseño físico en el lenguaje de consulta SPARQL.

A lo largo de esta investigación nos encontramos con una gran dificultad en la búsqueda de trabajos o publicaciones que sean relacionadas o que tengan como tema principal las guías de diseño en una BDOG por lo que este trabajo es una contribución en este sentido.

Todo el material con el que se trabajó en este trabajo de título se encuentra en un repositorio que se precisa en el anexo B con las respectivas instrucciones para su implementación. Igualmente, se almacenará cada carga de trabajo en dos formatos: el primero corresponde a Neo4j para ser cargado directamente en el SGBD y el segundo es con el formato nativo del Datagen para ser trabajado cualquiera sea el objetivo.

Anexo A

Diseño solución

A.1. Consulta BI3

El documento de LDBC SNB plantea que esta consulta requiere capturar las etiquetas que hayan usado los mensajes según un mes y año entregado, además de las etiquetas que fueron usadas el mes siguiente. Para las etiquetas y para ambos meses se debe contar la cantidad de mensajes. En la Figura A.1 podemos observar el plan de ejecución original de la consulta BI3 que se compone de los siguientes operadores:

1. Dos operadores *Projection* en el cual se asignan las variables para el año y el mes requerido, y efectuar el conteo de los mensajes.

2. Dos operadores *OptionalExpand* el primero es para la búsqueda de los mensajes dada la fecha indicada y el segundo es para la búsqueda de las etiquetas de los mensajes el mes siguiente

3. Un operador *Apply* que une las variables definidas y la búsqueda de las etiquetas del mes y año de los mensajes.

4. Dos operadores *EaggerAggregation* que efectúan la operación del conteo de la cantidad de mensajes que poseen las etiquetas encontradas.

5. Un operador *Top* que retorna los primeros 100 resultados.

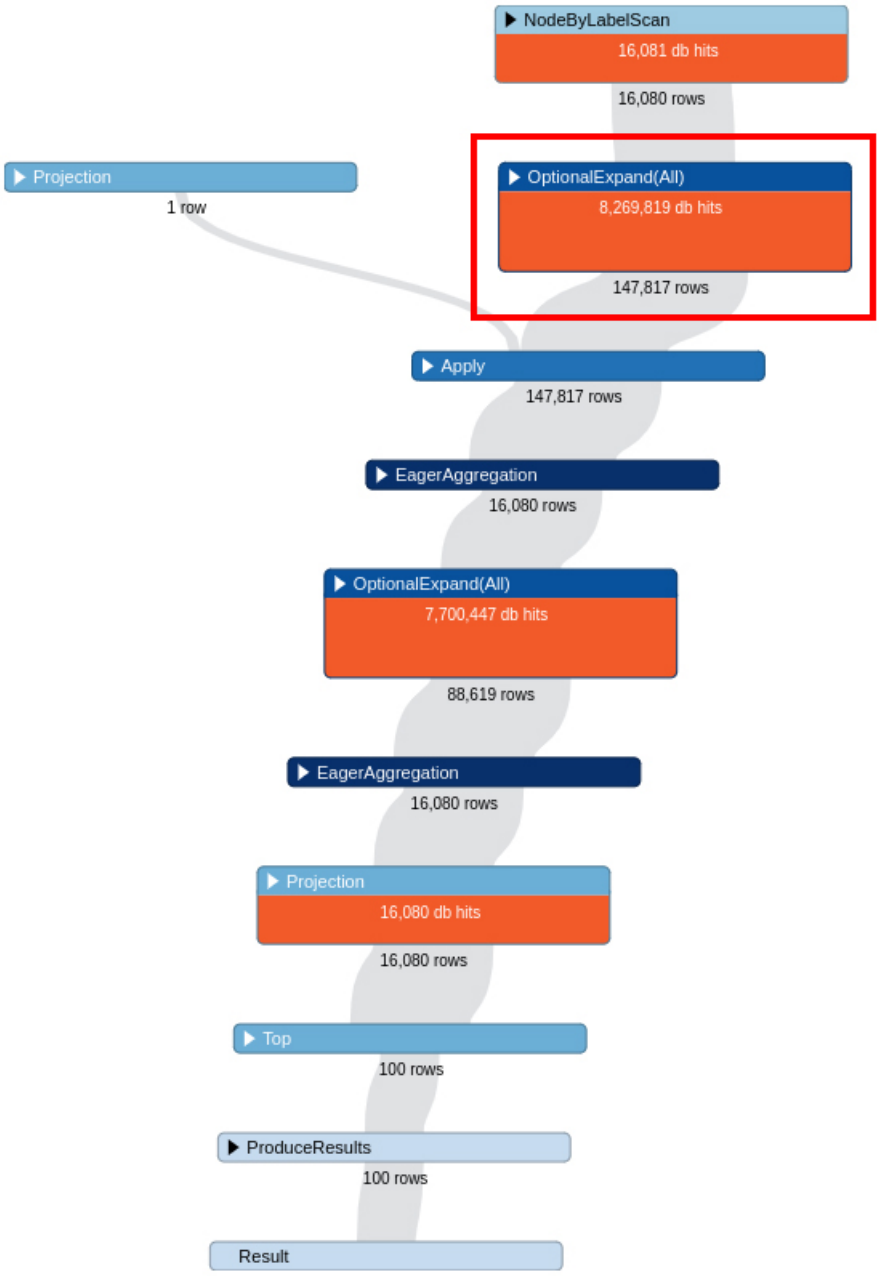


Figura A.1: Plan de Ejecución Original Consulta BI3.

El punto crítico de la consulta se halla enmarcado en el recuadro de color en el plan de ejecución. Este punto crítico corresponden a la relación *HAS_TAG* cuya diferencia se explica por las operaciones de condición que se deben ejecutar. En este caso el recuadro

posee una mayor cantidad de registros en comparación al segundo operador *OptionalExpand* y en consecuencia la operación de filtro es más costosa. Analizando el código nos damos cuenta que se puede emplear la estrategia Materialización de Caminos para abordar el punto crítico. Esto posibilita eliminar las condiciones que se presentan en la sentencia *WHERE* y establecer una relación directa entre ambos nodos, pero tomando en cuenta sólo los registros que se requieren. En la Figura A.2 se aprecia la implementación de la segunda guía de diseño.

```
WITH
  2010 AS year1,
  10 AS month1,
  2010 + toInteger(10 / 12.0) AS year2,
  10 % 12 + 1 AS month2
// year-month 1
MATCH (message1:Message)-[:HAS_TAG]->(tag:Tag)
WHERE message1.creationDate/10000000000000   = year1
    AND message1.creationDate/100000000000%100 = month1
CREATE (message1)-[:BI3_message1_tag]->(tag);
```

Figura A.2: Solución Propuesta Materialización de Caminos Consulta BI3.

Una vez aplicada la estrategia nuestro plan de ejecución original se ve modificado, puesto que se logra el objetivo de reducir considerablemente el costo de DB Hits consiguiendo de la misma forma reducir el tiempo de ejecución. Todo lo anterior lo podemos ver a continuación en la Figura A.3.

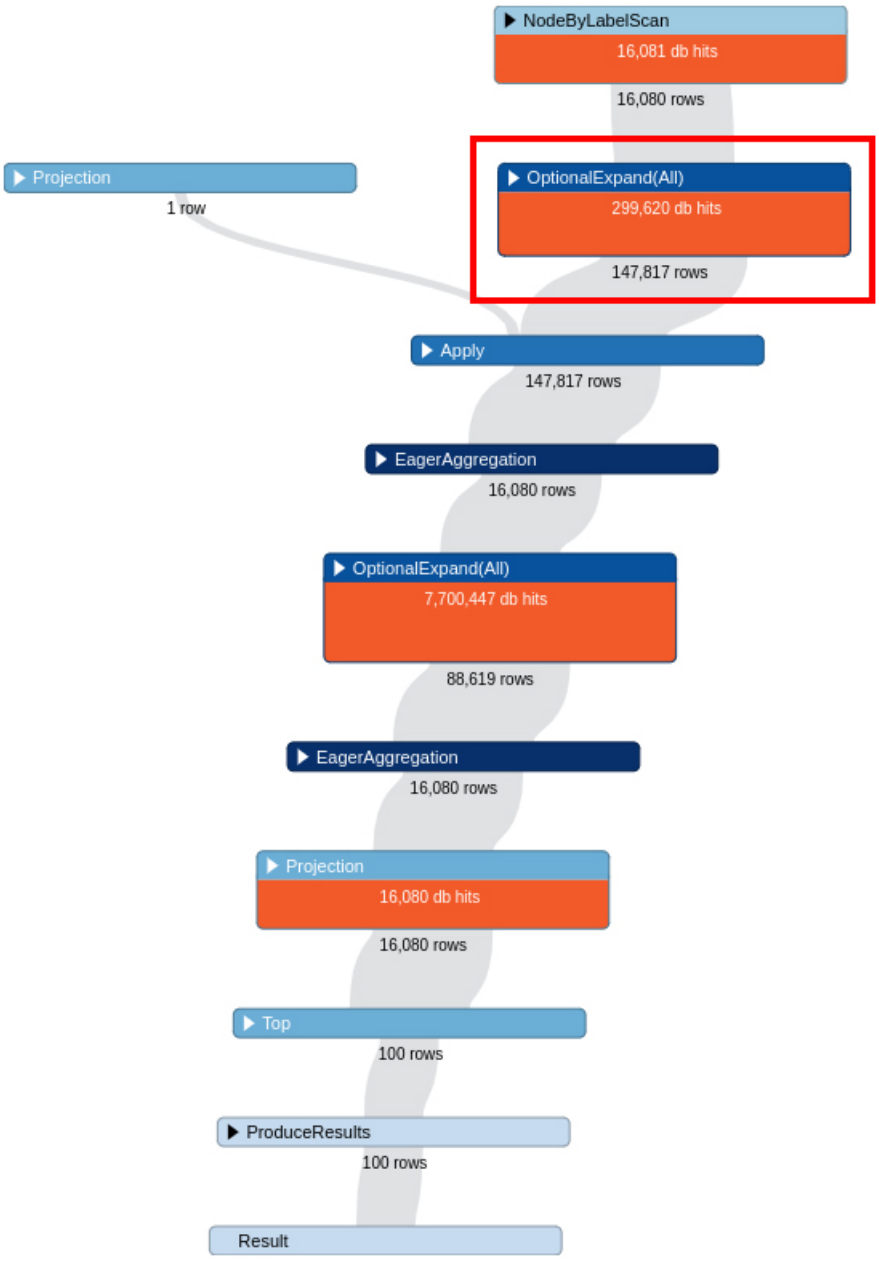


Figura A.3: Plan de Ejecución Solución Propuesta Consulta BI3.

A.2. Consulta BI4

En el documento de especificación se indica para la consulta BI4 que se debe buscar todos los foros creados en un país que contengan al menos una publicación con una etiqueta que pertenezca a una clase de etiquetas especifica, el país y la clase de etiqueta son asignados por el documento. En la Figura A.4 se puede apreciar un segmento del plan de ejecución de la consulta BI4 que se compone de los siguientes operadores:

1. Un operador *scan* para buscar todas las clases de etiquetas.
2. Cuatro operadores *Filter* que condicionan los resultados obtenidos según una clase de etiqueta y un país definido.
3. Tres operadores *Expand* que despliegan las relaciones
4. Un operador *NodeHashJoin* que une los resultados de ambas ramas de búsqueda.
5. Un operador *EagerAggregation* que efectúa el acceso a las propiedades de los nodos con el fin de ordenar los registros.
6. Un operador *Top* que retorna los primeros 100 resultados.

Podemos apreciar en el primer recuadro del plan de ejecución que existen dos operadores *EXPAND*, estos operadores hacen referencia a los siguientes nodos y relaciones, respectivamente:

- *(forum:Forum) - [:CONTAINER_OF]* $\rightarrow$ *(post:Post)*
- *(post:Post) - [:HAS_TAG]* $\rightarrow$ *(:Tag)*

Estas dos relaciones componen el punto crítico que se observa en el plan de ejecución, donde al revisar el código nos damos cuenta que existe sólo una sentencia *MATCH* para efectuar la búsqueda. De la misma forma, percibimos que existe una sentencia *LIMIT* en la parte final del código. Se descartan las estrategias de la primera guía de diseño con excepción de la técnica de Propiedades Limitadas, puesto que no existen condiciones apropiadas para su implementación. En segunda instancia buscamos probar la factibilidad de la incorporación de la tercera guía de diseño. Al analizar la estructura de la consulta concluimos que es posible de efectuar sobre los nodos *Country* y *TagClass*, sin embargo, el costo de procesamiento de esas operaciones es insignificante para el motor de ejecución por tal motivo descartamos esa posibilidad. Finalmente, decidimos incorporar la segunda guía de diseño en la cual materializamos la primera relación de la lista. Con esto no sólo materializamos dicha la relación sino que también podemos eliminar la relación *HAS_TAG* y en consecuencia el nodo *Tag*. De esta forma logramos eliminar completamente el punto

Figura A.4: Segmento Plan de Ejecución Consulta BI4.

crítico consiguiendo una reducción considerable en la búsqueda de los registros por parte del motor de ejecución. La aplicación de la segunda guía de diseño se puede apreciar en la Figura A.5.

```
MATCH
  (:Country {name: 'Burma'})<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-
  (person:Person)<-[:HAS_MODERATOR]-(forum:Forum)-[:CONTAINER_OF]->
  (post:Post)-[:HAS_TAG]->(:Tag)-[:HAS_TYPE]->(:TagClass {name: 'MusicalArtist'})
CREATE (forum)-[:BI4_forum_post_tagClass]->(post);
```

Figura A.5: Solución Propuesta Materialización de Caminos Consulta BI4.

Una vez aplicada la segunda guía de diseño es el turno de la estrategia Propiedades Limitadas que se presenta en en el segundo recuadro. El motivo de su aplicación es producto del operador *count* que se exhibe en la sentencia *RETURN*. La Figura A.6 demuestra que se puede modificar la estructura del código cambiando el orden, buscando sólo efectuar las operaciones necesarias previo al acceso de las propiedades. Habiendo aplicada la tercera técnica de la primera guía de diseño se puede visualizar el código final en la Figura A.6b.

```
RETURN
  forum.id,
  forum.title,
  forum.creationDate,
  person.id,
  count(DISTINCT post) AS postCount
ORDER BY
  postCount DESC,
  forum.id ASC
LIMIT 20
```

(a) Segmento Código Original

```
WITH
  forum,
  person,
  count(DISTINCT post) AS postCount
ORDER BY
  postCount DESC,
  forum.id ASC
LIMIT 20
RETURN
  forum.id,
  forum.title,
  forum.creationDate,
  person.id,
  postCount
```

(b) Segmento Código Optimizado

Figura A.6: Solución Propuesta Propiedades Limitadas Consulta BI4.

En la Figura A.7 vemos los resultados de combinar estas dos estrategias. En el primer recuadro logramos eliminar un camino por completo dejando así el plan de ejecución en un sólo camino y reduciendo el valor total de DB Hits. Por otro lado, en el segundo recuadro la aplicación de la estrategia nos permite rebajar a cero el valor de DB Hits en el operador *EagerAggregation* lo que permite aminorar aún más las acciones de debe realizar el motor de ejecución en la búsqueda de los registros.

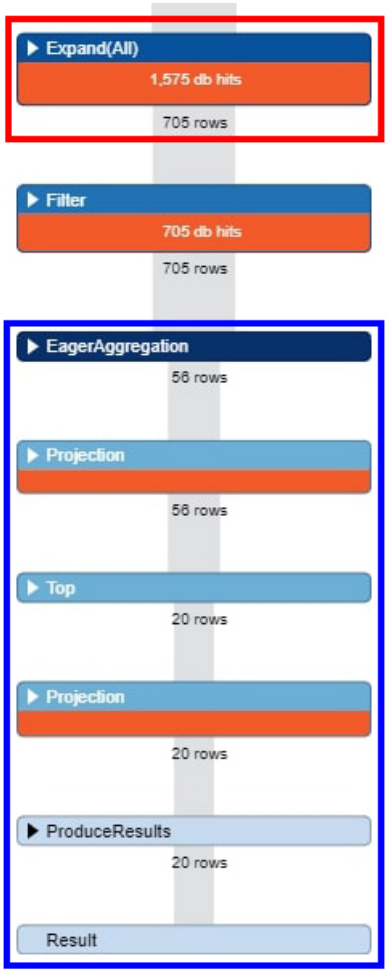


Figura A.7: Segmento Plan de Ejecución Solución Propuesta Consulta BI4.

A.3. Consulta BI5

El objetivo de la consulta BI5 es buscar los foros más populares dado un país, donde la popularidad del foro se mide por el número de miembros pertenecientes al país. Para cada miembro se debe contar el número de publicaciones en cualquiera de los 100 foros más populares. En la Figura A.4 podemos observar un segmento del plan de ejecución donde el punto crítico se encuentra en el primer recuadro. A continuación, se explican los operadores que se aprecian en este segmento.

1. Dos operadores *Argument* que es responsable del traslado de la variable *popularForums* que contiene los 100 foros más populares.

2. Cuatro operadores *Filter* los que tienen el objetivo de filtrar las personas que hayan publicado en un foro y que ese foro pertenezca a los 100 foros definidos. Además, de filtrar las personas que sean miembros de los foros.

3. Cuatro operadores *Expand* que buscan los miembros que se encuentren en los 100 foros más populares y que dichos miembros hayan publicado en estos foros.

4. Dos operadores *Apply* que validan que las publicaciones se encuentren de los foros más populares.

5. Un operador *Unwind* que permite transformar una lista en filas.

6. Tres operadores *EagerAggregation* los cuales se encargan de acceder a las propiedades de los nodos.

7. Dos operadores *Top* que retornan los primeros 100 resultados.

El primer recuadro enmarca el punto crítico de la consulta, al revisar el código de la consulta nos dimos cuenta de diferentes situaciones. La primera es que no nos permite incorporar estrategias de la primera guía de diseño dejando fuera la técnica de Propiedades Limitadas, debido a que la estructura necesaria para ser empleada se evidencia en la parte final de la consulta. El segundo caso corresponde a que existe sólo una condición, la cual valida que las personas pertenezcan a los foros que se encuentren dentro del grupo de los 100 más populares. Esto visibiliza que es una condición entre segmentos de registros de manera que no se pueden usar la primera o tercera guía de diseño. De igual forma las operaciones de agregación que existen son únicamente de conteo, de modo que nos impide modificar la estructura de la consulta. Finalmente, la estructura de la consulta se compone en base a las sentencias *MATCH* y *OPTIONAL MATCH* por ende la única solución posible es la Materialización de Caminos en conjunto de la de Propiedades Limitadas. La segunda guía de diseño es aplicada en la relación *HAS_TAG* de la segunda sentencia antes mencionada lo que nos propicia eliminar el segundo operador *EXPAND* y el filtro que acompaña a este operador. En la siguiente imagen podemos ver el código de la implementación de la segunda guía de diseño.

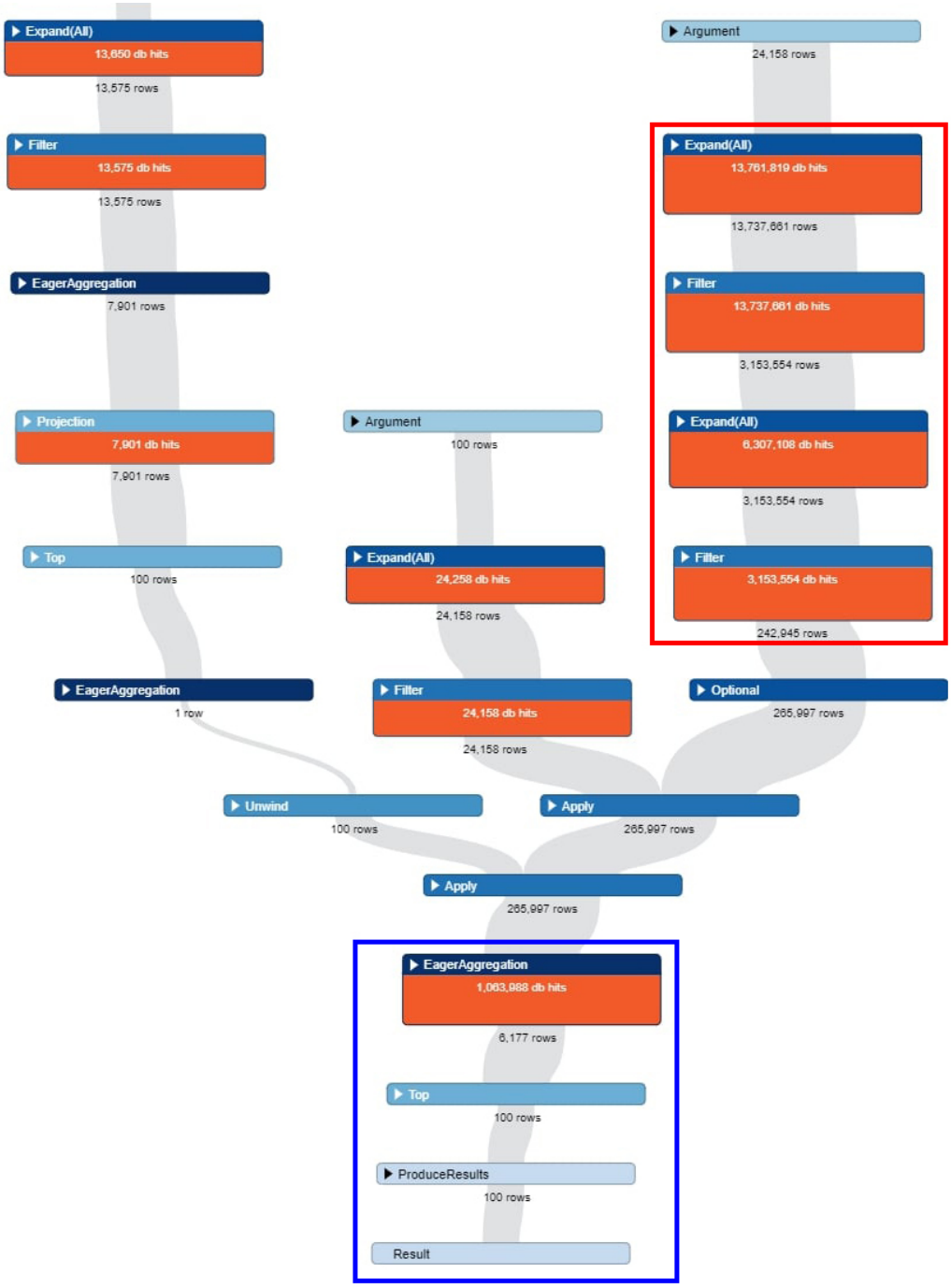


Figura A.8: Segmento Plan de Ejecución Original Consulta BI5.

```
MATCH
  (:Country {name: 'Belarus'})<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-
  (person:Person)<-[:HAS_MEMBER]-(forum:Forum)
WITH forum, count(person) AS numberOfMembers
ORDER BY numberOfMembers DESC, forum.id ASC
LIMIT 100
WITH collect(forum) AS popularForums
UNWIND popularForums AS forum
MATCH
  (forum)-[:HAS_MEMBER]->(person:Person)
MATCH
  (person)<-[:HAS_CREATOR]-(post:Post)<-[:CONTAINER_OF]-(popularForum:Forum)
WHERE popularForum IN popularForums
CREATE (person)<-[:BI5_post_person]-(post);
```

Figura A.9: Solución Propuesta Materialización de Caminos Consulta BI5.

Con respecto a la tercera estrategia de la primera guía de diseño se debe reestructurar el código original para lograr realizar en primera instancia el conteo de publicaciones para luego ser ordenados y acceder a sus propiedades. La aplicación de estrategia se aprecia en la Figura A.10.

```
RETURN
  person.id,
  person.firstName,
  person.lastName,
  person.creationDate,
  count(DISTINCT post) AS postCount
ORDER BY
  postCount DESC,
  person.id ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  person,
  count(DISTINCT post) AS postCount
ORDER BY
  postCount DESC,
  person.id ASC
LIMIT 100
RETURN
  person.id,
  person.firstName,
  person.lastName,
  person.creationDate,
  postCount
```

(b) Segmento Código Optimizado

Figura A.10: Solución Propuesta Propiedades Limitadas Consulta BI5.

Una vez aplicadas ambas estrategias de diseño físico conseguimos reducir considerablemente tanto el tiempo de ejecución y los DB Hits. Con la aplicación de la segunda guía de diseño que se aprecia en el primer recuadro logramos aminorar el costo de DB Hits en el primer operador y eliminar el operador *Filter* vinculado. Además, eliminamos completamente un camino del plan de ejecución disminuyendo las acciones y costos asociados. Con la implementación de la técnica Propiedades Limitadas vemos que en el segundo recuadro el costo de procesamiento del operador *EagerAggregation* se redujo a cero. De todas formas, se generó otro costo asociado en los operadores *Projection*, no obstante, este costo de

procesamiento corresponde al 0,1 % del costo original del operador *EagerAggregation*. A continuación, se expone el segmento del plan de ejecución con diseño físico en la Figura A.11.

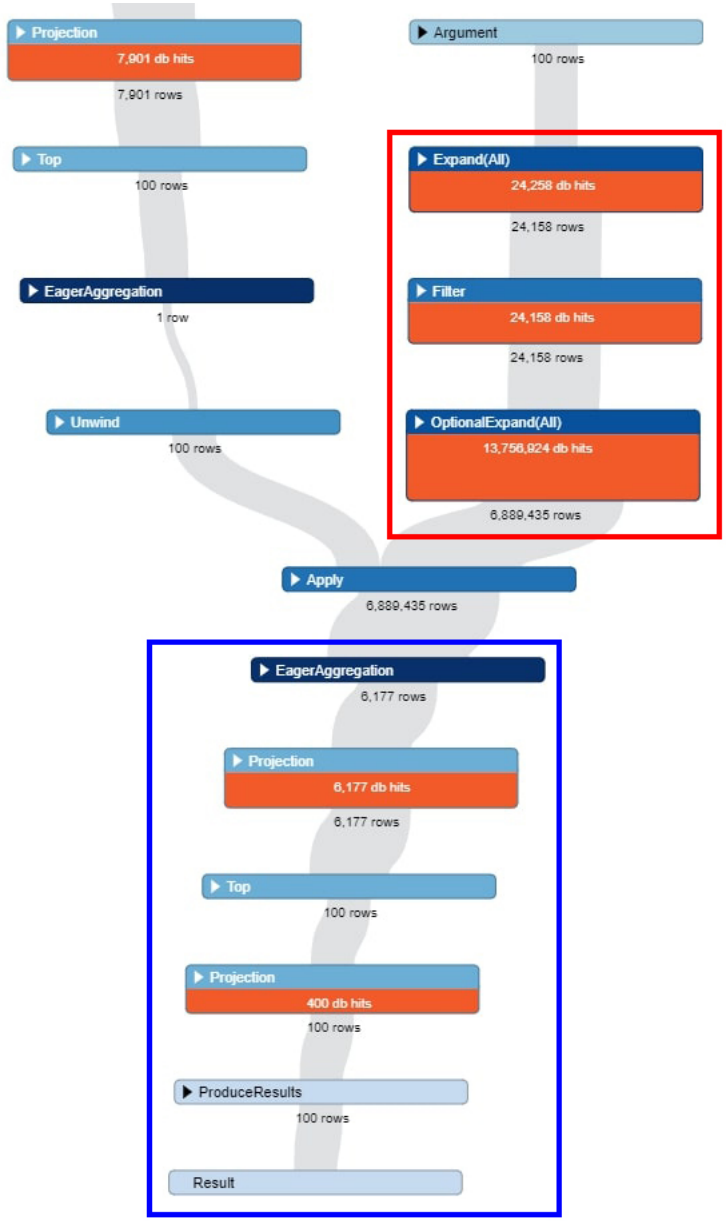


Figura A.11: Segmento Plan de Ejecución Solución Propuesta Consulta BI5.

A.4. Consulta BI6

En la consulta BI6 debemos localizar a las personas que hayan creado un mensaje con una etiqueta adjunta que es entregada por LDBC SNB. Adicionalmente, para cada mensaje por persona tenemos que efectuar las siguientes operaciones:

- Contar la cantidad de mensajes (*messageCount*).
- Contar la cantidad de *likes* de estos mensajes (*likeCount*).
- Contar los comentarios en respuesta a los mensajes (*replyCount*).

La puntuación es calculada de acorde a la siguiente formula:

$$1 * messageCount + 2 * replyCount + 10 * likeCount$$

En la Figura A.12 podemos apreciar el plan de ejecución cuyo punto crítico de la consulta se encuentra enmarcado en el recuadro. A continuación, se explican los operadores que se visualizan en el plan de ejecución.

1. Un operador *scan*
2. Tres operadores *Filter*.
3. Tres operadores *Expand* que tienen por objetivo ubicar a las personas que hayan creado mensajes que posean la etiqueta indicada.
4. Dos operadores *OptionalExpand* los que buscan las personas que les hayan dado *like* a los mensajes y a su vez los comentarios en respuesta a los mensajes.
5. Un operador *EagerAggregation* que ejecuta las operaciones de agregación.
6. Un operador *Projection* el que se ocupa de calcular la puntuación.
7. Un operador *Top* que retorna los primeros 100 resultados.

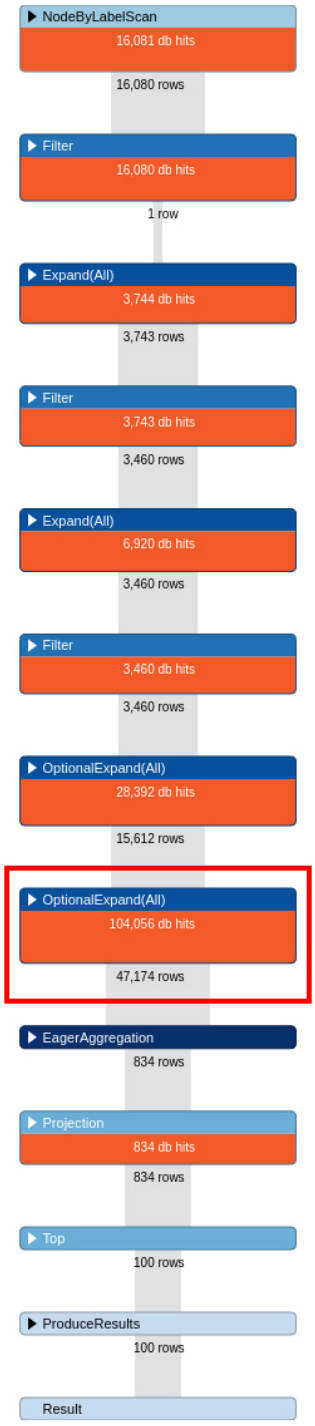


Figura A.12: Plan de Ejecución Original Consulta BI6.

Revisando el código de la consulta vemos que no existen condiciones ni operaciones aritméticas salvo el cálculo de la puntuación, imposibilitando el uso de las estrategias de la primera guía de diseño. Con respecto a la tercera guía de diseño se puede indicar que es factible su implementación en la etiqueta dada, no obstante, el costo de ese filtro es cercana a una sexta parte del costo del punto crítico, por ende, se desestima esta opción. En consecuencia, se decide optar por la segunda guía de diseño materializando la relación *REPLY_OF* que une los nodos *Message* y *Comment*. Con lo anterior conseguimos reducir notablemente el costo de procesamiento y el tiempo de ejecución. En la Figura A.13 podemos ver el código de la materialización de la relación antes mencionada.

```
MATCH (tag:Tag {name: 'Abbas_I_of_Persia'})<-[:HAS_TAG]-(message:Message)-[:HAS_CREATOR]->
 (person:Person)-[like:LIKES]->(message)<-[:REPLY_OF]-(comment:Comment)
CREATE (comment)-[:BI6_comment_message]->(message);
```

Figura A.13: Solución Propuesta Materialización de Caminos Consulta BI6.

El resultado de la segunda guía de diseño en el plan de ejecución es una considerable disminución de casi dos tercios del costo original de DB Hits que se encuentran el punto crítico identificado permitiéndonos acercar el costo de este operador al costo promedio que se evidencian en el resto de los operadores.

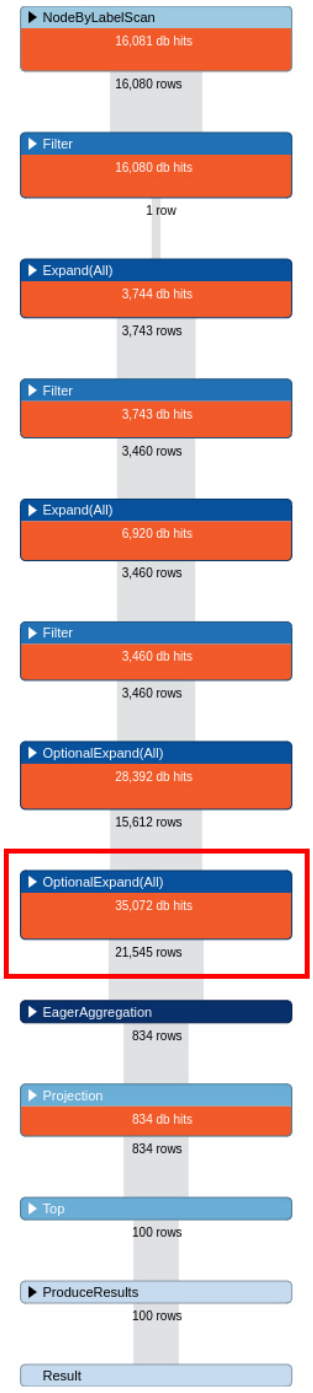


Figura A.14: Plan de Ejecución Solución Propuesta Consulta BI6.

A.5. Consulta BI8

La consulta BI8 establece la búsqueda de todos los mensajes que posean una etiqueta adjunta. Además de hallar las etiquetas relacionadas adjuntas a los comentarios de respuesta directa de estos mensajes, pero sólo de los comentarios de respuesta que no tienen la etiqueta asignada. Por último, tenemos que agrupar las etiquetas por nombre y obtener la cantidad de replicas por grupo. En la Figura A.15 contemplamos un segmento del plan de ejecución donde se enmarca el punto crítico de la consulta que corresponde al primer recuadro en el que se encuentra la relación directamente asociada. A continuación, se explican los operadores que se aprecian en este segmento.

1. Un operador *Argument* que retoma los valores de los nodos *Comment* y *Tag*.
2. Dos operadores *Filter* que se aplican sobre los nodos *Message* y *Comment*.
3. Tres operadores *Expand* que se aplican sobre las relaciones y con el objetivo de conseguir los comentarios de respuesta y los comentarios que no poseen la etiqueta asignada.
4. Un operador *AntiSemiApply* que valida que los comentarios dada la etiqueta no se incorporen.
5. Un operador *EagerAggregation* que se encarga de ejecutar las acciones necesarias para las operaciones de agregación.
6. Un operador *Top* que retorna los primeros 100 resultados.

Según el análisis efectuado la consulta BI8 tiene sólo una sentencia *MATCH* con una condición que verifica que los comentarios de respuesta no dispongan de la etiqueta. También se aprecia una de operación de agregación correspondiente a *count*. Producto del análisis podemos descartar las primeras dos estrategias de la primera guía de diseño, no obstante, podemos usar la tercera estrategia Propiedades Limitadas. La única opción donde se puede utilizar la tercera guía de diseño es en el nombre de la etiqueta, pero no nos resulta conveniente para abordar el punto crítico. En consecuencia, se decide usar la segunda guía de diseño sobre el primer recuadro, debido a que el punto crítico se sitúa en la condición de la sentencia *WHERE*. La materialización es aplicada sobre la relación *HAS_TAG* a causa de que es la operación más costosa dentro de la sentencia *MATCH*, por lo tanto, al materializar esta relación nos permite eliminar la condición consiguiendo reducir el número de DB Hits y a su vez eliminando completamente el punto crítico. La forma en que se aplica la segunda guía de diseño se puede ver en la Figura A.16.

Al igual como se define en este trabajo la implementación de la estrategia de Propiedades Limitadas que busca retrasar el acceso a las propiedades de los nodos. Es por esto que

Figura A.15: Segmento Plan de Ejecución Original Consulta BI8.

```
MATCH
  (tag:Tag {name: 'Genghis_Khan'})<-[:HAS_TAG]-(message:Message)
  <-[:REPLY_OF]-(comment:Comment)-[:HAS_TAG]->(relatedTag:Tag)
WHERE NOT (comment)-[:HAS_TAG]->(tag)
CREATE (comment)-[:BI8_comment_tag]->(relatedTag);
```

Figura A.16: Solución Propuesta Materialización de Caminos Consulta BI8.

podemos ver en la Figura A.17 la comparación del código sin diseño físico y con diseño físico, donde se modifica la estructura consiguiendo cumplir nuestro objetivo de retrasar específicamente la función *count*.

```
RETURN
  relatedTag.name,
  count(DISTINCT comment) AS count
ORDER BY
  count DESC,
  relatedTag.name ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  relatedTag,
  count(DISTINCT comment) AS count
ORDER BY
  count DESC,
  relatedTag.name ASC
LIMIT 100
RETURN
  relatedTag.name,
  count
```

(b) Segmento Código Optimizado

Figura A.17: Solución Propuesta Propiedades Limitadas Consulta BI8.

El uso de la segunda guía de diseño nos permite eliminar la condición y reducir el número de caminos de dos a uno como se aprecia en el plan de ejecución. De esta forma se elimina el punto crítico y además se reduce la búsqueda dos nodos y una relación que corresponde a la relación ya materializada. Con respecto a la estrategia aplicada vemos que el costo de DB Hits del operador *EagerAggregation* se reduce a cero y una parte de este se redistribuye al primer operador *Projection* como se puede ver en la Figura A.18.

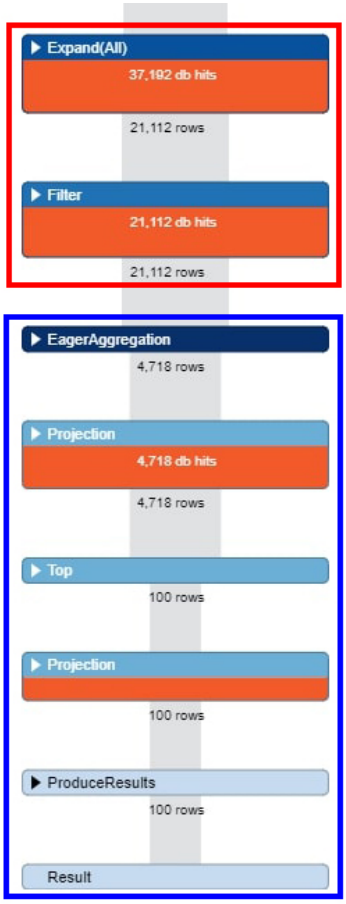


Figura A.18: Segmento Plan de Ejecución Solución Propuesta Consulta BI8.

A.6. Consulta BI9

Para la consulta BI9 se establece que dadas dos clases de etiquetas se deben buscar foros que cumplan con que:

- Al menos una publicación (*post1*) con una etiqueta que pertenezca directamente a la clase de etiqueta entregada (tagClass1).
- Al menos una publicación (*post2*) con una etiqueta que pertenezca directamente a la clase de etiqueta entregada (tagClass2).

Teniendo presente que el *post1* y el *post2* deben ser de la misma publicación y que además se debe considerar que los foros posean una número de miembros mayor al indicado, donde para cada foro se debe contar la cantidad de *post1* y *post2*. Una vez definida la

especificación de la consulta podemos ver que en la Figura A.19 se presenta el plan de ejecución de la consulta BI9. El recuadro número uno se sitúa el punto crítico que corresponde al primer operador *EXPAND* y su respectivo filtro en el operador *FILTER*. A continuación, se especifican los siguientes operadores que componen la Figura A.19:

1. Un operador *scan* para obtener los nodos *Person*.
2. Ocho operadores *Filter* que se ocupan de filtrar las publicaciones pertenecientes a los fotos, las etiquetas de esas publicaciones y que las etiquetas pertenezcan a la clase de la etiqueta.
3. Ocho operadores *Expand* responsables de desplegar las relaciones de los nodos *Forum Person*, *Tag* y *TagClass*.
4. Tres operadores *EagerAggregation* que se encarga de ejecutar las acciones necesarias para las operaciones de agregación.
5. Dos operadores *Projection* que se encargan de ordenar los registros.
6. Un operador *Top* que retorna los primeros 100 resultados.

Estudiando el código de la consulta BI9 apreciamos que existe una condición y hace referencia al número de miembros con la que cuenta cada foro. De la misma forma, nos damos cuenta que la única operación de agregación corresponde al conteo de la cantidad miembros y publicaciones distintas para el *post1* y *post2*, respectivamente. Al no contar con operaciones de agregación u operaciones aritméticas en las que tengamos la posibilidad de modificar la estructura de la consulta tenemos que descartar la opción de la primera guía de diseño. Por otro lado, la tercera guía de diseño puede ser aplicada sobre el nodo *TagClass* y su propiedad *name*, no obstante, no cumplimos con el objetivo de resolver el inconveniente del punto crítico. Como se indicaba anteriormente el punto crítico se encuentra en el primer operador *EXPAND* de la consulta que proviene de la relación *HAS_TYPE* perteneciente a los nodos *Tag* y *TagClass*. Se decide aplicar la segunda guía de diseño en el punto crítico que se ubica en el segundo operador *EXPAND* que es la operación de menor costo. Su implementación nos posibilita eliminar tanto el punto crítico como también el tercer y cuarto operador *EXPAND* al igual que sus respectivos operadores *FILTER*, incluyendo la condición y búsqueda asociada a los foros que cuenten con más de 200 miembros. La forma en que se implementó la segunda guía de diseño físico se puede ver en la Figura A.20.

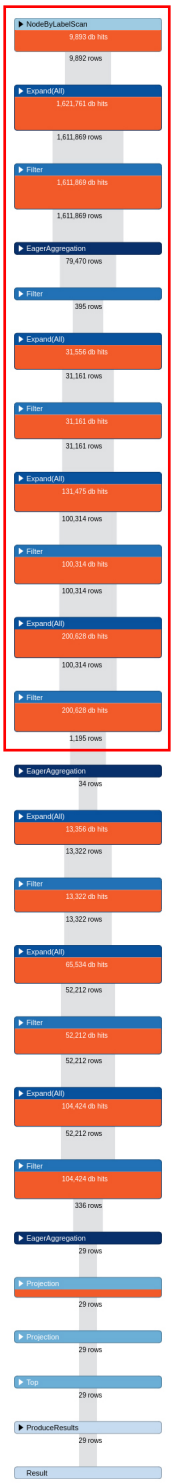


Figura A.19: Plan de Ejecución Consulta BI9.

```
MATCH
  (forum:Forum)-[:HAS_MEMBER]->(person:Person)
WITH forum, count(person) AS members
WHERE members > 200
MATCH
  (forum)-[:CONTAINER_OF]->(post1:Post)-[:HAS_TAG]->
  (:Tag)-[:HAS_TYPE]->(:TagClass {name: 'BaseballPlayer'})
CREATE (forum)-[:BI9_forum_post1]->(post1);
```

Figura A.20: Solución Propuesta Materialización de Caminos Consulta BI9.

Al revisar el plan de ejecución resultante podemos darnos cuenta que el primer recuadro se reduce a dos operadores consiguiendo una gran optimización que se ve reflejado en la disminución de DB Hits por ello también se reduce el tiempo de ejecución. Lo antes descrito se puede apreciar en la Figura A.21.

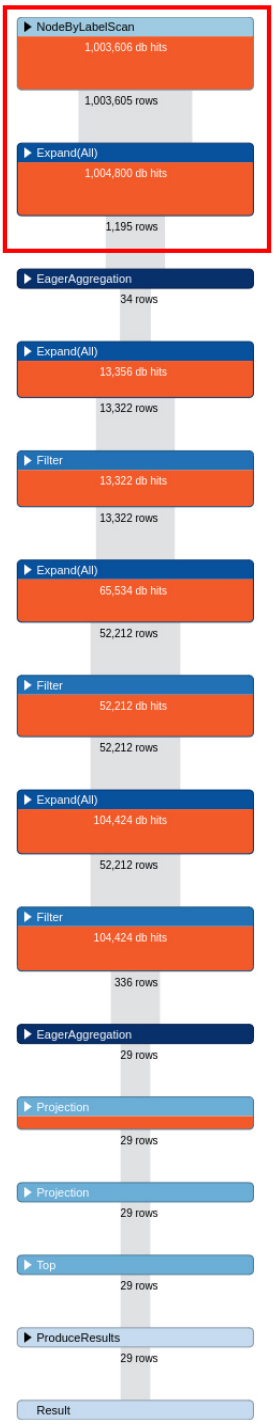


Figura A.21: Plan de Ejecución Solución Propuesta Consulta BI9.

A.7. Consulta BI10

En el documento de especificación se establece para la consulta BI10 que para una etiqueta dada, se deben buscar a todas las personas que se encuentren interesadas en la etiqueta y/o hayan escrito un mensaje (*Comment* o *Post*) después de una fecha indicada y que posea dicha etiqueta. Para cada persona se debe calcular una puntuación que corresponde a la suma de los siguientes dos factores:

- Asignar un valor de 100 si la persona tiene o se interesa en la etiqueta, y de no ser así entregar un valor de 0.
- La cantidad de mensajes que tenga esa persona con esa etiqueta.

Además, para cada persona, se debe sumar la puntuación de que hayan obtenido sus amistades. En la Figura A.22 podemos ver un segmento del plan de ejecución de la consulta BI10 en los cuales se mencionan los operadores más relevantes.

1. Dos operadores *EagerAggregation* que se responsabiliza de efectuar las operaciones de agregación.
2. Un operador *Unwind* que permite transformar una lista en filas.
3. Un operador *Distinct* cuya función es validar que las etiquetas y personas sean diferentes.
4. Cuatro operadores *Argument* que retorna las personas que hayan cumplido con las condiciones.
5. Seis operadores *Projection* que se ocupan de realizar los cálculos necesarios para obtener la puntuación.
6. Un operador *OptionalExpand* que debe ubicar las amistades de la persona.
7. Siete operadores *Expand* que mediante las relaciones realizan los diferentes cálculos para conseguir la puntuación de la persona y de sus amistades.
8. Un operador *Top* que retorna los primeros 100 resultados.

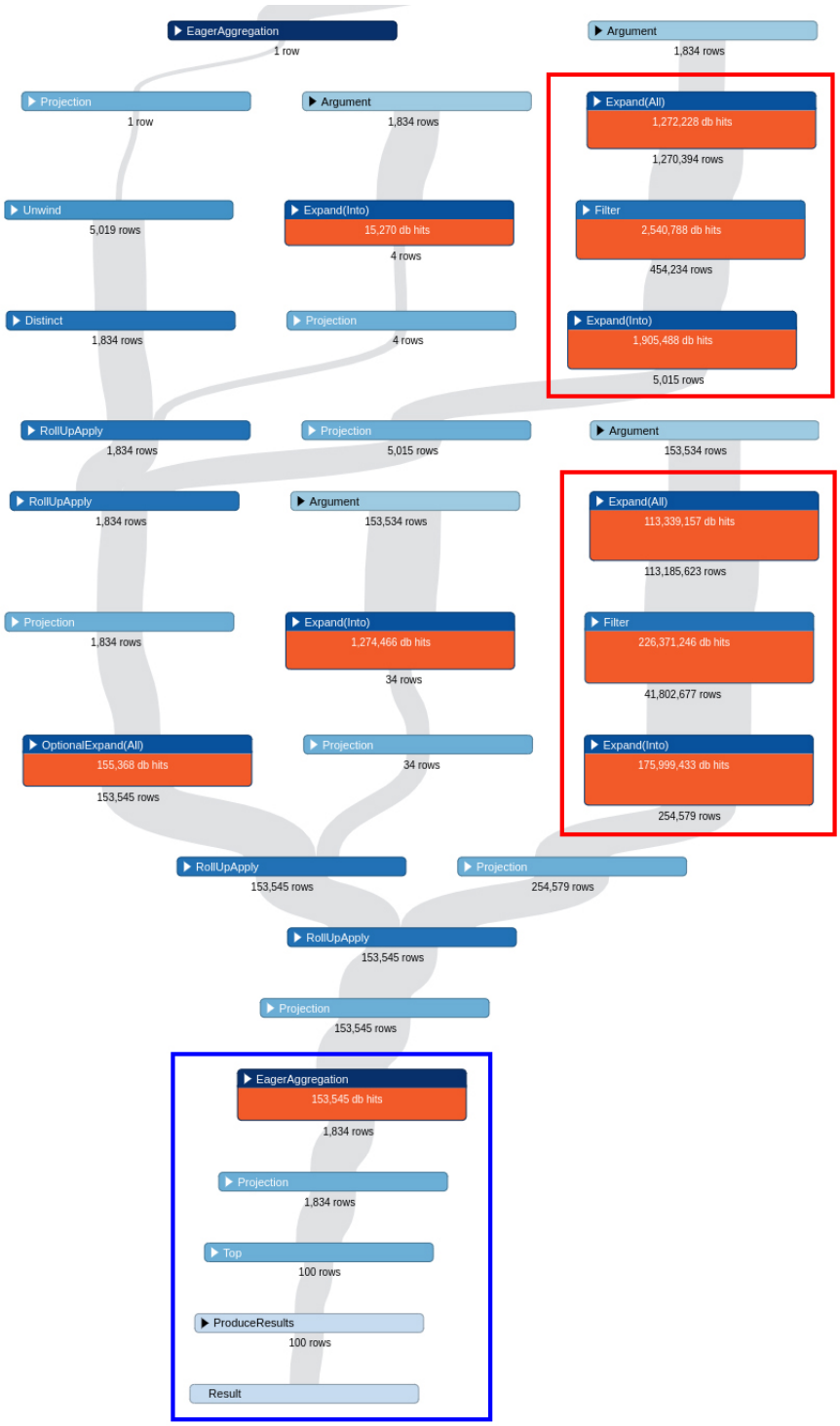


Figura A.22: Segmento Plan de Ejecución Consulta BI10.

Al revisar el código de la consulta BI10 podemos darnos cuenta que se compone de una sentencia *MATCH* y tres sentencias *OPTIONAL MATCH*. La primera está orientada en la búsqueda de las etiquetas indicadas por el documento de especificación. Sobre las tres sentencias *OPTIONAL MATCH* podemos ver que la primera se encarga de obtener las personas que le interesa la etiqueta encontrada, cuyos resultados son almacenados en una lista. La segunda sentencia busca las personas que hayan creado al menos un mensaje con esa etiqueta desde una fecha indicada, este filtro lo realiza mediante una condición en una sentencia *WHERE* para almacenar el resultado en una lista y sumar ambas. Por último, la tercera sentencia ratifica que se conozcan las personas que se encuentran en la lista y sus amistades. Con respecto a las sentencias *WITH* nos encontramos con cinco casos en los cuales se trabaja con la listas, se verifica que las etiquetas sean distintas y el cálculo de la puntuación para las personas y sus amistades. En la parte final del código podemos ver las sentencias *ORDER BY* y *LIMIT*. Una vez analizado el código de la consulta podemos indicar acerca de la primera guía de diseño que se puede implementar exclusivamente la estrategia de Propiedades Limitadas ya que la consulta no presenta redundancia, esto se enmarca en el tercer recuadro de color azul. También nos percatamos que en la búsqueda de las personas y amistades no podemos reutilizar variables para eliminar una supuesta redundancia que en este caso no existe. La tercera guía de diseño puede ser aplicada en las propiedades *name* o *creationDate* de los nodos *Tag* y *Message*, respectivamente. De igual forma se evalúa obteniendo una pequeña disminución en relación con los costos de procesamiento iniciales tomando la decisión de descartar esta guía de diseño. Al revisar minuciosamente el plan de ejecución vemos que el punto crítico se encuentra en una búsqueda al momento de realizar el cálculo de la puntuación de las amistades correspondiente al segundo recuadro. En ese recuadro el punto crítico se halla en la validación de la fecha de creación del mensaje, por lo tanto, se propone materializar la relación del primer operador de este recuadro. Este operador presenta los nodos *Message* y *Person* que son unidos por la relación *HAS_CREATOR*, integrando la sentencia *WHERE* y su respectivo condicionante que se aprecia en la Figura A.23.

```
MATCH
  (tag:Tag {name: 'John_Rhys-Davies'})<-[:HAS_TAG]-(message:Message)
  -[:HAS_CREATOR]->(person:Person)
WHERE
  message.creationDate > 20120122000000000
CREATE (message)-[:BI10_message_person]->(person);
```

Figura A.23: Solución Propuesta Materialización de Caminos Consulta BI10.

Como se indicaba anteriormente existe la posibilidad de emplear una estrategia de la primera guía de diseño ya que se presenta la función *sum* dentro de la sentencia *RETURN* como se ve en la Figura A.24a permitiéndonos de esta forma modificar la estructura apartando esta función y ubicándola en una sentencia *WITH* que corresponde al código

optimizado situado en la Figura A.24b.

```
RETURN
  person.id,
  score,
  sum(friendScore) AS friendsScore
ORDER BY
  score + friendsScore DESC,
  person.id ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  person,
  score,
  sum(friendScore) AS friendsScore
ORDER BY
  score + friendsScore DESC,
  person.id ASC
LIMIT 100
RETURN
  person.id,
  score,
  friendsScore
```

(b) Segmento Código Optimizado

Figura A.24: Solución Propuesta Propiedades Limitadas Consulta BI10.

Una vez implementadas ambas estrategias vemos en el plan de ejecución que al materializar la relación logramos reducir significativamente el costo de DB Hits del primer operador, donde conseguimos remover el filtro del segundo operador y el tercer operador ya que la búsqueda se redujo a una relación. Al analizar el plan de ejecución también nos damos cuenta que ambas operaciones de cálculo de puntuación son idénticas como se ve en el primer recuadro enmarcado en el plan de ejecución. En consecuencia, esto nos permite reutilizar la relación materializada obteniendo una disminución del costo de procesamiento del primer operador y la eliminación de los dos operadores restantes. En la Figura A.25 se puede observar lo antes descrito.

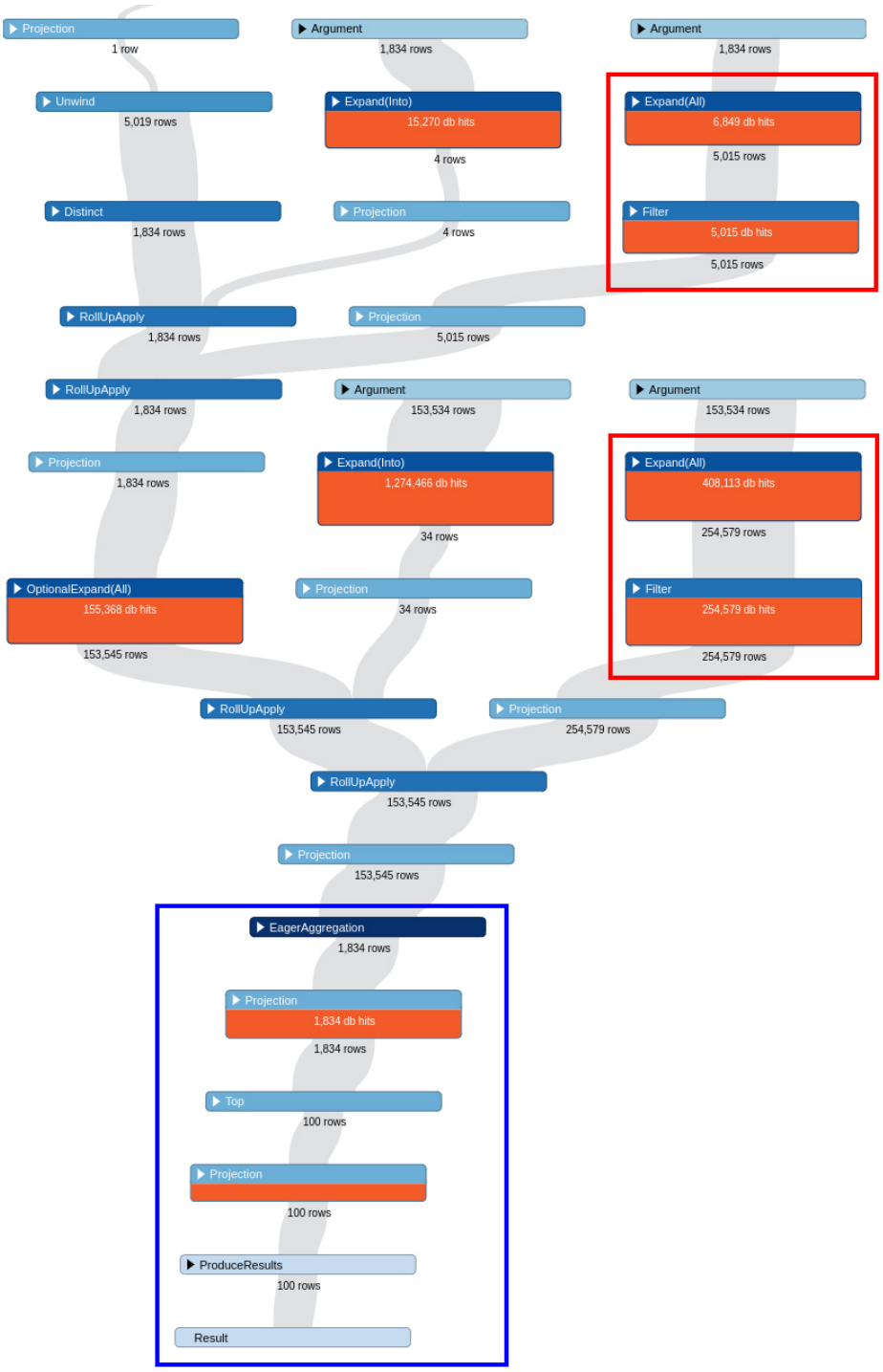


Figura A.25: Segmento Plan de Ejecución Solución Propuesta Consulta BI10.

A.8. Consulta BI11

La consulta BI11 tiene por objetivo encontrar a las personas de un país determinado que respondieron a cualquier mensaje, de modo que la respuesta no tenga ninguna etiqueta en común con el mensaje. Considerar sólo las respuestas que no contienen ninguna palabra de una lista negra dada. Para cada Persona y respuesta válida, se deben retornar las etiquetas asociadas con la respuesta y además devolver el número de *likes* en la respuesta. Las condiciones para verificar las palabras en la lista negra son actualmente las siguientes:

- Las palabras no tienen que estar separadas, es decir, si la palabra "Green" está en la lista negra, "South-Greenland" no puede incluirse en los resultados.
- La comparación debe hacerse de forma sensible a mayúsculas y minúsculas.

Una vez definida la consulta pasamos a revisar un segmento del plan de ejecución original que se encuentra en la Figura A.26. Allí el punto crítico se sitúa en la segundo camino que se enmarca en el primer recuadro precisamente en el operador *Expand (Into)*. Por otro lado, debemos hacer hincapié en el segundo recuadro puesto que genera un gran costo de procesamiento en las operaciones de agregación de la consulta. A continuación, se explican los operadores más importantes que se aprecian en este segmento.

1. Un operador *Argument* para los nodos mensajes y réplicas.
2. Seis operadores *Filter* de trabajar los mensajes en referencia a las restricciones de etiqueta, mensaje, país y persona.
3. Cinco operadores *Expand* de los cuales dos se destinan para la búsqueda de los mensajes y los restantes para la búsqueda de los mensajes que posen alguna palabra de la lista negra.
4. Un operador *AntiSemiApply* para descartar los mensajes que posean una palabra perteneciente a la lista negra.
5. Un operador *OptionalExpand* para validar que las respuesta no contengan alguna de las palabras de la lista negra.
6. Un operador *EagerAggregation* que se encarga de ejecutar las acciones necesarias para las operaciones de agregación.
7. Un operador *Top* que retorna los primeros 100 resultados.

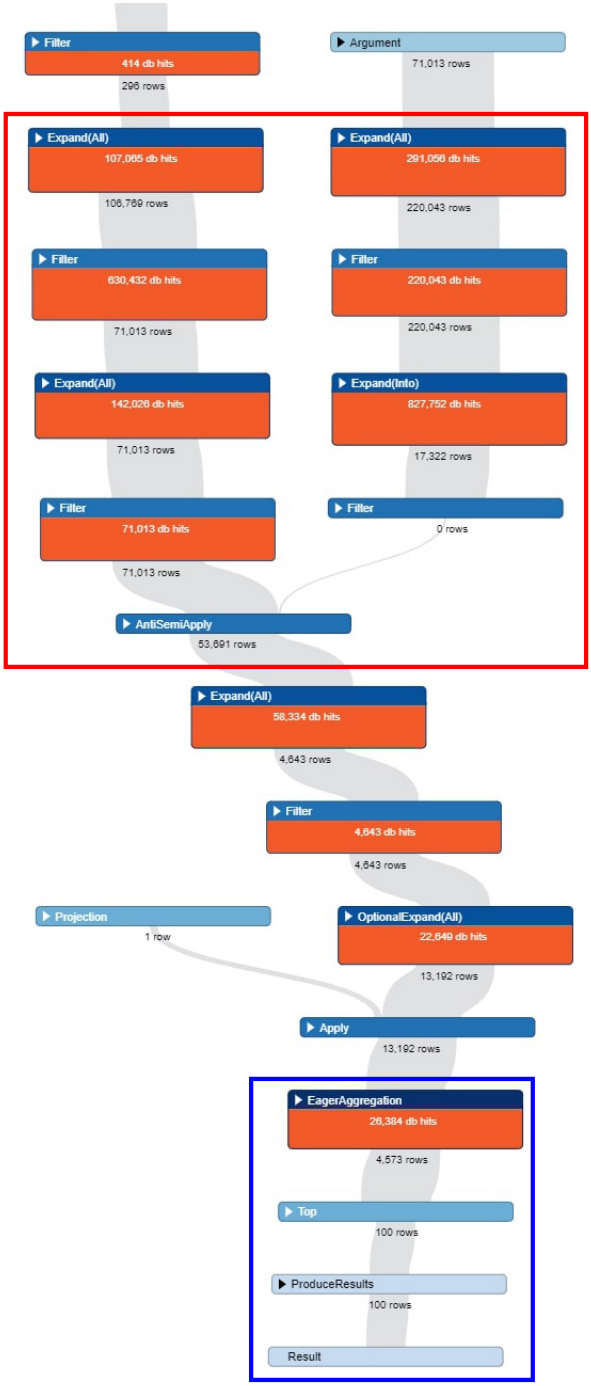


Figura A.26: Segmento Plan de Ejecución Original Consulta BI11.

El código posee una sentencia *MATCH* y *OPTIONAL MATCH*, donde la primera tiene dos condiciones en la sentencia *WHERE*. La primera de ellas trata el requerimiento de que la respuesta no tenga ninguna etiqueta en común con el mensaje, en cambio, la segunda verifica que las réplicas no contengan una palabra de la lista negra. Acerca de la sentencia *OPTIONAL MATCH* se obtienen los *likes* que hayan recibido las réplicas a los mensajes. Al final de la consulta podemos identificar dos operadores *count* en la sentencia *RETURN*. La tercera guía de diseño puede ser aplicada sobre el nombre de las etiquetas o el nombre de las clases de etiqueta, no obstante, ninguna de las dos resuelve el inconveniente con el punto crítico por lo que no son considerada. La segunda guía de diseño propone materializar la relación *REPLY_OF* que corresponde al primer camino del primer recuadro. Esto nos permite eliminar las condiciones que se buscan en la segunda columna, por lo tanto, se lograría resolver el problema del punto crítico. Finalmente, se ve restringida la implementación de la primera guía de diseño ya que no se aprecian operaciones de agregación ni operaciones aritméticas salvo la función *count* en dos oportunidades. Al contar con una sentencia *MATCH* se ve limitada la aplicación de la estrategia de Descomposición de Consulta. En referencia a la estrategia de Propiedades Limitadas observamos que si se cumplen las condiciones para incorporarla al código. La aplicación de la de segunda guía de diseño se observa en la Figura A.27 según las observaciones antes indicadas.

```
WITH ['also', 'Pope', 'that', 'James', 'Henry', 'one', 'Green'] AS blacklist
MATCH
  (country:Country {name: 'Germany'})<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-
  (person:Person)<-[:HAS_CREATOR]-(reply:Comment)-[:REPLY_OF]->(message:Message),
  (reply)-[:HAS_TAG]->(tag:Tag)
WHERE NOT (message)-[:HAS_TAG]->(:Tag)<-[:HAS_TAG]-(reply)
  AND size([word IN blacklist WHERE reply.content CONTAINS word | word]) = 0
CREATE (reply)-[:BI11_reply_person]->(person);
```

Figura A.27: Solución Propuesta Materialización de Caminos Consulta BI11.

El uso de la estrategia de Propiedades Limitadas se aprecia en la Figura A.28 la que se constituye de dos figuras que nos permiten efectuar la comparación del código sin diseño físico y con diseño físico. Se puede ver que se reubican los operadores *count* en la sentencia *WITH* permitiendo cambiar la posición de la sentencia *RETURN* como se aprecia a continuación.

```
RETURN
  person.id,
  tag.name,
  count(DISTINCT like) AS countLikes,
  count(DISTINCT reply) AS countReplies
ORDER BY
  countLikes DESC,
  person.id ASC,
  tag.name ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  tag,
  person,
  count(DISTINCT like) AS countLikes,
  count(DISTINCT reply) AS countReplies
ORDER BY
  countLikes DESC,
  person.id ASC,
  tag.name ASC
LIMIT 100
RETURN
  person.id,
  tag.name,
  countLikes,
  countReplies
```

(b) Segmento Código Optimizado

Figura A.28: Solución Propuesta Propiedades Limitadas Consulta BI11.

La implementación de ambas estrategias generan un impacto positivo como se ve en el plan de ejecución con diseño físico. Debido al uso de la segunda guía de diseño logramos remover completamente el segundo camino que nos generaba el mayor costo de DB Hits de la consulta. Igualmente, el primer camino ubicado en el primer recuadro del plan de ejecución original se redujo a dos operadores. En referencia a la estrategia de Propiedades Limitadas podemos ver que elimina el 66 % aproximadamente del costo de DB Hits del operador *EagerAggregation* redistribuyendo en los operadores *Projection* como se visualiza en el segundo recuadro enmarcado de la Figura A.29.

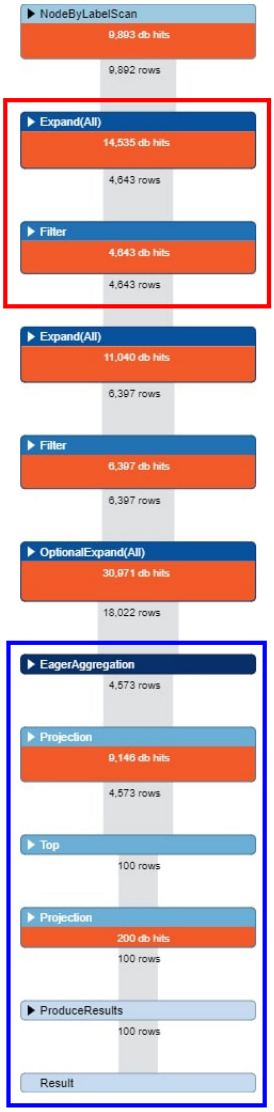


Figura A.29: Segmento Plan de Ejecución Solución Propuesta Consulta BI11.

A.9. Consulta BI12

La consulta BI12 establece que debemos buscar todos los mensajes creados después de una fecha indicada, estos mensajes deben haber recibido más de un número determinado de *likes* con la finalidad de entregar el nombre de sus creadores y el número de *likes* recibidos. En la Figura A.30 podemos observar el plan de ejecución original donde se halla el punto crítico en el recuadro enmarcado. A continuación, se explican los operadores que

se aprecian en este segmento.

1. Un operador *scan* que busca en primera instancia los nodos *Person*.
2. Tres operadores *Filter* cuyo objetivo es filtrar los mensajes dada la fecha y que cumplan con que posean una cantidad mayor a la solicitada.
3. Dos operadores *Expand* sobre las relaciones *HAS_CREATOR* y *LIKES* los que nos permiten obtener las personas que hayan creado los mensajes que cumplen con la condición.
4. Un operador *EagerAggregation* que se encarga de ejecutar las acciones necesarias para las operaciones de agregación.
5. Dos operadores *Projection*
6. que se ocupan de realizar los cálculos necesarios para obtener las propiedades de los nodos.
7. Un operador *Top* que retorna los primeros 100 resultados.

En la consulta BI12 existen dos condiciones *WHERE* y un operador de agregación, siendo la condición de la fecha nuestro punto crítico. Sin embargo, la operación de agregación está vinculada a la segunda condición la cual valida la cantidad de *likes*. En referencia a la tercera guía de diseño se puede emplear en la propiedad *creationDate* del nodo *Message* cumpliendo el fin de resolver el impedimento del punto crítico. No obstante, al aplicar esta estrategia nos genera un nuevo plan de ejecución en el cual se observa que su beneficio es casi nulo, por lo tanto, se rechaza esta alternativa. La posibilidad que nos queda es la segunda guía de diseño la cual se aplica sobre la relación *HAS_CREATOR*, esta nos permite reducir considerablemente el costo de procesamiento de la operación que filtra los mensajes por fecha. Asimismo, se ven afectados positivamente el primer y tercer recuadro e inclusive el operador de filtro que se encuentra posterior al tercer recuadro. Por lo tanto, al materializar la relación conseguimos reducir los costos de procesamiento para cuatro operadores. El código que materializa la nueva relación aplicando la segunda guía de diseño se puede ver a continuación.

En el primer recuadro se aprecia una disminución significativa y es responsabilidad de la segunda guía de diseño. Lo que vemos en este recuadro es consecuencia de establecer bajo una relación materializada sólo los registros que requerimos consiguiendo así pasar de un conjunto de datos a el subconjunto que posee los registros necesarios para avanzar en la búsqueda de las filas que cumplan con todas las condiciones. Acerca el uso de la estrategia de Propiedades Limitadas vemos que no se genera ningún cambio como se había indicado, no obstante, al momento de aumentar el volumen de datos lograremos trabajar

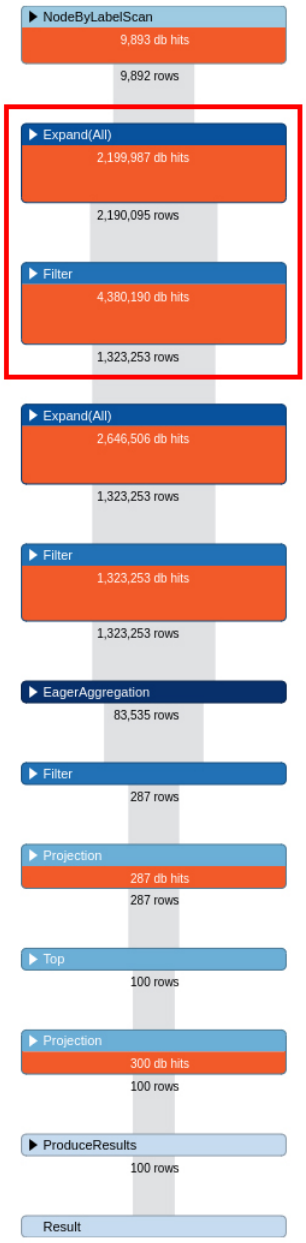


Figura A.30: Plan de Ejecución Original Consulta BI12.

```
MATCH
  (message:Message)-[:HAS_CREATOR]->(creator:Person),
  (message)<-[like:LIKES]-(:Person)
WHERE message.creationDate > 20110721220000000
WITH message, creator, count(like) AS likeCount
WHERE likeCount > 400
CREATE (message)-[:BI12_message_creator]->(creator);
```

Figura A.31: Solución Propuesta Materialización de Caminos Consulta BI12.

en primera instancia con el subconjunto de datos y luego de tener los resultados acceder a sus propiedades reduciendo el costo de procesamiento. Todo lo antes descrito se puede observar en el plan de ejecución con ambas estrategias en la Figura A.32.

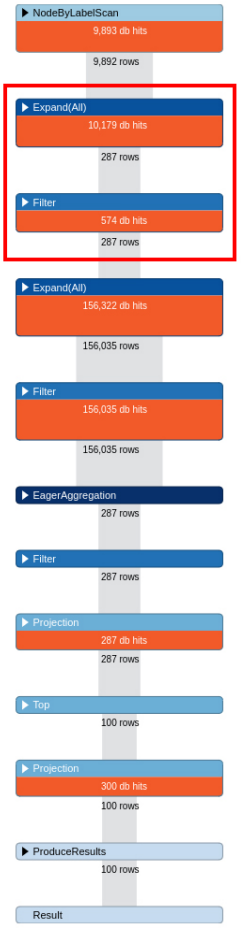


Figura A.32: Plan de Ejecución Solución Propuesta Consulta BI12.

A.10. Consulta BI13

El documento de especificación menciona que se deben hallar todos los mensajes dado un país y sus etiquetas. Los mensajes deben ser agrupados por año y mes de creación. Para cada grupo, se deben encontrar las 5 etiquetas más populares, donde la popularidad es la cantidad de mensajes en la que aparece la etiqueta. Podemos ver un segmento del plan de ejecución en cual el punto crítico de la consulta se encuentran en el operador *Projection* correspondiente al recuadro enmarcado en la Figura A.33. A continuación, se explican los operadores que se aprecian en el plan de ejecución.

1. Un operador *scan* para el nodo *Country*.
2. Dos operadores *Filter* para filtrar los registros según el país, año y mes.
3. Un operador *Expand* que se encarga de retornar los mensajes pertenecientes al país indicado.
4. Un operador *OptionalExpand* que debe retornar las etiquetas pertenecientes a esos mensajes.
5. Cinco operadores *Projection* que se encargan de efectuar las diferentes operaciones de agregación y operaciones aritméticas.
6. Un operador *Sort* que ordena los registros.
7. Un operador *EagerAggregation* que se encarga de ejecutar las acciones necesarias para las operaciones de agregación.
8. Un operador *Top* que retorna los primeros 100 resultados.

La estructura de la consulta BI13 se conforma dos tipos de sentencias que son *MATCH* y *OPTIONAL MATCH*, donde se relacionan cuatro sentencias *WITH* para el trabajo sobre los registros. Habiendo identificado la composición de la consulta descartamos de plano el uso de la tercera guía de diseño, debido a que existen dos propiedades candidatas: *name* del nodo *Country* y *creationDate* del nodo *Message*. Al evaluar su uso percibimos que no aborda el punto crítico e incluso el beneficio es casi nulo, es por esta razón que se descarta su aplicación. Con respecto a la primera guía de diseño el código presenta un alto número de operaciones de agregación, pero existe una complicación dado que estas operaciones se efectúan con la finalidad de crear una lista que almacene las etiquetas más populares. Esto significa que cada operación es requerida para el siguiente cálculo dejándonos sin la posibilidad de optimizar este proceso. Sin otra alternativa nos queda estudiar la segunda guía de diseño, esta se debe aplicar en el recuadro del plan de ejecución cuya relación es *HAS_TAG* que vincula los nodos *Message* y *Tag*. La materialización de

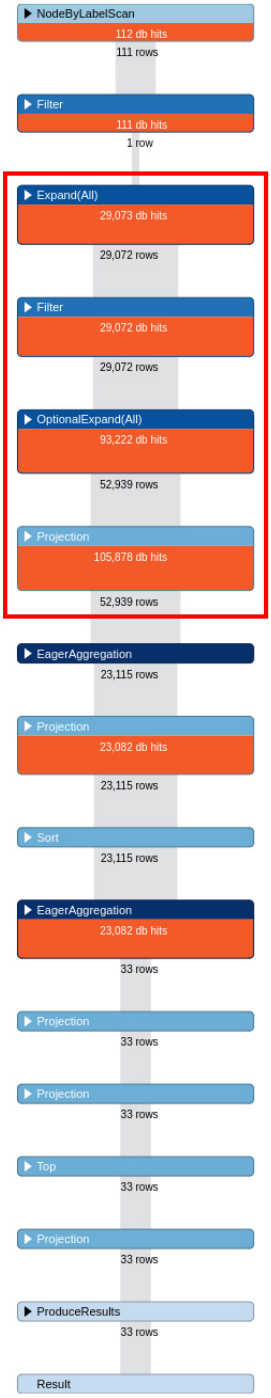


Figura A.33: Plan de Ejecución Consulta BI13.

esta relación nos retorna la eliminación de todas las operaciones previas salvo la operación *scan* que modifica su parámetro al nodo *Tag*. Pero aún se mantiene el punto crítico, no obstante, la materialización nos facilita reducir el costo de DB Hits a tan sólo 1/6 del costo inicial. La implementación de la segunda guía de diseño se aprecia a continuación.

```
MATCH
  (c:Country)<-[:IS_LOCATED_IN]-(message:Message)-[:HAS_TAG]->(tag:Tag)
WHERE c.name = 'Burma'
CREATE (message)-[:BI13_message_tag]->(tag);
```

Figura A.34: Solución Propuesta Materialización de Caminos Consulta BI13.

Lo que vemos en el recuadro del plan de ejecución con diseño físico es consecuencia de establecer bajo una relación materializada sólo los registros requeridos consiguiendo así pasar de un conjunto de datos a el subconjunto que posee los registros necesarios para avanzar en la búsqueda de las filas que cumplan con todas las condiciones. Lo descrito se puede observar en el plan de ejecución de la Figura A.35.

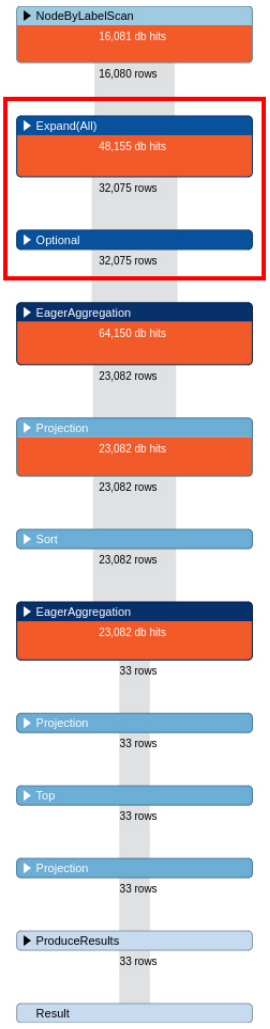


Figura A.35: Plan de Ejecución Solución Propuesta Consulta BI13.

A.11. Consulta BI14

La consulta BI14 establece que para cada persona debemos contar el número de publicaciones en un intervalo de tiempo y la cantidad de réplicas. El resultado de la consulta debe retornar el número de publicaciones y el recuento de todas las réplicas que aparecieron en los árboles de respuesta (incluida la publicación en la raíz del árbol) que creó cada persona. En la Figura A.36 se observan tres recuadros, el primero que corresponde al punto crítico, el segundo que posee relación con este y en el tercero se presenta otra opción a optimizar. A continuación, se especifican los operadores que se presentan en el plan de

ejecución.

1. Un operador *scan* para obtener las publicaciones.
2. Tres operadores *Filter* para que los mensajes y publicaciones se sitúen dentro del intervalo de tiempo.
3. Un operador *VarLengthExpand* que explora la cantidad de réplicas a cada publicación.
4. Un operador *Expand* que obtiene las personas, publicaciones y mensajes (réplicas).
5. Un operador *EagerAggregation* donde se cuantifica las publicaciones y mensajes validando que sean diferentes.
6. Un operador *Top* que retorna los 100 primeros registros.

Analizando el código de la consulta BI14 nos damos cuenta que se compone de una sola sentencia *MATCH* y cuatro condiciones mediante la función *WHERE*. Asimismo, en el fragmento final se visualiza un operador *EagerAggregation* con un alto costo en DB Hits. En el plan de ejecución podemos notar que en el primer recuadro se localiza el punto crítico, el cual corresponde a los filtros del intervalo de tiempo de las publicaciones. Al ser tan reducido el código la única opción que podemos usar de la primera guía de diseño corresponde a la técnica Propiedades Limitadas, donde mediante el uso de esta conseguimos reducir el operador *EagerAggregation*. Acerca de la tercera guía de diseño vemos que puede ser implementada en la propiedad del nodo *Post* obteniendo el mismo resultado que nos permite eliminar el punto crítico, a esto se le suma la reducción del costo de procesamiento del operador *scan* llegando a tan sólo el 5 % del costo original. En cuanto a la segunda guía de diseño, esta se puede utilizar en la relación *HAS_CREATOR* de los nodos *Person* y *Post* ya que los filtros se aplican sobre la propiedad *creationDate* del segundo nodo. Esta relación se halla delimitada en el segundo recuadro del plan de ejecución, con su uso logramos eliminar directamente el punto crítico y a su vez reducir el costo de procesamiento de ambos operadores del segundo recuadro. Después de un exhaustivo análisis se decide incorporar la tercera estrategia de la primera guía de diseño y la segunda guía de diseño, donde esta última si bien consigue un resultado semejante al de la tercera guía de diseño, sabemos también que existe una diferencia ya que reduce el costo de procesamiento de los operadores del segundo recuadro. En la Figura A.37 apreciamos la aplicación de la segunda guía de diseño.

La implementación de la tercera estrategia de la primera guía de diseño se puede ver en la Figura A.38. En primer lugar observamos el código original en la Figura A.38a, donde nos damos cuenta que existen dos funciones *count* los cuales son el motivo del alto

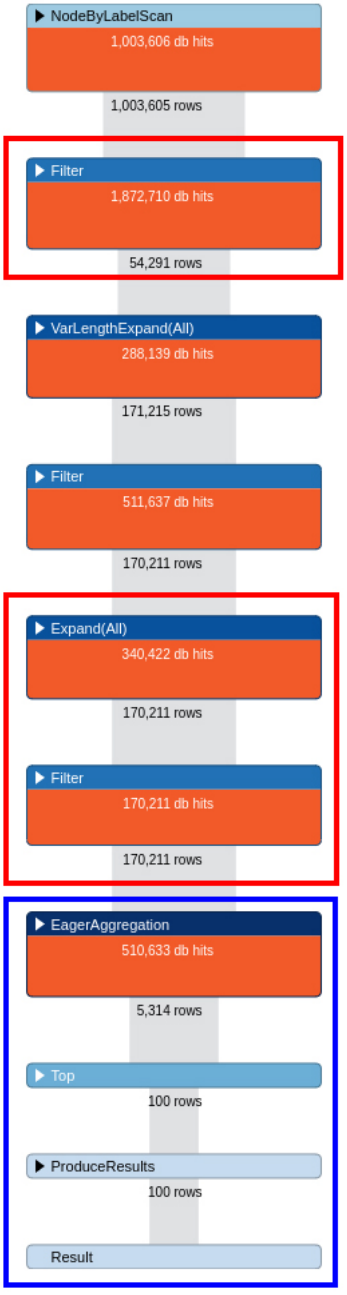


Figura A.36: Plan de Ejecución Consulta BI14.

```
MATCH
  (person:Person)<-[:HAS_CREATOR]-(post:Post)
WHERE  post.creationDate >= 20120531220000000
  AND  post.creationDate <= 20120630220000000
CREATE (post)-[:BI14_post_person]->(person);
```

Figura A.37: Solución Propuesta Materialización de Caminos Consulta BI14.

costo del operador *EagerAggregation*. Sin embargo, esto lo podemos solucionar reubicando estos dos operadores en una sentencia *WITH* y modificando la estructura original del código como se encuentra en la Figura A.38b.

```
RETURN
  person.id,
  person.firstName,
  person.lastName,
  count(DISTINCT post) AS threadCount,
  count(DISTINCT reply) AS messageCount
ORDER BY
  messageCount DESC,
  person.id ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  person,
  count(DISTINCT post) AS threadCount,
  count(DISTINCT reply) AS messageCount
ORDER BY
  messageCount DESC,
  person.id ASC
LIMIT 100
RETURN
  person.id,
  person.firstName,
  person.lastName,
  threadCount,
  messageCount
```

(b) Segmento Código Optimizado

Figura A.38: Solución Propuesta Propiedades Limitadas Consulta BI14.

Si bien lo que vemos en la Figura A.39 es sólo la eliminación de un operador *Filter* en comparación del plan ejecución original, debemos precisar que logramos el punto crítico. También se consiguió disminuir el valor de los primeros dos operadores del primer recuadro que hacen referencia al segundo recuadro de la Figura A.36. Lo anterior fue el resultado del uso de la segunda guía de diseño, sin embargo, se aplicó de la misma forma la tercera estrategia de la primera guía de diseño logrando aminorar a tan sólo el 10 % del costo original.

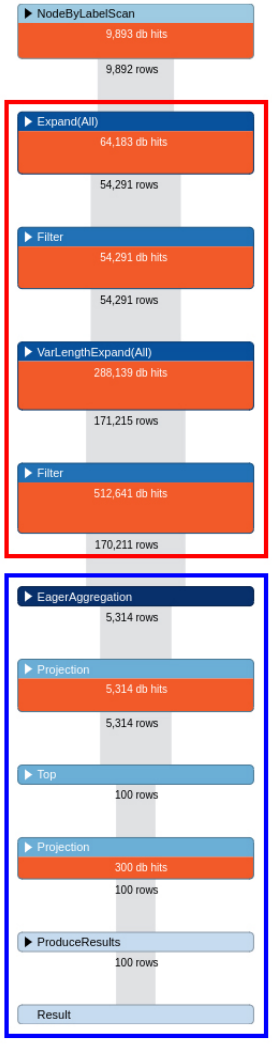


Figura A.39: Plan de Ejecución Solución Propuesta Consulta BI14.

A.12. Consulta BI15

El documento de especificación para la consulta BI15 establece que dado un país se debe obtener el "*social normal*", esta consiste en calcular la cantidad promedio de amistades de una persona, donde la persona y sus amistades debe pertenecer al mismo país. Una vez calculado el "*social normal*" debemos ubicar a todas las personas cuyo número de amistades en el país sea igual a este valor. En la Figura A.40 podemos ver un segmento del plan de ejecución donde se ubica el punto crítico de esta consulta. A continuación, se explican los operadores que se aprecian en este segmento.

1. Un operador *scan* aplicado sobre el nodo *Country*.
2. Tres operadores *Argument* que retoman las variables ya trabajadas.
3. Ocho operadores *Filter* aplicados sobre las relaciones y validaciones respectivas por nodo.
4. Diez operadores *Expand* se encarga de encontrar los registros para luego ser filtrados y obtener la información requerida.
5. Un operador *Optional* que es usado para resolver una sentencia *OPTIONAL MATCH*.
6. Un operador *Apply* que une ambos resultados de búsqueda.
7. Un operador *EagerAggregation* que realiza un fragmento del cálculo del "*social normal*.

En la consulta BI15 se visualiza que no existen candidatos al uso de la tercera guía de diseño por lo que se rechaza su aplicación. Con respecto a la primera guía de diseño se puede observar que existen variadas operaciones de agregación, sin embargo, nos encontramos con la situación con que cada una es esencial para el cálculo de la siguiente operación lo que no se puede abordar. Al estudiar la posibilidad de incorporar la segunda guía de diseño nos percatamos que existe una duplicidad de caminos que se aprecia en las áreas enmarcadas del plan de ejecución. Al efectuar distintos cálculos no podemos simplificar el código de la consulta, sin embargo, se puede materializar la relación *KNOWS* que une dos nodos *Person*. Sabemos que el punto crítico de la consulta se halla en la relación *IS_PART_OF* que vincula los nodos *Country* y *City*, pero al materializar la relación *KNOWS* conseguimos reducir para cada operador el costo de procesamiento. La implementación de la segunda guía de diseño se puede ver en la Figura A.41.

```
MATCH
  (country:Country {name: 'Burma'})
MATCH
  (country)<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(person1:Person)
OPTIONAL MATCH
  (country)<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(friend1:Person),
  (person1)-[:KNOWS]-(friend1)
WITH country, person1, count(friend1) AS friend1Count
WITH country, avg(friend1Count) AS socialNormalFloat
WITH country, floor(socialNormalFloat) AS socialNormal
MATCH
  (country)<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(person2:Person)
MATCH
  (country)<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(friend2:Person)-[:KNOWS]->(person2)
CREATE (friend2)-[:BI15_friend_person]->(person2);
```

Figura A.41: Solución Propuesta Materialización de Caminos Consulta BI15.

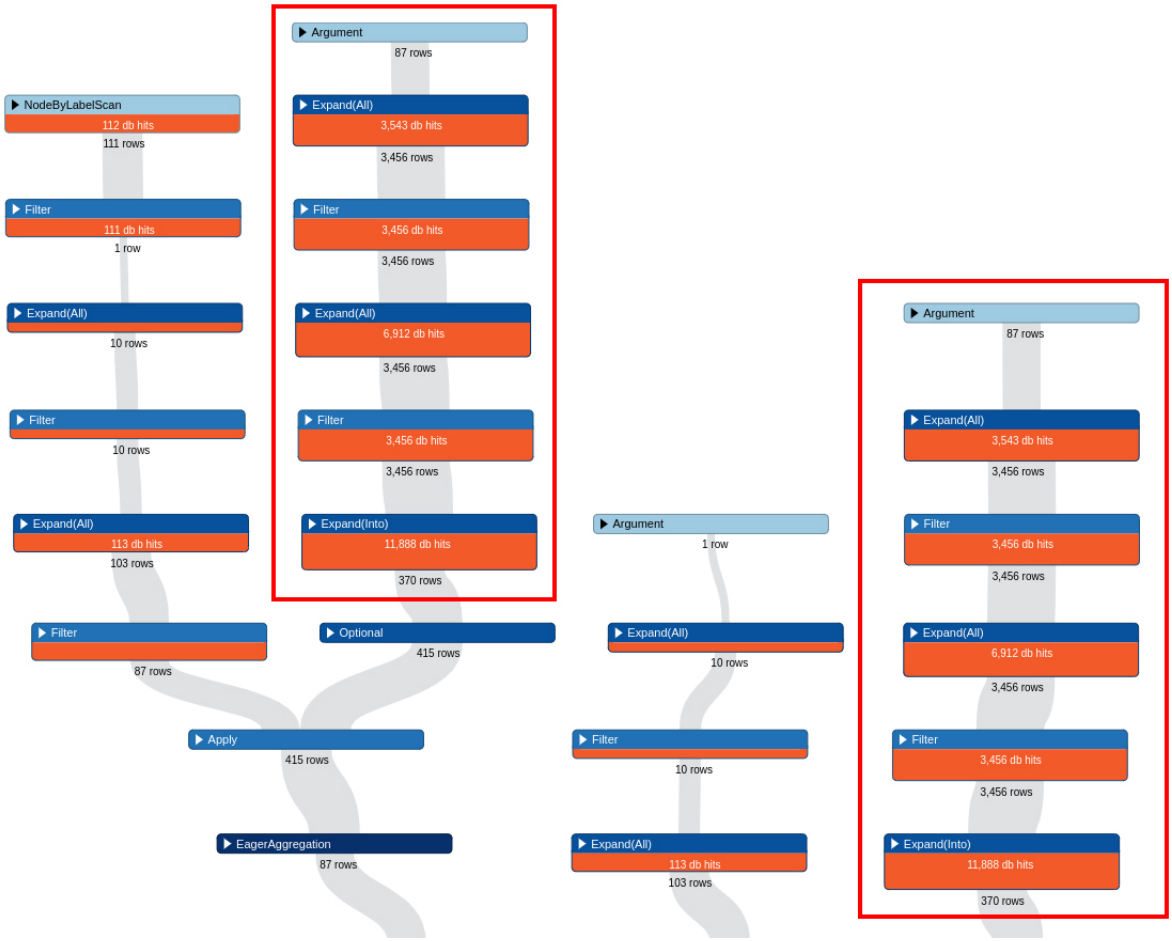


Figura A.40: Segmento Plan de Ejecución Consulta BI15.

El resultado del uso de la segunda guía de diseño se puede apreciar en el plan de ejecución de la Figura A.42. Podemos ver que en el primer recuadro se disminuyó un porcentaje de los costos de DB Hits de cada operador, no obstante, este resultado se replica en el segundo recuadro ya que se reutilizó la relación materializada en el camino duplicado consiguiendo reducir el costo de procesamiento para diez operaciones en total.

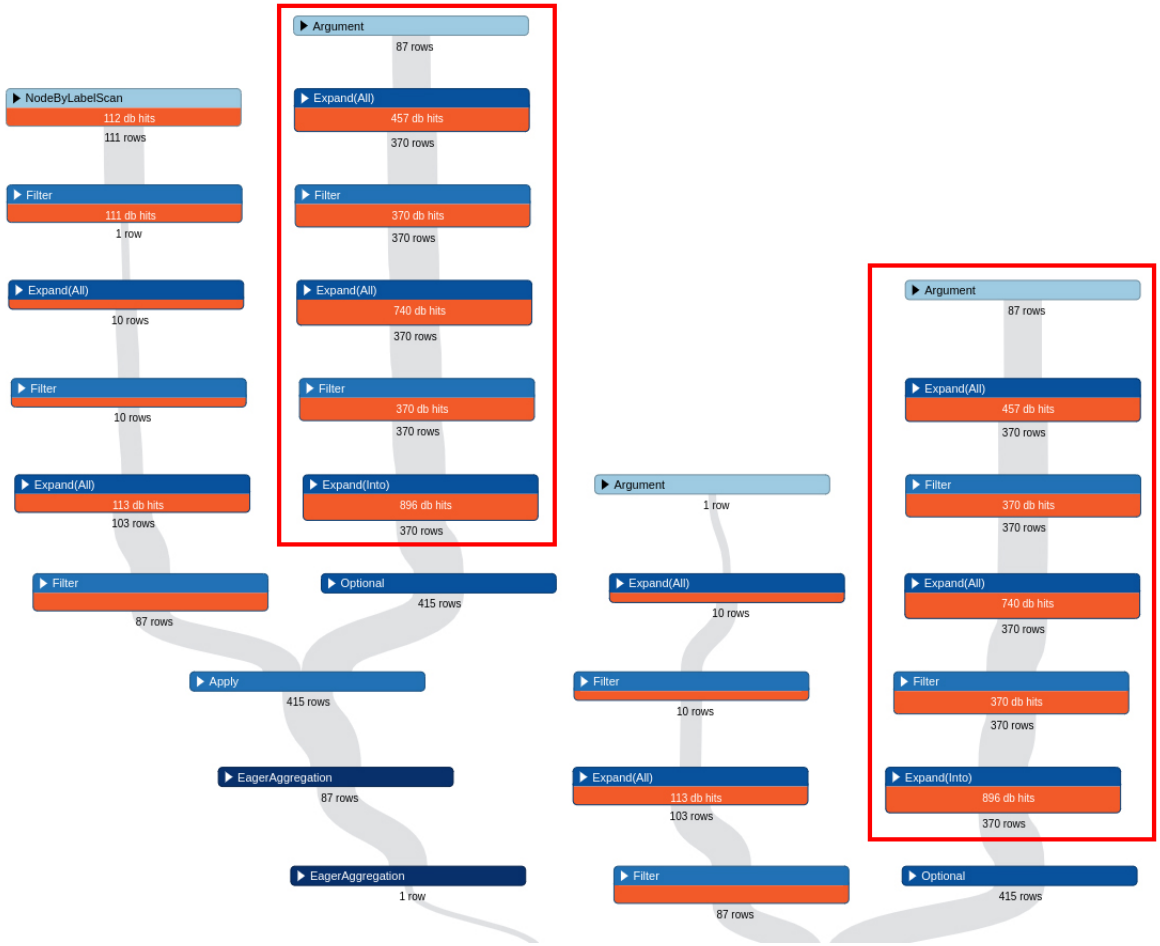


Figura A.42: Segmento Plan de Ejecución Solución Propuesta Consulta BI15.

A.13. Consulta BI16

La consulta BI16 establece que dada una persona, encontrar todas las otras personas que viven en un país determinado y que están conectadas a esa persona por un camino transitivo con una longitud en el rango [minimaRutaDistancia, maxRutaDistancia] a través de la relación *KNOWS*. Para cada una de estas personas, se debe retornar todos sus mensajes que contengan al menos una etiqueta que pertenezca a una clase de etiqueta indicada. De la misma forma, para cada mensaje se debe entregar todas sus etiquetas. Los resultados son agrupados por personas y etiquetas, luego se deben contar los mensajes por persona que tenga una etiqueta determinada. En la Figura A.43 podemos notar el plan de ejecución donde el punto crítico se realza en el primer recuadro y el segundo posee una relación directa. A continuación, se explican los operadores que se aprecian en este segmento.

1. Un operador *scan* para buscar la persona señalada por el documento.
2. Ocho operadores *Filter* que son aplicados en los nodos *Country*, *Message*, *Person*, *Tag* y *TagClass*.
3. Un operador *VarLengthExpand* para efectuar la búsqueda de personas según la longitud de rango entregada.
4. Un operador *Distinct* para obtener las personas que tengan relación con la persona inicial y estas no se repitan.
5. Seis operadores *Expand* que se encargan de buscar etiqueta, clase de etiqueta, mensajes, país y personas.
6. Un operador *EagerAggregation* que realiza el conteo del número total de mensajes distintos.
7. Un operador *Top* que retorna los 100 primeros registros.

El código de la consulta BI16 se constituye de tres sentencias *MATCH*, donde la primera se ocupa de encontrar a la persona inicial y sus conocidos. La segunda sentencia debe buscar a las personas conocidas según el país indicado, de la misma forma se aplica para la clase de etiqueta indicada que poseen los mensajes creados por las personas conocidas. Según lo analizado de la primera guía de diseño se puede usar la estrategia de Propiedades Limitadas ya que cumple se evidencia un operador *count* que permite su implementación. Al revisar el código percibimos que hay sólo una operación de agregación lo que impide el uso de la estrategia reescritura de la consulta, además no se observan funciones que nos permitan eliminar redundancia mediante la reutilización de variables ni descomponer la consulta. Para la tercera guía de diseño la propiedad candidata es *name* que pertenece a los nodos *Country*, *Tag* y *TagClass*. Cabe destacar que al aplicar la técnica se restringe a tan sólo un nodo, sin embargo, al ser utilizada en el nodo *TagClass* no reduce el costo de procesamiento con respecto al punto crítico, por esta razón no se considera la tercera guía de diseño. Finalmente, la segunda guía de diseño plantea materializar la relación del punto crítico, la cual se conforma por los nodos *Tag* y *TagClass* cuyo vinculo es la relación *HAS_TYPE*. Adicionalmente, al materializar la relación nos permite eliminar completamente el primer recuadro y a su vez reducir el costo de procesamiento. A continuación, en la Figura A.44 podemos ver la aplicación de la guía de diseño recién mencionada.

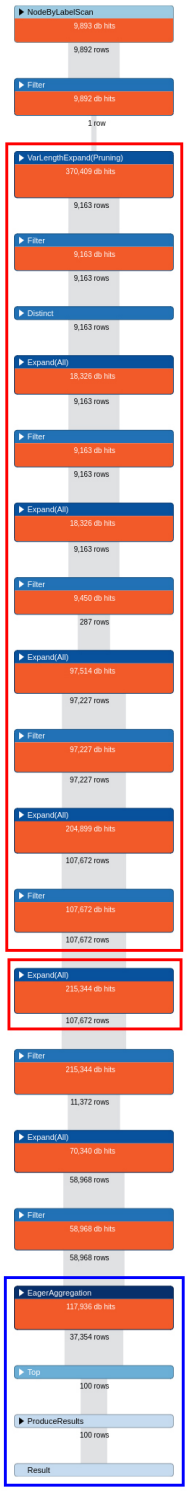


Figura A.43: Segmento Plan de Ejecución Consulta BI16.

```
MATCH
  (:Person {id: 19791209310731})-[:KNOWS*3..5]-(person:Person)
WITH DISTINCT person
MATCH
  (country:Country)<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-(person)
  <-[:HAS_CREATOR]-(message:Message)-[:HAS_TAG]->(:Tag)-[:HAS_TYPE]->
  (tagClass:TagClass)
WHERE country.name = 'Pakistan' AND tagClass.name = 'MusicalArtist'
CREATE (message)-[:BI16_message_person]->(person);
```

Figura A.44: Solución Propuesta Materialización de Caminos Consulta BI16.

Con respecto a la implementación de la tercera estrategia de la primera guía de diseño, esta se constituye con la localización del operador *count* en una sentencia *WITH* permitiéndonos modificar la estructura original de la parte final del código como se ve en la Figura A.45.

```
RETURN
  person.id,
  tag.name,
  count(DISTINCT message) AS messageCount
ORDER BY
  messageCount DESC,
  tag.name ASC,
  person.id ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  person,
  tag,
  count(DISTINCT message) AS messageCount
ORDER BY
  messageCount DESC,
  tag.name ASC,
  person.id ASC
LIMIT 100
RETURN
  person.id,
  tag.name,
  messageCount
```

(b) Segmento Código Optimizado

Figura A.45: Solución Propuesta Propiedades Limitadas Consulta BI16.

Vemos en la Figura A.46 que el primer recuadro del plan de ejecución original se convirtió en tan sólo dos operadores con un bajo costo puesto que la segunda guía de diseño rebajó el costo de DB Hits en un 90 % aproximadamente. Igualmente, con la aplicación de la segunda guía de diseño se consiguió reducir el costo de procesamiento del segundo recuadro. Finalmente, con el uso de la tercera estrategia de la segunda guía de diseño se pudo disminuir en un porcentaje cercano al 45 % el costo de procesamiento del operador *EagerAggregation* trasladándolo al operador *Projection*.

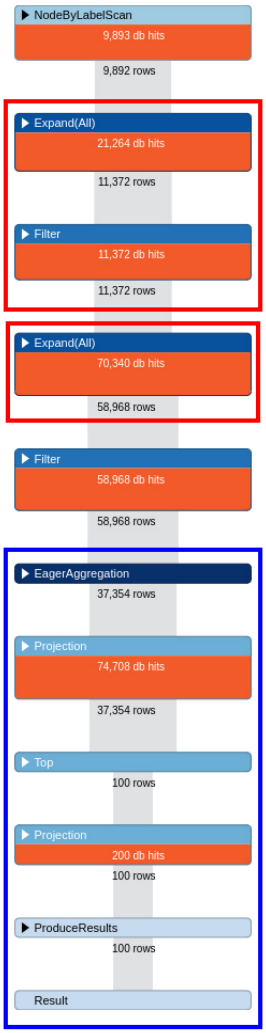


Figura A.46: Segmento Plan de Ejecución Solución Propuesta Consulta BI16.

A.14. Consulta BI18

El documento de especificación menciona que para la consulta BI18 se deben enumerar los mensajes hechos por una persona. Para los mensajes sólo contabilizar los que posean los siguientes atributos:

- Su contenido no este vacío.
- Su longitud se encuentre por debajo del umbral de longitud entregado.
- Su fecha de creación sea posterior a la fecha dada.

- Que este escrito en cualquiera de los idiomas indicados.

Para cada valor relacionado con la cantidad de mensajes, contar el número de personas con mensajes que cumplan con los atributos requeridos. En la Figura A.47 podemos contemplar un segmento del plan de ejecución donde el punto crítico se localiza en los operadores *Scan* y *Filter*. A continuación, se enumeran y definen los operadores que se aprecian en la Figura A.47.

1. Dos operadores *scan* para obtener a las personas y las publicaciones.
2. Dos operadores *Filter* que se ocupan de cumplir con los criterios requeridos para los mensajes.
3. Un operador *VarLengthExpand* que consigue los mensajes de réplica que haya hecho una persona en una publicación.
4. Un operador *Expand* para obtener las personas que hicieron un mensaje.
5. Un operador *Hash Join* el cual une el resultado de las personas y sus mensajes.

El código de la consulta se constituye de una sentencia *MATCH* y *OPTIONAL MATCH*. Por otro lado, vemos las condiciones que buscan cumplir el requerimiento mediante la sentencia *WHERE*. Al revisar minuciosamente podemos indicar que no existe ninguna posibilidad de implementar la primera guía de diseño, ya que no contamos con las operaciones de agregación, la estructura necesaria para la reutilización de variables ni la limitación de las operaciones. Acerca de la tercera guía de diseño es posible su uso en las propiedades *content*, *length* y *creationDate* por parte del nodo *Message*, y la propiedad *language* del nodo *Post*. Hay que remarcar que aplicar dicha estrategia consta de sólo tratar una propiedad lo que entrega un resultado que no se considera conveniente. La segunda guía de diseño plantea materializar la relación *HAS_CREATOR* que une a los nodos *Message* y *Person*, que corresponde al operador *Expand*. La implementación de esta guía de diseño se puede ver en la siguiente figura.

```
MATCH
  (person:Person)<-[:HAS_CREATOR]-(message:Message)-[:REPLY_OF*0..]->(post:Post)
WHERE message.content IS NOT NULL
  AND message.length < 20
  AND message.creationDate > 20110722000000000
  AND post.language IN ['ar']
CREATE (message)-[:BI18_message_person]->(person);
```

Figura A.48: Solución Propuesta Materialización de Caminos Consulta BI18.

La implementación de la segunda guía de diseño nos entrega como resultado la eliminación de todos los operadores previos al operador *Expand* como se ve en la Figura A.49.

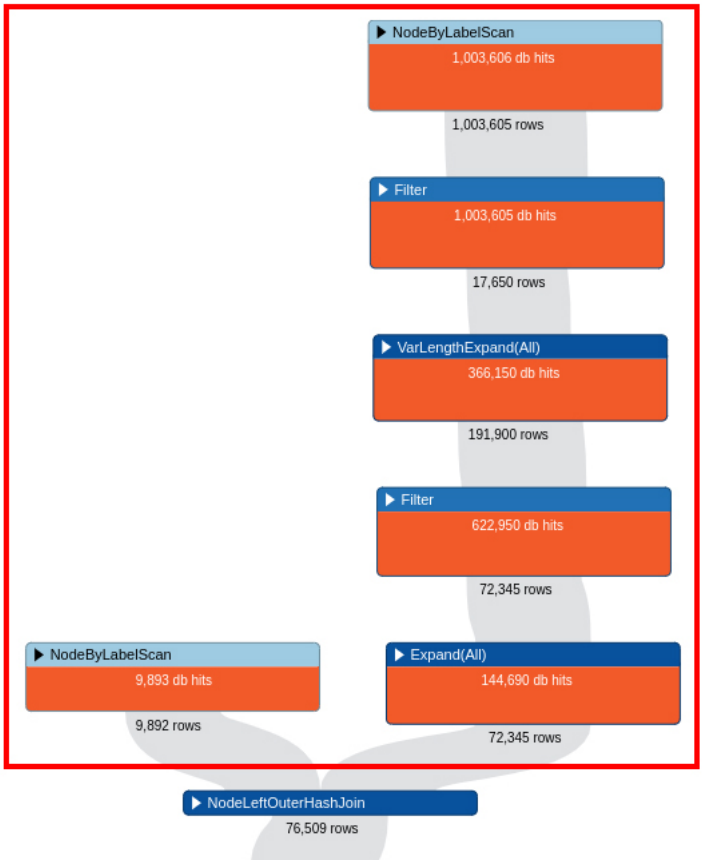


Figura A.47: Segmento Plan de Ejecución Consulta BI18.

Como se indicó previamente el factor que influyó en la decisión de cuál guía de diseño se aplicaría fue la reducción del número de DB Hits ya que el resultado de la tercera guía de diseño no se compara al resultado que apreciamos con la segunda guía de diseño. Por lo anterior, se decide trabajar con la segunda guía de diseño.

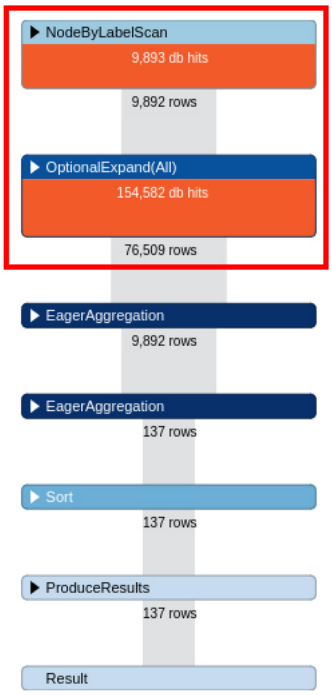


Figura A.49: Segmento Plan de Ejecución Solución Propuesta Consulta BI18.

A.15. Consulta BI19

El objetivo de la consulta BI19 es que para todas las personas (person) nacidas después de una determinada fecha, encontrar a todos los extraños (strangers) con los que interactuaron, donde los extraños son personas que no conocen a otra. No hay restricción en la fecha en que nacieron los extraños. Sólo considerar a los extraños que sean:

- miembros de foros etiquetados con una clase de etiqueta (*tagClass1*).
- miembros de foros etiquetados con una clase de etiqueta (*tagClass2*).

La interacción es definida como la persona que ha respondido a un mensaje de un extraño. El resultado de la consulta es que para cada persona, se deben contabilizar el número de extraños con los que interactuaron y el número total de veces que interactuaron con ellos. En la Figura A.50 podemos apreciar el segmento del plan de ejecución donde se sitúa el punto crítico en el interior del segundo recuadro correspondiente al operador *Expand (Into)*. A continuación, se explican los operadores que se aprecian en este segmento.

1. Dos operadores *Argument* para retomar los registros de mensajes y extraños.

2. Dos operadores *Distinct* para validar que los extraños no se repitan.
3. Dos operadores *AntiSemiApply* para cumplir con que la persona no conozca al extraño y que la persona no haya replicado un mensaje del extraño.
4. Dos operadores *VarLengthExpand* para obtener todas las réplicas a los mensajes.
5. Ocho operadores *Filter* que se encargan de cumplir con los criterios requeridos.
6. Siete operadores *Expand* para buscar los registros de los nodos *Comment*, *Forum*, *Message*, *TagClass* y *Person*.
7. Un operador *Apply* para aplicar los resultados definidos del operador *AntiSemiApply*.
8. Un operador *EagerAggregation* que realiza el conteo de la interacción y el número de extraños.
9. Un operador *Top* que retorna los 100 primeros registros.

El código de la consulta BI19 se integra por tres sentencias *MATCH* de las cuales las primeras dos se orientan en buscar y validar que los extraños sean distintos, en cambio, la tercera sentencia es la que hace cumplir las condiciones establecidas por la consulta. Allí nos encontrados con cuatro condiciones situadas en una sentencia *WHERE*, de las cuales dos están dirigidas a capturar las personas y extraños que se conocen y los mensajes de la persona que hayan sido replicados por un extraño para luego contraponer los resultados mencionados con los registros iniciales. Por último, identificamos dos operadores *count* en la sentencia *RETURN*. Ante lo indicado sólo se puede aplicar la tercera estrategia de la primera guía de diseño, dado que las dos primeras sentencias *MATCH* están sujetas a las condiciones de las clases de etiquetas y la tercera sentencia se establece para cumplir con las condiciones. De tal modo que imposibilita implementar las primera dos técnicas de la primera guía de diseño. La tercera guía de diseño se puede aplicar en las propiedades *birthday* del nodo *Person* y *name* del nodo *TagClass*, sin embargo, ninguno de los dos aborda el punto crítico en cuestión, desechando la tercera guía de diseño como alternativa. Con respecto a la segunda guía de diseño tenemos la posibilidad de crear una relación directa entre las personas y extraños permitiéndonos suprimir los nodos intermedios y a su vez cada una de las condiciones relacionadas.

Figura A.50: Segmento Plan de Ejecución Consulta BI19.

```
MATCH
  (person:Person)<-[:HAS_CREATOR]-(comment:Comment)-[:REPLY_OF*]->
  (message:Message)-[:HAS_CREATOR]->(stranger)
WHERE person.birthday > 19890101
  AND person <> stranger
  AND NOT (person)-[:KNOWS]-(stranger)
  AND NOT (message)-[:REPLY_OF*]->(:Message)-[:HAS_CREATOR]->(stranger)
CREATE (person)-[:BI19_person_stranger]->(stranger);
```

Figura A.51: Solución Propuesta Materialización de Caminos Consulta BI19.

En relación a la tercera estrategia de la primera guía de diseño podemos ver su aplicación en la Figura A.52. Allí posicionamos ambos operadores *count* en una sentencia *WITH* permitiendo la reestructuración de ese segmento de código que hace referencia al tercer recuadro de la Figura A.50.

```
RETURN
  person.id,
  count(DISTINCT stranger) AS strangersCount
  count(comment) AS interactionCount
ORDER BY
  interactionCount DESC,
  person.id ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  person,
  count(DISTINCT stranger) AS strangersCount,
  count(comment) AS interactionCount
ORDER BY
  interactionCount DESC,
  person.id ASC
LIMIT 100
RETURN
  person.id,
  strangersCount,
  interactionCount
```

(b) Segmento Código Optimizado

Figura A.52: Solución Propuesta Propiedades Limitadas Consulta BI19.

El uso de la segunda guía de diseño elimina completamente los dos primeras áreas delimitadas en el plan de ejecución, por ende se consigue remover completamente el punto crítico y otro segmento. También se logra reducir en un 93 % el costo de DB Hits del operador *EagerAggregation* siendo reubicado el porcentaje restante en el operador *Projection* como se ve en la Figura A.53.

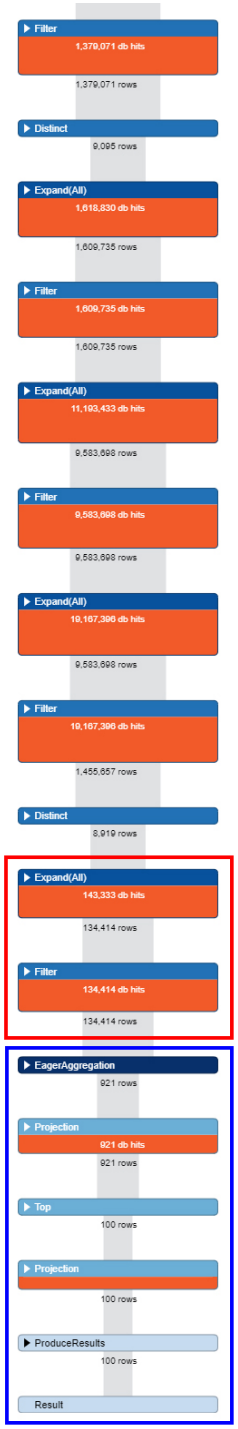


Figura A.53: Segmento Plan de Ejecución Solución Propuesta Consulta BI19.

A.16. Consulta BI20

El objetivo de la consulta BI20 es dada una clase de etiqueta, contar el número de mensajes que tiene una etiqueta que pertenece a esa clase de etiqueta o cualquiera de sus elementos secundarios. En la Figura A.54 podemos visualizar el plan de ejecución donde se ubica el punto crítico en los operadores *Expand(All)* y *Filter* del primer recuadro. A continuación, se explican los operadores que se aprecian en el plan de ejecución.

1. Un operador *scan* para obtener el nodo *TagClass* y los valores de la lista *tagClassName*.
2. Un operador *Unwind* para contener los valores de la lista *tagClassName*.
3. Un operador *Apply* que permite cruzar la información del nodo *TagClass* y los valores de la lista *tagClassName*.
4. Un operador *VarLengthExpand* que busca las etiqueta que pertenezcan a las clases indicadas.
5. Cuatro operadores *Filter* sobre los nodos *TagClass*, *Tag* y *Message*.
6. Dos operadores *Expand* sobre las relaciones *HAS_TYPE* y *HAS_TAG*.
7. Un operador *EagerAggregation* que se ocupan de ordenar los registros obtenidos.
8. Un operador *Top* que retorna los 100 primeros registros.

Al analizar el código en la consulta BI20 podemos ver que no existe redundancia que posibilite la primera estrategia de la primera guía de diseño, además al ser un código simple en cuanto a su estructura también restringe la alternativa de esta técnica. Por último, se observa la existencia de un operador *count* en la sentencia *RETURN* permitiendo el uso de la tercera estrategia de la primera guía de diseño. Por otro lado, nos damos cuenta que la única propiedad que se puede indexar en alusión a la tercera guía de diseño corresponde a la propiedad *name* del nodo *TagClass* que hace referencia a los nombres de las clases de etiquetas indicadas por el documento de especificación. Si bien esta guía de diseño se puede aplicar, el beneficio obtenido es insignificante e incluso no permite ocuparnos del punto crítico. La segunda guía de diseño es la candidata a ser utilizada ya que podemos establecer una vinculo directo en la relación *HAS_TAG* para los nodos *Message* y *Tag*. La implementación de la segunda guía de diseño se puede ver en la Figura A.55.

Figura A.54: Segmento Plan de Ejecución Consulta BI20.

```
UNWIND ['Writer', 'Single', 'Country'] AS tagClassName
MATCH
  (tagClass:TagClass {name: tagClassName})<-[:IS_SUBCLASS_OF*0..]-
  (:TagClass)<-[:HAS_TYPE]-(tag:Tag)<-[:HAS_TAG]-(message:Message)
WITH distinct (message), tagClass
CREATE (message)-[:BI20_message_tagClass]->(tagClass);
```

Figura A.55: Solución Propuesta Materialización de Caminos Consulta BI20.

En la Figura A.56 apreciamos la implementación de la tercera estrategia de la primera guía de diseño. La modificación realizada al código original es la reubicación del operador *count* en una sentencia *WITH* previo al orden y limitación de los resultados modificando así la estructura original de este segmento de código como se define en la Figura A.56b.

```
RETURN
  tagClass.name,
  count(DISTINCT message) AS messageCount
ORDER BY
  messageCount DESC,
  tagClass.name ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  tagClass,
  count(message) AS messageCount
ORDER BY
  messageCount DESC,
  tagClass.name ASC
LIMIT 100
RETURN
  tagClass.name,
  messageCount
```

(b) Segmento Código Optimizado

Figura A.56: Solución Propuesta Propiedades Limitadas Consulta BI20.

Podemos ver en la Figura A.57 que el primer recuadro del plan de ejecución original se reduce a sólo dos operadores, donde además el costo de procesamiento disminuye en un tercio para cada operador. Adicionalmente, conseguimos eliminar el costo de procesamiento del operador *EagerAggregation* con el uso de la técnica Propiedades Limitadas.

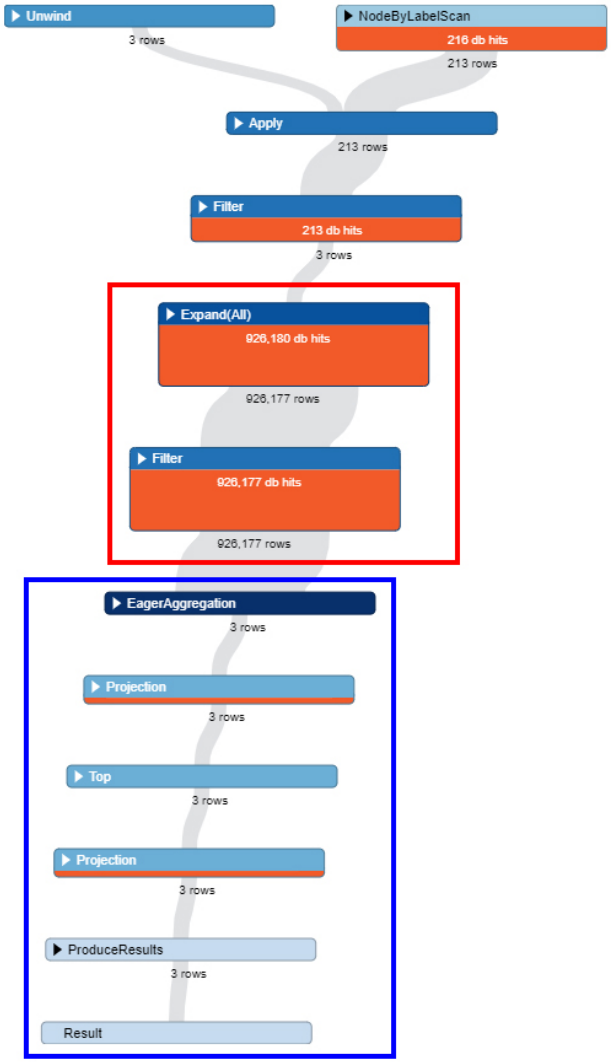


Figura A.57: Segmento Plan de Ejecución Solución Propuesta Consulta BI20.

A.17. Consulta BI22

El documento de especificación establece considerar para la consulta BI22 todos los pares de personas (*person1*, *person2*) de modo que uno esté ubicado en una ciudad del país (*pais1*) y el otro esté ubicado en una ciudad del país (*pais2*). Para cada ciudad del país *pais1*, se retorna el par de mayor puntuación. La puntuación de un par se define como la suma de las subpuntuaciones otorgadas por los siguientes tipos de interacción. Mencionar que la puntuación inicial es igual a 0.

1. Si *person1* ha replicado al menos un mensaje creado por *person2*: suma 4 puntos.
2. Si *person1* ha creado al menos un mensaje para el que *person2* haya replicado con un comentario: suma 1 punto.
3. Si *person1* y *person2* se conocen: suma 15 puntos.
4. Si *person1* le gustó al menos un mensaje realizado por *person2*: suma 10 puntos.
5. Si *person1* ha creado al menos un mensaje que le ha gustado a *person2*: suma 1 punto.

En consecuencia, la puntuación máxima que puede obtener un par es:

$$14 + 1 + 15 + 10 + 1 = 31$$

En la Figura A.58 observamos un segmento del plan de ejecución en el que se hallan dos tipos de recuadro los que hacen referencia a la réplica y el mensaje cuyos colores son rojo y azul, respectivamente. El punto crítico de la consulta se localiza en el tercer recuadro de color azul en el operador *Expand (Into)*. A continuación, se mencionan los operadores que se encuentran en los recuadros del plan de ejecución.

1. Tres operadores *Argument* que retoman los resultados de los registros ya trabajados.
2. Un operador *Optional* que complementa la búsqueda de la sentencia *OPTIONAL MATCH*.
3. Siete operadores *Filter* que filtran la información verificando que se cumplan las condiciones.
4. Diez operadores *Expand* que buscan los registros de los nodos *City*, *Comment*, *Country*, *Message* y *Person*.

El código de la consulta BI22 se constituye de una sentencia *MATCH* la cual se responsabiliza de buscar a las personas que se ubiquen en los países indicados (pais1 y pais2). Luego tenemos una sentencia *WITH* que establece la variable *score* en una valor 0. Igualmente, existen 5 sentencias *OPTIONAL MATCH* donde la primera y segunda se ocupan de la búsqueda de las personas que hayan replicado un mensaje o comentario. De la misma forma se comportan la cuarta y quinta sentencia que buscan a las personas (*person1* y *person2*) que les hayan dado *likes* a los mensajes creados por *person1* o *person2*. La última sentencia es la tercera que verifica que las personas se conozcan. Cabe precisar que cada una de las sentencias *OPTIONAL MATCH* viene acompañada de una sentencia *WITH* que va asignando el *score* correspondiente. Finalmente, apreciamos dos sentencias *ORDER BY*, donde la primera es seguida por una sentencia *WITH* que registra los resultados de

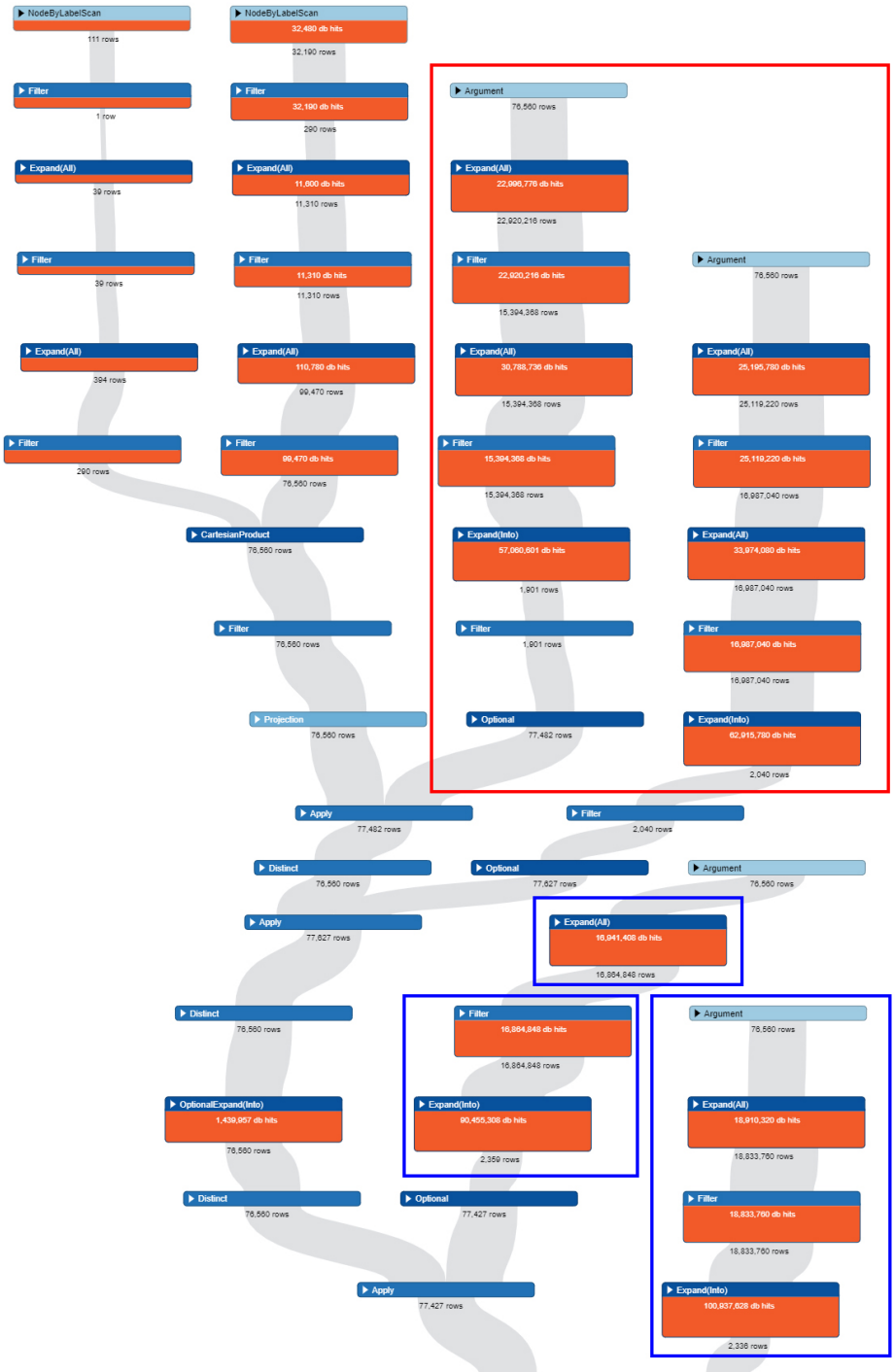


Figura A.58: Segmento Plan de Ejecución Consulta BI22.

person1 y *person2* en una lista declarada como *top*. Al tener tan estructurada la consulta queda en evidencia que no se puede aplicar ninguna de las estrategias de la primera guía de diseño debido a que sólo se cuenta con una operación de agregación que es la que trabaja la lista denominada *collect*. La propiedad candidata a usar la tercera guía de diseño es únicamente la propiedad (*name*) del nodo (*Country*), las operaciones que tienen relación con este último nodo son la búsqueda de personas por cada pais que se visualiza en el primer camino del primer recuadro de la Figura A.58 obligándonos a rechazar esta opción. Acerca de la segunda guía de diseño identificamos que se podría establecer una relación directa entre ambos nodos extremos de los puntos críticos de ambos recuadros enmarcados debido a que esta consulta es la más costosa de las 25 consultas. En el primer recuadro el punto crítico se localiza en el último operador, de igual forma sucede con el segundo recuadro. Para ambos casos se propone establecer una relación directa entre los nodos *person1* y *person2* consiguiendo remover los nodos y relaciones intermedias. La implementación de lo antes descrito en referencia a la segunda guía de diseño se puede ver en la Figura A.59.

```
//subscore 1 - 2

  MATCH
    (person1:Person)
  MATCH
    (person1)<-[:HAS_CREATOR]-(c:Comment)-[:REPLY_OF]->
    (:Message)-[:HAS_CREATOR]->(person2:Person)
  CREATE (person1)-[:BI22_person_person_subscore1_2]->(person2);

//subscore 4 - 5

  MATCH
    (person1:Person)
  MATCH
    (person1)-[:LIKES]->(m:Message)-[:HAS_CREATOR]->(person2:Person)
  CREATE (person1)<-[:BI22_person1_person2_subscore4_5]-(person2);
```

Figura A.59: Solución Propuesta Materialización de Caminos Consulta BI22.

El resultado de la implementación de la segunda guía de diseño en dos oportunidades se puede ver en la Figura A.60. Esta estrategia nos posibilitó eliminar los caminos de búsqueda enmarcados dejando así un exclusivo camino de búsqueda posterior a las primeras dos que se observan al lado izquierdo del primer recuadro delimitado. Se debe destacar que los recuadros de ambos colores se redujeron a tres operadores respectivamente, logrando de esta forma eliminar ambos puntos críticos y disminuyendo el costo de DB Hits en un 97 % aproximadamente.

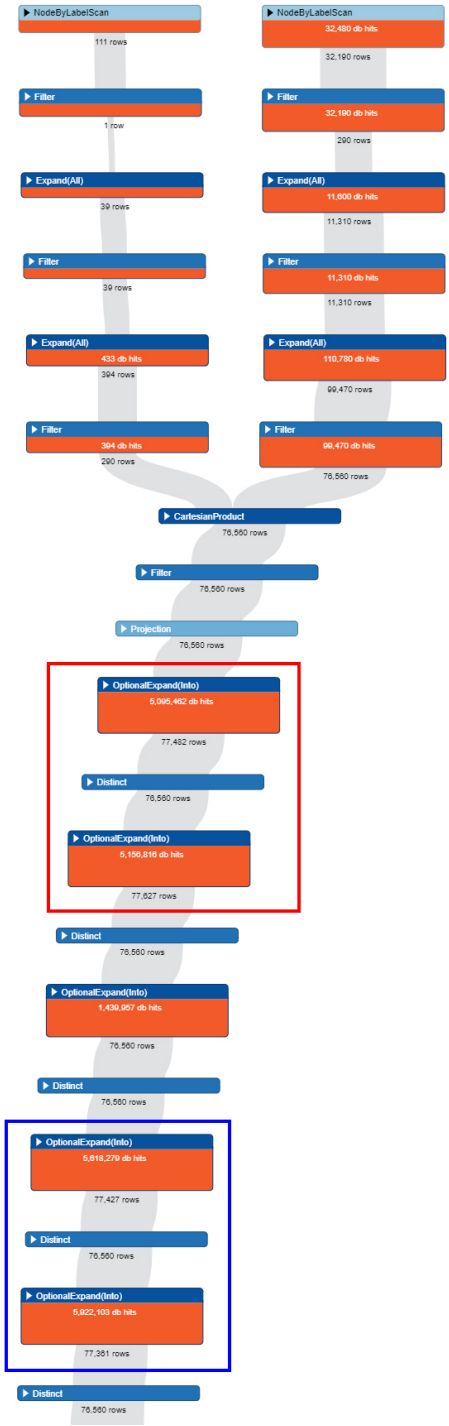


Figura A.60: Segmento Plan de Ejecución Solución Propuesta Consulta BI22.

A.18. Consulta BI23

El documento de especificación menciona que para la consulta BI23 se deben contabilizar los mensajes de todos los residentes de un país indicado (*home*), donde dichos mensajes hayan sido escritos en el extranjero. Estos mensajes deben ser agrupados por mes y destino (*destination*). Se considera un mensaje escrito en el extranjero si este se localiza en un país (*destination*) distinto al del residente (*home*). En la Figura A.61 podemos visualizar el plan de ejecución original cuyo punto crítico se ubica en el último operador del primer recuadro. A continuación, se explican los operadores que se aprecian en el plan de ejecución.

1. Un operador *scan* que obtiene los nodos *Country* (*home*).
2. Cinco operadores *Filter* que filtran la información verificando que se cumplan las condiciones.
3. Cuatro operadores *Expand* que buscan los continentes, etiquetas, mensajes y personas.
4. Un operador *Projection* para la obtención del mes de cada mensaje.
5. Un operador *EagerAggregation* para contabilizar la cantidad de mensajes.
6. Un operador *Top* que retorna los 100 primeros registros.

La estructura del código de la consulta BI23 se compone de una sentencia *MATCH* y una condición *WHERE*. La primera cumple con encontrar los registros solicitados, en cambio la segunda se responsabiliza de filtrar los mensajes que no cumplan con la condición de que hayan escrito el mensaje desde el extranjero. En el último segmento del código se ordena, agrupa y limita la cantidad de registros considerando exclusivamente los 100 mejores. Habiendo analizado el código podemos indicar que una alternativa es el uso de la estrategia Propiedades Limitadas de la primera guía de diseño, dado que se presenta un operador *count* en la sentencia *RETURN*. En segunda instancia, las propiedades candidatas a utilizar la tercera estrategia de diseño son la de *creationDate* y *name* de los nodos *Message* y *Country*, respectivamente. Al examinar está posibilidad nos damos cuenta que ninguna de las dos se encuentra en el punto crítico ni tampoco incide en su procesamiento lo que nos lleva a rechazar esta opción. La segunda guía de diseño es otra alternativa a considerar puesto que se puede materializar la relación del punto crítico en la que se ven asociados los nodos *Message* y *Country* con la salvedad que este hace referencia al destino (*destination*).

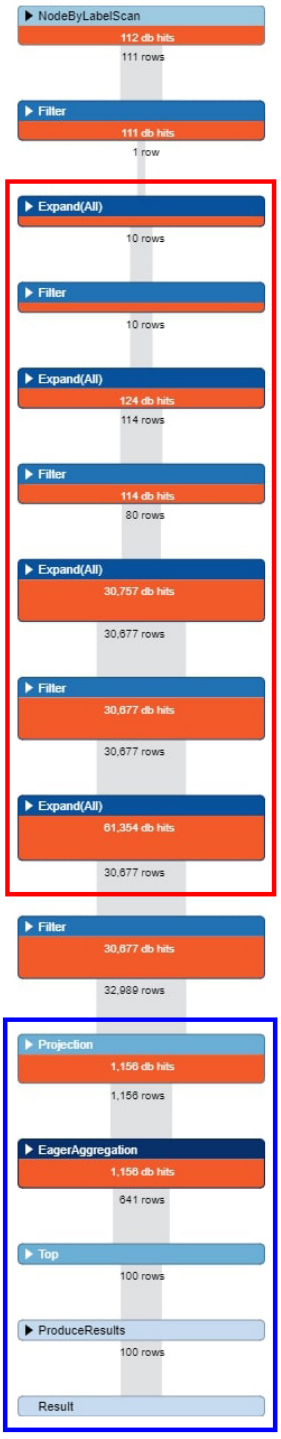


Figura A.61: Plan de Ejecución Consulta BI23.

```
MATCH
  (home:Country {name: 'Egypt'})<-[:IS_PART_OF]-(:City)<-[:IS_LOCATED_IN]-
  (person:Person)<-[:HAS_CREATOR]-(message:Message)-[:IS_LOCATED_IN]->(destination:Country)
WHERE home <> destination
CREATE (message)-[:BI23_message_person]->(destination);
```

Figura A.62: Solución Propuesta Materialización de Caminos Consulta BI23.

En la Figura A.63 encontramos la aplicación de la tercera estrategia de la primera guía de diseño. Allí podemos reubicar el operador *count* que se encuentra en la sentencia *RETURN* en el código original para ser trasladado en una sentencia *WITH*, donde adicionalmente se debe reestructurar el segmento de código dejando en última instancia la sentencia *RETURN* como se puede ver en la Figura A.63b.

```
RETURN
  count(message) AS messageCount,
  destination.name,
  month
ORDER BY
  messageCount DESC,
  destination.name ASC,
  month ASC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  count(message) AS messageCount,
  destination,
  message.creationDate/100000000000%100 AS month
ORDER BY
  messageCount DESC,
  destination.name ASC,
  month ASC
LIMIT 100
RETURN
  messageCount,
  destination.name,
  month
```

(b) Segmento Código Optimizado

Figura A.63: Solución Propuesta Propiedades Limitadas Consulta BI23.

La aplicación de la segunda guía de diseño nos permitió reducir el número de operadores del primer recuadro a sólo dos, donde cabe destacar que también que se pudo reducir el costo de procesamiento a un valor inferior al 2 % cumpliendo así el objetivo. Por otro lado, el uso de la tercera estrategia de la primera guía de diseño disminuyó cerca de un 55 % el costo de DB Hits del operador *Projection* como se aprecia en la Figura A.64.

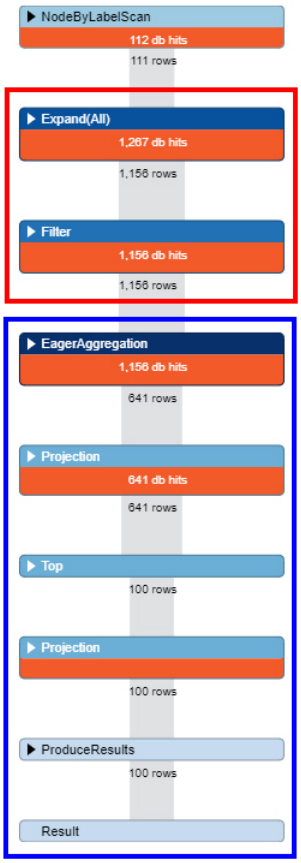


Figura A.64: Plan de Ejecución Solución Propuesta Consulta BI23.

A.19. Consulta BI24

En la consulta BI24 se deben encontrar todos los mensajes etiquetados con una etiqueta que sea del tipo de la clase de etiqueta dada, además hay que contar todos los mensajes y sus *likes* agrupados por continente, año y mes. En la Figura A.65 podemos visualizar el plan de ejecución donde el punto crítico se encuentra en el operador *OptionalExpand(All)*. Existen dos recuadros donde el primero se enmarca debido a que toda esa área corresponde a dos sentencias *MATCH* para la búsqueda de información requerida para encontrar lo solicitado, en cambio, el segundo nos entrega la posibilidad de complementar la opción principal. A continuación, se explican los operadores que se aprecian en el plan de ejecución.

1. Un operador *scan* que busca las etiqueta que pertenezcan a las clases indicadas.
2. Un operador *Argument* sobre los nodos *TagClass*, *Tag* y *Message*.

3. Cinco operadores *Filter* que filtran la información verificando que se cumplan las condiciones.
4. Cuatro operadores *Expand* que buscan los continentes, etiquetas, mensajes y personas.
5. Un operador *OptionalExpand* que es responsable de conseguir las personas que le hayan dado *like* a los mensajes.
6. Un operador *Distinct* filtra los mensajes que sean distintos.
7. Un operador *Apply* que une los resultados de la búsqueda de la rama izquierda con la rama derecha mediante el operador *Argument*.
8. Un operador *Projection* para la obtención del mes y año de cada mensaje.
9. Un operador *EagerAggregation* para contabilizar la cantidad de mensajes distintos y el número de *likes* de esos mensajes.
10. Un operador *Top* que retorna los 100 primeros registros.

Analizando el código de la consulta BI24 nos damos cuenta que existen dos operadores *count* situados en la sentencia *RETURN*, por otro lado, se observan dos operaciones aritméticas que se usan con el fin de convertir el valor de la propiedad *creationDate* del nodo *Message* en año y mes, además no existen condiciones mediante una sentencia *WHERE*. Finalmente, observamos que se hallan dos sentencias *MATCH* y una *OPTIONAL MATCH*, donde está última no presenta dificultades para el motor de ejecución. Después de estudiar las alternativas de uso de la primera guía de diseño se concluye que la única técnica que se puede utilizar es la de Propiedades Limitadas. Acerca de la factibilidad de la implementación de la tercera guía de diseño nos damos cuenta que se puede aplicar en la propiedad *creationDate*, sin embargo, el beneficio obtenido es nulo desechándose esta alternativa. Al revisar viabilidad de la segunda guía de diseño nos damos cuenta que con sólo materializar una relación directa entre los nodos *Message* y *Continent* logramos eliminar las relaciones *IS_LOCATED_IN* y *IS_PART_OF*, además del nodo *Country*. Esto lo podemos ver en la Figura A.66.

```
MATCH
  (:TagClass {name: 'Single'})<-[:HAS_TYPE]-(:Tag)<-[:HAS_TAG]-(message:Message)
WITH DISTINCT message
MATCH
  (message)-[:IS_LOCATED_IN]->(:Country)-[:IS_PART_OF]->(continent:Continent)
CREATE (message)-[:BI24_message_continent]->(continent);
```

Figura A.66: Solución Propuesta Materialización de Caminos Consulta BI24.

Figura A.65: Plan de Ejecución Consulta BI24.

En la Figura A.67 podemos ver la implementación de la estrategia Propiedades Limitadas. Como vemos ya existe previamente una sentencia *WITH* que se encarga de obtener el año y mes para cada mensaje, sin embargo, se hallan dos operadores *count* en la sentencia *RETURN* como ya habíamos mencionado previamente. Por esta razón se reubican ambos operadores en la sentencia *WITH* previa y se modifica la estructura del segmento original como se define en la Figura A.67b.

```
WITH
  message,
  message.creationDate/10000000000000   AS year,
  message.creationDate/100000000000%100 AS month,
  like,
  continent
RETURN
  count(DISTINCT message) AS messageCount,
  count(like) AS likeCount,
  year,
  month,
  continent.name
ORDER BY
  year ASC,
  month ASC,
  continent.name DESC
LIMIT 100
```

(a) Segmento Código Original

```
WITH
  message.creationDate/10000000000000   AS year,
  message.creationDate/100000000000%100 AS month,
  count(DISTINCT message) AS messageCount,
  count(like) AS likeCount,
  continent
ORDER BY
  year ASC,
  month ASC,
  continent.name DESC
LIMIT 100
RETURN
  messageCount,
  likeCount,
  year,
  month,
  continent.name
```

(b) Segmento Código Optimizado

Figura A.67: Solución Propuesta Propiedades Limitadas Consulta BI24.

El resultado de la aplicación de la segunda guía de diseño es la reducción del primer recuadro del plan de ejecución original a dos operadores. Si bien no se aborda el punto crítico de la consulta si logramos optimizar el primer recuadro, donde el costo total de procesamiento es significativamente mayor que el del punto crítico. Igualmente, se logra reducir en un 77,5 % el costo de DB Hits del primer recuadro quedando de esta forma un 22,5 % del costo de procesamiento original. Con respecto a la estrategia de Propiedades Limitadas observamos que el costo de procesamiento del operador *Projection* se redujo en casi un 100 %, aunque parte de ese costo se redistribuyó en el operador *EagerAggregation*. No obstante, de todas formas se disminuye el número de DB Hits en aproximadamente un tercio. Finalmente, todo lo antes descrito se puede observar en la Figura A.68.

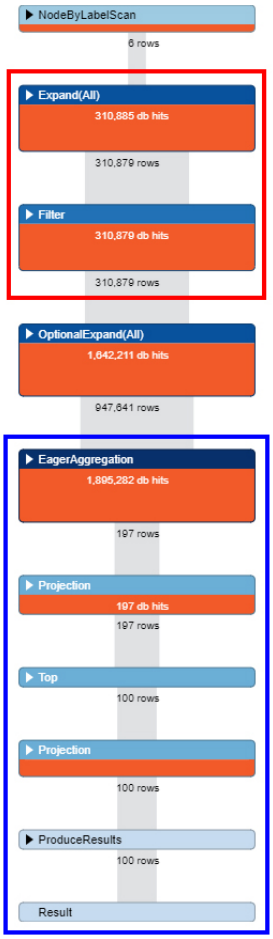


Figura A.68: Segmento Plan de Ejecución Solución Propuesta Consulta BI24.

A.20. Consulta BI25

El documento especifica que para la consulta BI25 dadas dos personas, se deben encontrar todos los caminos más cortos entre estas dos personas, en el subgrafo inducido por la relación *KNOWS*. Luego, para cada ruta encontrada calcular un peso. Los nodos en la ruta son personas, y el peso de una ruta es la suma de pesos entre cada par de personas consecutivos en la ruta. El peso para un par de personas se calcula en función de sus interacciones:

- Cada respuesta directa (de una de las personas) a una publicación (de la otra persona) contribuye 1.0.

- Cada respuesta directa (de una de las personas) a un comentario (de la otra persona) contribuye 0.5.

Por otro lado, considerar sólo los mensajes que se crearon en un foro, el cual se creó dentro de un intervalo de tiempo dado. Se deben retornar todos los caminos con los identificadores (*ID*) de las personas ordenados por sus pesos de forma descendente. En la Figura A.69 podemos visualizar un segmento del plan de ejecución en que se hallan dos tipos de recuadro los que hacen referencia a la réplica del comentario y de la publicación, cuyos colores son azul y rojo, respectivamente. El punto crítico de la consulta se localiza en el tercer recuadro de color rojo en el operador *Expand (Into)*. A continuación, se mencionan los operadores que se encuentran en los recuadros del plan de ejecución.

1. Un operador *scan* para obtener los registros de los foros.
2. Tres operador *Argument* que retoman los resultados de los registros ya trabajados.
3. Nueve operadores *Filter* que filtran la información verificando que se cumplan las condiciones.
4. Trece operadores *Expand* que buscan las personas, comentarios o réplicas, foros, personas y publicaciones, según lo requerido.

El código de la consulta BI25 se constituye de una sentencia *MATCH* que se encarga de obtener las diferentes relaciones entre el nodo inicial y el nodo final. Luego de esto siguen cuatro sentencias *OPTIONAL MATCH* cuya responsabilidad es encontrar a las personas que hayan replicado una publicación o un comentario. Lo anterior, se aplica desde la persona 1 hacia la persona 2 como también desde la persona 2 hacia la persona 1. Cada sentencia *OPTIONAL MATCH* viene acompañada de una condición que verifica que el foro haya sido creado en el intervalo de tiempo entregado. Adicionalmente, se incorpora una sentencia *WITH* que se encarga de asignar la puntuación a medida que se cumpla cada condición antes definida. Por último, tenemos una sentencia *WITH* cuya responsabilidad es obtener los identificadores (*id*) de las personas y de sumar los pesos de las relaciones para finalmente ordenar los resultados. Cabe mencionar que esta consulta tiene una gran similitud con la consulta BI22 que se encuentra en el anexo A.17. En el código de la consulta BI21 vemos una estructura bien definida y sin redundancia lo que nos imposibilita aplicar alguna de las estrategias de la primera guía de diseño. Acerca de la tercera guía de diseño, esta puede ser aplicada en las propiedades *id* y *creationDate* de los nodos *Person* y *Forum*, respectivamente. Sin embargo, el primero genera el efecto contrario aumentando el tiempo de ejecución y costo de procesamiento, en cambio, el segundo reduce un cierto porcentaje del costo de procesamiento pero no resuelve el problema de ambos puntos críticos por lo que no se considera esta opción. La segunda guía de diseño nos permite establecer una relación directa entre ambos nodos extremos de los dos tipos de consulta, esta acción nos

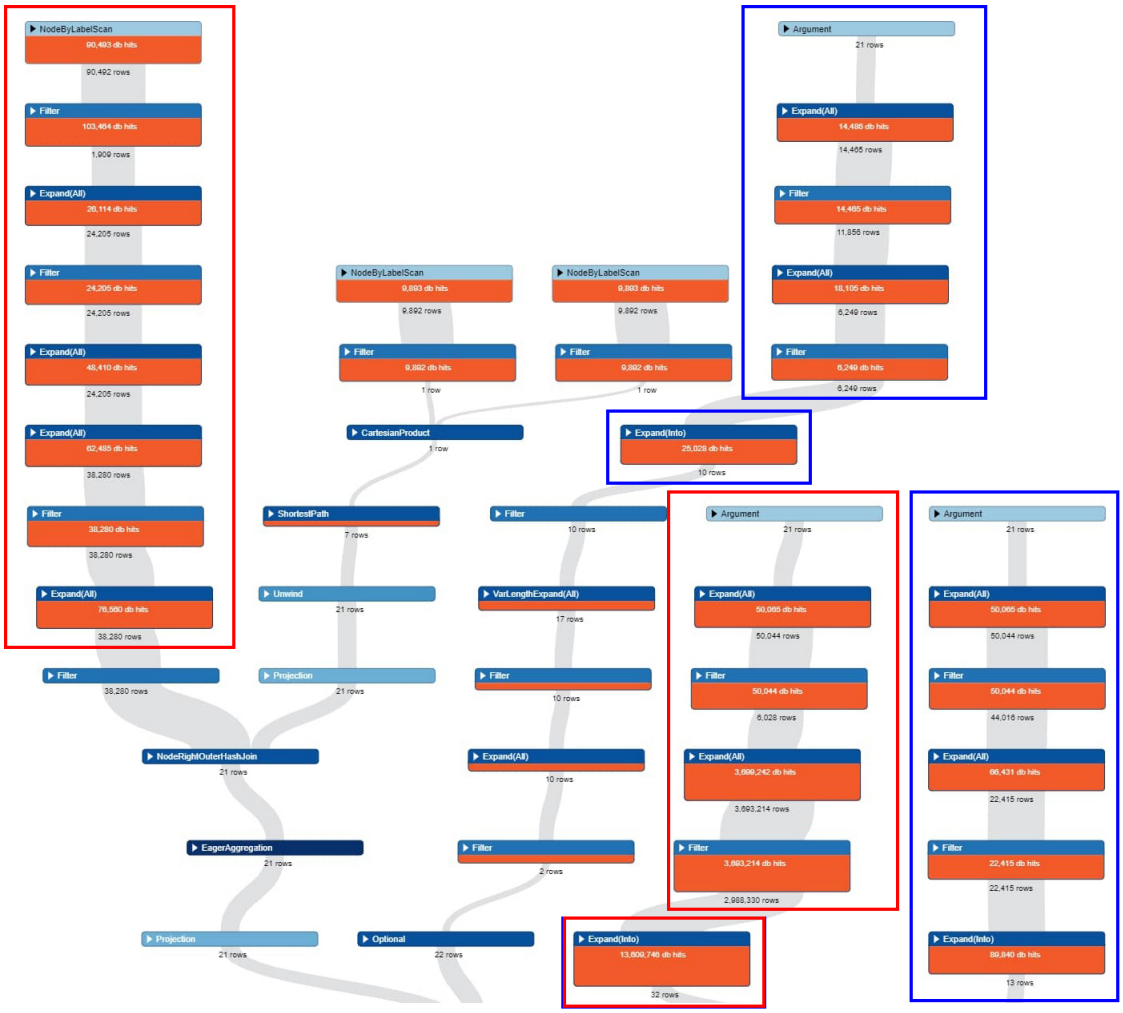


Figura A.69: Segmento Plan de Ejecución Consulta BI25.

facilita eliminar cada condición de la fecha de creación según el intervalo de tiempo para la sentencia *OPTIONAL MATCH*. Lo antes descrito se puede apreciar en la Figura A.70.

```
// case 1

  MATCH
    (pA)<-[:HAS_CREATOR]-(c:Comment)-[:REPLY_OF]->(post:Post)-[:HAS_CREATOR]->(pB),
    (post)<-[:CONTAINER_OF]-(forum:Forum)
  WHERE forum.creationDate >= 20101031230000000 AND forum.creationDate <= 20101130230000000
  CREATE (pA)-[:BI25_reply_post]->(pB);

// case 2

  MATCH
    (pA)<-[:HAS_CREATOR]-(c1:Comment)-[:REPLY_OF]->(c2:Comment)-[:HAS_CREATOR]->(pB),
    (c2)-[:REPLY_OF*]->(:Post)<-[:CONTAINER_OF]-(forum:Forum)
  WHERE forum.creationDate >= 20101031230000000 AND forum.creationDate <= 20101130230000000
  CREATE (pA)<-[:BI25_reply_comment]-(pB);
```

Figura A.70: Solución Propuesta Materialización de Caminos Consulta BI25.

El uso de la segunda guía de diseño nos posibilita eliminar ambos puntos críticos y disminuir considerablemente la cantidad de operadores a tan sólo tres operadores para cada caso, dejando de esta forma un exclusivo camino de búsqueda posterior a la búsqueda del nodo inicial y final que corresponde a los operadores que se encuentran entre los primeros recuadros de color rojo y azul en la Figura A.69. El resultado del uso de esta guía fue la reducción de un 99,5 % del tiempo de ejecución. El segmento de plan de ejecución con la segunda guía de diseño se puede ver en la Figura A.71.

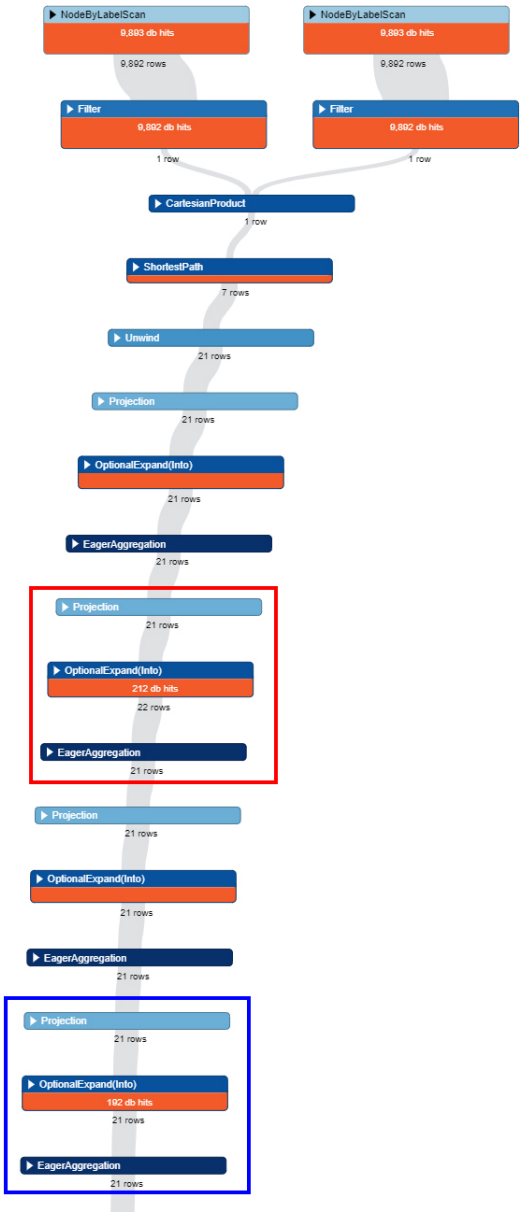


Figura A.71: Segmento Plan de Ejecución Consulta BI25.

Anexo B

Instructivo repositorio

El repositorio almacena la información más relevante de esta investigación, esta se sitúa en Google Drive y se debe solicitar el enlace al correo francisco.castillo@alumnos.uv.cl. Habiendo mencionado lo anterior debemos indicar que el repositorio consta de con tres secciones: Cargas de Trabajo, Consultas y Planes de Ejecución. La primera sección cuenta con cada carga de trabajo y en cada una están la base de datos generada por el Datagen y la base de datos de Neo4j. La segunda sección contiene el detalle de cada consulta encontrando la consulta sin diseño físico, la estrategia implementada y la estrategia con diseño físico. Finalmente, en la última sección se expone para la carga de trabajo de 1 GB el plan de ejecución sin diseño físico y con diseño físico para cada consulta.

Bibliografía

[1] Neo4j. The neo4j graph data platform. [Online]. Available: https://neo4j.com/product/

[2] R. Angles, J. B. Antal, A. Averbuch, P. Boncz, O. Erling, A. Gubichev, V. Haprian, M. Kaufmann, J. Lluís Larriba Pey, N. Martínez, J. Marton, M. Paradies, M.-D. Pham, A. Prat-Pérez, M. Spasić, B. A. Steer, G. Szárnyas, and J. Waudby, "The LDBC Social Network Benchmark," *arXiv e-prints*, p. arXiv:2001.02299, Jan. 2020.

[3] Talend. Sql vs nosql: Differences, databases, and decisions. [Online]. Available: https://www.talend.com/resources/sql-vs-nosql/

[4] E. Camacho. Nosql la evolución de las bases de datos. [Online]. Available: https://sg.com.mx/revista/28/nosql-evolucion-bases-datos

[5] B. M. Sasaki. Graph databases for beginners: Acid vs. base explained. [Online]. Available: https://neo4j.com/blog/acid-vs-base-consistency-models-explained/

[6] A. W. Services. What is NoSQL? [Online]. Available: https://aws.amazon.com/nosql/?nc1=h_ls

[7] G. Author. ¿qué es una base de datos nosql? [Online]. Available: https://blogs.oracle.com/spain/qu-es-una-base-de-datos-nosql

[8] J. Conesa and E. Rodríguez. Introducción a las bases de datos NoSQL en grafo. [Online]. Available: http://informatica.blogs.uoc.edu/2018/05/28/introduccion-a-las-bases-de-datos-nosql-en-grafo/

[9] J. C. Roma. Introducción al diseño de bases de datos. [Online]. Available: http://cv.uoc.edu/annotation/cb826b689abc472d8fb5b2519840058b/699689/PID_00213707/PID_00213707.html

[10] A. M. A. and R. López. Base de datos. [Online]. Available: http://repositorio.uchile.cl/bitstream/handle/2250/151632/Bases-de-datos.pdf?sequence=1&isAllowed=y

[11] LDBC. The graph & RDF benchmark reference. [Online]. Available: http://ldbcouncil.org/developer/snb

[12] I. Robinson, J. Webber, and E. Eifrem, "What is a graph?" in *Graph Databases*, 2nd ed., M. Beaugureau, Ed. O'Reilly Media, 6 2015, ch. 1, p. 1.

[13] ——, "Graph databases," in *Graph Databases*, 2nd ed., M. Beaugureau, Ed. O'Reilly Media, 6 2015, ch. 1, p. 5.

[14] C. C. Peña Álvarez, C. Pinilla, and M. Bello, "Bases de datos orientadas a grafos," *Tecnología Investigación y Academia*, vol. 5, no. 2, pp. 153–160, nov. 2017. [Online]. Available: https://revistas.udistrital.edu.co/index.php/tia/article/view/8769

[15] IONOS. Graph database: bases de datos para una interconexión eficiente. [Online]. Available: https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/graph-database/

[16] IBM. Physical database design. [Online]. Available: https://www.ibm.com/docs/en/db2-for-zos/11?topic=relationships-physical-database-design

[17] S. S. Lightstone, *Physical Database Design for Relational Databases*. Boston, MA: Springer US, 2009, pp. 2108–2114. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_644

[18] U. of Utah. Physical Design. [Online]. Available: https://utah.instructure.com/courses/108539/files/9491632/download?verifier=rjU97\QJQD4nt4ZkonCiXxYzzu5jiDok6PRE9pRGD&wrap=1

[19] R. Ramakrishnan and J. Gehrke, *Database Management Systems*, 3rd ed. USA: McGraw-Hill, Inc., 7 2002, ch. 20, pp. 650–653.

[20] LDBC. Why LDBC. [Online]. Available: http://ldbcouncil.org/public/why-ldbc

[21] ——. Technical documentation, code, features about graph and rdf benchmarking. [Online]. Available: http://ldbcouncil.org/developer/snb

[22] ——. LDBC-SNB Data Generator. [Online]. Available: https://github.com/ldbc/ldbc_snb_datagen

[23] Neo4j. What is a Graph Database? [Online]. Available: https://neo4j.com/developer/graph-database/#property-graph

[24] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*. O'Reilly Media, 2013.

[25] Neo4j. The internet-scale graph platform. [Online]. Available: https://neo4j.com/product/?ref=footer

[26] ——. 6.2. profiling a query. [Online]. Available: https://neo4j.com/docs/cypher-manual/current/query-tuning/

[27] ——. Chapter 7. Execution plans. [Online]. Available: https://neo4j.com/docs/cypher-manual/current/execution-plans/#execution-plans-dbhits

[28] ——. 2.7. Operators. [Online]. Available: https://neo4j.com/docs/cypher-manual/3.5/syntax/operators/

[29] ——. Neo4j 3.4 changelog. [Online]. Available: https://github.com/neo4j/neo4j/wiki/Neo4j-3.4-changelog

[30] ——. Neo4j 3.5 changelog. [Online]. Available: https://github.com/neo4j/neo4j/wiki/Neo4j-3.5-changelog

[31] ——. Introduction. [Online]. Available: https://neo4j.com/docs/operations-manual/3.5/clustering/introduction/

[32] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor, "Cypher: An evolving query language for property graphs," in *Proceedings of the 2018 International Conference on Management of Data*, ser. SIGMOD '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1433–1445. [Online]. Available: https://doi.org/10.1145/3183713.3190657

[33] Neo4j. Cypher query language. [Online]. Available: https://neo4j.com/developer/cypher/

[34] J. Marton, G. Szárnyas, and D. Varró, "Formalising opencypher graph queries in relational algebra," in *Advances in Databases and Information Systems*, M. Kirikova, K. Nørvåg, and G. A. Papadopoulos, Eds. Cham: Springer International Publishing, 2017, pp. 182–196.

[35] Neo4j. Indexes for search performance. [Online]. Available: https://neo4j.com/docs/cypher-manual/current/administration/indexes-for-search-performance/#administration-indexes-introduction

[36] C. Lei, R. Alotaibi, A. Quamar, V. Efthymiou, and F. Özcan, "Property Graph Schema Optimization for Domain-Specific Knowledge Graphs," *arXiv e-prints*, p. arXiv:2003.11580, Mar. 2020.

[37] M. Jota and R. Parra, "DISEÑO FíSICO PARA LA BASE DE DATOS SNB SOBRE UN GESTOR ORIENTADO A GRAFOS," *Universidad Simón Bolivar*, 2019.

[38] K. Seo, J. Ahn, and D.-H. Im, "Optimization of shortest-path search on rdbms-based graphs," *ISPRS International Journal of Geo-Information*, vol. 8, no. 12, p. 550, 2019.

[39] F. Rusu and Z. Huang, "In-depth benchmarking of graph database systems with the linked data benchmark council (LDBC) social network benchmark (SNB)," *CoRR*, vol. abs/1907.07405, 2019. [Online]. Available: http://arxiv.org/abs/1907.07405

[40] A. B. Ammar, "Query optimization techniques in graph databases," *CoRR*, vol. abs/1609.01893, 2016. [Online]. Available: http://arxiv.org/abs/1609.01893

[41] G. Szárnyas, J. Maginecz, and D. Varró, "Evaluation of optimization strategies for incremental graph queries," *Periodica Polytechnica Electrical Engineering and Computer Science*, vol. 61, no. 2, pp. 175–192, 2017.

[42] A. Gubichev, "Query processing and optimization in graph databases," Dissertation, Technische Universität München, München, 2015.

[43] J. Cheng, Y. Ke, and W. Ng, "Efficient query processing on graph databases," *ACM Trans. Database Syst.*, vol. 34, no. 1, Apr. 2009. [Online]. Available: https://doi.org/10.1145/1508857.1508859

[44] H. He and A. K. Singh, "Graphs-at-a-time: Query language and access methods for graph databases," in *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 405–418. [Online]. Available: https://doi.org/10.1145/1376616.1376660

[45] J. M. C. Lovelle. Metodología de la Investigación en Informática. [Online]. Available: http://di002.edv.uniovi.es/~cueva/asignaturas/doctorado/2008/MetodologiaInvestigacion1.pdf

[46] B. Lutkevich and A. S. Gillis. High availability. [Online]. Available: https://searchdatacenter.techtarget.com/definition/high-availability

[47] M. Hunger. 5 secrets to More Effective Neo4j 2.2 Query Tuning. [Online]. Available: https://neo4j.com/blog/neo4j-2-2-query-tuning/#:~:text=Query%20Tuning%20Tip%20%231%3A%20Use,is%20used%20in%20your%20query.

[48] IBM. Vistas materializadas. [Online]. Available: https://www.ibm.com/docs/es/psfa/7.1.0?topic=grammar-materialized-views

[49] Microsoft. Clustered and nonclustered indexes described. [Online]. Available: https://docs.microsoft.com/en-us/sql/relational-databases/indexes/clustered-and-nonclustered-indexes-described?view=sql-server-ver15#:

~:text=An%20index%20contains%20keys%20built,key%20values%20quickly%20and%20efficiently.&text=Clustered%20indexes%20sort%20and%20store,based%20on%20their%20key%20values.

[50] A. Networks. Buffer overflow. [Online]. Available: https://avinetworks.com/glossary/buffer-overflow/#main

[51] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965. [Online]. Available: http://www.jstor.org/stable/2333709

[52] F. Wilcoxon, *Individual Comparisons by Ranking Methods*. New York, NY: Springer New York, 1992, pp. 196–202. [Online]. Available: https://doi.org/10.1007/978-1-4612-4380-9_16

[53] R. Angles, J. B. Antal, A. Averbuch, P. Boncz, O. Erling, A. Gubichev, V. Haprian, M. Kaufmann, J. Lluís Larriba Pey, N. Martínez, J. Marton, M. Paradies, M.-D. Pham, A. Prat-Pérez, M. Spasić, B. A. Steer, G. Szárnyas, and J. Waudby, "The LDBC Social Network Benchmark (version 0.4.0-SNAPSHOT)," 2020.