INSTITUTO TECNOLÓGICO DE BUENOS AIRES – ITBA

ESCUELA DE INGENIERÍA Y GESTIÓN

Modelado y Consulta de Data Warehouses Usando Bases de Datos de Grafo

AUTORES: **Besteiro, Maria Florencia (Leg. Nº 51117)**

Valverde Melito, Maximiliano Javier (Leg. Nº 51158)

DOCENTE/S TITULAR/ES O TUTOR/ES: Vaisman, Alejandro

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE

INGENIERO EN INFORMÁTICA

Lugar: Buenos Aires, Argentina

Fecha: Junio, 2019

Tabla de Contenido

INTRODUCCION

El objetivo de este proyecto es poder modelar y consultar un data warehouse usando bases de datos de grafos, de forma de poder comparar la performance del mismo en contraste con su alternativa relacional. Para ello se definirán distintos tipos de consultas que luego serán ejecutadas en ambos modelos, con el fin de determinar que situaciones se destaca cada uno de ellos en términos de performance.

ESTADO DEL ARTE

Data Warehouse

A medida que las empresas crecen, el manejo y análisis de sus datos se vuelve más complejo lo que genera mayor dificultad para la toma de decisiones. Fue por esto que en 1988 nació la idea de **Data Warehouse** [1] [2] a manos de los investigadores de IBM, Barry Devlin y Paul Murphy, con el objetivo de proveer un modelo de arquitectura para el flujo de datos desde los sistemas operacionales a los entornos de soporte de decisiones.

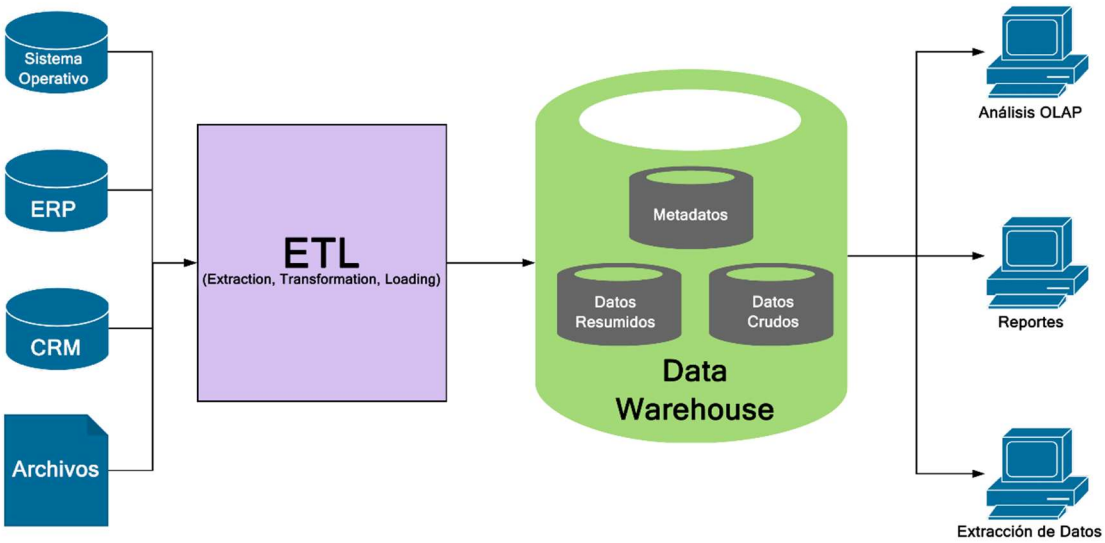


FIGURA 1 - ESQUEMA GENERAL DE UN DW

Este sistema es utilizado para **reportes** y **análisis de datos,** mientras que a su vez es considerado un componente central de **Inteligencia de Negocios,** es decir, el conjunto de metodologías, procesos, arquitecturas y tecnologías que son utilizadas para transformar datos crudos en información útil para la toma de decisiones.

Debido al volumen de información que poseen los data warehouses, los mismos requieren un nivel de normalización menor de forma de poder contar con la mejor performance al momento de utilizarlos. Esto se obtiene con **Modelado Multidimensional.**

Modelado Multidimensional

Organiza la información en *hechos* o *facts* relacionados con *dimensiones* o *dimensions*. Un hecho representa el foco del análisis, mientras que una dimensión provee de contexto a un hecho. A su vez pueden incluir atributos que forman jerarquías, por ejemplo, mes → cuatrimestre → año.

Dependiendo del caso existen las *medidas* o *measures* que suelen ser valores numéricos con los que se busca cuantificar un hecho. Por ejemplo, cantidad de ventas.

En caso de que en el modelado existan las medidas estas pueden ser usarse en métodos de agregación cuando se busca observar los datos desde otro nivel en la jerarquía de una dimensión.

Fact Table

Tabla principal en un modelo multidimensional. Contiene métricas opcionales y claves foráneas a las tablas de dimensión.

Dimension Table

Ofrecen características descriptivas de los hechos mediante sus atributos. Se relacionan a la *Fact Table* a través de claves foráneas. No existe un límite para la cantidad de dimensiones. Poseen una o más relaciones jerárquicas.

Bases de Datos de Grafos

Base de datos NoSql que usa el modelo de grafos para guardar, mapear y consultar información. En esencia es una colección de *nodos* y *arcos* donde cada nodo representa una entidad y cada arco una conexión o relación entre dos nodos.

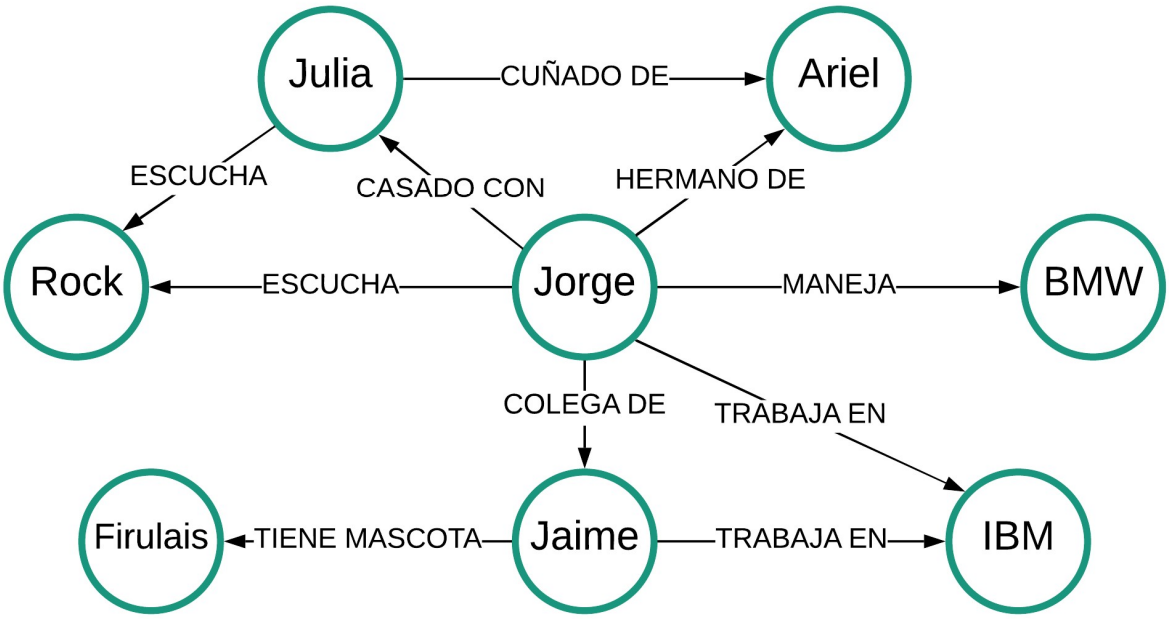


FIGURA 2 - BASE DE DATOS BASADO EN GRAFO

En la Figura 2, se muestran instancias (nodos) de distintas entidades como Julia (entidad Persona), Firulais (entidad Animal), IBM (entidad empresa) entre otros y relaciones entre los mismos como "cuñado de" entre entidades de tipo Persona, "trabaja en" conectando un nodo de tipo Persona con uno de tipo Empresa, etc.

Los casos en donde es recomendado usar este tipo de bases son

- Datos altamente conectados
- Necesidad de un esquema flexible
- Consultas similares a cómo piensan las personas

Las bases de datos de grafos [3] suelen utilizarse en aplicaciones tales como redes sociales, sistemas de recomendaciones, machine learning y detección de fraude. Esto se debe a que el foco de este tipo de bases yace en la forma de representar relaciones entre distintos puntos de datos, lo cual permite a sistemas de este tipo usar patrones con el objetivo de analizar de forma más inteligente la información disponible.

Modelo de Grafos de Propiedad

Un grafo está formado por nodos y arcos donde ambos pueden contener propiedades. Estas propiedades se almacenan siguiendo la forma clave-valor.

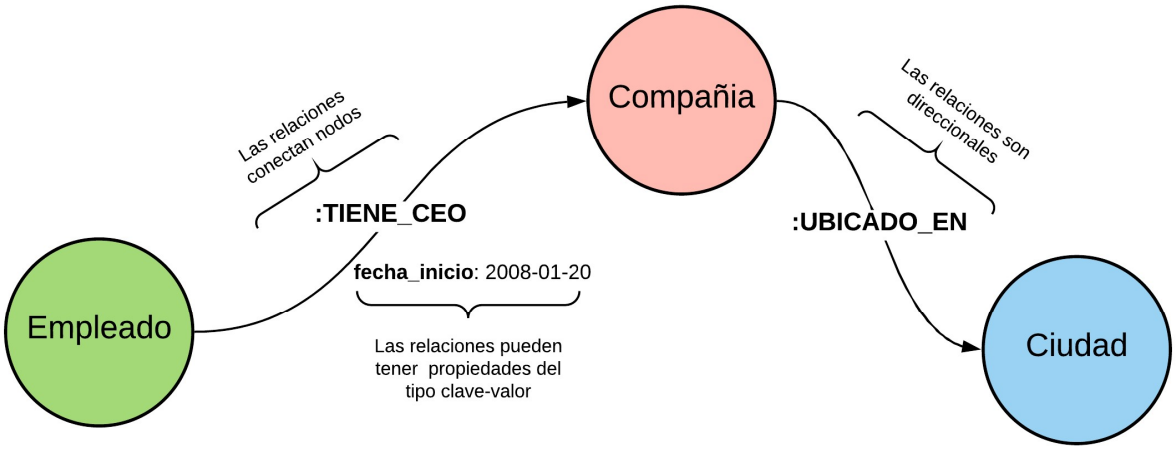


FIGURA 3 – MODELO DE GRAFOS DE PROPIEDAD [4]

En la Figura 3 contamos con las entidades Empleado, Compañía y Ciudad. Entre los nodos con etiqueta Empleado y Compañía existe un arco "TIENE_CEO" que posee una propiedad con clave "fecha_inicio" y valor "2008-01-20"

Nodos

Entidades del grafo. Pueden contener distintos atributos, en la forma clave-valor, llamados *propiedades* o *properties*. Pueden poseer *etiquetas* o *labels* que representen los diferentes roles que existen dentro del dominio. Estas pueden ser utilizadas para adjuntar metadatos (índices o restricciones) a ciertos nodos.

Arcos

Proveen relaciones directas entre dos nodos. Estos tienen una dirección, tipo, nodo origen y nodo destino. A su vez, así como los nodos, pueden poseer propiedades. Aun siendo dirigidas, las relaciones pueden navegarse de manera eficiente en cualquier dirección.

Este tipo de grafos posee una restricción conocida con el nombre de "*no broken links*" o "ausencia de links rotos", esto significa que todo arco debe poseer tanto nodo origen como nodo destino, de forma de asegurar que siempre que existan ambos extremos de una relación. Consecuentemente, la eliminación de un nodo origen o destino hará que el arco que existía entre ellos sea eliminado junto al nodo.

Algunas de las bases de datos existentes de este tipo son ArangoDB, OrientDB y Neo4j.

Arango DB

Base de datos open-source multimodelo que soporta grafos, pares clave-valor y documentos. Su motor de base de datos es schema-free implementado en C/C++ y Javascript y es compatible con distintos sistemas operativos como Linux, OS X, Windows, etc. Representa documentos en un formato propietario basado en JSON llamado VelocyPack. Su lenguaje declarativo es AQL (ArangoDB Query Language).

OrientDB

Open-source y multimodelo soporta objetos, documentos, pares clave-valor y grafos. Desarrollado en Java y multiplataforma. Utiliza un lenguaje SQL que provee extensiones específicas para manejo de grafos y especificación de patrones.

Neo4J

Base de datos de grafos open-source implementada en Java y Scala compatible con distintos sistemas operativos. Su modelo de datos consiste en objetos de tipo nodo que pueden ser conectados por nombre y dirigidos a objetos de tipo arcos. Se guarda la información de los grafos en diferentes archivos para cada parte del mismo, organizado de manera que facilite su procesamiento. Posee Cypher como lenguaje declarativo de consultas similar a SQL pero optimizado para el manejo de grafos.

En la Tabla 1 se puede ver un cuadro comparativo de distintas características entre las tres bases de grafos mencionadas y la base de datos relacional PostgreSQL.

Funcionalidad	ArangoDB	Neo4j	OrientDB	PostgreSQL
Categoría	NoSQL	NoSQL	NoSQL	Relacional
Lanzamiento	2012	2007	2010	1996
Modelo de Base de Datos	Grafo Documento Clave-Valor	Grafo	Grafo Documento Clave-Valor Objeto	Relacional Objeto
Modelo de Grafos	Grafo de Propiedad	Grafo de Propiedad	Grafo de Propiedad	-
Nativo de Grafos	Sí	Sí	Sí	-
Adyacencia sin Índice	No	Sí	Sí	-
Implantación	C++ JavaScript	Java Scala	Java	C
Índices	Sí, secundario	Sí, secundario	Sí, secundario	Sí, secundario
Transacciones	Sí, ACID	Sí, ACID	Sí, ACID	Sí, ACID
Esquema de Datos	Sin Esquema	Sin Esquema	Con/Sin Esquema	Con Esquema
Integridad Referencial	Sí, bordes	Sí, bordes	Sí, bordes	Sí
Tipado de Datos	Sí	Sí	Sí	Sí
Lenguaje de Consulta	AQL	Cypher	SQL Extendido	SQL
Stored Procedures	AQL JavaScript	Java	SQL JavaScript Groovy	PL/pgSQL PL/Tel PL/Perl PL/Python
Funciones de Grafos	Sí	Sí	Sí	No
Drivers	JavaScript Java PHP Python Perl .Net	Java C/C++ JavaScript PHP Ruby Python	Java JDBC JavaScript PHP Ruby Python	C/C++ JDBC PHP Ruby Python ODBC
Métodos de Acceso	REST HTTP	REST HTTP Java API	REST HTTP Java API Binary	JDBC C API
Triggers	Sí, vía FOXX Queues	Sí, vía Event Handler	Sí, vía Hooks	Sí
Concurrencia	Sí	Sí	Sí	Sí
Conceptos de Usuario	Sí	Sí	Sí	Sí
Durabilidad	Sí	Sí	Sí	Sí
Licencia Comunitaria	Apache v2	GPL v3	Apache v2	BSD
Replicación	Master / Master Master / Agent	Master / Slave	Master / Master Master / Slave	Master / Slave
Fragmentación de Datos	Sí	No	Sí	No
Caching	Datos y Resultados de Consultas	Datos y Planes de Consultas	Datos y Resultados de Consultas	Datos y Resultados de Consultas

TABLA 1 – BASES DE DATOS DE GRAFOS [5]

Bases de Datos Relacionales

Base de datos [6] basada en el *Modelo Relacional* [7] , un método de estructuración de la información usando relaciones representadas en tablas. Las mismas deben contener un encabezado y un cuerpo donde el primero consta de una lista de *atributos* en la forma de *columnas* de la relación y el segundo un set de *registros o tuplas* conteniendo la información de esta organizada en *filas*.

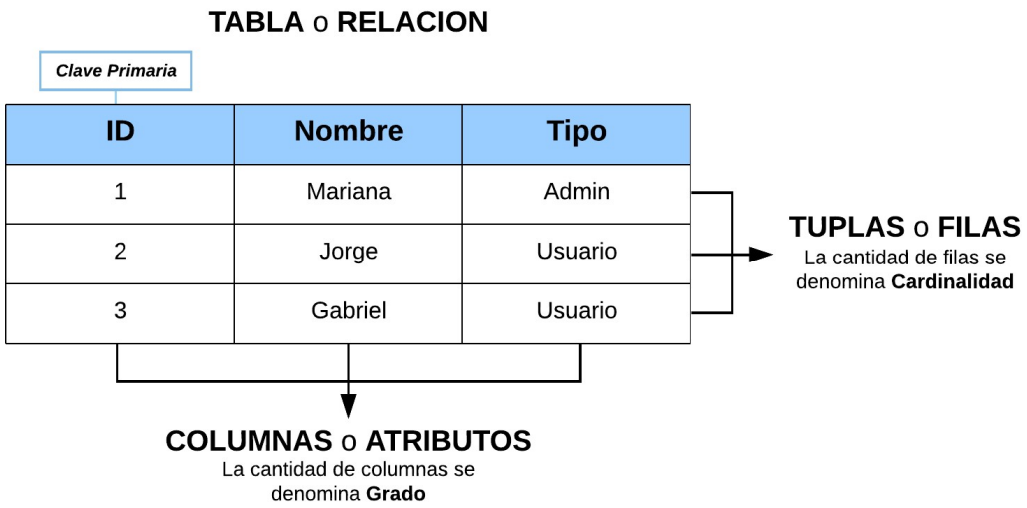


FIGURA 4 - EJEMPLO DEL MODELO RELACIONAL [8]

Como se puede ver en el ejemplo de la Figura 6, la tabla posee los atributos ID, Nombre y Tipo donde ID es la clave primaria de la misma. A su vez posee tres tuplas con información de los usuarios. Esta es una relación de grado y cardinalidad tres.

Estas bases de datos utilizan el lenguaje de *SQL (Structured Query Language)* para gestionar las mismas y manipular la información.

En la actualidad existen múltiples bases de datos de este tipo. Algunas de ellas son PostgreSQL, Microsoft SQL Server y Oracle entre otras. Una breve comparación entre ellas puede verse en la Tabla 2.

	Microsoft SQL Server	Oracle	PostgreSQL
Año de lanzamiento	1989	1980	1989
Desarrollador	Microsoft	Oracle	PostgreSQL Global Development Group
Licencia	Comercial	Comercial	Open Source
Lenguaje de implementación	C++	C y C++	C
Compatibilidad de servidores	Linux, Windows	AIX, HP-UX, Linux, OS X, Solaris, Windows, z/OS	FreeBSD, HP-UX, Linux, NetBSD, OpenBSD, OS X, Solaris, Unix, Windows
Esquema de datos	Si	Si	Si
APIs y otros métodos de accesos	OLE DB, TDS, ADO.NET, JDBC, ODBC	ODP.NET, OCI, JDBC, ODBC	Native C Library, Streaming API for large objects, ADO.NET, JDBC, ODBC
Lenguajes de programación soportados	C#, C++, Delphi, Go, Java, Javascript, PHP, Python, R, Ruby, Visual Basic	C, C#, C++, Clojure, Cobol, Delphi, Eiffel, Erlang, Fortran, Groovy, Haskell, Java, Javascript, Lisp, Objective C, OCaml, Perl, PHP, Python, R, Ruby, Scala, Tcl, Visual Basic	.Net, C, C++, Delphi, Java, Javascript, Perl, PHP, Python, Tcl
Conceptos de transacción	ACID	ACID	ACID

TABLA 2 – BASES DE DATOS RELACIONALES [9]

DESCRIPCIÓN DEL PROBLEMA

Para este proyecto buscamos realizar un estudio comparativo entre el rendimiento de una base de datos de grafos (NEO4J) con una base de datos relacional (POSTGRESQL). Para esto utilizaremos datos provenientes de Musicbrainz [10] , una enciclopedia abierta que recolecta información relacionada con la música. Para esto debemos analizar el esquema actual de la enciclopedia, diseñar un esquema de data warehouse para la base relacional y uno para la base de grafos, idear queries para correr en ambas y en base a los resultados sacar conclusiones en cuanto a las métricas establecidas.

Esquema de Musicbrainz[1]

MetaBrainz, una fundación estadounidense sin fines de lucro que cree en el libre acceso a la información. Esta base de datos crece gracias a la contribución de usuarios alrededor del mundo

[1] Referirse al Anexo para más detalle sobre el Esquema Parcial de la enciclopedia

que aportan contenido a la misma, cumpliendo con los requisitos y pasos a seguir indicados en su guía de uso de la enciclopedia.

Entidades Principales

- AREA: ubicación geográfica que puede ser del tipo país, ciudad, región, etc.
- ARTIST: musico, grupo u otros profesionales del ámbito.
- EVENT: recitales, festivales o cualquier otro tipo de evento relacionado al que personas puedan asistir.
- LABEL: sellos discográficos y/o nombres de las compañías que los controlan.
- PLACE: nombre de una ubicación no geográfica donde pueden llevarse a cabo eventos musicales. Por ejemplo, bares, estadios, teatros, etc.
- RECORDING: mezcla y/o ediciones únicas.
- RELEASE: entidad que engloba uno o más recordings. Representa una versión especifica que puede ser comprada de un álbum, EP, Single, etc.
- RELEASE GROUP: entidad que agrupa uno o varios los releases en una misma entidad lógica. Cada release puede pertenecer a solo un grupo. Contiene las distintas versiones de un reléase, por ejemplo, versiones para distintos países, versiones Deluxe, etc.
- SERIES: secuencias de release groups, releases individuales, recordings, obras o events con un tema común.
- URL: Representa una URL a un recurso externo a musicbrainz.
- WORK: composición detrás del recording

Entidades Secundarias

- ARTIST_CREDIT: Representa una lista de artistas que trabajaron juntos
- MEDIUM: Representa un segmento de un audio que está incluido en un reléase

Pondremos el foco en los e*ventos* (Events), *lanzamientos* (Releases) y las entidades relevantes con las que estos se relacionan como lo son el Á*rea*, *Artistas* (Artists), *Créditos de Artistas* (Artist Credit)

Diseño de Data Warehouse

Luego de analizar el esquema de Musicbrainz se decidió focalizar en los eventos y los lanzamientos ya que estos cumplen con los requisitos para ser utilizados como hechos en el diseño del data warehouse. Las dimensiones de estos están formadas por la fecha y el lugar donde ocurrieron y los artistas involucrados a la vez de una dimensión evento y una lanzamiento.

Para el diseño de las dimensiones se analizaron los niveles de jerarquía posibles. En el caso de la dimensión temporal fecha se definieron los niveles día, mes, bimestre, trimestre, cuatrimestre,

semestre y año. De esta manera un usuario podría buscar información agrupando por cualquiera de estos niveles si así lo quisiera.

Para el caso de la dimensión geográfica área solo se utilizaron dos niveles de agrupación, siendo estos ciudad y país. Se diseñaron de esta forma debido a que no todos los países utilizan los mismos conceptos para dividir su territorio, pero el concepto de ciudad es conocido y utilizado de manera global.

Las dimensiones restantes (artista, crédito de artista, evento y lanzamiento) no poseen niveles en los que puedan ser agrupados. El análisis de las mismas fue sobre qué información podría ser relevante para un usuario.

De esta manera el data warehouse relacional se divide en dos tablas de hechos, eventos y lanzamientos, y seis tablas de dimensiones que son artista, créditos de artistas, fecha, área, evento y lanzamiento.

Modelo Relacional

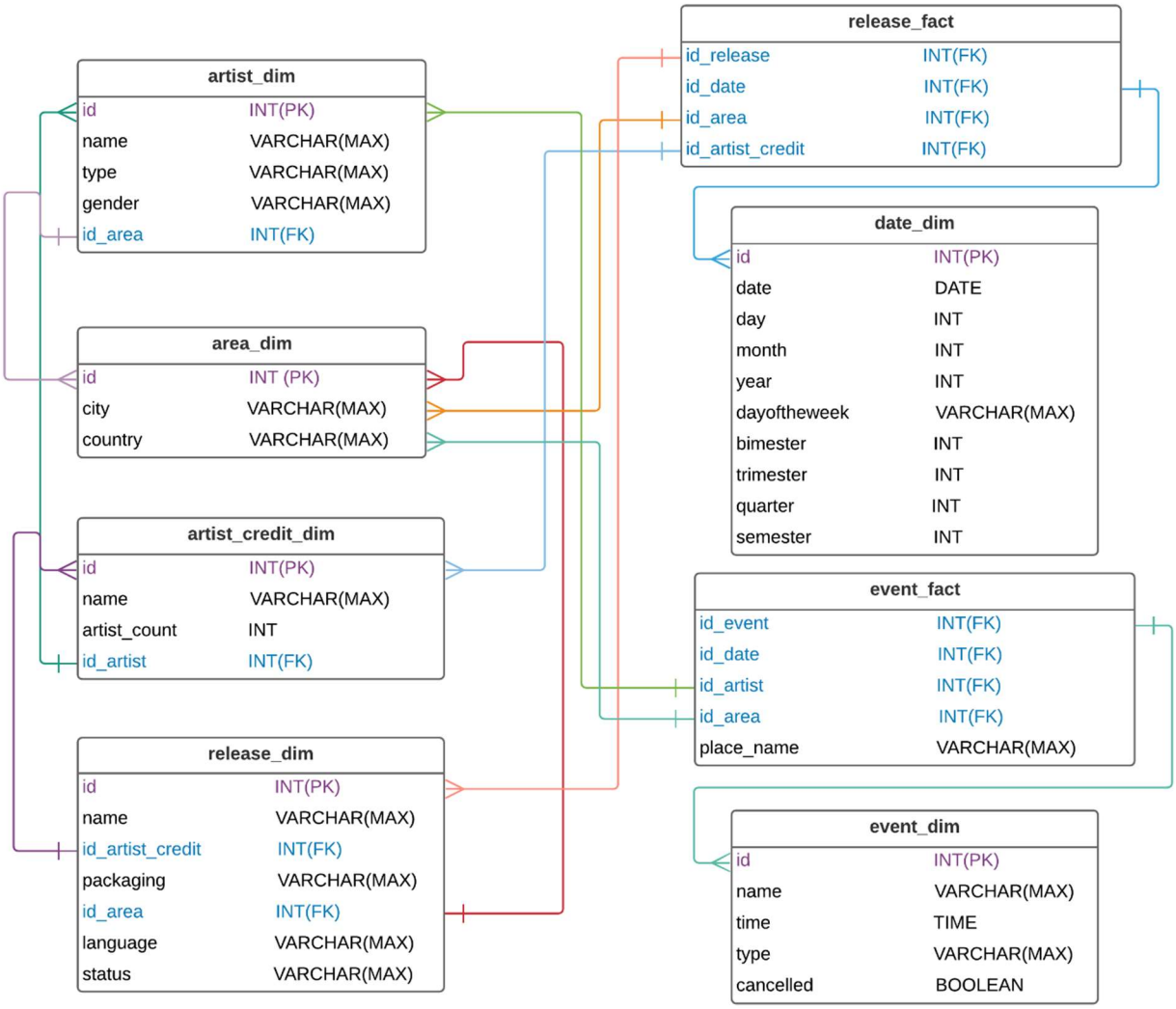


FIGURA 4 – ESQUEMA DE DATA WAREHOUSE RELACIONAL

Fact Tables

El esquema cuenta con dos fact tables: *Release Fact* y *Event Fact*.

Release Fact

Hechos referidos a los lanzamientos por un artista o un conjunto de artistas. Los mismos son descriptos por la fecha en la que ocurrieron, lugar y artistas que colaboraron. Contiene claves foráneas a *date_dim, area_dim, artist_credit_dim y release_dim*

Event Fact

Hechos referidos a eventos en el que participaron uno o más artistas. Los mismos son descriptos por la fecha en la que se realizaron, lugar, artista que participó y el nombre del lugar donde tuvo lugar (place_name) como por ejemplo "Queen's Hall". Contiene claves foráneas a *date_dim, artist_dim, area_dim* y *event_dim*

Dimension Tables

El esquema cuenta con seis dimension tables: *Artist Dim*, *Area Dim*, *Artist Credit Dim*, *Release Dim*, *Date Dim* y *Event Dim*.

Artist Dim

Describe a un artista.

Sus propiedades son

- **id**: Identificador del artista
- **name**: Nombre del artista
- **tipo**: Los tipos posibles de un artista son Orchestra, Choir, Group, Person, Character, Other
- **gender**: No se refiere al género musical, si no al propio del artista (Male, Female, Other)
- **id_area**: Clave foránea que refiere a un área de dónde es el artista. Puede ser el área donde comenzó su carrera o con que se lo identifica.

Artist Credit Dim

Describe los créditos de un artista. Este puede ser un grupo o artista solista, o una colaboración entre artistas. Por ejemplo "Queen & David Bowie"

Sus propiedades son

- **id**: Identificador
- **name**: Nombre del crédito de los artistas
- **artist_count**: Cantidad de artistas involucrados
    - En el ejemplo "Queen & David Bowie "su valor es 2
- **id_artist**: Clave foránea que relaciona el crédito con los artistas involucrados.
    - En el caso de "Queen & David Bowie" tendrá una referencia al artista "Queen" y otro al artista "David Bowie"

Release Dim

Describe los lanzamientos realizados por artistas y sus colaboraciones (artist_credit).

Sus propiedades son

- **id**: Identificador
- **name**: Nombre del lanzamiento
- **id_artist_credit**: Clave foránea que relaciona el lanzamiento con el crédito de los artistas involucrados.
- **packaging**: Tipo de packaging. Los posibles son Super Jewel Box, Digipak, Keep Case, Cardboard/Paper Sleeve, Cassette Case, Book, Fatbox, Snap Case, Gatefold Cover, Discbox Slider, Other, None, Jewel Case, Slim Jewel Case, Digibook
- **id_area**: Clave foránea que relaciona el lanzamiento con el área (país) donde se lanzó.
- **language**: Lenguaje del lanzamiento. Ejemplos de estos son English, Spanish, Zaiwa, etc.
- **status**: Estado del lanzamiento. Los estados posibles son Official, Promotion, Bootleg, Pseudo-Release

Event Dim

Describe eventos musicales donde uno o más artistas participan.

Sus propiedades son

- **id**: Identificador
- **name**: Nombre del evento
- **time**: Tiempo de duración del evento
- **type**: Tipo del evento. Los posibles son Concert, Festival, Launch event, Convention/Expo, Masterclass/Clinic
- **cancelled:** Informa si el evento fue cancelado

Area Dim

Describe distintos tipos de área. Los de mayor relevancia que encontramos y plasmamos en la tabla son ciudad (City) y país (Country). El registro representa el área de mayor detalle, es decir, si es una ciudad sería “id: 2372, city: Sevastopol’, country: Ukraine” y en el caso de un país sería “id: 219, city: Unknown, country: Ukraine”.

Sus propiedades son

- **id**: Identificador
- **city**: Nombre de la Ciudad. Si el registro es un país, city lleva el valor **“Unknown”**
- **country**: Nombre del país.

Para poder representar aquellas entidades en las que el área se desconoce, existe el registro del área desconocida. El mismo es

id	city	country
74489	Unknown	Unknown

REGISTRO 1 – AREA DESCONOCIDA

Date Dim

Dimensión que representa una fecha. Fue creada mediante un programa en c# para obtener toda la información de dicha fecha como a qué trimestre pertenece, qué día de la semana fue, etc. De esta manera se pueden hacer consultas por otras medidas de un periodo de tiempo (trimestre, cuatrimestre, etc).

Sus propiedades son

- **id:** Identificador
- **date:** Fecha
- **day:** Día de la fecha. En el caso de la fecha "15/06/2018" day sería 15
- **month:** Mes de la fecha. En el ejemplo del "15/06/2018" sería 06
- **year:** Año de la fecha. En el ejemplo del "15/06/2018" sería 2018
- **dayoftheweek:** Día de la semana de la fecha. En el ejemplo sería "Friday"
- **bimester:** Bimestre al que pertenece la fecha. Esta columna puede tomar cualquier número de 1 a 6
- **trimester:** Trimestre al que pertenece la fecha. Esta columna puede tomar cualquier número de 1 a 4
- **quarter:** Cuatrimestre al que pertenece la fecha. Esta columna puede tomar cualquier número de 1 a 3
- **semester:** Semestre al que pertenece la fecha. Esta columna puede tomar el valor 1 o 2

Para poder representar aquellas entidades en las que se desconoce la fecha, existe el registro de la fecha desconocida. El mismo es

id	date	day	month	year	dayoftheweek	bimester	trimester	quarter	semester
80000	NULL	NULL	NULL	NULL	Unknown	NULL	NULL	NULL	NULL

REGISTRO 2 – FECHA DESCONOCIDA

Modelo de Grafos

Como base para el diseño del grafo de propiedad, se utilizó el propuesto para el data warehouse relacional y se realizaron las modificaciones apropiadas para adaptarlo de manera acorde al modelo de grafos. Las entidades se tradujeron a nodos y se crearon etiquetas para representar cada uno de los hechos y dimensiones del warehouse. Las claves foráneas en ambos de estos casos fueron eliminadas y reemplazadas por arcos entre los nodos, como se puede ver en el caso Release Fact, donde se creó la relación *released_in* en lugar de depender del *id_area* como sucedía en el modelo relacional.

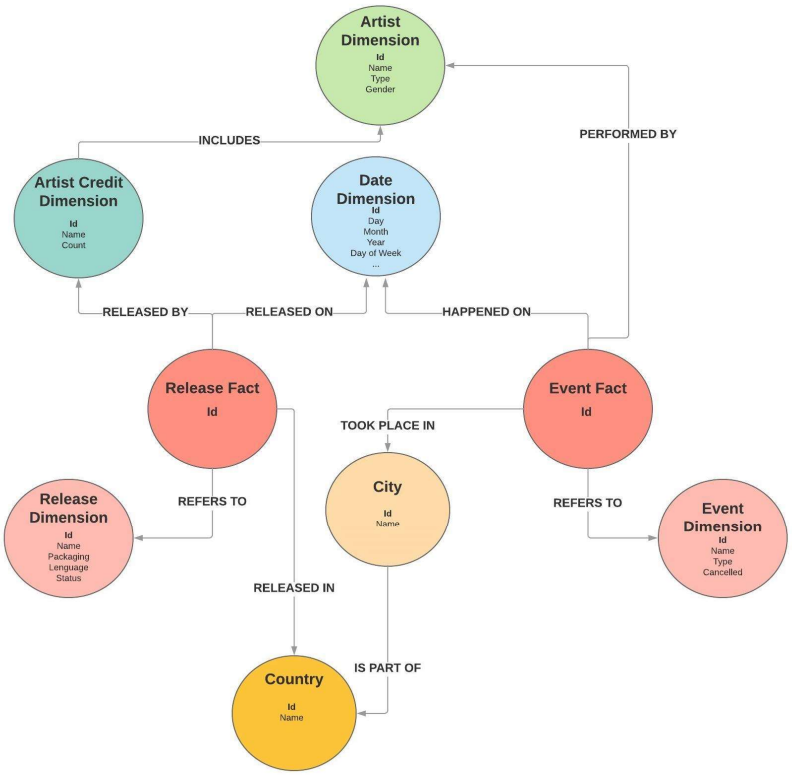


FIGURA 5 – ESQUEMA DE GRAFOS

Además, tal cómo se puede puede ver en la Figura 6, la entidad Area Dim que existía en el data warehouse relacional fue dividida en dos nodos: City y Country, aprovechando la facilidad con la que este tipo de jerarquías pueden representarse en los grafos de propiedad de forma de modelar de una manera más leal los datos.

Exportación de Data Warehouse

Debido a que la enciclopedia de Musicbrainz es almacenada en una base de datos relacional, se comenzó el desarrollo del proyecto por el diseño del data warehouse que sigue el mismo modelo y la migración de los datos a la nueva base. Para esto se crearon scripts de migración para cada entidad en particular, pero no sin antes analizar los datos para identificar y tratar aquellos casos que demandaban un análisis especial. Uno de estos casos es el área. Para poder contar con un mejor manejo de los datos, se decidió reducir la granularidad de este atributo hasta el nivel de *ciudad* en aquellos casos donde fue posible, conservando únicamente el *país* en aquellos donde no. Se analizaron en profundidad los datos y usando la estructura de estos se generaron scripts capaces de llevar la información al nivel buscado.

En el caso de las *fechas*, dado que en Musicbrainz eran manejadas simplemente como atributos, se planteó la existencia de una nueva entidad de forma de poder tratarla como dimensión. Para esto se creó un programa capaz de recibir un año inicial y uno final, y generar a partir de estos registros por día conteniendo toda la información propia de la dimensión, como, por ejemplo, día de la semana, semestre, mes, etc. Una vez poblada la tabla, se desarrollaron scripts que pudieran mapear uno a uno cada valor de fecha con un elemento de la tabla de dimensión.

Uno de los problemas encontrados durante esta etapa de la migración fue la existencia de registros de área que estaban aisladas y por ende no podían ser relacionadas con ningún país existente o estaban duplicadas en concepto, pero no en nombre. Estos casos particulares fueron resueltos individualmente en base a la información que se encontró tanto en Musicbrainz como en la web, o, en algunos casos, criterio propio. Un claro ejemplo de lo mencionado es el caso de un registro de área que llevaba el nombre de “worldwide”, el cual se decidió mapear al registro de *área desconocida*.

Otro de los problemas fue la existencia de registros repetidos una vez realizada la migración. Esto se debió a la utilización de las tablas de relaciones entre entidades, como por ejemplo *l_event_artist*, que en distintas oportunidades relacionan a las mismas entidades entre sí, a nivel registro, con el mismo tipo de conexión, conceptualmente, pero distinto nombre. Para poder saltear esta dificultad se desarrollaron scripts de depuración que eliminaran estas repeticiones sin sacrificar la integridad de los mismos ni eliminar información relevante para el análisis.

Una vez poblado el data warehouse relacional, lo único restante para dar por finalizada la migración era cargar los datos en NEO4J. Para esto, se exportó cada entidad del modelo relacional y se llevaron a cabo scripts que, utilizando los datos, crearan todos nodos y relaciones acorde a nuestro diseño.

Luego de finalizar la migración, se prosiguió con la etapa de validación, con el fin de comprobar que la información reflejada en ambos warehouses sea consistente, tanto entre ellos mismos como en Musicbrainz. Durante esa etapa se identificaron varias inconsistencias menores que surgieron durante la exportación; registros corrompidos a causa de problemas de encoding, información duplicada en Musicbrainz que no habíamos identificado previamente, y principalmente, nodos repetidos en el grafo a causa de la representación de relaciones de unos a muchos en el modelo relacional. Esto se midió principalmente en los nodos del tipo *Event Fact* y *Artist Credit*, en los cuales existían varias entradas con un mismo *id* por cada *Artist* con el que se relacionaban, por lo

que eran creadas como nodos separados, generando así nodos y arcos adicionales directamente proporcionales a la multiplicidad de la relación. Para solucionar esto, se tuvieron que eliminar todos los nodos con *id* repetido en Neo4j, dejando únicamente una ocurrencia de cada uno. Gracias a restricción de *no broken links* del modelo de grafos, este proceso aseguraba también la eliminación de los arcos repetidos.

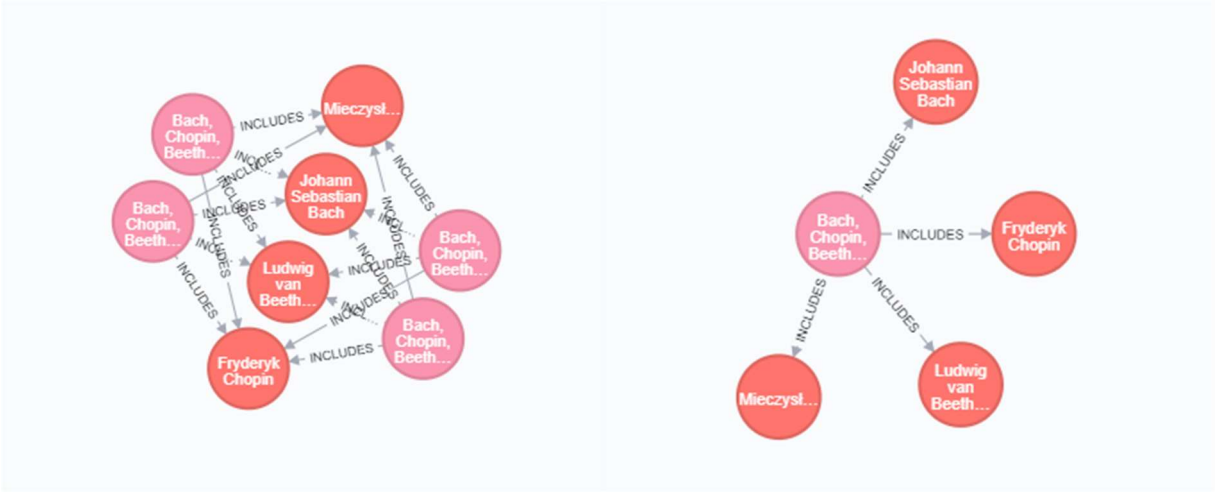


FIGURA 6 - ELIMINACIÓN DE REPETIDOS

En la Figura 7, se puede ver el resultado de una consulta simple realizada en Neo4J, cuyo objetivo es obtener los *Artist*, representado por nodos en rojo, pertenecientes a cierto *Artist Credit*, representados por nodos rosas. El grafo de la izquierda corresponde al de la base con nodos repetidos, donde se puede ver que el nodo de Artist Credit se encuentra repetido por cada Artist que contiene, mientras que el grafo de la derecha corresponde al resultado sobre la base depurada, donde esos nodos repetidos ya no existen.

ESTUDIO COMPARATIVO

Para la comparación se diseñaron distintos tipos de consultas

- Top N
- Conjuntos
- Recomendaciones

Tipos de Consultas

Q1: Top N

Son consultas que apuntan a obtener las entidades con mayor cantidad de relaciones u ocurrencias de algún tipo. Algunos tipos de consulta de este tipo que se van a tomar en consideración son

- Ciudades con mayor cantidad de Eventos (Q1.1)
- Artistas con más colaboraciones en Artist Credit (Q1.2)

Q2: Conjuntos

Para este tipo de consultas el enfoque se encuentra en los conjuntos de artistas que participaron de eventos y si los mismos colaboraron o no en lanzamientos.

Categorías

- Conjuntos de X artistas que participaron en N eventos juntos (Q2.1)
- Conjuntos de X artistas que participaron en N eventos juntos y colaboraron en M lanzamientos (Q2.1)
- Conjuntos de X artistas que participaron en N eventos juntos, pero no colaboraron en ningún lanzamiento. (Q2.3)

El parámetro N es un número entero mayor a 1 de artistas mientras que el parámetro M es un número entero mayor a 0 de eventos en el que los artistas participaron juntos. El parámetro P cumple con las características de M y representa la cantidad de lanzamientos en los que los artistas colaboraron.

Q3: Recomendaciones

Consultas que devuelven un set de datos a partir de ciertos parámetros.

Dos tipos de recomendaciones

- Recomendar artistas en base a un artista X dado como parámetro y una profundidad N de búsqueda (Q3.1)
- Recomendar eventos en base a un evento Y dado como parámetro donde el set de resultados este ordenado en base a la cantidad de matches de artistas que hay entre ambos eventos. (Q3.2)

Optimizaciones

Durante el proceso de probar las queries se implementaron ciertas optimizaciones para analizar si existían maneras de agregar cambios en el esquema que generaran una mejora en los resultados ya que los mismos no reflejaban lo que inicialmente suponíamos iba a suceder. Dado que es una consulta de camino el supuesto que manejamos es que la performance de Neo4j superaría a la de PostgreSQL.

Grafos

Consiste en agregar arcos entre los nodos que previamente se relacionaban mediante algún nodo de otro tipo. Con esto se busca optimizar la búsqueda de patrones y ofrecer atajos que pueden llegar a ser beneficiosos para el procesamiento. Uno de estos tipos de arcos es *Collab With* como se puede ver en la Figura 8, el cual se crea entre nodos del tipo *Artist* que hayan colaborado, es decir, que se conecten por medio de un mismo *Artist Credit*. Con la existencia de estos arcos se buscó principalmente mejorar la performance de las recomendaciones, ya que salteando los nodos de Artist Credit, se podría llegar a niveles más profundos en menos tiempo, reduciendo inclusive la complejidad de los patrones al momento de escribir las consultas.

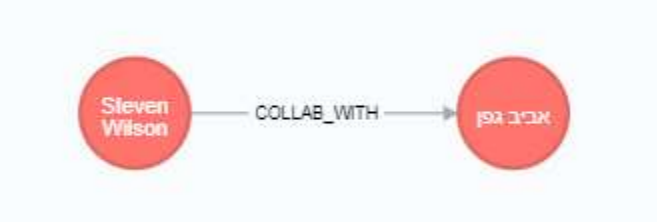


FIGURA 7 - ARCO DE COLABORACION

Se analizó a su vez la posibilidad de hacer algo similar para *Events* de forma de mejorar la velocidad de las consultas de conjuntos, pero al escribir las queries en Neo4j nos topamos con que trabajar de esta forma era contraproducente ya que conduce a caer en prácticas como manipulación de array y otras operaciones de texto y conjuntos que iban en contra del paradigma de grafos, por lo que se decidió no implementarla.

Relacional

Se agregaron dos tablas que relacionan dos artistas que colaboraron al lanzamiento en el que lo hicieron (collab_with) y dos artistas que tocaron juntos en un evento (play_with). Con estas nuevas tablas se reduciría la cantidad de joins necesarios para ejecutar las queries diseñadas, es decir, si se necesitaran pares de artistas que colaboraron juntos no se requeriría ningún join, para triplas uno, para cuádruplas aún se necesitaría un join en vez de tres y para quíntuplas solo dos.

Experimentos

Una vez diseñadas las consultas que fueran a ser utilizadas para la comparación se procedió a realizar las pruebas. Dependiendo del tipo de consulta se realizaron dos corridas para aquellas con tres parámetros o tres corridas para aquellas con dos, por conjunto de entradas calculando el tiempo de ejecución de estas y finalmente se usó el promedio entre estas para realizar la comparación y sacar conclusiones.

La métrica cuantitativa utilizada es el tiempo de ejecución de la consulta medida en milisegundos.

En esta sección se muestra el resultado final de las corridas, es decir, mostrando los promedios de las corridas obtenidas para cada set de parámetros. Los resultados completos se pueden observar en el anexo.

A su vez se presenta el código necesario para correr las distintas consultas. En esta sección se mostrará el código de la primera prueba de cada conjunto y en el anexo estarán en detalle las restantes.

Q1: Top N

Q1.1: Ciudades con mayor cantidad de Eventos

Parámetros

- ***N***: *Cantidad de Ciudades a devolver*

Código

En SQL se expresa como

```
SELECT A.id, A.city, Count(*) as Events
FROM  area_dim AS A
INNER JOIN event_fact AS E ON A.id = E.id_area
WHERE A.city != 'Unknown'
GROUP BY A.id, A.city
ORDER BY COUNT(*) DESC
LIMIT N
```

En Neo4j se expresa como

```
MATCH(c:City)--(e:EventFact)
WITH c, count(e) as count
RETURN c, count
ORDER BY count DESC
LIMIT 5
```

Resultados

N	SQL	NEO4J
1	59 ms	47 ms
3	55 ms	39 ms
5	63 ms	40 ms
10	57 ms	48 ms
15	63 ms	32 ms

TABLA DE TIEMPOS 1 – CIUDADES CON MAYOR CANTIDAD DE EVENTOS

Q1.2: Artistas con más colaboraciones en Artist Credit

Parámetros

- ***N***: *Cantidad de Artistas a devolver*

Código

En SQL se expresa como

```
SELECT AC.id_artist, AR.name, count(*) AS Collaborations
FROM  artist_credit_dim AC
INNER JOIN artist_dim AR ON AR.id = AC.id_artist
WHERE artist_count > 1
GROUP BY id_artist, AR.name
ORDER BY count(*) DESC
LIMIT N
```

En Neo4j se expresa como

```
MATCH (ac:ArtistCredit)-[r:INCLUDES]-(ar:Artist)
WHERE ac.artistCount > 1
RETURN ar.id AS ArtistId, ar.name AS Artist, count(r) AS collaborations
ORDER BY count(r) DESC LIMIT N
```

Resultados

N	SQL	NEO4J
1	2896 ms	3400 ms
3	2855 ms	3349 ms
5	2737 ms	3250 ms
10	2771 ms	3101 ms
15	2814 ms	3135 ms

TABLA DE TIEMPOS 2 - ARTISTAS CON MÁS COLABORACIONES EN ARTIST CREDIT

Q2: Conjuntos

Q2.1: Artistas que participaron en Eventos juntos

Parámetros

- ***X***: *Cantidad de artistas*
- ***N***: *Cantidad de eventos en los que participaron juntos*

Q2.1.1: Pares

Pares de artistas que participaron en N o más eventos juntos

Parámetros

X	N
2	[1 , 5]

Código

En SQL se expresa como

```
SELECT E1.id_artist, E2.id_artist, COUNT(*) as EventCount
FROM  event_fact AS E1
INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
WHERE E1.id_artist < E2.id_artist
GROUP BY E1.id_artist, E2.id_artist
HAVING COUNT(*) > N
ORDER BY COUNT(*) DESC
```

En Neo4j se expresa como

```
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, COLLECT(e) AS events
WHERE SIZE(events) > N
RETURN a1.id, a2.id
```

Resultados

N	SQL	NEO4J
1	399 ms	1050 ms
2	269 ms	785 ms
3	280 ms	831 ms
4	281 ms	813 ms
5	271 ms	797 ms

TABLA DE TIEMPOS 3 – PARES DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS

Q2.1.2: Triplas

Triplas de artistas que participaron en N o más eventos juntos

Parámetros

X	N
3	[1 , 5]

Resultados

N	SQL	NEO4J
1	9172 ms	34.970 ms
2	5552 ms	3.736 ms
3	5466 ms	2.498 ms
4	5416 ms	2.185 ms
5	5477 ms	2.216 ms

TABLA DE TIEMPOS 4 - TRIPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS

Q2.1.3: Cuádruplas

Cuádruplas de artistas que participaron en N o más eventos juntos

Parámetros

X	N
4	[1 , 5]

Resultados

N	SQL	NEO4J
1	323.000 ms	Sin Resultados
2	263.666 ms	22.215 ms
3	267.666 ms	17.775 ms
4	269.000 ms	17.022 ms
5	269.000 ms	17.948 ms

TABLA DE TIEMPOS 5 - CUÁDRUPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS

Q2.1.4: Quíntuplas

Quíntuplas de artistas que participaron en N o más eventos juntos

Parámetros

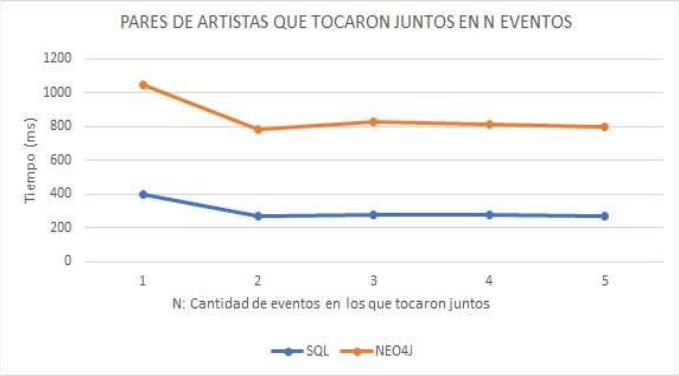
X	N
5	[1 , 5]

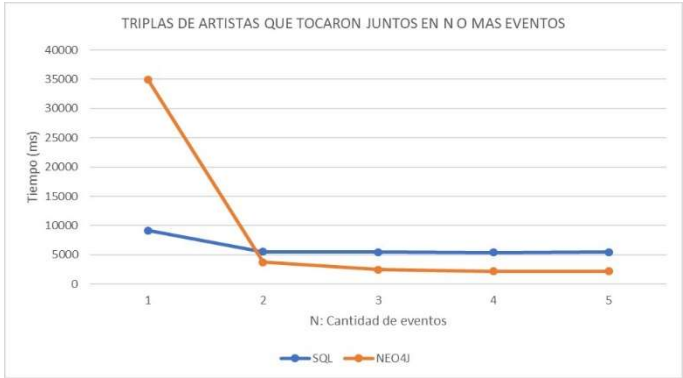
Resultados

N	SQL	NEO4J
1	Sin Resultados	Sin Resultados
2	Sin Resultados	Sin Resultados
3	Sin Resultados	Sin Resultados
4	2.380.000 ms	Sin Resultados
5	2.740.000 ms	Sin Resultados

TABLA DE TIEMPOS 6 - QUÍNTUPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS

Observaciones





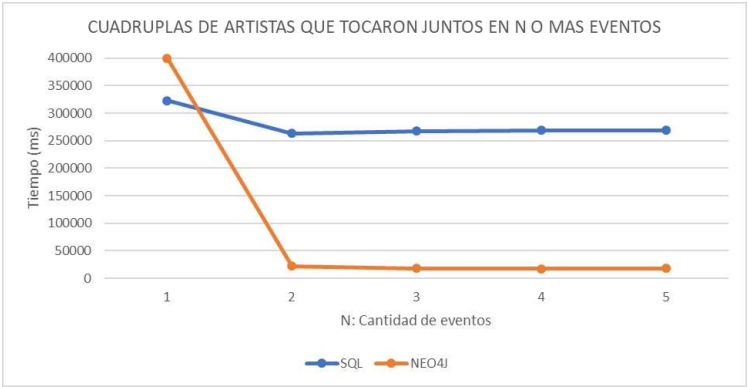


GRÁFICO 1 - RESULTADOS DE CONSULTAS Q2.1

Como podemos observar, para N = 1 en todas las corridas del conjunto se puede ver una ventaja de SQL sobre Neo4j. En las corridas para pares de artistas (Q2.1.1) se observa una mejor performance de SQL mientras que para las consultas de triplas (Q2.1.2) y cuádruplas (Q2.1.3), las corridas con un N mayor a 1 los tiempos de la base de grafos son mejores que su competencia relacional. Tanto Neo4j como SQL no terminaron de ejecutar las consultas de quíntuplas de artistas (Q2.1.4).

Q2.2: Artistas que participaron en Eventos juntos y no colaboraron en ningún Release

Parámetros

- ***X***: *Cantidad de artistas*
- ***N***: *Cantidad de eventos en los que participaron juntos*

Q2.2.1: Pares

Pares de artistas que participaron en N o más eventos juntos y no colaboraron en ningún reléase

Parámetros

X	N
2	[1 , 5]

Código

En SQL se expresa como

```
(
   SELECT E1.id_artist, E2.id_artist
   FROM event_fact AS E1
   INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
   WHERE E1.id_artist < E2.id_artist
   GROUP BY E1.id_artist, E2.id_artist
   HAVING COUNT(*) > N
   ORDER BY  COUNT(*) DESC
)
EXCEPT
(
   SELECT AC1.id_artist, AC2.id_artist
   FROM artist_credit_dim AS AC1
   INNER JOIN artist_credit_dim AS AC2 ON AC1.id = AC2.id
   INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
   WHERE AC1.id_artist < AC2.id_artist
)
```

En Neo4j se expresa como

```
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, COLLECT(e) AS events
WHERE SIZE(events) > N
WITH a1, a2, events
OPTIONAL MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit)-[:INCLUDES]->(a2)
WITH a1, a2, ac, events
OPTIONAL MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, events, collect(r) AS releases
WHERE SIZE(releases) = 0
RETURN a1.id, a2.id
```

Resultados

N	SQL	NEO4J
1	4.020 ms	3.928 ms
2	4.026 ms	1.027 ms
3	3.901 ms	860 ms
4	3.865 ms	809 ms
5	3.929 ms	792 ms

TABLA DE TIEMPOS 7 – PARES DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS Y NO COLABORARON EN NINGÚN RELEASE

Q.2.2.2: Triplas

Triplas de artistas que participaron en N o más eventos juntos y no colaboraron en ningún release

Parámetros

X	N
3	[1 , 5]

Resultados

N	SQL	NEO4J
1	15.820 ms	75.333 ms
2	10.740 ms	5.474 ms
3	10.596 ms	3.639 ms
4	10.687 ms	4.382 ms
5	10.910 ms	3.342 ms

TABLA DE TIEMPOS 8 - TRIPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS Y NO COLABORARON EN NINGÚN RELEASE

Q2.2.3: Cuádruplas

Cuádruplas de artistas que participaron en N o más eventos juntos y no colaboraron en ningún release

Parámetros

X	N
4	[1 , 5]

Resultados

N	SQL	NEO4J
1	599.333 ms	Sin Resultados
2	342.666 ms	32.284 ms
3	348.666 ms	27.439 ms
4	350.333 ms	25.613 ms
5	341.666 ms	26.178 ms

TABLA DE TIEMPOS 9 - CUÁDRUPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS Y NO COLABORARON EN NINGÚN RELEASE

Q2.2.4: Quíntuplas

Quíntuplas de artistas que participaron en N o más eventos juntos y no colaboraron en ningún reléase

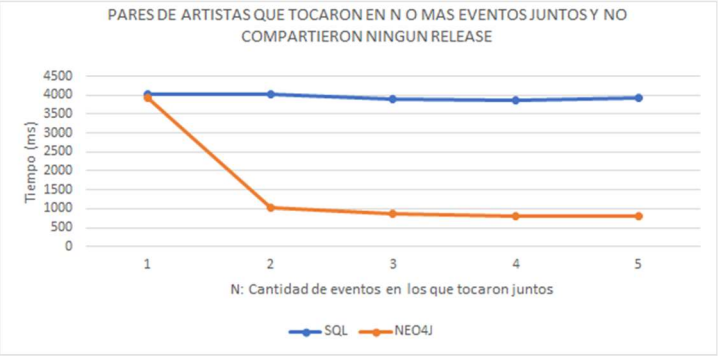
Parámetros

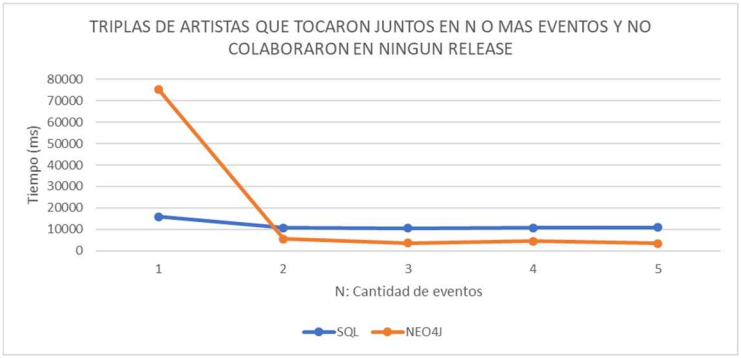
X	N
5	[1 , 5]

Resultados

Estas pruebas corrieron por más de doce horas sin resultados.

Observaciones





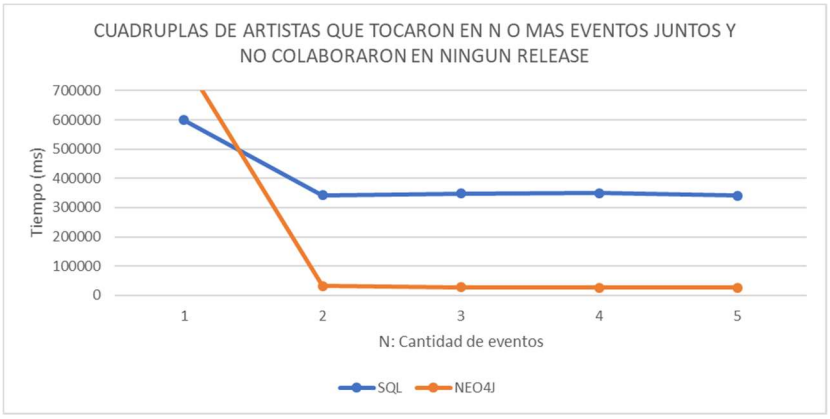


GRÁFICO 2 - RESULTADOS DE CONSULTAS Q2.2

Se puede observar un comportamiento similar a el conjunto Q2.1 donde las consultas devuelven un tiempo menor en las corridas de SQL cuando N es igual a 1, mientras que en este caso tanto para pares (Q2.2.1), triplas (Q2.2.2) como para cuádruplas (Q2.2.3) en las corridas con N mayor a uno, Neo4j supera en performance a PostgreSQL.

Q.2.3: Artistas que participaron en Eventos y Releases juntos

Parámetros

- $\boldsymbol{X}$: *Cantidad de artistas*
- $\boldsymbol{N}$: *Cantidad de eventos en los que participaron juntos*
- $\boldsymbol{M}$: *Cantidad de releases en los que colaboraron*

Q.2.3.1: Pares

Pares de artistas que participaron en N o más eventos juntos y colaboraron en M o más releases juntos

Parámetros

X	N	M
2	[1,3]	[1,5]

Código

En SQL se expresa como

```
(
    SELECT E1.id_artist, E2.id_artist
    FROM event_fact AS E1
    INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
    WHERE E1.id_artist < E2.id_artist
    GROUP BY E1.id_artist, E2.id_artist
    HAVING COUNT(*) > N
    ORDER BY COUNT(*) DESC
 )
    INTERSECT
 (
    SELECT AC1.id_artist, AC2.id_artist
    FROM artist_credit_dim AS AC1
    INNER JOIN artist_credit_dim AS AC2 ON AC1.id = AC2.id
    INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
    WHERE AC1.id_artist < AC2.id_artist
    GROUP BY AC1.id_artist, AC2.id_artist
    HAVING COUNT(*) > M
    ORDER BY COUNT(*) DESC
 )
```

En Neo4j se expresa como

```
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, COLLECT(e) AS events
WHERE SIZE(events) > N
WITH a1, a2, events
MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit)-[:INCLUDES]->(a2)
WITH a1, a2, ac, events
MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, events, collect(r) AS releases
WHERE SIZE(releases) > M
RETURN a1.id, a2.id
```

Resultados

Q2.3.1.1: $N = 1$

M	SQL	NEO4J
1	6.548 ms	2.728 ms
2	3.850 ms	2.868 ms
3	3.792 ms	2.672 ms
4	4.001 ms	2.595 ms
5	3.927 ms	2.694 ms

TABLA DE TIEMPOS 10 – PARES DE ARTISTAS QUE PARTICIPARON EN UNO O MÁS EVENTOS Y M O MÁS RELEASES JUNTOS

Q2.3.1.3: $N = 2$

M	SQL	NEO4J
1	3.877 ms	995 ms
2	3.597 ms	1.006 ms
3	3.643 ms	953 ms
4	3.717 ms	1.000 ms
5	3.661 ms	906 ms

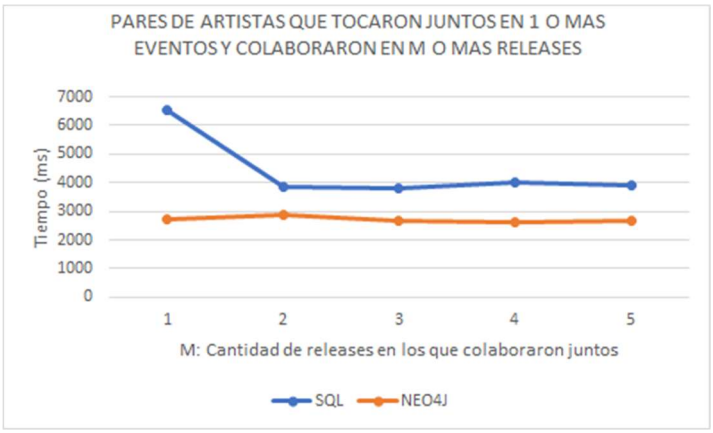
TABLA DE TIEMPOS 11 - PARES DE ARTISTAS QUE PARTICIPARON EN DOS O MÁS EVENTOS Y M O MÁS RELEASES JUNTOS

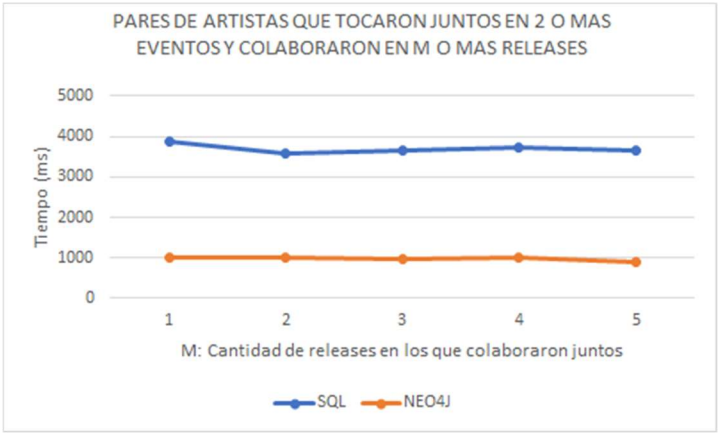
Q2.3.1.3: $N = 3$

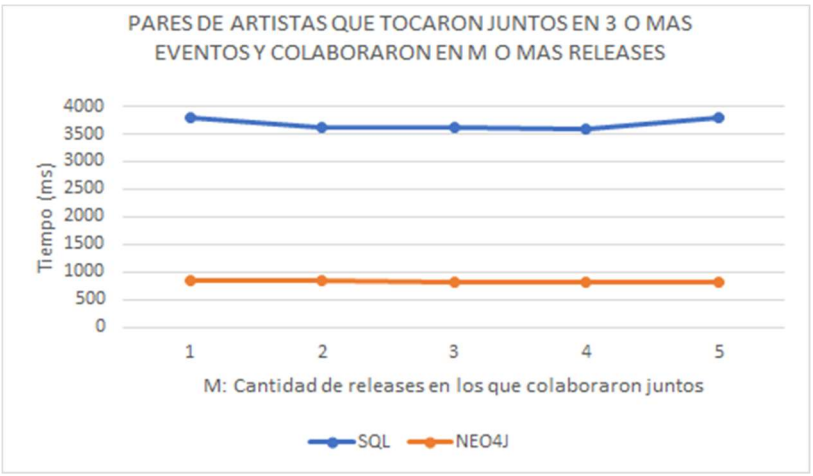
M	SQL	NEO4J
1	3.795 ms	849 ms
2	3.627 ms	835 ms
3	3.622 ms	820 ms
4	3.597 ms	814 ms
5	3.779 ms	827 ms

TABLA DE TIEMPOS 12 - PARES DE ARTISTAS QUE PARTICIPARON EN TRES O MÁS EVENTOS Y M O MÁS RELEASES JUNTOS

Observaciones







GRÁFICOS 3 - RESULTADOS DE CONSULTAS Q2.3.1

En este conjunto de pruebas podemos observar que Neo4j supera en performance a PostgreSQL en todas las combinaciones de parámetros posibles para este tipo consulta.

Q2.3.2: Triplas

Triplas de artistas que participaron en N o más eventos juntos y colaboraron en M o más releases juntos

Parámetros

X	N	M
3	[1 , 3]	[1 , 5]

Resultados

Q2.3.2.1: $N = 1$

M	SQL	NEO4J
1	17.233 ms	79.000 ms
2	17.261 ms	78.500 ms
3	16.691 ms	79.000 ms
4	17.024 ms	78.000 ms
5	16.454 ms	79.500 ms

TABLA DE TIEMPOS 13 - TRIPLAS DE ARTISTAS QUE PARTICIPARON EN UNO O MÁS EVENTOS Y M O MÁS RELEASES JUNTOS

Q2.3.2.2: $N = 2$

M	SQL	NEO4J
1	11.335 ms	5.079 ms
2	11.125 ms	5.069 ms
3	10.982 ms	4.941 ms
4	11.004 ms	4.960 ms
5	10.965 ms	4.384 ms

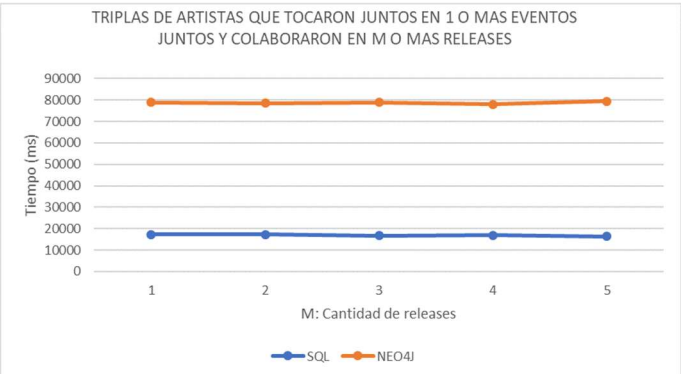
TABLA DE TIEMPOS 14 - TRIPLAS DE ARTISTAS QUE PARTICIPARON EN DOS O MÁS EVENTOS Y M O MÁS RELEASES JUNTOS

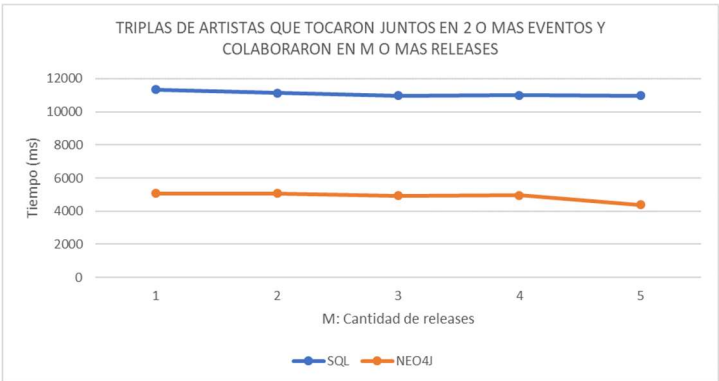
Q2.3.2.3: $N = 3$

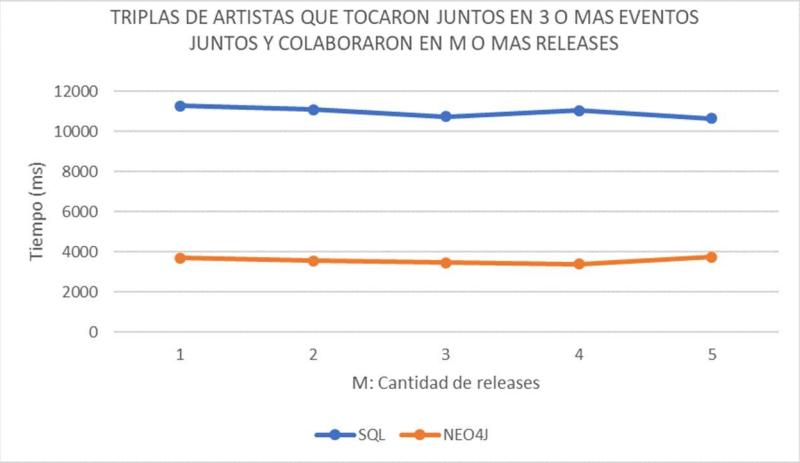
M	SQL	NEO4J
1	11.281 ms	3.685 ms
2	11.088 ms	3.544 ms
3	10.743 ms	3.460 ms
4	11.062 ms	3.398 ms
5	10.655 ms	3.739 ms

TABLA DE TIEMPOS 15 - TRIPLAS DE ARTISTAS QUE PARTICIPARON EN TRES O MÁS EVENTOS Y M O MÁS RELEASES JUNTOS

Observaciones







GRÁFICOS 4 - RESULTADOS DE CONSULTAS Q2.3.2

En este conjunto de pruebas podemos ver que para N igual a 1 (Q2.3.2.1) SQL supera en performance a Neo4j para todos los valores de M posibles. Sin embargo, para N mayor a 1 y todos los valores de M posibles, Neo4j muestra tiempos menores de ejecución en comparación a PostgreSQL.

Q2.3.3: Cuádruplas

Cuádruplas de artistas que participaron en N o más eventos juntos y colaboraron en M o más releases juntos

Parámetros

X	N	M
4	[1 , 3]	[1 , 5]

Resultados

Q2.3.3.1: $N = 1$

M	SQL	NEO4J
1	401.500 ms	Sin Resultados
2	388.500 ms	Sin Resultados
3	388.000 ms	Sin Resultados
4	516.000 ms	Sin Resultados
5	636.000 ms	Sin Resultados

TABLA DE TIEMPOS 16 - CUÁDRUPLAS DE ARTISTAS QUE PARTICIPARON EN UNO O MÁS EVENTOS Y M O MÁS RELEASES JUNTOS

Q2.3.2.3: $N = 2$

M	SQL	NEO4J
1	255.000 ms	26.885 ms
2	247.000 ms	27.179 ms
3	247.500 ms	26.654 ms
4	245.000 ms	28.967 ms
5	249.500 ms	28.862 ms

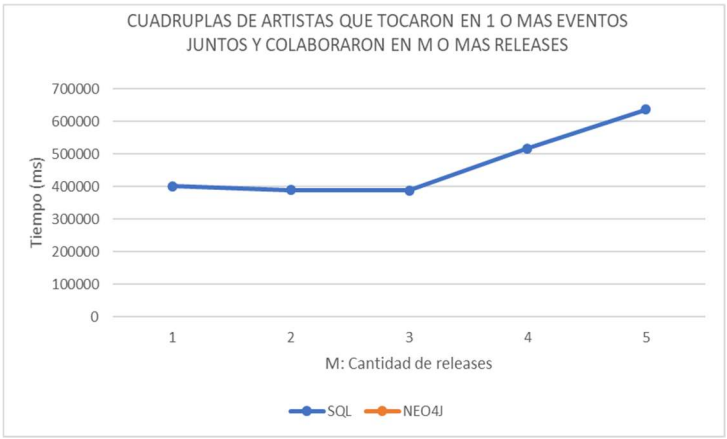
TABLA DE TIEMPOS 17 - CUÁDRUPLAS DE ARTISTAS QUE PARTICIPARON EN DOS O MÁS EVENTOS Y M O MÁS RELEASES JUNTOS

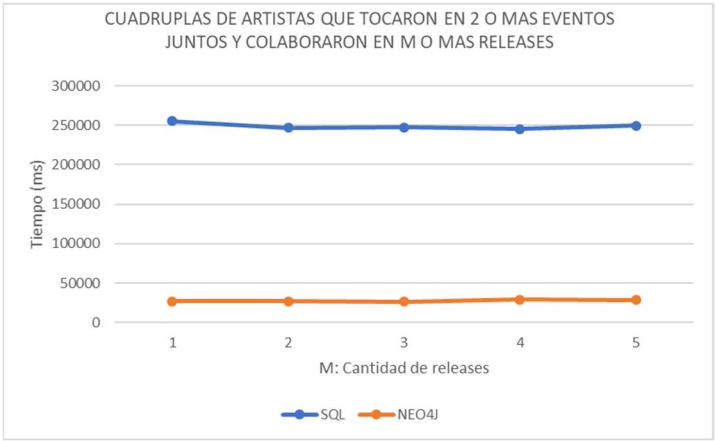
Q2.3.2.3: $N = 3$

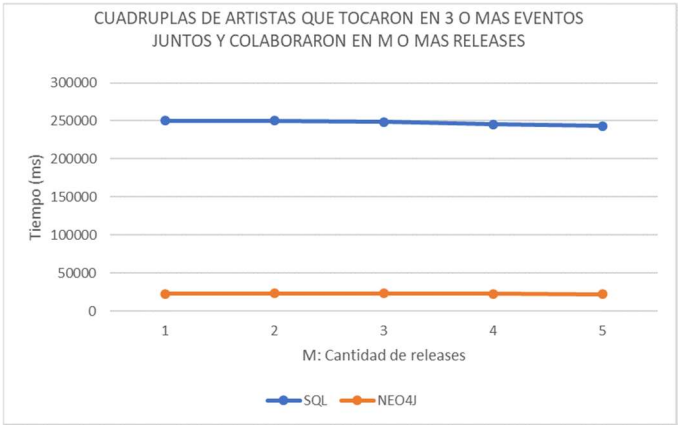
M	SQL	NEO4J
1	250.000 ms	22.881 ms
2	250.000 ms	23.188 ms
3	248.500 ms	23.190 ms
4	245.500 ms	23.061 ms
5	243.500 ms	22.811 ms

TABLA DE TIEMPOS 18 - CUÁDRUPLAS DE ARTISTAS QUE PARTICIPARON EN TRES O MÁS EVENTOS Y M O MÁS RELEASES JUNTOS

Observaciones







GRÁFICOS 5 - RESULTADOS DE CONSULTAS Q2.3.3

Para la N igual a 1 (Q2.3.3.1) Neo4j no devolvió resultados para ningún valor de M mientras PostgrSQL terminó la ejecución de todas sus pruebas bajo estos parámetros, en tiempos menores a los 10 minutos. Por otro lado para los casos de N mayor a 1 (Q2.3.3.2 y Q2.3.3.3) superó en performance a SQL para todos los valores posibles de M.

Q2.3.4: Quíntuplas

Quíntuplas de artistas que participaron en N o más eventos juntos y colaboraron en M o más releases juntos

Parámetros

X	N	M
5	[1 , 3]	[1 , 5]

Resultados

Tanto en SQL como en NEO4J las pruebas con este set de parámetros se demoraron más de ocho horas sin resultados.

Q3: Recomendaciones

Las consultas se realizaron tomando como punto de partido el artista 874111, dado que comprobó ser aquel con más ramificaciones de Artist Credit, lo cual es ideal para nuestras mediciones.

Parámetros

- $\boldsymbol{N}$: $profundidad$

Sin Optimización

En SQL se expresa como

```
 CREATE FUNCTION search_collabs(id_original_artist int, N int) RETURNS
TABLE(id_artist int)
 AS $$
 BEGIN
  IF N = 1 THEN
    RETURN QUERY
    SELECT DISTINCT(AC2.id_artist)
    FROM artist_credit_dim   AS  AC
    INNER JOIN artist_credit_dim   AS  AC2 ON AC.id = AC2.id
    WHERE AC.id_artist != AC2.id_artist AND AC.id_artist = id_original_artist
         AND AC.artist_count > 1;
  ElSE
    RETURN QUERY
    SELECT DISTINCT(AC4.id_artist)
    FROM artist_credit_dim AS AC3
    INNER JOIN artist_credit_dim AS AC4 ON AC3.id = AC4.id
    WHERE AC4.id_artist != id_original_artist AND AC3.id_artist IN (
         SELECT * FROM search_collabs(id_original_artist, N-1)
     );
    END IF;
 END;
 $$
 LANGUAGE plpgsql;
```

En Neo4j se expresa como

```
 WITH <_PARAM_> AS givenArtist
 MATCH (myArtist:Artist)-[:INCLUDES]-(:ArtistCredit)-[:INCLUDES*1..2N]-(otherArtists:Artist)
 WHERE (myArtist.id = givenArtist) AND NOT (otherArtists.id = givenArtist)
 RETURN DISTINCT otherArtists
```

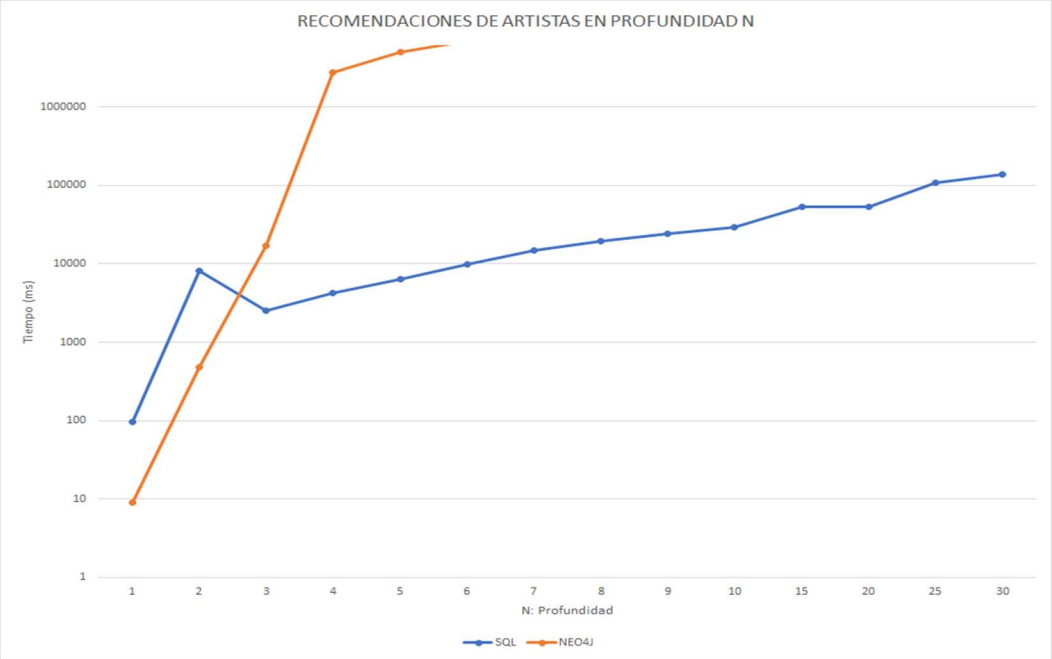


GRÁFICO 6- RECOMENDACIONES DE ARTISTAS SIN OPTIMIZACIÓN

En Neo4J el proceso se estancó al alcanzar profundidad 5 ($N = 5$), demorando mas de 8 horas sin obtener resultados. Se le dio un valor ilustrativo a dicha corrida para que pueda ser visualizado en el gráfico el crecimiento exponencial del tiempo de ejecución.

Con Optimización

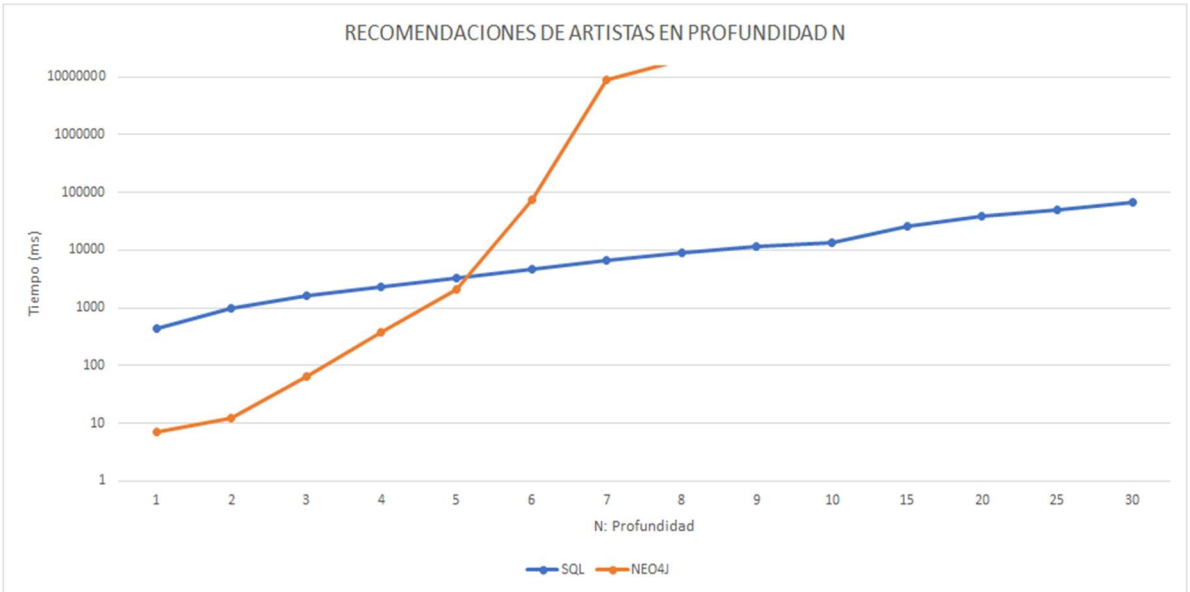


GRÁFICO 7 - RECOMENDACIONES DE ARTISTAS CON OPTIMIZACIÓN

Corriendo las mismas pruebas utilizando ahora las optimización, se logró mejorar la profundidad máxima alcanzada con Neo4j, siendo que previamente solo se podía llegar hasta nivel 5 y con estas nuevas pruebas se obtuvieron resultados hasta nivel 7 inclusive. Sin embargo, los tiempos medidos fueron considerablemente mayores a los obtenidos por PostgreSQL.

Problemas Encontrados

Durante el transcurso de las pruebas se vio que Neo4j no estaba dando buenos resultados en ninguna de las consultas planteadas. Tras revisar las queries y consultar con los tutores del proyecto, se encontró que era posible filtrar aún más los resultados parciales en las consultas de Neo4j, sumándole complejidad al código, pero ganando performance a la hora de trabajar con los datos en memoria. A continuación, se podrá ver una comparación de la primera versión donde se utilizó un único patrón para filtrar los resultados, contra la versión más reciente donde se usan patrones parciales más reducidos.

Comparación

Artistas que participaron en Eventos juntos

Triplas

Se puede observar una mejora en los tiempos de ejecución en la mayor parte de las pruebas para estos parámetros.

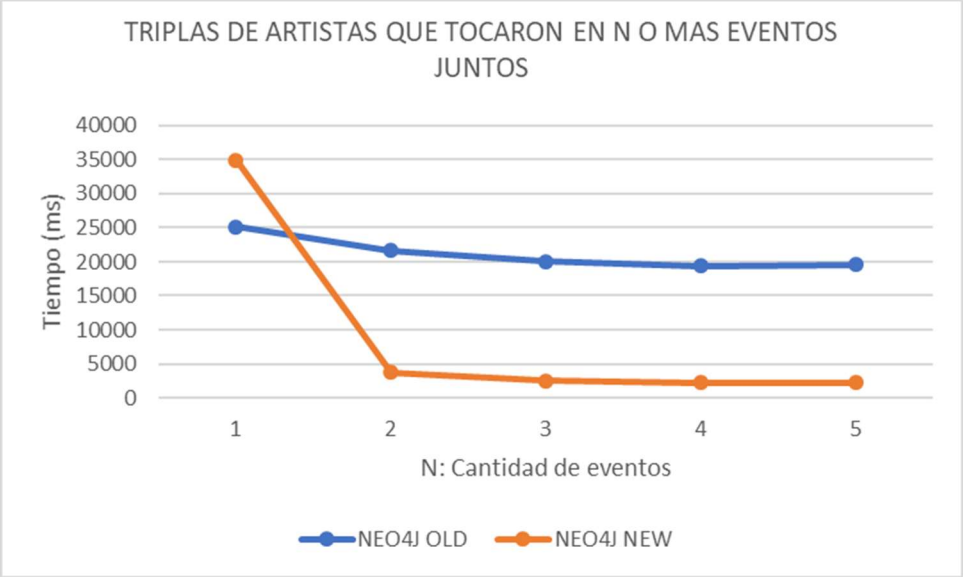


GRÁFICO 8 – COMPARACIÓN DE NEO4J PARA TRIPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS

Cuádruplas

En el caso de la cuádruplas, la mejora es clara ya que previamente Neo4j no lograba terminar de ejecutar las consultas a causa de falta de memoria. Filtrando los resultados parciales, esto no sucede ya que la cantidad de nodos a procesar en cada paso es menor, por lo que el uso de memoria se ve reducido.

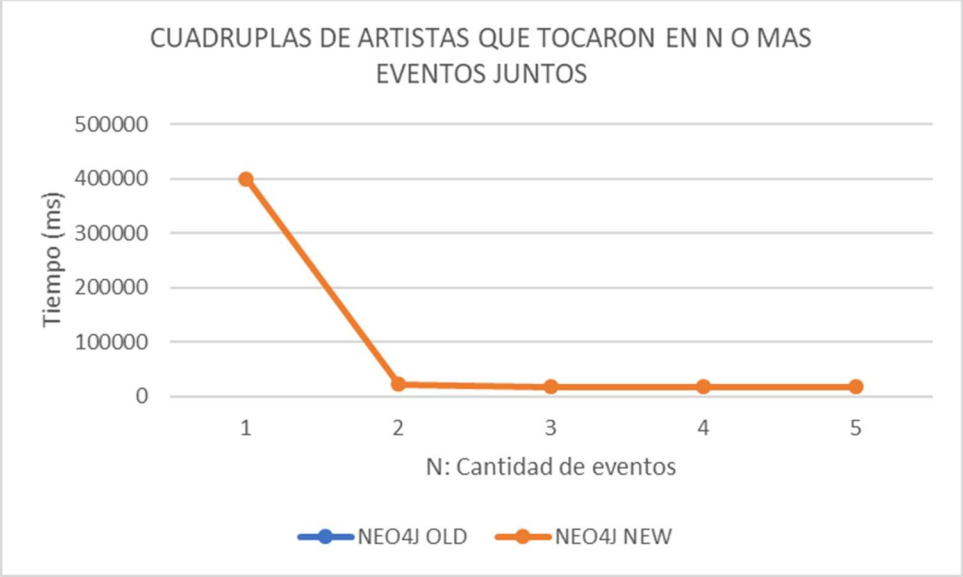


GRÁFICO 9 - COMPARACIÓN DE NEO4J PARA CUÁDRUPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS

Artistas que participaron en Eventos juntos y no colaboraron en ningún Release

Triplas

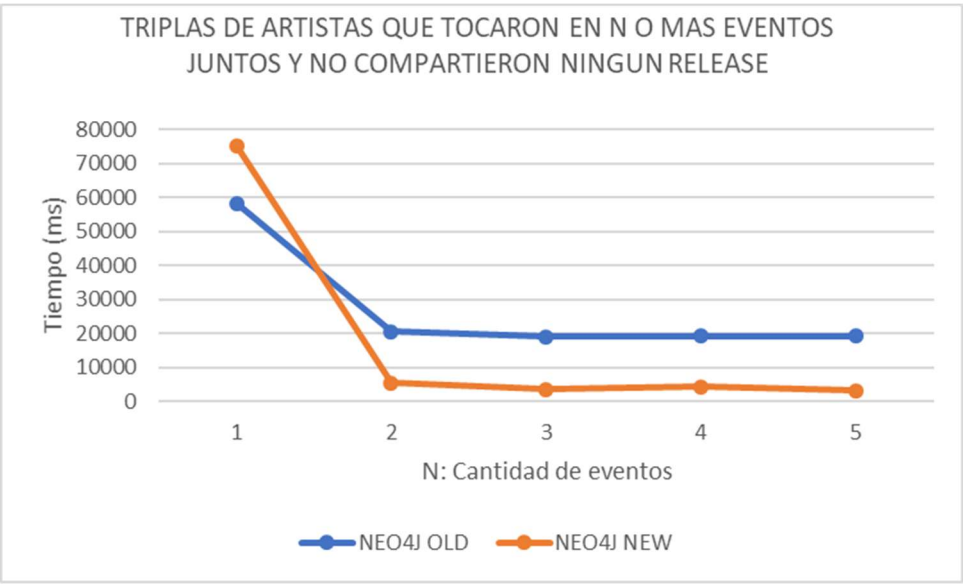


GRÁFICO 10 – COMPARACIÓN DE NEO4J PARA TRIPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS Y NO COLABORARON EN NINGÚN RELEASE

Cuádruplas

En este caso gracias a el cambio realizado pudimos obtener resultados para este set de parámetros y así compararlos con SQL.

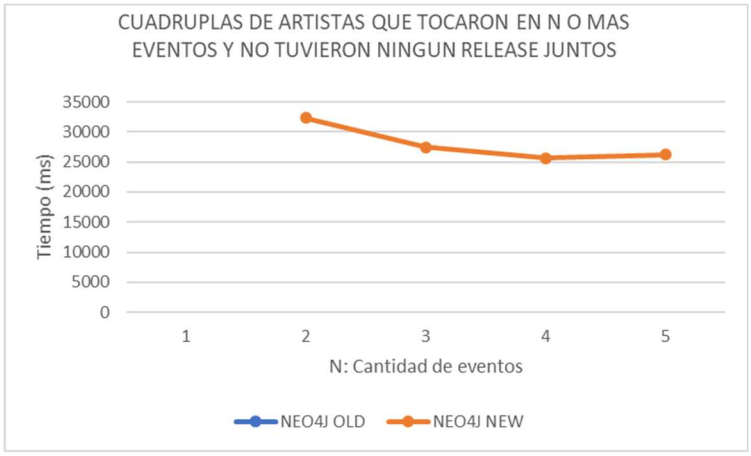
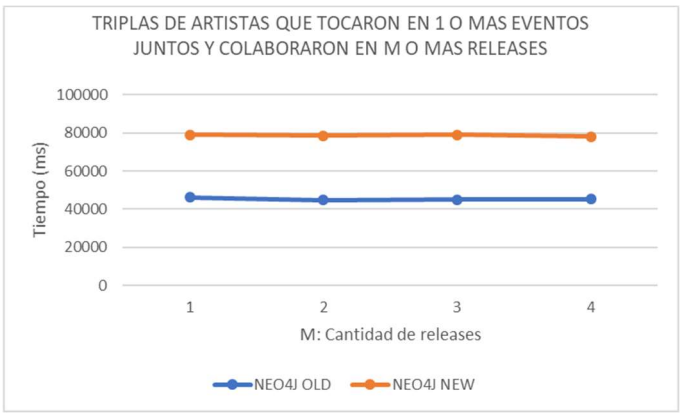
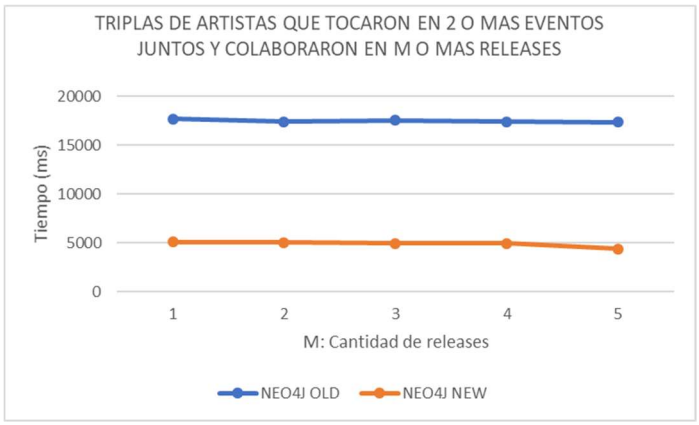


GRÁFICO 11 - COMPARACIÓN DE NEO4J PARA CUÁDRUPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS Y NO COLABORARON EN NINGÚN RELEASE

Artistas que participaron en Eventos y Releases juntos

Triplas





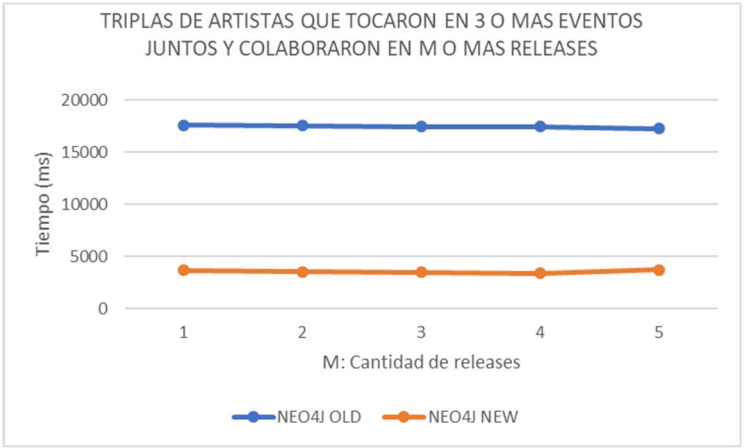
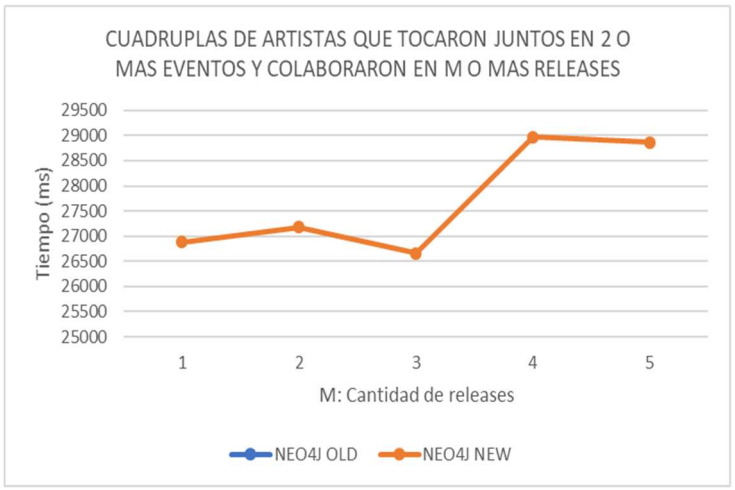


GRÁFICO 12 - COMPARACIÓN DE NEO4J PARA TRIPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS Y COLABORARON EN M O MÁS RELEASES

Cuádruplas



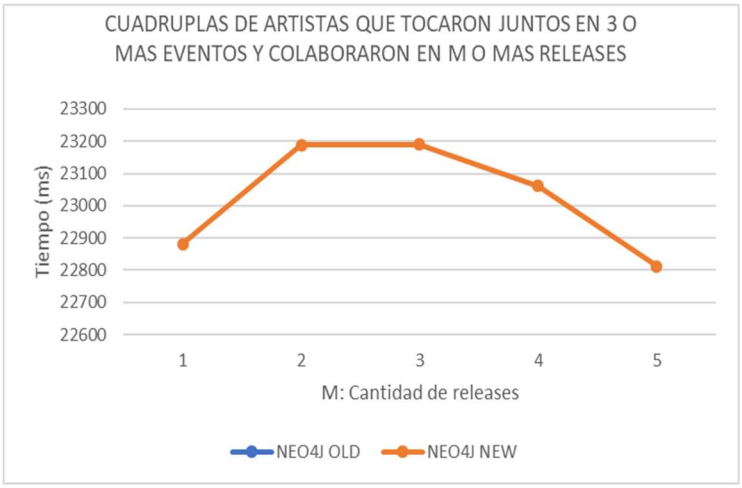


GRÁFICO 13 - COMPARACIÓN DE NEO4J PARA CUADRUPLAS DE ARTISTAS QUE PARTICIPARON EN EVENTOS JUNTOS Y COLABORARON EN M O MÁS RELEASES

Como podemos observar gracias a los cambios realizados para este set de pruebas logramos obtener resultados y así compararlos con aquellos provistos por SQL.

ANALISIS DE RESULTADOS

Evaluando los resultados se puede ver como la performance de Neo4J disminuye considerablemente cuando se trabaja con conjuntos grandes y restricciones de N bajas.

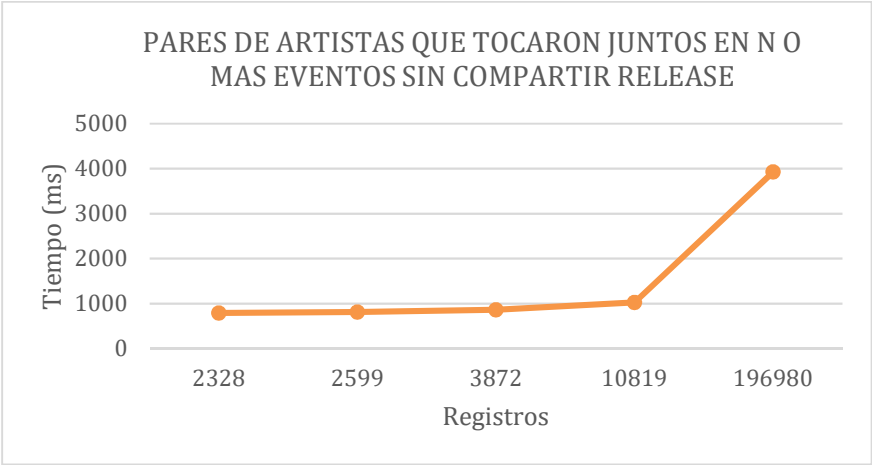


GRÁFICO 14 - TIEMPO EN FUNCIÓN DE REGISTROS PARA PARES[2]

[2] En este set de pruebas con $N = 1$ el tiempo de ejecución supero las 8 horas sin obtener resultados. El tiempo representado en el gráfico es ilustrativo para demostrar el salto en performance. Los datos de cantidad de registros para esos casos fueron sacados de los obtenidos en PostgreSQL.

A mayor cantidad de registros el tiempo suele aumentar, dando saltos considerables en el orden de los minutos o incluso horas para los $N = 1$, siendo este el menor nivel de restricción que presentamos en nuestras pruebas, lo cual causa que la cantidad de registros involucrados en el procesamiento de la consulta se vea maximizado.

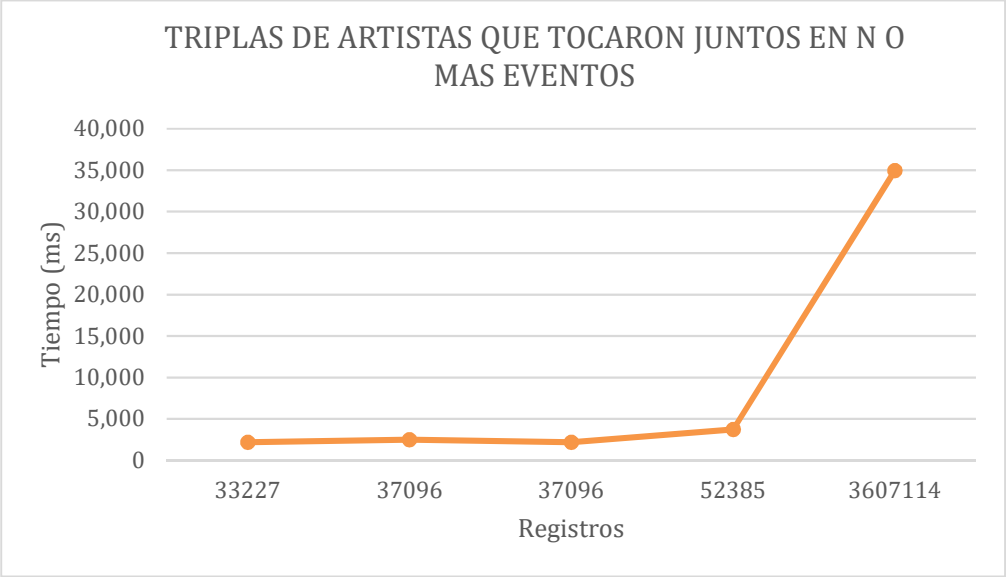


GRÁFICO 15 - TIEMPO EN FUNCIÓN DE REGISTROS PARA TRIPLAS

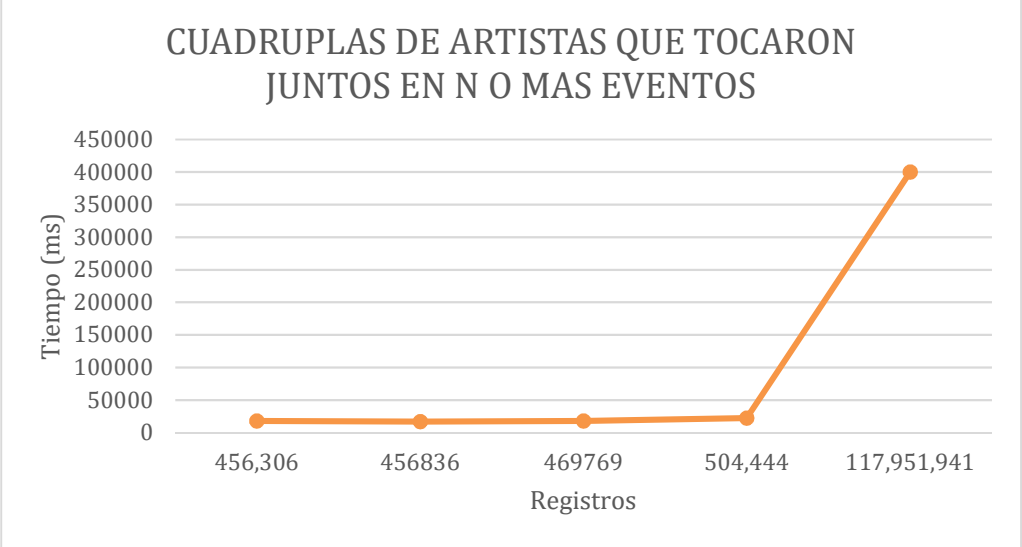


GRÁFICO 16 – TIEMPO EN FUNCIÓN DE REGISTROS PARA CUÁDRUPLAS

Luego de investigar en mayor profundidad la manera en la que esta base [11] de grafos maneja los datos en disco se pudo observar que Neo4j almacena la información en forma de listas enlazadas de elementos de tamaño fijo, dependiendo del tipo de información que guarden. Las propiedades se guardan en listas de registros de este tipo que consisten en un par *clave-valor* y un puntero a la siguiente propiedad. Las relaciones referencian al nodo-inicio y nodo-fin de la misma y, a su vez, a la relación anterior y la siguiente para cada uno de estos respectivamente. Tanto los nodos como las relaciones tienen una referencia a la primera propiedad de su lista si así la tuvieran. Todo eso le otorga un peso extra a cada nodo, por lo que traer grandes volúmenes de los mismos a memoria para procesar una consulta se traduce en una degradación de la performance que se pone en evidencia cuando se compara con otra bases que usan almacenamientos más simples o que manejen estructuras preparadas para traer grandes volúmenes. Fue teniendo esto en cuenta que se llevaron a cabo la modificaciones en las consultas de Conjuntos en Neo4j, donde se buscó la forma de filtrar los nodos en cada paso posible en lugar de obtenerlos todos usando un único patrón padre. Fue mediante esa reducción de nodos en memoria que se lograron obtener resultados de la forma original resultaban imposibles de obtener debido a la cantidad de memoria que exigían las consultas, y fue gracias a eso que se logró superar los tiempos de PostgreSQL en la mayoría de los casos. En el caso de las recomendaciones, no había cambio posible para hacerle a las consultas y aunque el tipo de optimizaciones planteado para las mismas mejoró considerablemente los tiempos, no fue suficiente para alcanzar a los de PosgreSQL. Sin embargo, si observamos la descomposición de los tiempos en las tablas de resultados para las recomendaciones, podemos ver que existen dos tiempos; el tiempo en el que se encuentran disponibles los datos, y el tiempo que lleva consumirlos. Este último es el que demora mayor cantidad de tiempo, mientras que el tiempo en obtenerlos es varios ordenes menor.

Profundidad	Disponible	Consumido
7	42ms	10555005ms
	245ms	9842516ms
	3ms	10113268ms

TABLA DE TIEMPOS 19 - DESCOMPOSICIÓN DE TIEMPO DE RECOMENDACIONES EN NEO4J

Es posible obtener mejores resultados teniendo más entidades o casos de uso que nos permitan definir patrones más específicos y consultas que requiera de mayor navegación de relaciones y menos tamaño de resultados en lugar de una devolución masiva de datos. Un ejemplo de eso sería la existencia de Usuarios, los cuales podrían tener tanto Artistas como Géneros o hasta Eventos favoritos y a su vez estos Usuarios tengan relaciones con otros Usuarios. De esta forma, se podría utilizar de una mejor manera los aspectos positivos de Neo4j, permitiendo describir patrones más específicos que lleven a una búsqueda más inteligente, reduciendo así la cantidad de nodos a procesar, de forma de disminuir el tiempo que se tarde en consumirlos. Casos como estos son los de las redes sociales y recomendaciones basadas en machine learning, donde gracias este tipo de datos y relaciones representadas en grafos se logra recorrer la información de manera eficiente, usando esas restricciones para crear caminos inteligentes que devuelvan un set acotado de información en base a una gran cantidad de datos y relaciones como contexto.

CONCLUSIONES

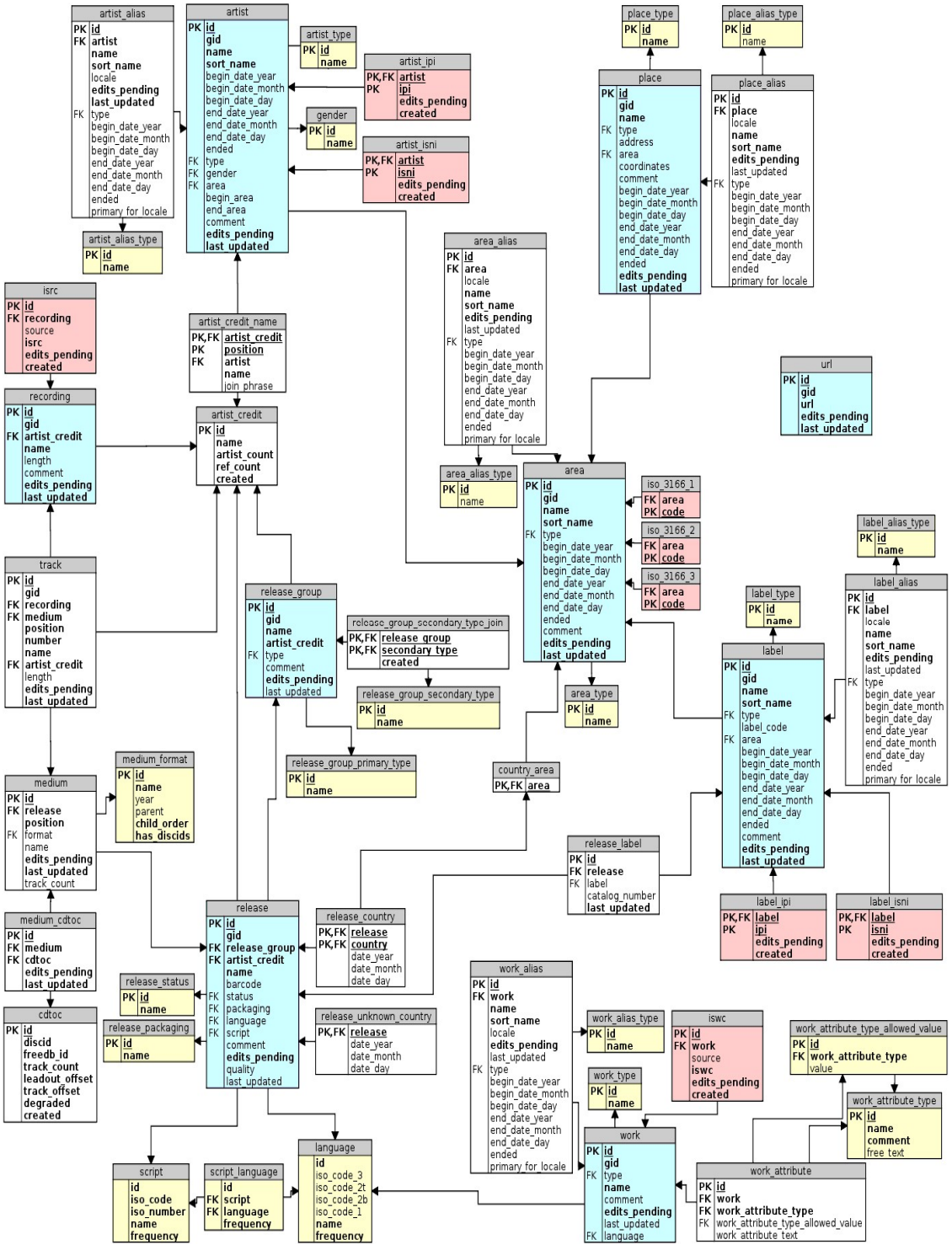
Se realizó un estudio comparativo entre una base de datos relacional (PostgreSQL) y una de grafos (Neo4j) tomando como valor cuantitativo el tiempo de ejecución de las consultas. Para esto se diseñó un data warehouse para cada modelo usando datos reales obtenidos de la enciclopedia musical Musicbrainz. Teniendo en consideración los tres grupos de consultas utilizadas para realizar este estudio se notó un comportamiento similar en cuanto a performance. Las consultas de ranking, referidas como *top N,* no otorgaron diversidad de resultados, mientras que en las consultas de *conjuntos* y *recomendaciones* se destacaron algunos casos. Con $N = 1$, las consultas de conjuntos favorecieron considerablemente a PostgreSQL, el cual superó los tiempos de Neo4j por órdenes de magnitud, existiendo casos en donde la ejecución del último se extendió por horas sin resultados. Sin embargo, en el resto de las pruebas donde se manejaban valores mayores de N, se observó el caso contrario, colocando a Neo4j por sobre PostgreSQL. Finalmente, en las consultas de recomendaciones la base de datos relacional mostró tiempos de ejecución menores que aquellos de la base de grafos. Todo esto deja en evidencia la estabilidad de PostreSQL en cuanto a performance, mientras que en Neo4j la misma se ve fuertemente influencia por las restricciones que se definan en la consulta. Ambas bases de datos poseen sus ventajas y desventajas, por lo que la elección de una u otra dependerá de las características del proyecto.

BIBLIOGRAFIA

1. Data warehouse: Diapositivas cursada “Análisis de datos de la Web Semántica” (ITBA)
2. Data warehouse: https://en.wikipedia.org/wiki/Data_warehouse
3. Bases de datos de grafos: https://whatis.techtarget.com/definition/graph-database
4. NEO4J: https://neo4j.com/developer/graph-database/#_what_is_neo4j
5. Graph databases benchmarking on the Italian Bussiness Register: https://www.researchgate.net/publication/326262082_Graph_Databases_Benchmarking_on_the_Italian_Business_Register/stats
6. Bases de datos relacionales: https://computer.howstuffworks.com/question599.htm
7. Modelo Relacional: https://www.techopedia.com/definition/24559/relational-model-database
8. Ejemplo del Modelo Relacional: https://www.guru99.com/relational-data-model-dbms.html
9. Cuadro comparativo de BD Relacionales: https://db-engines.com/en/system/Microsoft+SQL+Server%3BOracle%3BPostgreSQL
10. Musicbrainz : https://musicbrainz.org/
11. NEO4J Data on disk: https://neo4j.com/developer/kb/understanding-data-on-disk/

ANEXO

Esquema Parcial de Musicbrainz



CONSULTAS

CONJUNTOS

Pares de Artistas que tocaron en N o más Eventos juntos

PostgreSQL

```
SELECT E1.id_artist, E2.id_artist
FROM event_fact AS E1
INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
WHERE E1.id_artist < E2.id_artist
GROUP BY E1.id_artist, E2.id_artist
HAVING COUNT(*) >= 1 -- N
ORDER BY COUNT(*) DESC
```

Neo4J

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, COLLECT(e) AS events
WHERE SIZE(events) >= n
RETURN a1.id, a2.id;
```

Triplas de Artistas que tocaron en N o más Eventos juntos

PostgreSQL

```
SELECT E1.id_artist, E2.id_artist, E3.id_artist
FROM event_fact AS E1
INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
INNER JOIN event_fact AS E3 ON E2.id_event = E3.id_event
WHERE E1.id_artist < E2.id_artist AND E2.id_artist < E3.id_artist
GROUP BY E1.id_artist, E2.id_artist, E3.id_artist
HAVING COUNT(*) >= 1 -- N
ORDER BY COUNT(*) DESC
```

Neo4J

Primera Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact), (a2:Artist)<-[:PERFORMED_BY]-(e),
      (a3:Artist)<-[:PERFORMED_BY]-(e)
WHERE a1.id < a2.id < a3.id
WITH a1, a2, a3, COLLECT(e) AS events
WHERE SIZE(events) >= n
RETURN a1.id, a2.id, a3.id;
```

Segunda Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e0:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, n, COLLECT(e0) AS events0
WHERE SIZE(events0) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e1:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3:Artist)<-[:PERFORMED_BY]-(e1) WHERE a2.id < a3.id
WITH a1, a2, a3, n, COLLECT(e1) AS events1 WHERE SIZE(events1) >= n
RETURN a1.id, a2.id, a3.id;
```

Cuádruplas de Artistas que tocaron en N o más Eventos juntos

PostgreSQL

```
SELECT E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist
FROM event_fact AS E1
INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
INNER JOIN event_fact AS E3 ON E2.id_event = E3.id_event
INNER JOIN event_fact AS E4 ON E3.id_event = E4.id_event
WHERE E1.id_artist < E2.id_artist AND E2.id_artist < E3.id_artist
      AND E3.id_artist < E4.id_artist
GROUP BY E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist
HAVING COUNT(*) >= 1 -- N
ORDER BY COUNT(*) DESC
```

Neo4J

Primera Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact), (a2:Artist)<-[:PERFORMED_BY]-(e),
      (a3:Artist)<-[:PERFORMED_BY]-(e), (a4:Artist)<-[:PERFORMED_BY]-(e)
WHERE a1.id < a2.id < a3.id < a4.id
WITH a1, a2, a3, a4, COLLECT(e) AS events
WHERE SIZE(events) >= n
RETURN a1.id, a2.id, a3.id, a4.id;
```

Segunda Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e0:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, n, COLLECT(e0) AS events0
WHERE SIZE(events0) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e1:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3:Artist)<-[:PERFORMED_BY]-(e1) WHERE a2.id < a3.id
WITH a1, a2, a3, n, COLLECT(e1) AS events1 WHERE SIZE(events1) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e2:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3)<-[:PERFORMED_BY]-(e2)-[:PERFORMED_BY]->(a4:Artist) WHERE a3.id < a4.id
WITH a1, a2, a3, a4, n, COLLECT(e2) AS events2 WHERE SIZE(events2) >= n
RETURN a1.id, a2.id, a3.id, a4.id;
```

Quíntuplas de Artistas que tocaron en N o más Eventos juntos

PostgreSQL

```
SELECT E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist, E5.id_artist
FROM event_fact AS E1
INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
INNER JOIN event_fact AS E3 ON E2.id_event = E3.id_event
INNER JOIN event_fact AS E4 ON E3.id_event = E4.id_event
INNER JOIN event_fact AS E5 ON E4.id_event = E5.id_event
WHERE E1.id_artist < E2.id_artist AND E2.id_artist < E3.id_artist
      AND E3.id_artist < E4.id_artist AND E4.id_artist < E5.id_artist
GROUP BY E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist, E5.id_artist
HAVING COUNT(*) >= 1 -- N
ORDER BY COUNT(*) DESC
```

Neo4J

Primera Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact), (a2:Artist)<-[:PERFORMED_BY]-(e),
      (a3:Artist)<-[:PERFORMED_BY]-(e), (a4:Artist)<-[:PERFORMED_BY]-(e),
      (a5:Artist)<-[:PERFORMED_BY]-(e)
WHERE a1.id < a2.id < a3.id < a4.id < a5.id
WITH a1, a2, a3, a4, a5, COLLECT(e) AS events
WHERE SIZE(events) >= n
RETURN a1.id, a2.id, a3.id, a4.id, a5.id;
```

Segunda Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e0:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, n, COLLECT(e0) AS events0 WHERE SIZE(events0) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e1:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3:Artist)<-[:PERFORMED_BY]-(e1) WHERE a2.id < a3.id
WITH a1, a2, a3, n, COLLECT(e1) AS events1 WHERE SIZE(events1) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e2:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3)<-[:PERFORMED_BY]-(e2)-[:PERFORMED_BY]->(a4:Artist) WHERE a3.id < a4.id
WITH a1, a2, a3, a4, n, COLLECT(e2) AS events2 WHERE SIZE(events2) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e3:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3)<-[:PERFORMED_BY]-(e3)-[:PERFORMED_BY]->(a4)
MATCH (a5:Artist)<-[:PERFORMED_BY]-(e3) WHERE a4.id < a5.id
WITH a1, a2, a3, a4, a5, n, COLLECT(e3) AS events3 WHERE SIZE(events3) >= n
RETURN a1.id, a2.id, a3.id, a4.id, a5.id;
```

Pares de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

PostgreSQL

```
(
   SELECT E1.id_artist, E2.id_artist
   FROM event_fact AS E1
   INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
   WHERE E1.id_artist < E2.id_artist
   GROUP BY E1.id_artist, E2.id_artist
   HAVING COUNT(*) >= 1 -- N
   ORDER BY COUNT(*) DESC
)
   INTERSECT
(
   SELECT AC1.id_artist, AC2.id_artist
   FROM  artist_credit_dim AS AC1
   INNER JOIN artist_credit_dim AS AC2  ON AC1.id = AC2.id
   INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
   WHERE AC1.id_artist < AC2.id_artist
   GROUP BY AC1.id_artist, AC2.id_artist
   HAVING COUNT(*) >= 1 -- M
   ORDER BY COUNT(*) DESC
)
```

Neo4J

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, COLLECT(e) AS events
WHERE SIZE(events) >= n
WITH a1, a2, events
OPTIONAL MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit)-[:INCLUDES]->(a2)
WITH a1, a2, ac, events
OPTIONAL MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH  a1, a2, events, collect(r) AS releases
WHERE SIZE(releases) = 0
RETURN a1.id, a2.id;
```

Triplas de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

PostgreSQL

```
(
   SELECT E1.id_artist, E2.id_artist, E3.id_artist
   FROM event_fact AS E1
   INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
   INNER JOIN event_fact AS E3 ON E2.id_event = E3.id_event
   WHERE E1.id_artist < E2.id_artist AND E2.id_artist < E3.id_artist
   GROUP BY E1.id_artist, E2.id_artist, E3.id_artist
   HAVING COUNT(*) >= 1 -- N
   ORDER BY COUNT(*) DESC
)
   INTERSECT
(
   SELECT AC1.id_artist, AC2.id_artist, AC3.id_artist
   FROM artist_credit_dim AS AC1
   INNER JOIN artist_credit_dim AS AC2 ON AC1.id = AC2.id
   INNER JOIN artist_credit_dim AS AC3 ON AC2.id = AC3.id
   INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
   WHERE AC1.id_artist < AC2.id_artist AND AC2.id_artist < AC3.id_artist
   GROUP BY AC1.id_artist, AC2.id_artist, AC3.id_artist
   HAVING COUNT(*) >= 1 -- M
   ORDER BY COUNT(*) DESC
)
```

Neo4J

Primera Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact), (a2:Artist)<-[:PERFORMED_BY]-(e),
      (a3:Artist)<-[:PERFORMED_BY]-(e)
WHERE a1.id < a2.id < a3.id
WITH a1, a2, a3, COLLECT(e) AS events
WHERE SIZE(events) >= n
WITH a1, a2, a3, events
OPTIONAL MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac),
               (a3)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, ac, events
OPTIONAL MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, events, collect(r) AS releases
WHERE SIZE(releases) = 0
RETURN a1.id, a2.id, a3.id;
```

Segunda Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e0:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, n, COLLECT(e0) AS events0
WHERE SIZE(events0) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e1:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3:Artist)<-[:PERFORMED_BY]-(e1) WHERE a2.id < a3.id
WITH a1, a2, a3, n, COLLECT(e1) AS events1 WHERE SIZE(events1) >= n
OPTIONAL MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac),
               (a3)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, ac
OPTIONAL MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, collect(r) AS releases
WHERE SIZE(releases) = 0
RETURN a1.id, a2.id, a3.id;
```

Cuádruplas de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

PostgreSQL

```
(
   SELECT E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist
   FROM event_fact AS E1
   INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
   INNER JOIN event_fact AS E3 ON E2.id_event = E3.id_event
   INNER JOIN event_fact AS E4 ON E3.id_event = E4.id_event
   WHERE E1.id_artist < E2.id_artist AND E2.id_artist < E3.id_artist
         AND E3.id_artist < E4.id_artist
   GROUP BY E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist
   HAVING COUNT(*) >= 1 -- N
   ORDER BY COUNT(*) DESC
)
   INTERSECT
(
   SELECT AC1.id_artist, AC2.id_artist, AC3.id_artist, AC4.id_artist
   FROM artist_credit_dim AS AC1
   INNER JOIN artist_credit_dim AS AC2 ON AC1.id = AC2.id
   INNER JOIN artist_credit_dim AS AC3 ON AC2.id = AC3.id
   INNER JOIN artist_credit_dim AS AC4 ON AC3.id = AC4.id
   INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
   WHERE AC1.id_artist < AC2.id_artist AND AC2.id_artist < AC3.id_artist
         AND AC3.id_artist < AC4.id_artist
   GROUP BY AC1.id_artist, AC2.id_artist, AC3.id_artist, AC4.id_artist
   HAVING COUNT(*) >= 1 -- M
   ORDER BY COUNT(*) DESC
)
```

Neo4J

Primera Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact), (a2:Artist)<-[:PERFORMED_BY]-(e),
     (a3:Artist)<-[:PERFORMED_BY]-(e), (a4:Artist)<-[:PERFORMED_BY]-(e)
WHERE a1.id < a2.id < a3.id < a4.id
WITH a1, a2, a3, a4, COLLECT(e) AS events
WHERE SIZE(events) >= n
WITH a1, a2, a3, a4, events
OPTIONAL MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac),
               (a3)<-[:INCLUDES]-(ac), (a4)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, a4, ac, events
OPTIONAL MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, a4, events, collect(r) AS releases
WHERE SIZE(releases) = 0
RETURN a1.id, a2.id, a3.id, a4.id;
```

Segunda Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e0:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, n, COLLECT(e0) AS events0
WHERE SIZE(events0) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e1:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3:Artist)<-[:PERFORMED_BY]-(e1) WHERE a2.id < a3.id
WITH a1, a2, a3, n, COLLECT(e1) AS events1 WHERE SIZE(events1) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e2:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3)<-[:PERFORMED_BY]-(e2)-[:PERFORMED_BY]->(a4:Artist) WHERE a3.id < a4.id
WITH a1, a2, a3, a4, n, COLLECT(e2) AS events2 WHERE SIZE(events2) >= n
OPTIONAL MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac),
               (a3)<-[:INCLUDES]- (ac), (a4)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, a4, ac,
OPTIONAL MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, a4, collect(r) AS releases
WHERE SIZE(releases) = 0
RETURN a1.id, a2.id, a3.id, a4.id;
```

Quíntuplas de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

PostgreSQL

```
(
   SELECT E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist, E5.id_artist
   FROM event_fact AS E1
   INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
   INNER JOIN event_fact AS E3 ON E2.id_event = E3.id_event
   INNER JOIN event_fact AS E4 ON E3.id_event = E4.id_event
   INNER JOIN event_fact AS E5 ON E4.id_event = E5.id_event
   WHERE E1.id_artist < E2.id_artist AND E2.id_artist < E3.id_artist
         AND E3.id_artist < E4.id_artist AND E4.id_artist < E5.id_artist
   GROUP BY E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist, E5.id_artist
   HAVING      COUNT(*) >= 1 -- N
   ORDER BY    COUNT(*) DESC
)
   INTERSECT
(
   SELECT AC1.id_artist, AC2.id_artist, AC3.id_artist, AC4.id_artist, AC5.id_artist
   FROM artist_credit_dim AS AC1
   INNER JOIN artist_credit_dim AS AC2 ON AC1.id = AC2.id
   INNER JOIN artist_credit_dim AS AC3 ON AC2.id = AC3.id
   INNER JOIN artist_credit_dim AS AC4 ON AC3.id = AC4.id
   INNER JOIN artist_credit_dim AS AC5 ON AC4.id = AC5.id
   INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
   WHERE AC1.id_artist < AC2.id_artist AND AC2.id_artist < AC3.id_artist
         AND AC3.id_artist < AC4.id_artist AND AC4.id_artist < AC5.id_artist
   GROUP BY AC1.id_artist, AC2.id_artist, AC3.id_artist,
                                       AC4.id_artist, AC5.id_artist
   HAVING      COUNT(*) >= 1 -- M
   ORDER BY    COUNT(*) DESC
)
```

Neo4J

Primera Versión

```
WITH 1 as n
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact), (a2:Artist)<-[:PERFORMED_BY]-(e),
     (a3:Artist)<-[:PERFORMED_BY]-(e), (a4:Artist)<-[:PERFORMED_BY]-(e),
     (a5:Artist)<-[:PERFORMED_BY]-(e)
WHERE a1.id < a2.id < a3.id < a4.id < a5.id
WITH a1, a2, a3, a4, a5, COLLECT(e) AS events
WHERE SIZE(events) >= n
WITH a1, a2, a3, a4, a5, events
OPTIONAL MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac),
               (a3)<-[:INCLUDES]-(ac), (a4)<-[:INCLUDES]-(ac), (a5)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, a4, a5, ac, events
OPTIONAL MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, a4, a5, events, collect(r) AS releases
WHERE SIZE(releases) = 0
RETURN a1.id, a2.id, a3.id, a4.id, a5.id;
```

Segunda Versión

```
WITH 1 as n, 1 as m
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e0:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, n, m, COLLECT(e0) AS events0 WHERE SIZE(events0) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e1:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3:Artist)<-[:PERFORMED_BY]-(e1) WHERE a2.id < a3.id
WITH a1, a2, a3, n, m, COLLECT(e1) AS events1 WHERE SIZE(events1) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e2:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3)<-[:PERFORMED_BY]-(e2)-[:PERFORMED_BY]->(a4:Artist) WHERE a3.id < a4.id
WITH a1, a2, a3, a4, n, m, COLLECT(e2) AS events2 WHERE SIZE(events2) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e3:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3)<-[:PERFORMED_BY]-(e3)-[:PERFORMED_BY]->(a4)
MATCH (a5:Artist)<-[:PERFORMED_BY]-(e3) WHERE a4.id < a5.id
WITH a1, a2, a3, a4, a5, n, m, COLLECT(e3) AS events3 WHERE SIZE(events3) >= n
OPTIONAL MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac),
               (a3)<-[:INCLUDES]-(ac), (a4)<-[:INCLUDES]-(ac), (a5)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, a4, a5, ac, m
OPTIONAL MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, a4, a5, m, collect(r) AS releases
WHERE SIZE(releases) = 0
RETURN a1.id, a2.id, a3.id, a4.id, a5.id;
```

Pares de Artistas que tocaron en N o más Eventos juntos y compartieron M o más Releases

PostgreSQL

```
(
   SELECT E1.id_artist, E2.id_artist
   FROM event_fact AS E1
   INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
   WHERE E1.id_artist < E2.id_artist
   GROUP BY E1.id_artist, E2.id_artist
   HAVING COUNT(*) >= 1 -- N
   ORDER BY COUNT(*) DESC
)
   INTERSECT
(
   SELECT AC1.id_artist, AC2.id_artist
   FROM artist_credit_dim AS AC1
   INNER JOIN artist_credit_dim AS AC2 ON AC1.id = AC2.id
   INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
   WHERE AC1.id_artist < AC2.id_artist
   GROUP BY AC1.id_artist, AC2.id_artist
   HAVING COUNT(*) >= 1 -- M
   ORDER BY COUNT(*) DESC
)
```

Neo4J

Primera Versión

```
WITH 1 as n, 1 as m
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, COLLECT(e) AS events
WHERE SIZE(events) >= n
WITH a1, a2, events
MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit)-[:INCLUDES]->(a2)
WITH a1, a2, ac, events
MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, events, collect(r) AS releases
WHERE SIZE(releases) >= m
RETURN a1.id, a2.id;
```

Triplas de Artistas que tocaron en N o más Eventos juntos y compartieron M o más Releases

PostgreSQL

```
(
   SELECT E1.id_artist, E2.id_artist, E3.id_artist
   FROM event_fact AS E1
   INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
   INNER JOIN event_fact AS E3 ON E2.id_event = E3.id_event
   WHERE E1.id_artist < E2.id_artist AND E2.id_artist < E3.id_artist
   GROUP BY E1.id_artist, E2.id_artist, E3.id_artist
   HAVING COUNT(*) >= 1 -- N
   ORDER BY COUNT(*) DESC
)
   INTERSECT
(
   SELECT AC1.id_artist, AC2.id_artist, AC3.id_artist
   FROM artist_credit_dim AS AC1
   INNER JOIN artist_credit_dim AS AC2 ON AC1.id = AC2.id
   INNER JOIN artist_credit_dim AS AC3 ON AC2.id = AC3.id
   INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
   WHERE AC1.id_artist < AC2.id_artist AND AC2.id_artist < AC3.id_artist
   GROUP BY AC1.id_artist, AC2.id_artist, AC3.id_artist
   HAVING COUNT(*) >= 1 -- M
   ORDER BY COUNT(*) DESC
)
```

Neo4J

Primera Versión

```
WITH 1 as n, 1 as m
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact), (a2:Artist)<-[:PERFORMED_BY]-(e),
      (a3:Artist)<-[:PERFORMED_BY]-(e)
WHERE a1.id < a2.id < a3.id
WITH a1, a2, a3, COLLECT(e) AS events
WHERE SIZE(events) >= n
WITH a1, a2, a3, events
MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac), (a3)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, ac, events
MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, events, collect(r) AS releases
WHERE SIZE(releases) >= m
RETURN a1.id, a2.id, a3.id;
```

Segunda Versión

```
WITH 1 as n, 1 as m
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e0:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, n, m, COLLECT(e0) AS events0
WHERE SIZE(events0) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e1:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3:Artist)<-[:PERFORMED_BY]-(e1) WHERE a2.id < a3.id
WITH a1, a2, a3, n, m, COLLECT(e1) AS events1 WHERE SIZE(events1) >= n
MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac), (a3)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, ac, m
MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, m, collect(r) AS releases
WHERE SIZE(releases) >= m
RETURN a1.id, a2.id, a3.id;
```

Cuádruplas de Artistas que tocaron en N o más eventos juntos y compartieron M o más Releases

PostgreSQL

```
(
   SELECT E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist
   FROM event_fact AS E1
   INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
   INNER JOIN event_fact AS E3 ON E2.id_event = E3.id_event
   INNER JOIN event_fact AS E4 ON E3.id_event = E4.id_event
   WHERE E1.id_artist < E2.id_artist AND E2.id_artist < E3.id_artist
         AND E3.id_artist < E4.id_artist
   GROUP BY E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist
   HAVING COUNT(*) >= 1 -- N
   ORDER BY COUNT(*) DESC
)
   INTERSECT
(
   SELECT AC1.id_artist, AC2.id_artist, AC3.id_artist, AC4.id_artist
   FROM artist_credit_dim AS AC1
   INNER JOIN artist_credit_dim AS AC2 ON AC1.id = AC2.id
   INNER JOIN artist_credit_dim AS AC3 ON AC2.id = AC3.id
   INNER JOIN artist_credit_dim AS AC4 ON AC3.id = AC4.id
   INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
   WHERE AC1.id_artist < AC2.id_artist AND AC2.id_artist < AC3.id_artist
         AND AC3.id_artist < AC4.id_artist
   GROUP BY AC1.id_artist, AC2.id_artist, AC3.id_artist, AC4.id_artist
   HAVING COUNT(*) >= 1 -- M
   ORDER BY COUNT(*) DESC
)
```

Neo4J

Primera Versión

```
WITH 1 as n, 1 as m
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact), (a2:Artist)<-[:PERFORMED_BY]-(e),
      (a3:Artist)<-[:PERFORMED_BY]-(e), (a4:Artist)<-[:PERFORMED_BY]-(e)
WHERE a1.id < a2.id < a3.id < a4.id
WITH a1, a2, a3, a4, COLLECT(e) AS events
WHERE SIZE(events) >= n
WITH a1, a2, a3, a4, events
MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac), (a3)<-[:INCLUDES]-(ac),
      (a4)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, a4, ac, events
MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, a4, events, collect(r) AS releases
WHERE SIZE(releases) >= n
RETURN a1.id, a2.id, a3.id, a4.id;
```

Segunda Versión

```
WITH 1 as n, 1 as m
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e0:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, n, m, COLLECT(e0) AS events0
WHERE SIZE(events0) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e1:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3:Artist)<-[:PERFORMED_BY]-(e1) WHERE a2.id < a3.id
WITH a1, a2, a3, n, m, COLLECT(e1) AS events1 WHERE SIZE(events1) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e2:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3)<-[:PERFORMED_BY]-(e2)-[:PERFORMED_BY]->(a4:Artist) WHERE a3.id < a4.id
WITH a1, a2, a3, a4, n, m, COLLECT(e2) AS events2 WHERE SIZE(events2) >= n
MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac), (a3)<-[:INCLUDES]-(ac),
      (a4)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, a4, ac, m
MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, a4, m, collect(r) AS releases
WHERE SIZE(releases) >= m
RETURN a1.id, a2.id, a3.id, a4.id;
```

Quíntuplas de Artistas que tocaron en N o más Eventos juntos y compartieron M o más Releases

PostgreSQL

```
(
   SELECT E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist, E5.id_artist
   FROM event_fact AS E1
   INNER JOIN event_fact AS E2 ON E1.id_event = E2.id_event
   INNER JOIN event_fact AS E3 ON E2.id_event = E3.id_event
   INNER JOIN event_fact AS E4 ON E3.id_event = E4.id_event
   INNER JOIN event_fact AS E5 ON E4.id_event = E5.id_event
   WHERE E1.id_artist < E2.id_artist AND E2.id_artist < E3.id_artist
         AND E3.id_artist < E4.id_artist AND E4.id_artist < E5.id_artist
   GROUP BY E1.id_artist, E2.id_artist, E3.id_artist, E4.id_artist, E5.id_artist
   HAVING COUNT(*) >= 1 -- N
   ORDER BY COUNT(*) DESC
)
   INTERSECT
(
   SELECT AC1.id_artist, AC2.id_artist, AC3.id_artist, AC4.id_artist, AC5.id_artist
   FROM artist_credit_dim AS AC1
   INNER JOIN artist_credit_dim AS AC2 ON AC1.id = AC2.id
   INNER JOIN artist_credit_dim AS AC3 ON AC2.id = AC3.id
   INNER JOIN artist_credit_dim AS AC4 ON AC3.id = AC4.id
   INNER JOIN artist_credit_dim AS AC5 ON AC4.id = AC5.id
   INNER JOIN release_fact AS R ON R.id_artist_credit = AC1.id
   WHERE AC1.id_artist < AC2.id_artist AND AC2.id_artist < AC3.id_artist
         AND AC3.id_artist < AC4.id_artist AND AC4.id_artist < AC5.id_artist
   GROUP BY AC1.id_artist, AC2.id_artist, AC3.id_artist,
                                 AC4.id_artist, AC5.id_artist
   HAVING COUNT(*) >= 1 -- M
   ORDER BY COUNT(*) DESC
)
```

Neo4J

Primera Versión

```
WITH 1 as n, 1 as m
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e:EventFact), (a2:Artist)<-[:PERFORMED_BY]-(e),
      (a3:Artist)<-[:PERFORMED_BY]-(e), (a4:Artist)<-[:PERFORMED_BY]-(e),
      (a5:Artist)<-[:PERFORMED_BY]-(e)
WHERE a1.id < a2.id < a3.id < a4.id < a5.id
WITH a1, a2, a3, a4, a5, COLLECT(e) AS events
WHERE SIZE(events) >= n
WITH a1, a2, a3, a4, a5, events
MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac), (a3)<-[:INCLUDES]-(ac),
      (a4)<-[:INCLUDES]-(ac), (a5)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, a4, a5, ac, events
MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, a4, a5, events, collect(r) AS releases
WHERE SIZE(releases) >= m
RETURN a1.id, a2.id, a3.id, a4.id, a5.id;
```

Segunda Versión

```
WITH 1 as n, 1 as m
MATCH (a1:Artist)<-[:PERFORMED_BY]-(e0:EventFact)-[:PERFORMED_BY]->(a2:Artist)
WHERE a1.id < a2.id
WITH a1, a2, n, m, COLLECT(e0) AS events0 WHERE SIZE(events0) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e1:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3:Artist)<-[:PERFORMED_BY]-(e1) WHERE a2.id < a3.id
WITH a1, a2, a3, n, m, COLLECT(e1) AS events1 WHERE SIZE(events1) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e2:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3)<-[:PERFORMED_BY]-(e2)-[:PERFORMED_BY]->(a4:Artist) WHERE a3.id < a4.id
WITH a1, a2, a3, a4, n, m, COLLECT(e2) AS events2 WHERE SIZE(events2) >= n
MATCH (a1)<-[:PERFORMED_BY]-(e3:EventFact)-[:PERFORMED_BY]->(a2)
MATCH (a3)<-[:PERFORMED_BY]-(e3)-[:PERFORMED_BY]->(a4)
MATCH (a5:Artist)<-[:PERFORMED_BY]-(e3) WHERE a4.id < a5.id
WITH a1, a2, a3, a4, a5, n, m, COLLECT(e3) AS events3 WHERE SIZE(events3) >= n
MATCH (a1)<-[:INCLUDES]-(ac:ArtistCredit), (a2)<-[:INCLUDES]-(ac), (a3)<-[:INCLUDES]-(ac),
      (a4)<-[:INCLUDES]-(ac), (a5)<-[:INCLUDES]-(ac)
WITH a1, a2, a3, a4, a5, ac, m
MATCH (r:ReleaseFact)-[:RELEASED_BY]->(ac)
WITH a1, a2, a3, a4, a5, m, collect(r) AS releases
WHERE SIZE(releases) >= m
RETURN a1.id, a2.id, a3.id, a4.id, a5.id;
```

RECOMENDACIONES

Recomendaciones de Artistas

PostgreSQL

```
CREATE FUNCTION recommend_artists(id_original_artist int, N int)
RETURNS TABLE(id_artist int)
AS $$
BEGIN
   IF N = 1 THEN
       RETURN QUERY
       SELECT DISTINCT(AC2.id_artist)
       FROM artist_credit_dim AS AC
       INNER JOIN artist_credit_dim AS AC2 ON AC.id = AC2.id
       WHERE AC.id_artist != AC2.id_artist AND AC.id_artist = id_original_artist
             AND AC.artist_count > 1;
   ElSE
       RETURN QUERY
       SELECT DISTINCT(AC4.id_artist)
       FROM artist_credit_dim AS AC3
       INNER JOIN artist_credit_dim AS AC4 ON AC3.id = AC4.id
       WHERE AC4.id_artist != id_original_artist AND AC3.id_artist IN (
        SELECT * FROM recommend_artists (id_original_artist, N-1)
       );
   END IF;
END;
$$
LANGUAGE plpgsql;
```

Neo4J

```
WITH _PARAM_ AS givenArtist
MATCH (myArtist:Artist)-[:INCLUDES]-(:ArtistCredit)-[:INCLUDES*1..2N]-(otherArtists:Artist)
WHERE (myArtist.id = givenArtist) AND NOT (otherArtists.id = givenArtist)
RETURN DISTINCT otherArtists
```

Recomendaciones de Artistas con Optimización

PostgreSQL

```
CREATE FUNCTION recommend_artists_opt(id_original_artist integer, n integer)
RETURNS TABLE(id_artist integer)
AS $$
BEGIN
   IF N = 1 THEN
       RETURN QUERY
       SELECT DISTINCT(id_artist_2)
       FROM collab_with
       WHERE id_artist_1 = id_original_artist;
   ElSE
       RETURN QUERY
       SELECT DISTINCT(id_artist_2)
       FROM collab_with
       WHERE id_artist_2 != id_original_artist AND id_artist_1 IN (
        SELECT * FROM recommend_artists_opt(id_original_artist, N-1)
       );
   END IF;
END;
$$
LANGUAGE 'plpgsql'
```

Neo4J

```
WITH _PARAM_ AS givenArtist
MATCH (myArtist:Artist)-[:COLLAB_WITH*1..N]-(otherArtists:Artist)
WHERE (myArtist.id = givenArtist) AND NOT (otherArtists.id = givenArtist)
RETURN DISTINCT otherArtists
```

PRIMER SET DE RESULTADOS

Pares de Artistas que tocaron en N o más Eventos juntos

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	2	1	410ms	197974	1s 41ms	197974
2	2	1	395ms		1s 22ms	
3	2	1	393ms		1s 87ms	
1	2	2	278ms	10995	762ms	10995
2	2	2	269ms		784ms	
3	2	2	261ms		808ms	
1	2	3	302ms	3951	749ms	3951
2	2	3	259ms		795ms	
3	2	3	279ms		949ms	
1	2	4	272ms	2641	750ms	2641
2	2	4	291ms		838ms	
3	2	4	281ms		850ms	
1	2	5	256ms	2357	771ms	2357
2	2	5	288ms		773ms	
3	2	5	269ms		848ms	

Triplas de Artistas que tocaron en N o más Eventos juntos

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	3	1	9s 144ms	3607114	24s 755ms	3607114
2	3	1	9s 197ms		25s 698ms	
3	3	1	9s 175ms		24s 917ms	
1	3	2	5s 512ms	52385	24s 755ms	52385
2	3	2	5s 538ms		20s 459ms	
3	3	2	5s 608ms		19s 622ms	
1	3	3	5s 310ms	37096	22s 758ms	37096
2	3	3	5s 568ms		18s 504ms	
3	3	3	5s 520ms		18s 821ms	
1	3	4	5s 405ms	33540	19s 463ms	33540
2	3	4	5s 379ms		18s 606ms	
3	3	4	5s 464ms		19s 301ms	
1	3	5	5s 415ms	33227	20s 444ms	33227

2	3	5	5s 462ms		18s 559ms	
3	3	5	5s 538ms		19s 831ms	

Cuádruplas de Artistas que tocaron en N o más Eventos juntos

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	4	1	5m 24s	117951941	8hs+ Mem: 26GB	OUT OF HEAP
2	4	1	5m 26s			
3	4	1	5m 19s			
1	4	2	4m 26s	504444	8hs+ Mem: 26GB	OUT OF HEAP
2	4	2	4m 21s			
3	4	2	4m 24s			
1	4	3	4m 24s	469769	8hs+ Mem: 26GB	OUT OF HEAP
2	4	3	4m 30s			
3	4	3	4m 29s			
1	4	4	4m 29s	456836	8hs+ Mem: 26GB	OUT OF HEAP
2	4	4	4m 29s			
3	4	4	4m 29s			
1	4	5	4m 29s	456306	8hs+ Mem: 26GB	OUT OF HEAP
2	4	5	4m 29s			
3	4	5	4m 29s			

Quíntuplas de Artistas que tocaron en N o más Eventos juntos

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	4	1		NOT FINISHED		
2	4	1	18hs			
3	4	1				
1	4	2		NOT FINISHED		
2	4	2	18hs			
3	4	2				
1	4	3		NOT FINISHED		
2	4	3	18hs			
3	4	3				
1	4	4		NO DATA		

2	4	4	39m 40s			
3	4	4				
1	4	5		NO DATA		
2	4	5	45m 40s			
3	4	5				

Pares de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	2	1	4s 5ms	196980	3s 883ms	196980
2	2	1	3s 851ms		4s 195ms	
3	2	1	4s 205ms		3s 706ms	
1	2	2	3s 932ms	10819	1s 72ms	10819
2	2	2	4s 82ms		1s 1ms	
3	2	2	4s 64ms		1s 10ms	
1	2	3	3s 972ms	3872	836ms	3872
2	2	3	3s 861ms		858ms	
3	2	3	3s 871ms		886ms	
1	2	4	3s 925ms	2599	802ms	2599
2	2	4	3s 914ms		801ms	
3	2	4	3s 756ms		824ms	
1	2	5	3s 925ms	2328	798ms	2328
2	2	5	3s 851ms		792ms	
3	2	5	4s 13ms		786ms	

Triplas de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	3	1	15s 865ms	3606482	58s 741ms	3606482
2	3	1	15s 438ms		58s 384ms	
3	3	1	16s 157ms		57s 717ms	
1	3	2	10s 834ms	52360	21s 842ms	52360
2	3	2	10s 726ms		19s 902ms	
3	3	2	10s 659ms		20s 93ms	
1	3	3	11s 10ms	37089	19s 208ms	37089
2	3	3	9s 907ms		19s 61ms	

3	3	3	10s 871ms		19s 90ms	
1	3	4	10s 886ms	33539	19s 229ms	33539
2	3	4	10s 380ms		19s 37ms	
3	3	4	10s 794ms		19s 836ms	
1	3	5	11s 363ms	33226	19s 167ms	33226
2	3	5	10s 602ms		19s 526ms	
3	3	5	10s 766ms		19s 167ms	

Cuádruplas de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	3	1	10m 13s	117950708	20hs	NOT FINISHED
2	3	1	9m 52s			
3	3	1	9m 53s			
1	3	2	5m 39s	504429	20hs	NOT FINISHED
2	3	2	5m 39s			
3	3	2	5m 50s			
1	3	3	5m 43s	469768	20hs	NOT FINISHED
2	3	3	5m 44s			
3	3	3	5m 59s			
1	3	4	5m 54s	456836	20hs	NOT FINISHED
2	3	4	5m 49s			
3	3	4	5m 48s			
1	3	5	5m 49s	456306	20hs	NOT FINISHED
2	3	5	5m 43s			
3	3	5	5m 33s			

Quíntuplas de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	3	1	12hs	NOT FINISHED		
2	3	1				
3	3	1				

1	3	2	12hs	NOT FINISHED		
2	3	2				
3	3	2				
1	3	3	12hs	NOT FINISHED		
2	3	3				
3	3	3				
1	3	4	12hs	NOT FINISHED		
2	3	4				
3	3	4				
1	3	5	12hs	NOT FINISHED		
2	3	5				
3	3	5				

Pares de Artistas que tocaron en N o más Eventos juntos y compartieron M o más Releases

	PARAMETROS			SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**M**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	2	1	1	9s 82ms	994	2s 842ms	994
2	2	1	1	4s 15ms		2s 614ms	
1	2	1	2	3s 869ms	421	2s 723ms	414
2	2	1	2	3s 832ms		3s 13ms	
1	2	1	3	3s 795ms	256	2s 775ms	256
2	2	1	3	3s 790ms		2s 570ms	
1	2	1	4	3s 996ms	185	2s 556ms	185
2	2	1	4	4s 7ms		2s 635ms	
1	2	1	5	3s 860ms	152	2s 734ms	150
2	2	1	5	3s 995ms		2s 654ms	
1	2	2	1	3s 869ms	176	1s 41ms	176
2	2	2	1	3s 886ms		949ms	
1	2	2	2	3s 566ms	107	1s 14ms	106
2	2	2	2	3s 629ms		999ms	
1	2	2	3	3s 639ms	80	970ms	80
2	2	2	3	3s 648ms		937ms	
1	2	2	4	3s 656ms	57	1s 13ms	57
2	2	2	4	3s 778ms		987ms	
1	2	2	5	3s 719ms	48	931ms	48
2	2	2	5	3s 603ms		881ms	
1	2	3	1	3s 737ms	79	741ms	79

2	2	3	1	3s 854ms		957ms	
1	2	3	2	3s 646ms	52	894ms	52
2	2	3	2	3s 608ms		776ms	
1	2	3	3	3s 485ms	43	739ms	43
2	2	3	3	3s 760ms		902ms	
1	2	3	4	3s 612ms	27	923ms	27
2	2	3	4	3s 582ms		705ms	
1	2	3	5	3s 798ms	23	756ms	23
2	2	3	5	3s 761ms		899ms	

Triplas de Artistas que tocaron en N o más Eventos juntos y compartieron M o más Releases

	PARAMETROS			SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**M**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	3	1	1	17s 257ms	632	47s 94ms	632
2	3	1	1	17s 209ms		45s 930ms	
1	3	1	2	17s 273ms	62	45s 99ms	61
2	3	1	2	17s 250ms		44s 609ms	
1	3	1	3	16s 379ms	15	45s 61ms	15
2	3	1	3	17s 4m		44s 864ms	
1	3	1	4	17s 18ms	8	45s 841ms	8
2	3	1	4	17s 30ms		44s 519ms	
1	3	1	5	16s 462ms	5	44s 801ms	5
2	3	1	5	16s 447ms		45s 491ms	
1	3	2	1	11s 255ms	25	17s 368ms	25
2	3	2	1	11s 415ms		17s 965ms	
1	3	2	2	11s 333ms	7	17s 697ms	7
2	3	2	2	10s 918ms		17s 118ms	
1	3	2	3	11s 293ms	4	17s 206ms	4
2	3	2	3	10s 672ms		17s 818ms	
1	3	2	4	11s 280ms	1	17s 759ms	1
2	3	2	4	10s 728ms		17s 56ms	
1	3	2	5	10s 571ms	NO DATA	17s 122ms	NO DATA
2	3	2	5	11s 360ms		17s 588ms	
1	3	3	1	11s 263ms	7	17s 216ms	7
2	3	3	1	11s 300ms		17s 878ms	
1	3	3	2	10s 854ms	1	17s 686ms	1
2	3	3	2	11s 323ms		17s 346ms	

1	3	3	3	10s 740ms	1	17s 40ms	1
2	3	3	3	10s 747ms		17s 795ms	
1	3	3	4	11s 300ms	NO DATA	17s 539ms	NO DATA
2	3	3	4	10s 824ms		17s 289ms	
1	3	3	5	10s 521ms	NO DATA	16s 947ms	NO DATA
2	3	3	5	10s 789ms		17s 532ms	

Cuádruplas de Artistas que tocaron en N o más eventos juntos y compartieron M o más Releases

	PARAMETROS			**SQL**		**NEO4J**	
N° Corrida	**Artistas**	**N**	**M**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	4	1	1	6m 39s	1233	8h+ 26GB	OUT OF HEAP
2	4	1	1	6m 44s			
1	4	1	2	6m 29s	38	8h+ 26GB	OUT OF HEAP
2	4	1	2	6m 28s			
1	4	1	3	6m 28s	1	8h+ 26GB	OUT OF HEAP
2	4	1	3	6m 28s			
1	4	1	4	6m 28s	NO DATA	8h+ 26GB	OUT OF HEAP
2	4	1	4	10m 44s			
1	4	1	5	10m 39s	NO DATA	8h+ 26GB	OUT OF HEAP
2	4	1	5	10m 33s			
1	4	2	1	4m 28s	15	8h+ 26GB	OUT OF HEAP
2	4	2	1	4m 2s			
1	4	2	2	4m 7s	2	8h+ 26GB	OUT OF HEAP
2	4	2	2	4m 7s			
1	4	2	3	4m 13s	1	8h+ 26GB	OUT OF HEAP
2	4	2	3	4m 2s			
1	4	2	4	4m 8s	NO DATA	8h+ 26GB	OUT OF HEAP
2	4	2	4	4m 2s			
1	4	2	5	4m 7s	NO DATA	8h+ 26GB	OUT OF HEAP
2	4	2	5	4m 12s			
1	4	3	1	4m 7s	1	8h+ 26GB	OUT OF HEAP
2	4	3	1	4m 13s			
1	4	3	2	4m 7s	NO DATA	8h+ 26GB	OUT OF HEAP
2	4	3	2	4m 13s			
1	4	3	3	4m 7s	NO DATA	8h+ 26GB	OUT OF HEAP
2	4	3	3	4m 10s			
1	4	3	4	4m 2s	NO DATA	8h+ 26GB	

2	4	3	4	4m 9s			OUT OF HEAP
1	4	3	5	4m 5s	NO DATA	8h+ 26GB	OUT OF HEAP
2	4	3	5	4m 2s			

Quíntuplas de Artistas que tocaron en N o más Eventos juntos y compartieron M o más Releases

	PARAMETROS			SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**M**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	5	1	1	18hs	NOT FINISHED		
2	5	1	1				
1	5	1	2	18hs	NOT FINISHED		
2	5	1	2				
1	5	1	3	18hs	NOT FINISHED		
2	5	1	3				
1	5	1	4	18hs	NOT FINISHED		
2	5	1	4				
1	5	1	5	18hs	NOT FINISHED		
2	5	1	5				
1	5	2	1	18hs	NOT FINISHED		
2	5	2	1				
1	5	2	2	18hs	NOT FINISHED		
2	5	2	2				
1	5	2	3	18hs	NOT FINISHED		
2	5	2	3				
1	5	2	4	18hs	NOT FINISHED		
2	5	2	4				
1	5	2	5	18hs	NOT FINISHED		
2	5	2	5				
1	5	3	1	18hs	NOT FINISHED		
2	5	3	1				
1	5	3	2	18hs	NOT FINISHED		
2	5	3	2				
1	5	3	3	18hs	NOT FINISHED		
2	5	3	3				
1	5	3	4	18hs	NOT FINISHED		
2	5	3	4				
1	5	3	5	18hs			

2	5	3	5		NOT FINISHED		

Recomendaciones de Artistas

Id de artista seleccionado: 874111

	PARAMETROS		SQL		NEO4J			
N° Corrida	**OPT**	**Profundidad**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**	**Available**	**Consumed**
1	0	1	59ms	86	2ms	86	0ms	2ms
2	0	1	115ms		5ms		1ms	4ms
3	0	1	116ms		20ms		0ms	14ms
1	0	2	2s 951ms	708	196ms	708	1ms	195ms
2	0	2	1s 701ms		276ms		0ms	276ms
3	0	2	1s 503ms		983ms		0ms	983ms
1	0	3	2s 479ms	6567	7s 928ms	6567	2ms	7926ms
2	0	3	2s 715ms		18s 999ms		1ms	18998ms
3	0	3	2s 481ms		24s 584ms		1ms	24583ms
1	0	4	4s 169ms	35385	46m 2s	35385	33ms	2761772ms
2	0	4	4s 387ms		43m 53s		1ms	2633189ms
3	0	4	4s 168ms		46m 46s		0ms	2386323ms
1	0	5	6s 470ms	85185				
2	0	5	6s 436ms					
3	0	5	6s 330ms					
1	0	6	10s 3ms	149302				
2	0	6	9s 912ms					
3	0	6	10s 12ms					
1	0	7	14s 509ms	185241				
2	0	7	14s 549ms					
3	0	7	15s 43ms					
1	0	8	20s 113ms	199520				
2	0	8	18s 513ms					
3	0	8	20s 65ms					
1	0	9	23s 604ms	204853				
2	0	9	25s 121ms					
3	0	9	24s 147ms					
1	0	10	31s 347ms	207075				

2	0	10	26s 838ms					
3	0	10	30s 266ms					
1	0	15	53s 140ms	208682				
2	0	15	48s 323ms					
3	0	15	58s 37ms					
1	0	20	1m 20s	208772				
2	0	20	1m 9s					
3	0	20	1m 24s					
1	0	25	1m 47s	208778				
2	0	25	1m 52s					
3	0	25	1m 47s					
1	0	30	2m 13s	208778				
2	0	30	2m 18s					
3	0	30	2m 18s					

Recomendaciones de Artistas con Optimización.

Id de artista seleccionado: 874111

	PARAMETROS		SQL		NEO4J			
N° Corrida	**OPT**	**Profundidad**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**	**Available**	**Consumed**
1	1	1	419ms	87	16ms	86	15ms	1ms
2	1	1	448ms		2ms		1ms	1ms
3	1	1	444ms		3ms		2ms	1ms
1	1	2	982ms	708	25ms	708	17ms	8ms
2	1	2	997ms		7ms		1ms	6ms
3	1	2	1s 14ms		5ms		0ms	5ms
1	1	3	1s 597ms	6567	75ms	6567	15ms	60ms
2	1	3	1s 549ms		58ms		1ms	57ms
3	1	3	1s 603ms		60ms		0ms	60ms
1	1	4	2s 384ms	35385	406ms	35385	18ms	388ms
2	1	4	2s 300ms		375ms		2ms	373ms
3	1	4	2s 308ms		357ms		1ms	356ms
1	1	5	3s 296ms	85185	2s 100ms	85185	15ms	2085ms
2	1	5	3s 262ms		2s 226ms		0ms	2226ms

3	1	5	3s 334ms		1s 933ms		0ms	1933ms
1	1	6	4s 623ms	149302	1m 15s	149302	15ms	74532ms
2	1	6	4s 590ms		1m 13s		0ms	72961ms
3	1	6	4s 703ms		1m 12s		1ms	71984ms
1	1	7	6s 601ms	185241	2h 55m 55s	185241	42ms	10555005ms
2	1	7	6s 601ms		2h 2m 3s		245ms	9842516ms
3	1	7	6s 643ms		2h 33m 33s		3ms	10113268ms
1	1	8	8s 975ms	199520				
2	1	8	9s 12ms					
3	1	8	8s 863ms					
1	1	9	11s 440ms	204853				
2	1	9	11s 420ms					
3	1	9	12s 42ms					
1	1	10	13s 612ms	207075				
2	1	10	13s 761ms					
3	1	10	13s 740ms					
1	1	15	26s 521ms	208682				
2	1	15	25s 728ms					
3	1	15	25s 752ms					
1	1	20	38s 581ms	208772				
2	1	20	37s 587ms					
3	1	20	37s 322ms					
1	1	25	49s 706ms	208778				
2	1	25	50s 744ms					
3	1	25	49s 437ms					
1	1	30	1m 4s	208778				
2	1	30	1m 7s					
3	1	30	1m 7s					

SEGUNDO SET DE RESULTADOS

Triplas de Artistas que tocaron en N o más Eventos juntos

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	3	1	9s 144ms	3607114	35s 44ms	3607114
2	3	1	9s 197ms		35s 791ms	
3	3	1	9s 175ms		34s 76ms	
1	3	2	5s 512ms	52385	3s 746ms	52385
2	3	2	5s 538ms		3s 655ms	
3	3	2	5s 608ms		3s 817ms	
1	3	3	5s 310ms	37096	2s 560ms	37096
2	3	3	5s 568ms		2s 453ms	
3	3	3	5s 520ms		2s 483ms	
1	3	4	5s 405ms	33540	2s 152ms	33540
2	3	4	5s 379ms		2s 240ms	
3	3	4	5s 464ms		2s 163ms	
1	3	5	5s 415ms	33227	2s 165ms	33227
2	3	5	5s 462ms		2s 113ms	
3	3	5	5s 538ms		2s 369ms	

Cuádruplas de Artistas que tocaron en N o más Eventos juntos

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	4	1	5m 24s	117951941	36hs	NO RESULTS
2	4	1	5m 26s			
3	4	1	5m 19s			
1	4	2	4m 26s	504444	22s 264ms	504444
2	4	2	4m 21s		22s 510ms	
3	4	2	4m 24s		21s 870ms	
1	4	3	4m 24s	469769	17s 398ms	469769
2	4	3	4m 30s		16s 856ms	
3	4	3	4m 29s		19s 70ms	
1	4	4	4m 29s	456836	16s 993ms	456836
2	4	4	4m 29s		16s 559ms	
3	4	4	4m 29s		17s 514ms	

1	4	5	4m 29s	456306	17s 319ms	456306
2	4	5	4m 29s		19s 786ms	
3	4	5	4m 29s		16s 739ms	

Quíntuplas de Artistas que tocaron en N o más Eventos juntos

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	4	1	18hs	NO RESULTS	18hs	NO RESULTS
2	4	1				
3	4	1				
1	4	2	18hs	NO RESULTS	18hs	NO RESULTS
2	4	2				
3	4	2				
1	4	3	18hs	NO RESULTS	18hs	NO RESULTS
2	4	3				
3	4	3				
1	4	4	39m 40s	0	18hs	NO RESULTS
2	4	4				
3	4	4				
1	4	5	45m 40s	0	18hs	NO RESULTS
2	4	5				
3	4	5				

Triplas de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	3	1	15s 865ms	3606482	1m 18s	3606482
2	3	1	15s 438ms		1m 19s	
3	3	1	16s 157ms		1m 9s	
1	3	2	10s 834ms	52360	4s 971ms	52360
2	3	2	10s 726ms		5s 564ms	
3	3	2	10s 659ms		5s 888ms	
1	3	3	11s 10ms	37089	3s 777ms	37089

2	3	3	9s 907ms		3s 405ms	
3	3	3	10s 871ms		3s 736ms	
1	3	4	10s 886ms		3s 320ms	
2	3	4	10s 380ms	33539	6s 633ms	33539
3	3	4	10s 794ms		3s 195ms	
1	3	5	11s 363ms		3s 760ms	
2	3	5	10s 602ms	33226	3s 58ms	33226
3	3	5	10s 766ms		3s 210ms	

Cuádruplas de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	3	1	10m 13s			
2	3	1	9m 52s	117950708		
3	3	1	9m 53s			
1	3	2	5m 39s		32s 160ms	
2	3	2	5m 39s	504429	32s 93ms	504429
3	3	2	5m 50s		32s 599ms	
1	3	3	5m 43s		26s 914ms	
2	3	3	5m 44s	469768	27s 203ms	469768
3	3	3	5m 59s		28s 201ms	
1	3	4	5m 54s		26s 49ms	
2	3	4	5m 49s	456836	25s 471ms	456836
3	3	4	5m 48s		25s 319ms	
1	3	5	5m 49s		25s 945ms	
2	3	5	5m 43s	456306	25s 892ms	456306
3	3	5	5m 33s		26s 697ms	

Quíntuplas de Artistas que tocaron en N o más Eventos juntos y no compartieron Release

	PARAMETROS		SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**

1	3	1	12hs	NO RESULTS		
2	3	1				
3	3	1				
1	3	2	12hs	NO RESULTS		
2	3	2				
3	3	2				
1	3	3	12hs	NO RESULTS		
2	3	3				
3	3	3				
1	3	4	12hs	NO RESULTS		
2	3	4				
3	3	4				
1	3	5	12hs	NO RESULTS		
2	3	5				
3	3	5				

Triplas de Artistas que tocaron en N o más Eventos juntos y compartieron M o más Releases

	PARAMETROS			SQL		NEO4J	
N° Corrida	**Artistas**	**N**	**M**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	3	1	1	17s 257ms	632	1m 19s	632
2	3	1	1	17s 209ms		1m 19s	
1	3	1	2	17s 273ms	62	1m 20s	61
2	3	1	2	17s 250ms		1m 17s	
1	3	1	3	16s 379ms	15	1m 22s	15
2	3	1	3	17s 4m		1m 16s	
1	3	1	4	17s 18ms	8	1m 21s	8
2	3	1	4	17s 30ms		1m 15s	
1	3	1	5	16s 462ms	5	1m 22s	5
2	3	1	5	16s 447ms		1m 17s	
1	3	2	1	11s 255ms	25	4s 886ms	25
2	3	2	1	11s 415ms		5s 273ms	
1	3	2	2	11s 333ms	7	4s 846ms	7
2	3	2	2	10s 918ms		5s 293ms	
1	3	2	3	11s 293ms	4	4s 974ms	4
2	3	2	3	10s 672ms		4s 909ms	
1	3	2	4	11s 280ms	1	5s 059ms	1
2	3	2	4	10s 728ms		4s 862ms	
1	3	2	5	10s 571ms	0	4s 504ms	0

2	3	2	5	11s 360ms		4s 265ms	
1	3	3	1	11s 263ms	7	3s 840ms	7
2	3	3	1	11s 300ms		3s 531ms	
1	3	3	2	10s 854ms	1	3s 853ms	1
2	3	3	2	11s 323ms		3s 236ms	
1	3	3	3	10s 740ms	1	3s 704ms	1
2	3	3	3	10s 747ms		3s 217ms	
1	3	3	4	11s 300ms	0	3s 447ms	0
2	3	3	4	10s 824ms		3s 350ms	
1	3	3	5	10s 521ms	0	4s 114ms	0
2	3	3	5	10s 789ms		3s 364ms	

Cuádruplas de Artistas que tocaron en N o más Eventos juntos y compartieron M o más Releases

	PARAMETROS			**SQL**		**NEO4J**	
N° Corrida	**Artistas**	**N**	**M**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	4	1	1	6m 39s	1233	4hs OUT OF HEAP	NO RESULTS
2	4	1	1	6m 44s			
1	4	1	2	6m 29s	38	4hs OUT OF HEAP	NO RESULTS
2	4	1	2	6m 28s			
1	4	1	3	6m 28s	1	4hs OUT OF HEAP	NO RESULTS
2	4	1	3	6m 28s			
1	4	1	4	6m 28s	0	4hs OUT OF HEAP	NO RESULTS
2	4	1	4	10m 44s			
1	4	1	5	10m 39s	0	4hs OUT OF HEAP	NO RESULTS
2	4	1	5	10m 33s			
1	4	2	1	4m 28s	15	26s 528ms	15
2	4	2	1	4m 2s		27s 243ms	
1	4	2	2	4m 7s	2	27s 233ms	2
2	4	2	2	4m 7s		27s 126ms	
1	4	2	3	4m 13s	1	27s 129ms	1
2	4	2	3	4m 2s		26s 179ms	
1	4	2	4	4m 8s	0	28s 989ms	0
2	4	2	4	4m 2s		28s 945ms	
1	4	2	5	4m 7s	0	27s 911ms	0
2	4	2	5	4m 12s		29s 813ms	
1	4	3	1	4m 7s	1	22s 750ms	1
2	4	3	1	4m 13s		23s 12ms	

1	4	3	2	4m 7s	0	22s 885ms	0
2	4	3	2	4m 13s		23s 491ms	
1	4	3	3	4m 7s	0	23s 293ms	0
2	4	3	3	4m 10s		23s 87ms	
1	4	3	4	4m 2s	0	22s 975ms	0
2	4	3	4	4m 9s		23s 147ms	
1	4	3	5	4m 5s	0	22s 773ms	0
2	4	3	5	4m 2s		22s 850ms	

Quíntuplas de Artistas que tocaron en N o más Eventos juntos y compartieron M o más Releases

	PARAMETROS			**SQL**		**NEO4J**	
N° Corrida	**Artistas**	**N**	**M**	**TIEMPO**	**REGISTROS**	**TIEMPO**	**REGISTROS**
1	5	1	1	18hs	NO RESULTS		
2	5	1	1				
1	5	1	2	18hs	NO RESULTS		
2	5	1	2				
1	5	1	3	18hs	NO RESULTS		
2	5	1	3				
1	5	1	4	18hs	NO RESULTS		
2	5	1	4				
1	5	1	5	18hs	NO RESULTS		
2	5	1	5				
1	5	2	1	18hs	NO RESULTS		
2	5	2	1				
1	5	2	2	18hs	NO RESULTS		
2	5	2	2				
1	5	2	3	18hs	NO RESULTS		
2	5	2	3				
1	5	2	4	18hs	NO RESULTS		
2	5	2	4				
1	5	2	5	18hs	NO RESULTS		
2	5	2	5				
1	5	3	1	18hs	NO RESULTS		
2	5	3	1				

1	5	3	2	18hs	NO RESULTS		
2	5	3	2				
1	5	3	3	18hs	NO RESULTS		
2	5	3	3				
1	5	3	4	18hs	NO RESULTS		
2	5	3	4				
1	5	3	5	18hs	NO RESULTS		
2	5	3	5				