ITESO, Universidad Jesuita de Guadalajara

Minería de Grafos

Proyecto análisis de Netflix

Estudiantes:

- Medina Munguia, Iñaki.
- Sedano Paz, Marian Montserrat
- Velásquez Borda, Mateo Josué.

Profesor:

- Ortega Guzmán, Victor Hugo

Fecha de entrega:

- 24 de Abril de 2025

Índice

Introducción **4**
Comprensión del Proceso Analítico **5**
Identificar objetivos del Negocio 5
Identificar requisitos de información 5
Identificar métricas exitosas 7
Identificar el plan del proyecto 7
Alcance del proyecto 7
Duración del proyecto 8
Entregables acordados 8
Definir preguntas para la toma de decisiones 8
Construcción del Grafo **8**
Selección de formato de almacenamiento de datos 9
Extraer, transformar y cargar datos **10**
Análisis básico exploratorio **17**
Preguntas exploratorias 17
Respuestas a las preguntas exploratorias 18
Esquema general del grafo actualizado 30
Creación de relaciones para Recomendaciones **30**
Creación de Tablero de Recomendaciones básicas **36**
Tablero de recomendaciones 36
Minería de Grafos **44**
Seleccionar algoritmos de minería de Grafos 44
Selección de relaciones 45
Aplicación de algoritmos de centralidad **45**
(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie) 45
(m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie) 51
(m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie) 56
(m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie) 62
(m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie) 67
Aplicación de los algoritmos de comunidad **73**
(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie) 73
(m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie) 80
(m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie) 87
(m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie) 94
(m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie) 101
Aplicación de los algoritmos de similaridad **108**
(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie): 108
(m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie): 109
(m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie): 111
(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie): 113
(m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie): 115
COMBINACIÒN DE RELACIONES: 116
Creación de Tablero de Recomendaciones complejas **119**

Codigo para tablero de recomendaciones Complejas (general) 119
Tablero de recomendaciones Complejas (general) 123
Tablero de recomendaciones Complejas (por relaciones específicas) 124
Conclusiones y recomendaciones: 138
Conclusiones en referencia a los hallazgos: 139
Recomendaciones: 141
Referencias bibliográficas: 141

Introducción

En la actualidad, el crecimiento exponencial de los datos ha impulsado el desarrollo de nuevas formas de almacenamiento y análisis que permitan descubrir patrones y relaciones complejas. En este contexto, las bases de datos basadas en grafos han emergido como una poderosa herramienta para modelar datos altamente interconectados, proporcionando una representación más intuitiva y eficiente que las bases de datos relacionales tradicionales. Su capacidad para gestionar y analizar estructuras de datos complejas hace que sean especialmente valiosas en diversos dominios de aplicación, como los sistemas de información empresarial, los sistemas de detección de delitos financieros, los sistemas de información de transporte y los sistemas de recomendación [1].

Este proyecto tiene como objetivo principal la construcción y análisis de una base de datos de grafos a partir de información extraída de un DUMP que contiene datos sobre la plataforma de Netflix, que inicialmente contiene el siguiente esquema:

En esta situación utilizaremos la metodología propuesta por el artículo “A Methodology for Knowledge Discovery in Labeled and Heterogeneous Graphs (KDG)” [1]. Además se utilizaron dos artículos adicionales: en primer lugar el trabajo recepcional de la Ing. Sarahi Partida Ochoa [2], y el artículo de “Analítica de grafos para identificar entidades relevantes y comunidades en Mercado Libre: un estudio de caso” [3] cómo guía en algunas cuestiones prácticas (En esta situación se utilizó principalmente el estudio 3).

A partir del archivo DUMP decidimos crear un archivo CSV que contenga información de futuros nodos ‘Profile’. En este contexto se crearon 51000 de tipo ADD relaciones aproximadamente con archivos .py en dónde se hizo un top “esperado” de 20 películas/series de nuestra elección para que sean notorias. Además se transformarán los datos en un modelo gráfico que permitirá aplicar algoritmos de análisis de redes. Con esto, se busca identificar patrones clave en la interacción entre las pelculas, determinando su nivel de influencia y agrupando comunidades en función de su conectividad dentro del grafo.

Objetivos secundarios:

- Construir una base de datos basada en grafos a partir de los archivos CSV generados con Python y con los datos cargados previamente en DUMP
- Elaborar archivos CSV de los Profiles.
- Generar relaciones ADD entre los Profiles y Movies, mediante funciones en Python.

- Aplicar algoritmos de centralidad para identificar las películas más influyentes dentro de la red de la plataforma Netflix.
- Implementar técnicas de detección de comunidades para agrupar películas y analizar sus interacciones dentro del grafo.
- Visualizar la estructura y las relaciones de los datos mediante herramientas especializadas en análisis de grafos.

El dominio de los datos utilizados en este estudio provienen de un archivo DUMP de la plataforma Netflix. Estos datos permiten construir una red de relaciones que refleje la dinámica social dentro de la plataforma de Netflix, proporcionando una base sólida para el análisis posterior y para generar futuras recomendaciones.

Comprensión del Proceso Analítico

Identificar objetivos del Negocio

En este apartado logramos identificar los siguientes objetivos del negocio:

- Mejorar la experiencia del usuario mediante recomendaciones personalizadas que aumenten la satisfacción y el tiempo de permanencia en la plataforma.
- Aumentar el consumo de contenido por parte de los usuarios al ofrecer películas relevantes basadas en intereses reales y patrones comunitarios de visualización.
- Optimizar el catálogo de contenido identificando películas infrautilizadas pero con alto potencial de recomendación según su centralidad en la red.
- Segmentar la audiencia de forma más precisa a través de la detección de comunidades con intereses comunes, permitiendo campañas de marketing o lanzamientos de contenido más dirigidos.
- Reducir la tasa de abandono de usuarios (churn) mediante sistemas de recomendación más efectivos que mantengan al usuario enganchado con el contenido.

Identificar requisitos de información

El presente proyecto de investigación tiene como finalidad el diseño e implementación de un sistema de recomendación de películas utilizando técnicas de minería de grafos. En este contexto, es fundamental definir de manera precisa, concreta y factible los requisitos de información necesarios para lograr los objetivos propuestos. Esta sección presenta dichos requisitos, siguiendo los principios de especificidad, viabilidad técnica, relevancia y orientación a resultados medibles dentro de un marco temporal definido.

- **Modelo de Datos**: La base de datos está estructurada en torno a un modelo orientado a grafos, almacenado en Neo4j, que incluye los siguientes nodos y relaciones:
    - Nodos:
        - (:Movie) – Representa películas o series.
        - (:Profile) – Representa perfiles de usuario.
        - (:Person) – Representa actores y directores.

        - (:Category) – Representa los géneros o etiquetas de contenido.
        - (:Country) – Representa el país de origen de la película o de la persona.
    - Relaciones:
        - (:Person)-[:ACTED_IN]->(:Movie)
        - (:Person)-[:DIRECTED]->(:Movie)
        - (:Person)-[:IS_CITIZEN_OF]->(:Country)
        - (:Movie)-[:FROM]->(:Country)
        - (:Movie)-[:LISTED_IN]->(:Category)
        - (:Profile)-[:ADD]->(:Movie)
- **Requisitos Funcionales**: Se requiere que la base de datos cumpla con las siguientes condiciones para ser considerada adecuada para el análisis:
    - Existencia de una cantidad significativa de relaciones entre perfiles y películas (:ADD) que permita inferir preferencias y similitudes entre usuarios.
    - Presencia de actores y directores que participen en múltiples películas, favoreciendo la conectividad del grafo.
    - Asignación de múltiples categorías por película, con el fin de identificar clústeres temáticos.
- **Tecnologías y herramientas utilizadas**: El entorno tecnológico seleccionado permite cumplir con los requisitos analíticos y computacionales del proyecto:
    - Neo4j (VERSIÓN 5.25.1 [De la base de datos]): como sistema gestor de base de datos orientado a grafos.
    - Cypher: como lenguaje de consulta principal.
    - Neo4j Graph Data Science (GDS), como biblioteca analítica para ejecutar los algoritmos de análisis de grafos.
    - APOC, para operaciones avanzadas sobre el grafo.
    - Neo4j Browser: Interfaz gráfica para la carga, consulta y manipulación de los datos dentro de la base de datos.
    - Neo4j Bloom: Herramienta de visualización avanzada para explorar la estructura del grafo de manera interactiva.
    - Neo4j NeoDash: Plataforma para la creación de dashboards que facilitarán la visualización y el análisis de los datos.
    - Gephi: Software de análisis de redes que complementará la visualización y exploración de la estructura del grafo.
- **Algoritmos Requeridos**:
    - Algoritmos de Centralidad:
        - Degree Centrality: Para evaluar la cantidad de conexiones directas de un nodo.
        - Closeness Centrality: Para estimar la eficiencia de acceso a otros nodos desde uno dado.
        - Betweenness Centrality: Para detectar nodos que actúan como puentes entre comunidades.
        - PageRank: Para identificar nodos relevantes por su influencia estructural.
    - Algoritmos de Detección de Comunidades:
        - Louvain Modularity: Para identificar agrupamientos de nodos densamente conectados.

- Label Propagation: Para clasificar nodos en comunidades de forma no supervisada.
- Triangle Count – Para cuantificar la cantidad de triángulos en los que participa un nodo, útil para medir cohesión local.
- Strongly Connected Components (SCC): Para identificar subgrafos fuertemente conectados.
- Weakly Connected Components (WCC): Para identificar grupos conectados sin dirección.
- K1-coloring: Para estudiar la posibilidad de particionar nodos en grupos mutuamente excluyentes.

Identificar métricas exitosas

Dado que el proyecto se enfoca en la exploración de algoritmos de centralidad y detección de comunidades aplicados sobre una base de datos de películas y series de Netflix, es crucial establecer métricas claras para evaluar la calidad y validez de los patrones descubiertos. A continuación, se detallan las métricas de éxito que guiarán la evaluación del progreso y los resultados obtenidos a lo largo del desarrollo del proyecto:

Métrica	Criterio de éxito estimado
Número de nodos Profile	> 1000 nodos de tipo Profile
Número de relaciones ADD	> 10000 relaciones de tipo ADD
Cantidad de comunidades encontradas con Label Propagation	< 20 comunidades detectadas (comunidad razonable)
Cantidad de comunidades encontradas con k1-coloring	> 10 comunidades detectadas
Cantidad de comunidades encontradas con WCC (Weakly Connected Components)	< 5 comunidades detectadas

Identificar el plan del proyecto

Este será el documento del proyecto. Por consiguiente, definimos los siguientes puntos:

Alcance del proyecto

Este proyecto está diseñado para explorar la estructura de grafos de Netflix y generar recomendaciones basadas en la detección de comunidades y centralidad. El alcance está claramente delimitado a un análisis exploratorio y no abarca la creación de sistemas de recomendación comerciales o el uso de modelos predictivos complejos. Las métricas y algoritmos seleccionados guiarán el análisis, y los resultados se usarán para obtener patrones significativos dentro del grafo de Netflix, sin extenderse a aspectos como preferencias de usuario individuales o la integración de fuentes de datos externas.

Duración del proyecto

La duración estimada de este proyecto abarca desde las 13:00 del jueves 7 de abril hasta las 13:00 del martes 24 de abril. Lo que nos deja un aproximado de 17 días.

Entregables acordados

Para esta actividad se acordaron los siguientes entregables:

- Archivo DUMP de la base de datos.
- Archivo PDF que sirva como un informe del proyecto (este documento).
- Archivo JSON del tablero creado en NeoDash
- Archivos CQL utilizado para escribir el código necesario para cada una de las consultas a la base de datos.
- Archivos PY necesarios para crear los CSV para los Profiles, y además las relaciones ADD.
- Archivos CSV utilizados para crear la base de datos.
- Presentación de power point para difundir y compartir los descubrimientos, resultados y conclusiones obtenidos en el proyecto.

Definir preguntas para la toma de decisiones

El objetivo principal de la elaboración de estas preguntas es facilitar la recopilación de información relevante, evaluar las opciones disponibles y considerar los factores que influyen en la toma de decisiones bien informada y efectiva. [1]

En este apartado, se presentan una serie de preguntas de interés general que serán abordadas a lo largo de todo el proyecto de investigación:

- ¿Cuáles son las 5 películas/series más vistas y que otras recomendaremos a los usuarios para que sigan en la plataforma?
- ¿Cómo se podrían relacionar las películas entre sí?
- ¿Se pueden identificar sagas con algún algoritmo de centralidad?
- En caso de identificar sagas ¿Podríamos recomendar otras sagas?
- ¿Cuáles son las posibles recomendaciones entre películas y/o series de la misma categoría?
- ¿Cómo afecta la nacionalidad de las películas a las recomendaciones?
- ¿Es buena idea recomendar películas/series si son del mismo director? ¿Y si lo pensamos con actores?

Construcción del Grafo

Ahora basándonos en la metodología previamente establecida deberíamos seguir con los siguientes puntos: (2.1: Análisis exploratorio de datos) que implica realizar un análisis exploratorio de la información, y (2.2: Diseño de modelo de grafos) en dónde formulamos el modelo de grafos [1]. Sin embargo, la información del DUMP que nos brindaron no fue suficiente, por lo que fue necesario crear Perfiles utilizando Python y metiendolos en un archivo de profile.csv, luego de eso fue necesario nuevamente con Python crear las

relaciones ADD entre los Profile y las Movie. Por consiguiente es innecesario realizar los puntos 2.1 y 2.2 de la metodología. Así que continuamos directamente con el punto (2.3).

Siguiendo la metodología propuesta “A Methodology for Knowledge Discovery in Labeled and Heterogeneous Graphs”, este capítulo tiene como propósito abordar las fases iniciales del análisis del grafo, específicamente las etapas (2.1) Análisis exploratorio de datos y (2.2) Diseño del modelo de grafos. No obstante, debido a las particularidades del conjunto de datos proporcionado para este proyecto, se establecieron excepciones justificadas que se detallan a continuación:

- El archivo DUMP entregado como base para esta investigación presentaba limitaciones significativas en cuanto a la información de perfiles de usuarios y sus interacciones con el contenido audiovisual de la plataforma. En concreto, no se incluían nodos de tipo Profile, ni relaciones del tipo ADD que representaran la interacción entre usuarios y películas. Esto imposibilitó realizar un análisis exploratorio significativo (2.1) sobre dichos nodos, así como diseñar un modelo de grafo completo (2.2) exclusivamente a partir de la fuente de datos original.
- Frente a esta situación, se optó por una estrategia alternativa de construcción de datos complementarios, con el objetivo de garantizar la viabilidad del análisis posterior centrado en algoritmos de centralidad y detección de comunidades:
    - Se generaron perfiles de usuario sintéticos utilizando Python, modelados de forma que reflejen interacciones verosímiles con las películas disponibles en el conjunto de datos.
    - Se establecieron relaciones (:Profile)-[:ADD]->(:Movie), también mediante procesamiento en Python, simulando los patrones de consumo de contenido por parte de los usuarios. Estas relaciones fueron exportadas a archivos .csv e importadas en Neo4j para integrarse con el resto del modelo.
- Gracias a estas acciones, el grafo resultante contiene una estructura suficientemente rica y coherente como para permitir la aplicación de métricas topológicas avanzadas y técnicas de análisis estructural.

Selección de formato de almacenamiento de datos

Dado el carácter heterogéneo y altamente relacional del conjunto de datos, que involucra entidades como: Movie, Person, Profile, Country y Category. Es que se optó por emplear Neo4j como plataforma de almacenamiento, gestión y consulta del grafo.

Neo4j es una base de datos orientada a grafos que ofrece múltiples ventajas clave para el presente proyecto:

- Modelo de datos natural: Permite representar entidades como nodos y relaciones explícitas, facilitando la semántica del dominio.
- Lenguaje Cypher: Brinda una sintaxis declarativa específica para grafos, permitiendo expresar con claridad algoritmos de análisis como centralidad, detección de comunidades y patrones de conectividad.
- Alto rendimiento en consultas complejas: Es capaz de ejecutar búsquedas y métricas topológicas de forma eficiente incluso en grafos con decenas de miles de nodos y relaciones.
- Integración con librerías analíticas: La herramienta Neo4j Graph Data Science (GDS) permite ejecutar algoritmos como Degree, PageRank, Louvain Modularity, Triangle

Count, Label Propagation, Strongly y Weakly Connected Components, entre otros, de manera nativa sobre el grafo almacenado.

- Importación desde archivos CSV: Se integraron nodos Profile y relaciones ADD a partir de archivos .csv previamente generados con Python, lo cual resultó compatible con el sistema de importación por lotes de Neo4j.

En este contexto, Neo4j no sólo cumple con los requisitos técnicos del proyecto, sino que constituye un componente estratégico fundamental, al permitir la ejecución eficiente de las tareas analíticas requeridas por los objetivos de investigación, tales como el análisis de comunidades, detección de nodos clave mediante centralidad, y evaluación de la estructura global de la red.

Por tanto, se concluye que Neo4j representa el formato de almacenamiento óptimo para este proyecto, en términos de representación, rendimiento, expresividad y escalabilidad.

Extraer, transformar y cargar datos

La etapa de Extracción, Transformación y Carga de Datos (ETL) representa un proceso fundamental dentro del pipeline de integración de datos para grafos heterogéneos, tal como se establece en la metodología [1]. Esta fase tiene como objetivo adquirir, limpiar, estructurar y cargar la información proveniente de múltiples fuentes hacia el sistema de almacenamiento previamente seleccionado (en este caso, Neo4j), asegurando así la consistencia y compatibilidad semántica del modelo de grafo.

Las etapas realizadas fueron las siguientes:

1. **Extracción de datos**: Los datos fueron extraídos de un DUMP inicial incompleto correspondiente al catálogo de películas y series de Netflix. Este DUMP contenía información parcial sobre entidades como películas (Movie), personas (Person), países (Country) y categorías (Category). Sin embargo, se identificó una ausencia crítica de datos relativos a perfiles (Profile), los cuales son imprescindibles para la posterior generación de recomendaciones.
    a. Para suplir esta carencia, se procedió a la generación sintética de nodos Profile utilizando scripts en Python, asegurando un volumen de 5000 perfiles, para facilitar el análisis estadístico posterior.
    b. Asimismo, se crearon de manera programática aproximadamente 90.000 relaciones ADD, que vinculan a los perfiles con las películas vistas, simulando interacciones realistas de consumo.
2. **Transformación de datos**: Durante esta fase se realizó un proceso de limpieza, estandarización y adecuación semántica de los datos, con el fin de garantizar su correcta integración en el modelo de grafo. Las transformaciones incluyeron:
    a. Se creó un archivo Python dónde se dió una lista de 415 nombres y 140 apellidos, en dónde se concatenaron de forma aleatoria hasta que generaron 5000 nombres para Profile.
    b. Formateo de identificadores únicos (id) para los nodos Profile y Movie.
3. **Carga en Neo4j**: La carga de datos se efectuó utilizando:
    a. El comando LOAD CSV de Cypher, propio de Neo4j, lo cual permitió una incorporación eficiente de nodos y relaciones en la base de datos orientada a grafos. Este procedimiento facilitó la correcta estructuración de las entidades y vínculos, respetando el modelo conceptual previamente definido.
    b. Generando la base de datos a partir del DUMP inicial.

c. Corriendo el Archivo Python que generó las relaciones ADD.

Código empleado:

Generar perfiles:

```
import csv
import random

# Lista de Nombres:
nombres = [
        "Agustín", "Agustina", "Alejandro", "Alejandra", "Alfredo",
"Alicia", "Alan", "Alma", "Andrés", "Andrea",
     "Antonio", "Ana", "Axel", "Abril", "Ariel", "Ariana", "Alberto",
"Antonella", "Abel", "Ailén",
        "Alonso", "Aitana", "Amador", "Amalia", "Adriel", "Amanda",
"Aarón", "Anahí", "Américo", "Ayelén",
       "Asier", "Aurora", "Arón", "Abigail", "Arnaldo", "Anastasia",
"Aldo", "Araceli", "Adrián", "Alina",
       "Benjamín", "Brenda", "Bruno", "Bianca", "Bautista", "Belén",
"Braian", "Bárbara", "Boris", "Beatriz",
      "Blas", "Brisa", "Bastián", "Berenice", "Baltasar", "Betsabé",
"Bernardo", "Bernarda", "Bladimir", "Bonnie",
     "Basilio", "Bettina", "Braulio", "Brigitte", "Bayron", "Betania",
"Benicio", "Bellatrix", "Bartolomé", "Bricia",
      "Beltrán", "Bilha", "Branko", "Blanda", "Benilda", "Bismarck",
"Bibiana", "Boanerges", "Baya", "Bashir",
       "Camila", "Carlos", "Carla", "Cristian", "Catalina", "César",
"Celeste", "Claudio", "Clara", "Ciro",
     "Cecilia", "Cristóbal", "Carmen", "Caleb", "Constanza", "Conrado",
"Cintia", "Cirilo", "Claribel", "Cornelio",
      "Coral", "Cipriano", "Chloe", "Crisanto", "Candela", "Casandra",
"Cayetano", "Carlota", "Candelaria", "Cloe",
     "Diego", "Daniela", "David", "Delfina", "Darío", "Dana", "Damián",
"Diana", "Denis", "Daiana",
         "Dante", "Dulce", "Dylan", "Deborah", "Dilan", "Damaris",
"Domingo", "Doris", "Darian", "Dominika",
         "Demian", "Dunia", "Denzel", "Dalma", "Donovan", "Dasha",
"Desiderio", "Dafne", "Donato", "Dalila",
      "Elías", "Elena", "Emanuel", "Estefanía", "Eduardo", "Emilia",
"Ezequiel", "Eva", "Erik", "Ema",
       "Esteban", "Eloísa", "Enzo", "Esmeralda", "Edgar", "Eugenia",
"Efraín", "Elvira", "Emiliano", "Elsa",
         "Federico", "Florencia", "Facundo", "Fátima", "Fernando",
"Fiorella", "Fabio", "Fernanda", "Flavio", "Fabiana",
```

```
      "Félix", "Francisca", "Fabián", "Frida", "Faustino", "Gabriel",
"Gabriela", "Guillermo", "Guillermina", "Germán",
      "Graciela", "Genaro", "Gloria", "Gastón", "Georgina", "Gonzalo",
"Gema", "Gervasio", "Giuliana", "Gregorio",
       "Héctor", "Helena", "Hugo", "Hortensia", "Horacio", "Haydée",
"Hernán", "Helena", "Hermes", "Harmony",
      "Ignacio", "Isabel", "Iván", "Ingrid", "Ian", "Irina", "Iker",
"Ileana", "Ismael", "Ivana",
     "Imanol", "Iris", "Isaías", "Indira", "Italo", "Juan", "Julieta",
"Joaquín", "Jimena", "Jorge",
      "Jazmín", "Jeremías", "Juliana", "Javier", "Johanna", "Josué",
"Josefina", "Julián", "Juana", "Jael",
     "Kevin", "Karen", "Katherina", "Kenia", "Khalil", "Luis", "Laura",
"Leonardo", "Lucia", "Lorenzo",
    "Lara", "Lucas", "Lourdes", "Leandro", "Liliana", "Lázaro", "Lola",
"Luciano", "Lía", "Luisana",
       "Leonardo", "Lila", "Lionel", "Leticia", "Lucas", "Lorenza",
"Lisandro", "Leila", "Lucho", "Libia",
     "Lino", "Laurana", "Luisa", "Lázara", "Luisito", "María", "Mateo",
"Martín", "Mariana", "Mara",
      "Micaela", "Mario", "Marcela", "Mauricio", "Melissa", "Melanie",
"Maxima", "Manuel", "Margarita", "Milton",
       "Mirella", "Marcos", "Mónica", "Miguel", "Marisol", "Matías",
"Melisa", "Moisés", "Mila", "Mauro",
       "Nicolás", "Noelia", "Néstor", "Natalia", "Nacho", "Norma",
"Nahuel", "Nadia", "Nicanor", "Nina",
    "Omar", "Oliver", "Oscar", "Olivia", "Oriana", "Octavio", "Onelia",
"Osvaldo", "Otilia", "Orlando",
     "Pablo", "Patricia", "Pedro", "Paula", "Piero", "Pilar", "Paloma",
"Priscila", "Pascual", "Penélope",
    "Patricio", "Pabla", "Pablo", "Plácido", "Paola", "Pompeyo", "Pía",
"Pablina", "Prudencia", "Primo",
      "Pancho", "Pilar", "Placidia", "Pancracio", "Paolo", "Quiana",
"Quentin", "Quetzal", "Queralt", "Quirina",
       "Roberto", "Raquel", "Ricardo", "Rosa", "Ramiro", "Renata",
"Rafael", "Regina", "Rodrigo", "Rocío",
     "Rocco", "Raúl", "Rubén", "Remedios", "Renzo", "Rita", "Rolando",
"Rebeca", "Romeo", "Rosaura",
      "Sebastián", "Sofía", "Samuel", "Sabrina", "Sergio", "Silvia",
"Santiago", "Sara", "Salvatore", "Selene",
       "Simón", "Susana", "Sebastián", "Stella", "Santos", "Sonia",
"Saúl", "Sol", "Santiago", "Sarah",
```

```
    "Tomás", "Teresa", "Thiago", "Tatiana", "Tadeo", "Tania", "Tobías",
"Tamara", "Teodoro", "Trini",
    "Ulises", "Úrsula", "Ubaldo", "Uriel", "Uma", "Vicente", "Valeria",
"Víctor", "Vanessa", "Valentino",
        "Violeta",  "Verónica",  "Valentín",  "Virginia",  "Vladimiro",
"William", "Wendy", "Walter", "Waldo", "Ximena",
    "Yasmín", "Yago", "Yolanda", "Zoe", "Zacarías"
]

# Lista de apellidos
apellidos = [
    'Almeida', 'Anziani', 'Arnaudo', 'Ayala', 'Barrionuevo', 'Benítez',
'Bermúdez', 'Bonelli', 'Borda', 'Cabral',
         'Cáceres',  'Calderón',  'Camolotto',  'Cantero',  'Cardozo',
'Carrizo', 'Castillo', 'Castro', 'Cerda', 'Cervantes',
     'Chávez', 'Cisneros', 'Contreras', 'Cornejo', 'Crespo', 'Cuevas',
'Del Valle', 'Delgado', 'Díaz', 'Domínguez',
         'Escobar',  'Esperanza',  'Espinosa',  'Figueroa',  'Fonseca',
'Franco', 'Gaitán', 'Gallardo', 'Galván', 'García',
      'Gómez', 'González', 'Godoy', 'Grey', 'Gutiérrez', 'Hernández',
'Ibarra', 'Iñíguez', 'Juárez', 'Lagos',
       'Lannister',  'Lara',  'Ledesma',  'Leiva',  'Lencina',  'Lépore',
'Lopez', 'Luque', 'Luna', 'Maciel',
      'Magni', 'Malvarez', 'Marín', 'Martínez', 'Maturana', 'Medina',
'Méndez', 'Mercado', 'Messi', 'Micolini',
      'Molina', 'Monzón', 'Montenegro', 'Morán', 'Moreno', 'Munguia',
'Nardón', 'Nieto', 'Nicolás', 'Ojeda',
       'Oliva',  'Ortega',  'Ortiz',  'Palacios',  'Paniagua',  'Parra',
'Peñalosa', 'Perotti', 'Pérez', 'Pizarro',
      'Quinteros', 'Quival', 'Quiroga', 'Ramírez', 'Ramos', 'Reyna',
'Reynoso', 'Rico', 'Riera', 'Rivero',
    'Rodríguez', 'Rojas', 'Rojo', 'Russo', 'Sad', 'Salazar', 'Salinas',
'Sánchez', 'Santamaría', 'Santos',
       'Schmitt',  'Sedano',  'Silva',  'Stoffel',  'Suárez',  'Tapia',
'Toledo', 'Torres', 'Trejo', 'Valdez',
     'Valenzuela', 'Vásquez', 'Vega', 'Velásquez', 'Vera', 'Vergara',
'Villalba', 'Zambrano', 'Zarate', 'Zitto',
       'Acosta',  'Aguilar',  'Alarcón',  'Alfaro',  'Amador',  'Amaya',
'Andrada', 'Aparicio', 'Arce', 'Arias',
       'Avendaño',  'Azcurra',  'Báez',  'Becerra',  'Bello',  'Blanco',
'Bogado', 'Borja', 'Brizuela', 'Bustos',
     'Caballero', 'Campos', 'Cano', 'Carballo', 'Carrera', 'Carvajal',
'Casas', 'Catalán', 'Cazares', 'Celis',
```

```
      'Centeno', 'Cervini', 'Chacón', 'Chaparro', 'Chirino', 'Colina',
'Colombo', 'Comas', 'Concha', 'Cordero',
        'Coria', 'Cornejo', 'Cortés', 'Cuello', 'Cuevas', 'Curcio',
'Dávalos', 'De la Fuente', 'De la Vega', 'Delgado',
      'Díaz', 'Duarte', 'Durán', 'Enríquez', 'Escalante', 'Esquivel',
'Estévez', 'Farías', 'Fernández', 'Ferreyra',
       'Figueroa', 'Flores', 'Fuentes', 'Gallego', 'Gallo', 'Gálvez',
'García', 'Garnica', 'Giménez', 'Gómez',
           'Gonzaga', 'González', 'Guerrero', 'Guillén', 'Guzmán',
'Hernández', 'Herrera', 'Ibáñez', 'Infante', 'Insaurralde',
        'Iribarren', 'Jaimes', 'Jara', 'Jofré', 'Lagos', 'Lamadrid',
'Lara', 'León', 'Llamas', 'Lobo',
      'Lozano', 'Lucero', 'Luján', 'Macías', 'Maldonado', 'Manríquez',
'Marchese', 'Martel', 'Martínez', 'Massa',
      'Matías', 'Medrano', 'Mejía', 'Melgarejo', 'Merino', 'Miranda',
'Molina', 'Monje', 'Morales', 'Moreira',
        'Moyano', 'Muñoz', 'Mussi', 'Navarrete', 'Navarro', 'Núñez',
'Obando', 'Obregón', 'Olguín', 'Olivera',
       'Orellana', 'Orlando', 'Ortíz', 'Otero', 'Pacheco', 'Padilla',
'Páez', 'Palma', 'Pardo', 'Paredes',
         'Parodi', 'Pastor', 'Peña', 'Perales', 'Pérez', 'Pineda',
'Portillo', 'Posadas', 'Prieto', 'Puente',
        'Puga', 'Quevedo', 'Quiñones', 'Ramos', 'Rangel', 'Ravelo',
'Redondo', 'Reyes', 'Riquelme', 'Rivera',
        'Robledo', 'Roca', 'Rodríguez', 'Roldán', 'Romero', 'Rosas',
'Ruiz', 'Salguero', 'Salinas', 'Salvatierra',
      'Sánchez', 'Sandoval', 'Sanhueza', 'Santana', 'Santi', 'Santos',
'Saravia', 'Saucedo', 'Serrano', 'Sibila',
     'Silva', 'Solano', 'Sosa', 'Suárez', 'Taborda', 'Tapia', 'Tejada',
'Toledo', 'Torrico', 'Torres',
      'Trejo', 'Trujillo', 'Ulloa', 'Valdés', 'Valencia', 'Valiente',
'Varela', 'Vargas', 'Vázquez', 'Vega',
      'Velázquez', 'Vera', 'Vergara', 'Vidal', 'Villagra', 'Villalba',
'Villanueva', 'Villarreal', 'Vives', 'Zalazar'
]



# Generar CSV con 2000 perfiles
def generar_csv_con_nombres_y_apellidos(output_file, cantidad):
     with open(output_file, mode='w', newline='', encoding='utf-8') as
file:
        writer = csv.writer(file)
```

```
                writer.writerow(['id_profile',  'nombre_completo'])    #
Encabezados

        for i in range(1, cantidad + 1):
            nombre = random.choice(nombres)
            apellido = random.choice(apellidos)
            nombre_completo = f"{nombre} {apellido}"
            writer.writerow([i, nombre_completo])

    print(f"Archivo generado exitosamente con {cantidad} perfiles.")

# Ejecutar la función
generar_csv_con_nombres_y_apellidos('profiles.csv', 5000)
```

Cargar perfiles:

```
// Agregamos los Profiles creados en el CSV 'profiles.csv'
LOAD CSV WITH HEADERS FROM 'file:///profiles.csv' AS row
CREATE (:Profile{
    id: row.id_profile,
    name: row.nombre_completo
});
```

Generar relaciones:

```
from neo4j import GraphDatabase
import random
from dotenv import load_dotenv # Nos va a permitir leer las variables
de entorno
import os

# Comado del PipInstall:
# python -m pip install neo4j

# Cargar las variables de entorno
load_dotenv()

# Aca simplemente pasamos los datos de las variables de entorno
uri = os.getenv('uri')
username = os.getenv('usernameNeo4j')
password = os.getenv('password')
driver = GraphDatabase.driver(uri, auth=(username, password))

# Definir los IDs de los perfiles y de las películas
```

```
perfil_ids = range(1,5001) # El id de los perfiles va desde el 1 al
5000 inclusive
pelicula_ids = [f's{num}' for num in range(1, 8808)] # El id de las
películas va desde el 1 al 8807 inclusive

IDPeliselegidas = [28, 33, 34, 83, 173, 312, 339, 391, 452, 460,
                   527, 575, 602, 603, 604, 632, 684, 779, 856, 936,
                   1013, 1139, 1307, 1333, 1386, 1407, 1449, 1571,
1617, 1879,
                   1996, 2011, 2057, 2418, 2583, 3097, 3775, 4814,
6201, 6326,
                   7009, 7443, 7444, 7448, 8068, 8069, 8312, 8342,
8581, 8617]

pelicula_ids_Potentes = [f's{num}' for num in IDPeliselegidas]

pelicula_ids_NoPotentes = list(set(pelicula_ids) -
set(pelicula_ids_Potentes))

# Crear relaciones entre perfiles y películas
def crear_relaciones(session, perfil_ids, pelicula_ids,
max_relaciones):
    contador = 0  # Inicializamos el contador

    while contador < max_relaciones:
        # Seleccionamos un perfil y una película aleatoriamente
        perfil_id = random.choice(perfil_ids)
        pelicula_id = random.choice(pelicula_ids)

        # Creamos la consulta para la relación entre perfil y película
        query = (
            f"MERGE (p:Profile {{id:'{perfil_id}'}}) "
            f"MERGE (m:Movie {{id:'{pelicula_id}'}}) "
            "MERGE (p)-[:ADD]->(m)"
        )

        # Ejecutamos la consulta
        # print(f"Creando perfil {perfil_id} relación :ADD y película
{pelicula_id}")
        session.run(query)

        # Incrementamos el contador
        contador += 1
```

```
# Ejecutar el bucle para crear relaciones
with driver.session() as session:
    crear_relaciones(session, perfil_ids, pelicula_ids_NoPotentes,
36000) #36814
    crear_relaciones(session, perfil_ids, pelicula_ids_Potentes, 55000)
    # Total: 85271

# Cerrar la conexión al terminar
driver.close()
```

A través de este proceso de ETL, se logró materializar con éxito el grafo heterogéneo sobre el cual se aplicarán los algoritmos de análisis de grafos. Este grafo consolidado representa no sólo relaciones de colaboración artística (actores, directores), sino también comportamientos de visualización simulados por los perfiles, lo cual habilita el estudio profundo de patrones de comunidad, centralidad y afinidad estructural, fundamentales para el sistema de recomendación propuesto.

Análisis básico exploratorio

Dado que el presente proyecto tiene como objetivo principal generar un sistema de recomendaciones de películas basado en algoritmos de centralidad y detección de comunidades en grafos, y considerando que los datos disponibles provienen de un DUMP parcial de origen desconocido junto con un conjunto sintético de 5.000 perfiles generados aleatoriamente, resulta indispensable formular un conjunto de preguntas que orienten el análisis estructural y funcional de la base de datos basada en grafos.

Estas preguntas no solo permiten comprender la morfología y conectividad de la red, sino que también su propósito es fomentar una mirada reflexiva, con foco en los objetivos del sistema de recomendación, y documentar los hallazgos relevantes obtenidos durante la fase de exploración y análisis del grafo.

Preguntas exploratorias

Las preguntas exploratorias planteadas son las siguientes:

1. ¿Cómo es el esquema general del grafo?
2. ¿Cuáles tipos de nodos existen y qué atributos tienen?
3. ¿Cuántos nodos de tipo Movie existen?
4. ¿Cuántos nodos de tipo Person existen?
5. ¿Cuántos nodos de tipo Profile existen?
6. ¿Cuántos nodos de tipo Country existen?
7. ¿Cuántos nodos de tipo Category existen?
8. ¿Cuáles tipos de relaciones existen y qué atributos tienen?
9. ¿Cuántas relaciones "ACTED_IN" existen?
10. ¿Cuántas relaciones "DIRECTED" existen?
11. ¿Cuántas relaciones "IS_CITIZEN_OF" existen?

12. ¿Cuántas relaciones "FROM" existen?
13. ¿Cuántas relaciones "LISTED_IN" existen?
14. ¿Cuántas relaciones "ADD" existen?
15. ¿Cuál es el diámetro del grafo?
16. ¿Cuál es la densidad del grafo?
17. ¿Cuáles son las 50 películas más añadidas a los perfiles?
18. ¿Existen películas que nunca fueron añadidas a los perfiles?
19. ¿Quiénes son los 20 actores más famosos?
20. ¿Quiénes son los 10 directores a los que más películas han generado un ADD?
21. ¿Qué 5 países tienen más actores?
22. ¿De qué país son las 50 películas más añadidas?
23. ¿Existen perfiles que nunca hicieron ADD a una serie/película?
24. ¿Cuáles son las 5 categorías favoritas de la gente?
25. ¿Quiénes son los 10 actores que actuaron en más películas?

Respuestas a las preguntas exploratorias

Las respuestas a las preguntas exploratorias planteadas son las siguientes:

1. **¿Cómo es el esquema general del grafo?**

```
CALL db.schema.visualization;
```

2. **¿Cuáles tipos de nodos existen y qué atributos tienen?**

```
CALL db.schema.nodeTypeProperties;
```

```
neo4j$ CALL db.schema.nodeTypeProperties;
```

nodeType	nodeLabels	propertyName	propertyTypes	mandatory
":`Person`"	["Person"]	"name"	["String"]	true
":`Country`"	["Country"]	"name"	["String"]	true
":`Country`"	["Country"]	"point"	["Point"]	false
":`Category`"	["Category"]	"name"	["String"]	true
":`Profile`"	["Profile"]	"id"	["String"]	true
":`Profile`"	["Profile"]	"name"	["String"]	true
":`Movie`"	["Movie"]	"id"	["String"]	true
":`Movie`"	["Movie"]	"dateAdded"	["String"]	false
":`Movie`"	["Movie"]	"releaseYear"	["Long"]	true
":`Movie`"	["Movie"]	"title"	["String"]	true

3. **¿Cuántos nodos de tipo Movie existen?**

```
MATCH (m:Movie)
RETURN COUNT(m) AS Total_Nodos_Movie;
```

```
1 MATCH (m:Movie)
2 RETURN COUNT(m) AS Total_Nodos_Movie;
```

	Total_Nodos_Movie
1	8807

4. **¿Cuántos nodos de tipo Person existen?**

```
MATCH (p:Person)
RETURN COUNT(p) AS Total_Nodos_Person;
```

```
1 MATCH (p:Person)
2 RETURN COUNT(p) AS Total_Nodos_Person;
```

	Total_Nodos_Person
1	40948

5. **¿Cuántos nodos de tipo Profile existen?**

```
MATCH (p:Profile)
RETURN COUNT(p) AS Total_Nodos_Profile;
```

```
1 MATCH (p:Profile)
2 RETURN COUNT(p) AS Total_Nodos_Profile;
```

Table / Text

	Total_Nodos_Profile
1	5000

6. **¿Cuántos nodos de tipo Country existen?**

```
MATCH (c:Country)
RETURN COUNT(c) AS Total_Nodos_Country;
```

```
1 MATCH (c:Country)
2 RETURN COUNT(c) AS Total_Nodos_Country;
```

Table / Text

	Total_Nodos_Country
1	215

7. **¿Cuántos nodos de tipo Category existen?**

```
MATCH (c:Category)
RETURN COUNT(c) AS Total_Nodos_Category;
```

```
1 MATCH (c:Category)
2 RETURN COUNT(c) AS Total_Nodos_Category;
```

Table / Text

	Total_Nodos_Category
1	42

8. ¿Cuáles tipos de relaciones existen y qué atributos tienen?

```
CALL db.schema.relTypeProperties;
```

```
neo4j$ CALL db.schema.relTypeProperties;
```

Table / Text / Code

relType	propertyName	propertyTypes	mandatory
":`FROM`"	null	null	false
":`LISTED_IN`"	null	null	false
":`DIRECTED`"	null	null	false
":`IS_CITIZEN_OF`"	null	null	false
":`ACTED_IN`"	null	null	false
":`ADD`"	null	null	false

9. ¿Cuántas relaciones "ACTED_IN" existen?

```
MATCH ()-[r:ACTED_IN]->()
RETURN COUNT(r) AS Total_Relaciones_ACTED_IN;
```

```
1 MATCH ()-[r:ACTED_IN]→()
2 RETURN COUNT(r) AS Total_Relaciones;
```

	Total_Relaciones
1	64124

10. ¿Cuántas relaciones "DIRECTED" existen?

```
MATCH ()-[r:DIRECTED]->()
RETURN COUNT(r) AS Total_Relaciones_DIRECTED;
```

```
1 MATCH ()-[r:DIRECTED]→()
2 RETURN COUNT(r) AS Total_Relaciones_DIRECTED;
```

	Total_Relaciones_DIRECTED
1	6977

11. ¿Cuántas relaciones "IS_CITIZEN_OF" existen?

```
MATCH ()-[r:IS_CITIZEN_OF]->()
RETURN COUNT(r) AS Total_Relaciones_IS_CITIZEN_OF;
```

```
1 MATCH ()-[r:IS_CITIZEN_OF]→()
2 RETURN COUNT(r) AS Total_Relaciones_IS_CITIZEN_OF;
```

	Total_Relaciones_IS_CITIZEN_OF
1	349

12. ¿Cuántas relaciones "FROM" existen?

```
MATCH ()-[r:FROM]->()
RETURN COUNT(r) AS Total_Relaciones_FROM;
```

```
1 MATCH ()-[r:FROM]→()
2 RETURN COUNT(r) AS Total_Relaciones_FROM;
```

	Total_Relaciones_FROM
1	10019

13. ¿Cuántas relaciones "LISTED_IN" existen?

```
MATCH ()-[r:LISTED_IN]->()
```

```
RETURN COUNT(r) AS Total_Relaciones_LISTED_IN;
```

```
1 MATCH ()-[r:LISTED_IN]→()
2 RETURN COUNT(r) AS Total_Relaciones_LISTED_IN;
```

	Total_Relaciones_LISTED_IN
1	19323

14. ¿Cuántas relaciones "ADD" existen?

```
MATCH ()-[r:ADD]->()
RETURN COUNT(r) AS Total_Relaciones_ADD;
```

```
1 MATCH ()-[r:ADD]→()
2 RETURN COUNT(r) AS Total_Relaciones_ADD;
```

	Total_Relaciones_ADD
1	85271

15. ¿Cuál es el diámetro del grafo?

Diámetro de la red	7

En este contexto esto señala que necesito cómo máximo 7 saltos para llegar desde cualquier nodo de la red a cualquier otro nodo de la red (fue realizado con Gephi)

16. ¿Cuál es la densidad del grafo?

Densidad de grafo	0

En realidad la densidad del grafo no es 0, lo que pasa es que es tan pequeña que la herramienta de Gephi la hace redondearse a 0 (Fue realizado con Gephi). Si calculamos el total de nodos multiplicado por la cantidad de posibles nodos a los que puede conectarse tendríamos: $55.012 \times 55.011 = 3.026.265.132$ Para que el grafo tenga una relación entre cualquier par de nodos, lo cual es demasiado.

17. ¿Cuáles son las 50 películas más añadidas a los perfiles?

```
MATCH ()-[r:ADD]->(m:Movie)
RETURN m.id AS IDMovie,
    m.title AS Nombre_SerieOPelicula,
    COUNT(m.id) AS Total_ADDs
ORDER BY Total_ADDs DESC
LIMIT 50;
```

IDMovie	Nombre_SerieOPelicula	Total_ADDs
"s3775"	"Black Mirror"	1046
"s6201"	"Avengers: Infinity War"	1040

"s2011"	"How to Train Your Dragon 2"	1026
"s2583"	"Madagascar: Escape 2 Africa"	1020
"s1571"	"Once Upon a Time"	1018
"s602"	"The Karate Kid"	1015
"s7009"	"Hotel Transylvania 3: Summer Vacation"	1014
"s3097"	"You"	1012
"s1996"	"The Devil All The Time"	1010
"s452"	"Twilight"	1010
"s8068"	"Spider-Man 3"	1010
"s1407"	"Penguins of Madagascar: The Movie"	1010
"s173"	"School of Rock"	1006
"s575"	"Kung Fu Panda 2"	1004
"s684"	"A Family"	1002
"s1386"	"Demon Slayer: Kimetsu no Yaiba"	1001
"s1333"	"ICarly"	1001
"s856"	"Jurassic World Camp Cretaceous"	1001
"s7443"	"Men in Black"	1000
"s7444"	"Men in Black II"	998
"s33"	"Sex Education"	998
"s28"	"Grown Ups"	996
"s7448"	"Merlí"	996

"s2418"	"13 Reasons Why"	994
"s1449"	"Cobra Kai"	991
"s8617"	"Transcendence"	991
"s1139"	"The Pianist"	989
"s1879"	"The 100"	988
"s8581"	"Thor: Ragnarok"	987
"s632"	"The Seven Deadly Sins"	984
"s83"	"Lucifer"	983
"s8342"	"The Help"	980
"s2057"	"Hotel for Dogs"	978
"s8069"	"Spider-Man: Into the Spider-Verse"	976
"s6326"	"Black Panther"	975
"s603"	"The Karate Kid Part II"	974
"s1013"	"Luis Miguel - The Series"	972
"s4814"	"Hotel Transylvania"	971
"s527"	"Orange Is the New Black"	954
"s312"	"Control Z"	954
"s460"	"Never Have I Ever"	954
"s936"	"Madagascar 3: Europe's Most Wanted"	950
"s339"	"Hunter X Hunter (2011)"	947
"s779"	"Attack on Titan"	946
"s8312"	"The Fosters"	944

"s34"	"Squid Game"	944
"s391"	"The Walking Dead"	941
"s1617"	"Monster House"	935
"s604"	"The Karate Kid Part III"	929
"s1307"	"Good Girls"	924

18. ¿Existen películas que nunca fueron añadidas a los perfiles?

```
MATCH (m:Movie)
WHERE NOT ( ()-[:ADD]->(m) )
RETURN COUNT(DISTINCT m) AS Total_NO_ADDs;
```

```
1 MATCH (m:Movie)
2 WHERE NOT ( ()-[:ADD]→(m) )
3 RETURN COUNT(DISTINCT m) AS Total_NO_ADDs;
```

Table | Text

	Total_NO_ADDs
1	0

Lo que se traduce cómo que todas las películas alguna vez fueron vistas

19. ¿Quiénes son los 20 actores más famosos?

```
MATCH (p:Person)-[:ACTED_IN]->(m:Movie)<-[r:ADD]-(pr:Profile)
WITH p.name AS Nombre_Actor,
    COUNT(r) AS Total_ADDs
RETURN Nombre_Actor, Total_ADDs
ORDER BY Total_ADDs DESC
LIMIT 20;
```

```
MATCH (p:Person)-[:ACTED_IN]→(m:Movie)←[r:ADD]-(pr:Profile)
WITH p.name AS Nombre_Actor,
    COUNT(r) AS Total_ADDs
RETURN Nombre_Actor, Total_ADDs
ORDER BY Total_ADDs DESC
LIMIT 20;
```

	Nombre_Actor	Total_ADDs
1	"Martin Kove"	3913
2	"Ralph Macchio"	3911
3	"Benedict Cumberbatch"	3065
4	"Bryce Dallas Howard"	3054
5	"Yuki Kaji"	3041
6	"Chris Rock"	3003

20. ¿Quiénes son los 10 directores a los que más películas han generado un ADD?

```
MATCH (p:Person)-[:DIRECTED]->(m:Movie)<-[r:ADD]-(pr:Profile)
WITH p.name AS Nombre_Director,
    COUNT(r) AS Total_ADDs
RETURN Nombre_Director, Total_ADDs
ORDER BY Total_ADDs DESC
LIMIT 10;
```

```
MATCH (p:Person)-[:DIRECTED]→(m:Movie)←[r:ADD]-(pr:Profile)
WITH p.name AS Nombre_Director,
    COUNT(r) AS Total_ADDs
RETURN Nombre_Director, Total_ADDs
ORDER BY Total_ADDs DESC
LIMIT 10;
```

	Nombre_Director	Total_ADDs
1	"Eric Darnell"	2980
2	"John G. Avildsen"	2925
3	"Barry Sonnenfeld"	2002
4	"Tom McGrath"	1970
5	"Anthony Russo"	1040
6	"Joe Russo"	1040

21. ¿Qué 5 países tienen más actores?

```
MATCH (c:Country)<-[:IS_CITIZEN_OF]-(p:Person)-[:ACTED_IN]->(m:Movie)
RETURN c.name AS Pais,
    COUNT(DISTINCT p.name) AS Total_Actores
ORDER BY Total_Actores DESC
LIMIT 5;
```

```
1 MATCH (c:Country)←[:IS_CITIZEN_OF]-(p:Person)-[:ACTED_IN]→(m:Movie)
2 RETURN c.name AS Pais,
3     COUNT(DISTINCT p.name) AS Total_Actores
4 ORDER BY Total_Actores DESC
5 LIMIT 5;
```

Table | Text | Code

	Pais	Total_Actores
1	"United States of America"	17
2	"India"	3
3	"Nigeria"	3
4	"Indonesia"	2
5	"New Zealand"	2

Lo cual tiene sentido porque hay muy pocas relaciones IS_CITIZEN_OF. Por lo tanto, esta pregunta es importante para saber que no está la información completa de estas relaciones.

22. ¿De qué país son las 50 películas más añadidas?

```
MATCH ()-[r:ADD]->(m:Movie)-[:FROM]->(c:Country)
RETURN m.id AS IDMovie,
    m.title AS Nombre_SerieOPelicula,
    c.name AS Pais,
    COUNT(m.id) AS Total_ADDs
ORDER BY Total_ADDs DESC
LIMIT 50;
```

```
MATCH ()-[r:ADD]→(m:Movie)-[:FROM]→(c:Country)
RETURN m.id AS IDMovie,
    m.title AS Nombre_SerieOPelicula,
    c.name AS Pais,
    COUNT(m.id) AS Total_ADDs
ORDER BY Total_ADDs DESC
LIMIT 50;
```

Table | Text | Code

	IDMovie	Nombre_SerieOPelicula	Pais	Total_ADDs
1	"s3775"	"Black Mirror"	"United Kingdom"	1046
2	"s6201"	"Avengers: Infinity War"	"United States"	1040
3	"s2011"	"How to Train Your Dragon 2"	"United States"	1026
4	"s2583"	"Madagascar: Escape 2 Africa"	"United States"	1020
5	"s1571"	"Once Upon a Time"	"Egypt"	1018
6	"s602"	"The Karate Kid"	"United States"	1015

23. ¿Existen perfiles que nunca hicieron ADD a una serie/película?

```
MATCH (p:Profile)
WHERE NOT ( (p)-[:ADD]->() )
RETURN COUNT(DISTINCT p.id) AS Total_NO_ADDs;
```

```
MATCH (p:Profile)
WHERE NOT ( (p)-[:ADD]→() )
RETURN COUNT(DISTINCT p.id) AS Total_NO_ADDs;
```

Table | Text

	Total_NO_ADDs
1	0

Lo que se traduce cómo que todos los Perfiles alguna vez añadieron películas/series

24. ¿Cuáles son las 5 categorías favoritas de la gente?

```
MATCH (c:Category)<-[:LISTED_IN]-(m:Movie)<-[r:ADD]-(:Profile)
RETURN c.name AS Category,
    COUNT(r) AS Total_ADDs_Por_Categoria
ORDER BY Total_ADDs_Por_Categoria DESC
LIMIT 5;
```

```
MATCH (c:Category)←[:LISTED_IN]-(m:Movie)←[r:ADD]-(:Profile)
RETURN c.name AS Category,
    COUNT(r) AS Total_ADDs_Por_Categoria
ORDER BY Total_ADDs_Por_Categoria DESC
LIMIT 5;
```

Table | Text | Code

	Category	Total_ADDs_Por_Categoria
1	"Comedies"	19620
2	"TV Dramas"	18715
3	"Dramas"	16784
4	"International TV Shows"	16382
5	"Action & Adventure"	13438

25. ¿Quiénes son los 10 actores que actuaron en más películas?

```
MATCH (p:Person)-[:ACTED_IN]->(:Movie)<-[r:ADD]-(:Profile)
RETURN p.name AS Category,
    COUNT(r) AS Total_ADDs_Por_Actor
ORDER BY Total_ADDs_Por_Actor DESC
LIMIT 10;
```

```
MATCH (p:Person)-[:ACTED_IN]→(:Movie)←[r:ADD]-(:Profile)
RETURN p.name AS Category,
    COUNT(r) AS Total_ADDs_Por_Actor
ORDER BY Total_ADDs_Por_Actor DESC
LIMIT 10;
```

Table | Text | Code

	Category	Total_ADDs_Por_Actor
1	"Martin Kove"	3913
2	"Ralph Macchio"	3911
3	"Benedict Cumberbatch"	3065
4	"Bryce Dallas Howard"	3054
5	"Yuki Kaji"	3041
6	"Chris Rock"	3003

Esquema general del grafo actualizado

Creación de relaciones para Recomendaciones

En este apartado dejamos documentada la secciones de código implementadas para realizar recomendaciones:

```
// 1. (m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
MATCH
(c:Category)<-[:LISTED_IN]-(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Mo
vie)-[:LISTED_IN]->(c2:Category)
WHERE m.id <> m2.id AND c.name = c2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_CATEGORY{quantity:qty}]->(m2);
```

```
1 MATCH (c:Category)←[:LISTED_IN]-(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)-
  [:LISTED_IN]→(c2:Category)
2 WHERE m.id <> m2.id AND c.name = c2.name
3 WITH m, m2, COUNT(p.id) AS qty
4 MERGE (m)-[:WATCH_TOG_SAME_CATEGORY{quantity:qty}]→(m2);
```

Table

Set 159924 properties, created 159924 relationships, completed after 9077 ms.

```
//2. (m:Movie)-[r:WATCH_TOG_SAME_CATEGORY_DRAMAS]->(m2:Movie)
MATCH
(c:Category)<-[:LISTED_IN]-(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Mo
vie)-[:LISTED_IN]->(c2:Category)
WHERE m.id <> m2.id AND c.name = c2.name AND c.name = "Dramas"
WITH m, m2, COUNT(p.id) AS qty
```

```
MERGE (m)-[:WATCH_TOG_SAME_CATEGORY_DRAMAS{quantity:qty}]->(m2);
```

```
MATCH (c:Category)←[:LISTED_IN]-(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)-[:LISTED_IN]→(c2:Category)
WHERE m.id <> m2.id AND c.name = c2.name AND c.name = "Dramas"
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_CATEGORY_DRAMAS{quantity:qty}]→(m2);
```

Table

Set 38758 properties, created 38758 relationships, completed after 1250 ms.

```
//3. (m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie)
MATCH
(c:Country)<-[:FROM]-(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)-[
:FROM]->(c2:Country)
WHERE m.id <> m2.id AND c.name = c2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_COUNTRY{quantity:qty}]->(m2);
```

```
MATCH (c:Country)←[:FROM]-(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)-[:FROM]→(c2:Country)
WHERE m.id <> m2.id AND c.name = c2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_COUNTRY{quantity:qty}]→(m2);
```

Table

Set 171236 properties, created 171236 relationships, completed after 5031 ms.

```
//4. (m:Movie)-[r:WATCH_TOG_SAME_COUNTRY_MEXICO]->(m2:Movie)
MATCH
(c:Country)<-[:FROM]-(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)-[
:FROM]->(c2:Country)
WHERE m.id <> m2.id AND c.name = c2.name AND c.name = "Mexico"
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_COUNTRY_MEXICO{quantity:qty}]->(m2);
```

```
MATCH (c:Country)←[:FROM]-(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)-[:FROM]→(c2:Country)
WHERE m.id <> m2.id AND c.name = c2.name AND c.name = "Mexico"
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_COUNTRY_MEXICO{quantity:qty}]→(m2);
```

Table

Set 346 properties, created 346 relationships, completed after 90 ms.

```
//5. (m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie)
MATCH
(pe:Person)-[:ACTED_IN]->(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movi
e)<-[:ACTED_IN]-(pe2:Person)
WHERE m.id <> m2.id AND pe.name = pe2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_ACTOR{quantity:qty}]->(m2);
```

```
MATCH (pe:Person)-[:ACTED_IN]→(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)←[:ACTED_IN]-(pe2:Person)
WHERE m.id <> m2.id AND pe.name = pe2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_ACTOR{quantity:qty}]→(m2);
```

Table

Set 3016 properties, created 3016 relationships, completed after 48938 ms.

```
//6. (m:Movie)-[r:WATCH_TOG_SAME_ACTOR_TOM_HOLLAND]->(m2:Movie)
MATCH
(pe:Person)-[:ACTED_IN]->(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)<-[:ACTED_IN]-(pe2:Person)
WHERE m.id <> m2.id AND pe.name = pe2.name AND pe.name = "Tom Holland"
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_ACTOR_TOM_HOLLAND{quantity:qty}]->(m2);
```

```
MATCH (pe:Person)-[:ACTED_IN]→(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)←[:ACTED_IN]-(pe2:Person)
WHERE m.id <> m2.id AND pe.name = pe2.name AND pe.name = "Tom Holland"
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_ACTOR_TOM_HOLLAND{quantity:qty}]→(m2);
```

Table

Set 2 properties, created 2 relationships, completed after 42 ms.

```
//7. (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie)
MATCH
(pe:Person)-[:DIRECTED]->(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)<-[:DIRECTED]-(pe2:Person)
WHERE m.id <> m2.id AND pe.name = pe2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_DIRECTOR{quantity:qty}]->(m2);
```

```
MATCH (pe:Person)-[:DIRECTED]→(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)←[:DIRECTED]-(pe2:Person)
WHERE m.id <> m2.id AND pe.name = pe2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_DIRECTOR{quantity:qty}]→(m2);
```

Table

Set 60 properties, created 60 relationships, completed after 907 ms.

```
//8. (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR_MARTIN_SCORSESE]->(m2:Movie)
MATCH
(pe:Person)-[:DIRECTED]->(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)<-[:DIRECTED]-(pe2:Person)
WHERE m.id <> m2.id AND pe.name = pe2.name AND pe.name = "Martin Scorsese"
WITH m, m2, COUNT(p.id) AS qty
MERGE
(m)-[:WATCH_TOG_SAME_DIRECTOR_MARTIN_SCORSESE{quantity:qty}]->(m2);
```

```
MATCH (pe:Person)-[:DIRECTED]→(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)←[:DIRECTED]-(pe2:Person)
WHERE m.id <> m2.id AND pe.name = pe2.name AND pe.name = "Martin Scorsese"
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_DIRECTOR_MARTIN_SCORSESE{quantity:qty}]→(m2);
```

Table

Set 2 properties, created 2 relationships, completed after 36 ms.

```
//9. (m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie)
MATCH (m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)
WHERE m.id <> m2.id AND m.releaseYear = m2.releaseYear
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_RELEASE_YEAR{quantity:qty}]->(m2);
```

```
MATCH (m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)
WHERE m.id <> m2.id AND m.releaseYear = m2.releaseYear
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_RELEASE_YEAR{quantity:qty}]→(m2);
```

Table

Set 52658 properties, created 52658 relationships, completed after 2303 ms.

```
//10. (m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR_2019]->(m2:Movie)
MATCH (m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)
WHERE m.id <> m2.id AND m.releaseYear = m2.releaseYear AND
m.releaseYear = 2019
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_RELEASE_YEAR_2019{quantity:qty}]->(m2);
```

```
MATCH (m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)
WHERE m.id <> m2.id AND m.releaseYear = m2.releaseYear AND m.releaseYear = 2019
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_RELEASE_YEAR_2019{quantity:qty}]→(m2);
```

Table

Set 10552 properties, created 10552 relationships, completed after 645 ms.

```
//11. (m:Movie)-[r:WATCH_TOG_SAME_ACTOR_COUNTRY]->(m2:Movie)
MATCH
(co:Country)<-[:IS_CITIZEN_OF]-(pe:Person)-[:ACTED_IN]->(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)<-[:ACTED_IN]-(pe2:Person)-[:IS_CITIZEN_OF]->(co2:Country)
WHERE m.id <> m2.id AND co.name = co2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_ACTOR_COUNTRY{quantity:qty}]->(m2);
```

```
MATCH (co:Country)←[:IS_CITIZEN_OF]-(pe:Person)-[:ACTED_IN]→(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)←[:ACTED_IN]-(pe2:Person)-[:IS_CITIZEN_OF]→(co2:Country)
WHERE m.id <> m2.id AND co.name = co2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_ACTOR_COUNTRY{quantity:qty}]→(m2);
```

Table

Set 6 properties, created 6 relationships, completed after 82 ms.

```
//12. (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR_COUNTRY]->(m2:Movie)
```

```
MATCH
(co:Country)<-[:IS_CITIZEN_OF]-(pe:Person)-[:DIRECTED]->(m:Movie)<-[:AD
D]-(p:Profile)-[:ADD]->(m2:Movie)<-[:DIRECTED]-(pe2:Person)-[:IS_CITIZE
N_OF]->(co2:Country)
WHERE m.id <> m2.id AND co.name = co2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_DIRECTOR_COUNTRY{quantity:qty}]->(m2);
```

```
1 MATCH (co:Country)←[:IS_CITIZEN_OF]-(pe:Person)-[:DIRECTED]→(m:Movie)←[:ADD]-(p:Profile)-
  [:ADD]→(m2:Movie)←[:DIRECTED]-(pe2:Person)-[:IS_CITIZEN_OF]→(co2:Country)
2 WHERE m.id <> m2.id AND co.name = co2.name
3 WITH m, m2, COUNT(p.id) AS qty
4 MERGE (m)-[:WATCH_TOG_SAME_DIRECTOR_COUNTRY{quantity:qty}]→(m2);
```

Table

Set 2040 properties, created 2040 relationships, completed after 201 ms.

```
//13. (m:Movie)-[r:WATCH_TOG_SAME_ACTOR_MOVIE_COUNTRY]->(m2:Movie)
MATCH
(co:Country)<-[:IS_CITIZEN_OF]-(pe:Person)-[:ACTED_IN]->(m:Movie)<-[:AD
D]-(p:Profile)-[:ADD]->(m2:Movie)<-[:ACTED_IN]-(pe2:Person)-[:IS_CITIZE
N_OF]->(co2:Country),
      (c3:Country)<-[:FROM]-(m),
      (m2)-[:FROM]->(c4:Country)
WHERE m.id <> m2.id AND co.name = co2.name AND c3.name = c4.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_ACTOR_MOVIE_COUNTRY{quantity:qty}]->(m2);
```

```
1 MATCH (co:Country)←[:IS_CITIZEN_OF]-(pe:Person)-[:ACTED_IN]→(m:Movie)←[:ADD]-(p:Profile)-
  [:ADD]→(m2:Movie)←[:ACTED_IN]-(pe2:Person)-[:IS_CITIZEN_OF]→(co2:Country),
2       (c3:Country)←[:FROM]-(m),
3       (m2)-[:FROM]→(c4:Country)
4 WHERE m.id <> m2.id AND co.name = co2.name AND c3.name = c4.name
5 WITH m, m2, COUNT(p.id) AS qty
6 MERGE (m)-[:WATCH_TOG_SAME_ACTOR_MOVIE_COUNTRY{quantity:qty}]→(m2);
```

Table

Set 4 properties, created 4 relationships, completed after 58 ms.

```
//14. (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR_MOVIE_COUNTRY]->(m2:Movie)
MATCH
(co:Country)<-[:IS_CITIZEN_OF]-(pe:Person)-[:DIRECTED]->(m:Movie)<-[:AD
D]-(p:Profile)-[:ADD]->(m2:Movie)<-[:DIRECTED]-(pe2:Person)-[:IS_CITIZE
N_OF]->(co2:Country),
      (c3:Country)<-[:FROM]-(m),
      (m2)-[:FROM]->(c4:Country)
WHERE m.id <> m2.id AND co.name = co2.name AND c3.name = c4.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_DIRECTOR_MOVIE_COUNTRY{quantity:qty}]->(m2);
```

```
MATCH (co:Country)←[:IS_CITIZEN_OF]-(pe:Person)-[:DIRECTED]→(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)←[:DIRECTED]-(pe2:Person)-[:IS_CITIZEN_OF]→(co2:Country),
      (c3:Country)←[:FROM]-(m),
      (m2)-[:FROM]→(c4:Country)
WHERE m.id <> m2.id AND co.name = co2.name AND c3.name = c4.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_DIRECTOR_MOVIE_COUNTRY{quantity:qty}]→(m2);
```

Table

Set 1612 properties, created 1612 relationships, completed after 216 ms.

```
//15. (m:Movie)-[r:WATCH_TOG_SAME_CATEGORY_COUNTRY]->(m2:Movie)
MATCH
(c:Category)<-[:LISTED_IN]-(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)-[:LISTED_IN]->(c2:Category),
      (c3:Country)<-[:FROM]-(m),
      (m2)-[:FROM]->(c4:Country)
WHERE m.id <> m2.id AND c.name = c2.name AND c3.name = c4.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_CATEGORY_COUNTRY{quantity:qty}]->(m2);
```

```
MATCH (c:Category)←[:LISTED_IN]-(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)-[:LISTED_IN]→(c2:Category),
      (c3:Country)←[:FROM]-(m),
      (m2)-[:FROM]→(c4:Country)
WHERE m.id <> m2.id AND c.name = c2.name AND c3.name = c4.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_CATEGORY_COUNTRY{quantity:qty}]→(m2);
```

Table

Set 34244 properties, created 34244 relationships, completed after 4641 ms.

```
//16. (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR_CATEGORY]->(m2:Movie)
MATCH
(pe:Person)-[:DIRECTED]->(m:Movie)<-[:ADD]-(p:Profile)-[:ADD]->(m2:Movie)<-[:DIRECTED]-(pe2:Person),
      (c:Category)<-[:LISTED_IN]-(m),
      (m2)-[:LISTED_IN]->(c2:Category)
WHERE m.id <> m2.id AND pe.name = pe2.name AND c.name = c2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_DIRECTOR_CATEGORY{quantity:qty}]->(m2);
```

```
MATCH (pe:Person)-[:DIRECTED]→(m:Movie)←[:ADD]-(p:Profile)-[:ADD]→(m2:Movie)←[:DIRECTED]-(pe2:Person),
      (c:Category)←[:LISTED_IN]-(m),
      (m2)-[:LISTED_IN]→(c2:Category)
WHERE m.id <> m2.id AND pe.name = pe2.name AND c.name = c2.name
WITH m, m2, COUNT(p.id) AS qty
MERGE (m)-[:WATCH_TOG_SAME_DIRECTOR_CATEGORY{quantity:qty}]→(m2);
```

Table

Set 50 properties, created 50 relationships, completed after 989 ms.

Creación de Tablero de Recomendaciones básicas

En este apartado dejaremos documentados los tableros solicitados en el punto 4.4 de la consigna del proyecto 2 de la materia de Minería de Grafos:

Tablero de recomendaciones

1. **Mostrar el top 5 de recomendaciones basadas en categoría**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en categoría

Movie	RecommendedMovie	RecommendationsCount
Men in Black	Thor: Ragnarok	615
Thor: Ragnarok	Men in Black	615
Men in Black II	Men in Black	579
Men in Black	Men in Black II	579
Thor: Ragnarok	Men in Black II	558

2. **Mostrar el top 5 de recomendaciones basadas en categoría Dramas**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_CATEGORY_DRAMAS]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en categoría Dramas

Movie	RecommendedMovie	RecommendationsCount
The Devil All The Time	The Karate Kid Part II	221
The Karate Kid Part II	The Devil All The Time	221
The Pianist	The Karate Kid Part II	214
The Devil All The Time	A Family	214
The Karate Kid Part II	The Pianist	214

3. **Mostrar el top 5 de recomendaciones basadas en país**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en país

Movie	RecommendedMovie	RecommendationsCount
The Pianist	Transcendence	398
Transcendence	The Pianist	398
Hotel Transylvania	The 100	370
The 100	Hotel Transylvania	370
Avengers: Infinity War	Madagascar: Escape 2 Africa	246

4. **Mostrar el top 5 de recomendaciones basadas en país México**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_COUNTRY_MEXICO]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en México como país

Movie	RecommendedMovie	RecommendationsCount
Luis Miguel - The Series	Control Z	182
Control Z	Luis Miguel - The Series	182
Luis Miguel - The Series	Selena: The Series	5
Selena: The Series	Luis Miguel - The Series	5
Luis Miguel - The Series	Rezeta	4

5. **Mostrar el top 5 de recomendaciones basadas en un actor**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en un actor

Movie	RecommendedMovie	RecommendationsCount
Madagascar: Escape 2 Africa	Madagascar 3: Europe's Most W...	1,267
Madagascar 3: Europe's Most W...	Madagascar: Escape 2 Africa	1,267
The Karate Kid	The Karate Kid Part II	792
The Karate Kid Part II	The Karate Kid	792
Thor: Ragnarok	Avengers: Infinity War	784

6. **Mostrar el top 5 de recomendaciones basadas en un actor de tu elección: Tom Holland**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_ACTOR_TOM_HOLLAND]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en Tom Holland

Movie	RecommendedMovie	RecommendationsCount
The Devil All The Time	Avengers: Infinity War	226
Avengers: Infinity War	The Devil All The Time	226

7. **Mostrar el top 5 de recomendaciones basadas en un director**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en un director

Movie	RecommendedMovie	RecommendationsCount
Madagascar 3: Europe's Most W...	Madagascar: Escape 2 Africa	362
Madagascar: Escape 2 Africa	Madagascar 3: Europe's Most W...	362
Madagascar: Escape 2 Africa	Penguins of Madagascar: The M...	222
Penguins of Madagascar: The M...	Madagascar: Escape 2 Africa	222
Penguins of Madagascar: The M...	Madagascar 3: Europe's Most W...	217

8. **Mostrar el top 5 de recomendaciones basadas en un director de tu elección: Martin Scorsese**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR_MARTIN_SCORSESE]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en el director Martin Scorsese

Movie	RecommendedMovie	RecommendationsCount
Shutter Island	No Direction Home: Bob Dylan	1
No Direction Home: Bob Dylan	Shutter Island	1

9. **Mostrar el top 5 de recomendaciones basadas en un año de lanzamiento**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en el año de lanzamiento

Movie	RecommendedMovie	RecommendationsCount
Once Upon a Time	You	229
You	Once Upon a Time	229
13 Reasons Why	The 100	227
The 100	13 Reasons Why	227
Black Mirror	Once Upon a Time	219

10. **Mostrar el top 5 de recomendaciones basadas en un año de lanzamiento de tu elección: 2019**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR_2019]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en el año de lanzamiento 2019

Movie	RecommendedMovie	RecommendationsCount
Once Upon a Time	You	229
You	Once Upon a Time	229
Once Upon a Time	Black Mirror	219
Black Mirror	Once Upon a Time	219
You	Demon Slayer: Kimetsu no Yaiba	215

11. Mostrar el top 5 de recomendaciones basadas en actores del mismo país

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_ACTOR_COUNTRY]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en actores del mismo país

Movie	RecommendedMovie	RecommendationsCount
The Outlaw Josey Wales	That '70s Show	1
That '70s Show	The Outlaw Josey Wales	1
Wives on Strike	Norman Lear: Just Another Versi...	1
Team America: World Police	Ghost of the Mountains	1
Norman Lear: Just Another Versi...	Wives on Strike	1

12. Mostrar el top 5 de recomendaciones basadas en directores del mismo país

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR_COUNTRY]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en directores del mismo país

Movie	RecommendedMovie	RecommendationsCount
The Karate Kid Part II	Kung Fu Panda 2	215
Kung Fu Panda 2	The Karate Kid Part II	215
The Karate Kid	Kung Fu Panda 2	208
Kung Fu Panda 2	The Karate Kid	208
Grown Ups	The Karate Kid Part II	200

13. **Mostrar el top 5 de recomendaciones basadas en actores y directores y películas del mismo país**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_ACTOR_MOVIE_COUNTRY]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en actores y directores y pelícu

Movie	RecommendedMovie	RecommendationsCount
The Outlaw Josey Wales	That '70s Show	1
That '70s Show	The Outlaw Josey Wales	1
Team America: World Police	Ghost of the Mountains	1
Ghost of the Mountains	Team America: World Police	1

14. **Mostrar el top 5 de recomendaciones basadas en directores y películas del mismo país**

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR_MOVIE_COUNTRY]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en directores y películas del m

Movie	RecommendedMovie	RecommendationsCount
The Karate Kid Part II	Kung Fu Panda 2	215
Kung Fu Panda 2	The Karate Kid Part II	215
The Karate Kid	Kung Fu Panda 2	208
Kung Fu Panda 2	The Karate Kid	208
Grown Ups	The Karate Kid Part II	200

15. Mostrar el top 5 de recomendaciones basadas en películas de cierta categoría por país

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_CATEGORY_COUNTRY]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en películas de cierta categoría

Movie	RecommendedMovie	RecommendationsCount
Men in Black	Thor: Ragnarok	615
Thor: Ragnarok	Men in Black	615
Men in Black II	Men in Black	579
Men in Black	Men in Black II	579
Thor: Ragnarok	Men in Black II	558

16. Mostrar el top 5 de recomendaciones basadas en directores de películas de la misma categoría

```
MATCH (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR_CATEGORY]->(m2:Movie)
RETURN m.title AS Movie, m2.title AS RecommendedMovie, r.quantity AS
RecommendationsCount
ORDER BY RecommendationsCount DESC
LIMIT 5;
```

Top 5 de recomendaciones basadas en directores de películas de la

Movie	RecommendedMovie	RecommendationsCount
Madagascar: Escape 2 Africa	Madagascar 3: Europe's Most W...	724
Madagascar 3: Europe's Most W...	Madagascar: Escape 2 Africa	724
Men in Black	Men in Black II	579
Men in Black II	Men in Black	579
Penguins of Madagascar: The M...	Madagascar: Escape 2 Africa	444

Minería de Grafos

La minería de grafos es una técnica potente que permite analizar diversas propiedades de los grafos, predecir sus estructuras y relaciones, y modelar patrones presentes en grafos reales [1].

En este apartado presentaremos un procedimiento para realizar un exhaustivo análisis de centralidad y de comunidades.

Seleccionar algoritmos de minería de Grafos

Para realizar el análisis del grafo, se implementarán distintos algoritmos en dos áreas clave: centralidad y detección de comunidades.

- Algoritmos de centralidad:
  - Degree Centrality: Evalúa la cantidad de conexiones directas de un nodo dentro del grafo.
  - Closeness Centrality: Determina la proximidad de un nodo con respecto a los demás, considerando su accesibilidad dentro de la red.
  - Betweenness Centrality: Identifica los nodos que actúan como puentes entre diferentes partes del grafo.
  - PageRank: Calcula la importancia de cada nodo en función de la cantidad y calidad de sus conexiones.
- Algoritmos de detección de comunidades:
  - Triangle Count: Este algoritmo cuenta la cantidad de triángulos en un grafo, es decir, conjuntos de tres nodos que están todos conectados entre sí.

- Clustering Coefficient: Analiza la densidad de conexiones entre nodos vecinos.
- Weakly Connected Components: Identifica grupos de nodos débilmente conectados dentro del grafo
- Label Propagation: Agrupa los nodos en comunidades basándose en la propagación de etiquetas a través de la red.
- Louvain Modularity: Detecta comunidades optimizando la modularidad de la estructura del grafo.

Selección de relaciones

En esta parte dejaremos establecido a qué relaciones de las creadas anteriormente se decide aplicar los algoritmos previamente seleccionados:

- **(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)**: Esta relación nos permitirá hacer recomendaciones de las películas vistas por la misma categoría
- **(m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie)**: Esta relación nos permitirá hacer recomendaciones de las películas vistas en el mismo actor
- **(m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie)**: Esta relación nos permitirá hacer recomendaciones de las películas vistas que contengan los mismos actores (al menos 1).
- **(m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie)**: Esta relación nos permitirá hacer recomendaciones de las películas vistas que hayan sido dirigidas por el mismo director.
- **(m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie)**: Esta relación nos permitirá hacer recomendaciones de las películas vistas que se hayan estrenado el mismo año.

Aplicación de algoritmos de centralidad

En este apartado, establecemos todos los elementos necesarios para aplicar los algoritmos de centralidad de manera eficiente y efectiva.

Si bien la metodología que vamos a seguir incluye la etapa (4.2) "Incluir atributos estructurados", en este caso es más conveniente agregar dichos atributos en una etapa anterior, antes de responder las preguntas de centralidad. Por lo tanto dedicamos toda esta sección a incluir los atributos a los nodos.

(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)

DEGREE:

```
// DEGREE CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphDEG',
```

```
    'Movie',
    {WATCH_TOG_SAME_CATEGORY:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
1 CALL gds.graph.project(
2     'myGraphDEG',
3     'Movie',
4     {WATCH_TOG_SAME_CATEGORY:
5         {
6             orientation: 'UNDIRECTED'
7         }
8     }
9 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_CATEGORY" } }	"myGraphDEG"	8807	319848	733

```
// Paso 2: Calculo la memoria necesaria:
CALL gds.degree.write.estimate(
    'myGraphDEG', {writeProperty: 'degreeSameCategory'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo:
CALL gds.degree.stream(
    'myGraphDEG'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS id, gds.util.asNode(nodeId).name
AS Movie, score
ORDER BY score DESC, Movie ASC;

// Paso 4: Implementamos
CALL gds.degree.write('myGraphDEG', { writeProperty:
'degreeSameCategory' })
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
centralityDistribution.mean AS meanScore,
centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

```
CALL gds.degree.write('myGraphDEG', { writeProperty: 'degreeSameCategory' })
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore,
centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

minimumScore	meanScore	maxScore	nodePropertiesWritten
0.0	36.3175777375962	4238.031249999999	8807

```
// Paso 5: Probamos
MATCH(m:Movie)
RETURN m.name AS producto,
    m.degreeSameCategory AS Grado
ORDER BY Grado DESC;
```

CLOSENESS CENTRALITY:

```
// CLOSENESS CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphCC',
    'Movie',
    'WATCH_TOG_SAME_CATEGORY'
);
```

```
CALL gds.graph.project(
    'myGraphCC',
    'Movie',
    'WATCH_TOG_SAME_CATEGORY'
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
`{ "Movie": { "label": "Movie", "properties": { } } }`	`{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_CATEGORY" } }`	"myGraphCC"	8807	159924	81

```
// PASO 2: calcular la memoria
CALL gds.closeness.write.estimate(
    'myGraphCC', { writeProperty: 'closenessSameCategory' }
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.closeness.stream(
    'myGraphCC'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
```

```
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.closeness.write('myGraphCC', {writeProperty:
'closenessSameCategory'})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
       centralityDistribution.max AS maxScore,
       centralityDistribution.mean AS meanScore,
       nodePropertiesWritten;
```

```
1 CALL gds.closeness.write('myGraphCC', {writeProperty: 'closenessSameCategory'})
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimumScore,
4        centralityDistribution.max AS maxScore,
5        centralityDistribution.mean AS meanScore,
6        nodePropertiesWritten;
```

	minimumScore	maxScore	meanScore	nodePropertiesWritten
1	0.0	1.000007629394531	0.3916640298461568	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.closenessSameCategory AS closenessSameCategory
ORDER BY closenessSameCategory DESC;
```

BETWEENNESS CENTRALITY:

```
// BETWEENNESS CENTRALITY:
// PASO 1:
CALL gds.graph.project(
    'myGraphBC',
    'Movie',
    'WATCH_TOG_SAME_CATEGORY'
);
```

```
1 CALL gds.graph.project(
2     'myGraphBC',
3     'Movie',
4     'WATCH_TOG_SAME_CATEGORY'
5 );
```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1	{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_CATEGORY" } }	"myGraphBC"	8807	159924	35

```
// PASO 2: calcular la memoria
CALL gds.betweenness.write.estimate(
    'myGraphBC', {writeProperty: 'betweennessSameCategory'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.betweenness.stream('myGraphBC')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC;

// PASO 4: Escribimos el resultado como un atributo extra en cada nodo
CALL          gds.betweenness.write('myGraphBC',          {writeProperty:
"betweennessSameCategory"})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimunScore,
        centralityDistribution.mean AS meanScore,
        centralityDistribution.max AS maxScore,
        nodePropertiesWritten;
```

```
1 CALL gds.betweenness.write('myGraphBC', {writeProperty: "betweennessSameCategory"})
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimunScore,
4        centralityDistribution.mean AS meanScore,
5        centralityDistribution.max AS maxScore,
6        nodePropertiesWritten;
```

Table / Text

	minimunScore	meanScore	maxScore	nodePropertiesWritten
1	0.0	6771.810245775102	4706143.999999999	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.betweennessSameCategory AS betweennessSameCategory
ORDER BY betweennessSameCategory DESC;
```

PAGERANK:

```
// PAGERANK:
//PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphPR',
    'Movie',
    'WATCH_TOG_SAME_CATEGORY'
);
```

```
CALL gds.graph.project(
    'myGraphPR',
    'Movie',
    'WATCH_TOG_SAME_CATEGORY'
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_CATEGORY" } }	"myGraphPR"	8807	159924	56

```
// PASO 2: calcular la memoria
// Aca estamos calculando el pageRank
CALL gds.pageRank.write.estimate(
    'myGraphPR', {
        writeProperty: 'pagerankSameCategory',
        maxIterations: 20,
        dampingFactor: 0.85}
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.pageRank.stream(
    'myGraphPR'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
        writeProperty: 'pagerankSameCategory'
    }
)
YIELD nodePropertiesWritten, ranIterations;
```

```
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
        writeProperty: 'pagerankSameCategory'
    }
)
YIELD nodePropertiesWritten, ranIterations;
```

nodePropertiesWritten	ranIterations
8807	20

```
// Paso 5: Verificar
MATCH (m:Movie)
RETURN m.name AS producto,
    m.pagerankSameCategory AS pageRank
ORDER BY pageRank DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie)

DEGREE:

```
// DEGREE CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphDEG',
    'Movie',
    {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphDEG',
    'Movie',
    {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_COUNTRY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_COUNTRY" } }	"myGraphDEG"	8807	342472	78

```
// Paso 2: Calculo la memoria necesaria:
CALL gds.degree.write.estimate(
    'myGraphDEG', {writeProperty: 'degreeSameCountry'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo:
CALL gds.degree.stream(
    'myGraphDEG'
)
YIELD nodeId, score
RETURN  gds.util.asNode(nodeId).id  AS  id,  gds.util.asNode(nodeId).name
AS Movie, score
ORDER BY score DESC, Movie ASC;

// Paso 4: Implementamos
CALL        gds.degree.write('myGraphDEG',        {        writeProperty:
'degreeSameCountry' })
YIELD centralityDistribution, nodePropertiesWritten
RETURN        centralityDistribution.min        AS        minimumScore,
centralityDistribution.mean AS meanScore,
centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

```
1 CALL gds.degree.write('myGraphDEG', { writeProperty: 'degreeSameCountry' })
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore,
4 centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

	minimumScore	meanScore	maxScore	nodePropertiesWritten
1	0.0	38.88644126076913	3766.0156249999995	8807

```
// Paso 5: Probamos
MATCH(m:Movie)
RETURN m.name AS producto,
    m.degreeSameCountry AS Grado
ORDER BY Grado DESC;
```

CLOSENESS CENTRALITY:

```
// CLOSENESS CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphCC',
    'Movie',
    'WATCH_TOG_SAME_COUNTRY'
);
```

```
CALL gds.graph.project(
    'myGraphCC',
    'Movie',
    'WATCH_TOG_SAME_COUNTRY'
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_COUNTRY": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_COUNTRY" } }	"myGraphCC"	8807	171236	41

```
// PASO 2: calcular la memoria
CALL gds.closeness.write.estimate(
    'myGraphCC', { writeProperty: 'closenessSameCountry' }
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.closeness.stream(
    'myGraphCC'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.closeness.write('myGraphCC', {writeProperty:
'closenessSameCountry'})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
        centralityDistribution.max AS maxScore,
        centralityDistribution.mean AS meanScore,
        nodePropertiesWritten;
```

```
CALL gds.closeness.write('myGraphCC', {writeProperty: 'closenessSameCountry'})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
        centralityDistribution.max AS maxScore,
        centralityDistribution.mean AS meanScore,
        nodePropertiesWritten;
```

minimumScore	maxScore	meanScore	nodePropertiesWritten
0.0	1.000007629394531	0.22624062138246978	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
```

```
    m.closenessSameCountry AS closenessSameCountry
ORDER BY closenessSameCountry DESC;
```

BETWEENNESS CENTRALITY:

```
// BETWEENNESS CENTRALITY:
// PASO 1:
CALL gds.graph.project(
    'myGraphBC',
    'Movie',
    'WATCH_TOG_SAME_COUNTRY'
);
```

```
1 CALL gds.graph.project(
2     'myGraphBC',
3     'Movie',
4     'WATCH_TOG_SAME_COUNTRY'
5 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_COUNTRY": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_COUNTRY" } }	"myGraphBC"	8807	171236	50

```
// PASO 2: calcular la memoria
CALL gds.betweenness.write.estimate(
    'myGraphBC', {writeProperty: 'betweennessSameCountry'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.betweenness.stream('myGraphBC')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC;

// PASO 4: Escribimos el resultado como un atributo extra en cada nodo
CALL          gds.betweenness.write('myGraphBC',          {writeProperty:
"betweennessSameCountry"})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimunScore,
        centralityDistribution.mean AS meanScore,
        centralityDistribution.max AS maxScore,
        nodePropertiesWritten;
```

```
CALL gds.betweenness.write('myGraphBC', {writeProperty: "betweennessSameCountry"})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimunScore,
       centralityDistribution.mean AS meanScore,
       centralityDistribution.max AS maxScore,
       nodePropertiesWritten;
```

	minimunScore	meanScore	maxScore	nodePropertiesWritten
1	0.0	11494.8084916974	8332799.999999999	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.betweennessSameCountry AS betweennessSameCountry
ORDER BY betweennessSameCountry DESC;
```

PAGERANK:

```
// PAGERANK:
//PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphPR',
    'Movie',
    'WATCH_TOG_SAME_COUNTRY'
);
```

```
CALL gds.graph.project(
    'myGraphPR',
    'Movie',
    'WATCH_TOG_SAME_COUNTRY'
);
```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1	{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_COUNTRY": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_COUNTRY" } }	"myGraphPR"	8807	171236	46

```
// PASO 2: calcular la memoria
// Aca estamos calculando el pageRank
CALL gds.pageRank.write.estimate(
    'myGraphPR', {
        writeProperty: 'pagerankSameCountry',
        maxIterations: 20,
        dampingFactor: 0.85}
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.pageRank.stream(
```

```
    'myGraphPR'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
        writeProperty: 'pagerankSameCountry'
    }
)
YIELD nodePropertiesWritten, ranIterations;
```

```
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
        writeProperty: 'pagerankSameCountry'
    }
)
YIELD nodePropertiesWritten, ranIterations;
```

nodePropertiesWritten	ranIterations
8807	20

```
// Paso 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.pagerankSameCountry AS pageRank
ORDER BY pageRank DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie)

DEGREE:

```
// DEGREE CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphDEG',
    'Movie',
    {WATCH_TOG_SAME_ACTOR:
        {
            orientation: 'UNDIRECTED'
```

```
            }
        }
);
```

```
CALL gds.graph.project(
    'myGraphDEG',
    'Movie',
    {WATCH_TOG_SAME_ACTOR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_ACTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_ACTOR" } }	"myGraphDEG"	8807	6032	59

```
// Paso 2: Calculo la memoria necesaria:
CALL gds.degree.write.estimate(
    'myGraphDEG', {writeProperty: 'degreeSameActor'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo:
CALL gds.degree.stream(
    'myGraphDEG'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS id, gds.util.asNode(nodeId).name
AS Movie, score
ORDER BY score DESC, Movie ASC;

// Paso 4: Implementamos
CALL gds.degree.write('myGraphDEG', { writeProperty: 'degreeSameActor'
})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
centralityDistribution.mean AS meanScore,
centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

```
CALL gds.degree.write('myGraphDEG', { writeProperty: 'degreeSameActor' })
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore,
centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

minimumScore	meanScore	maxScore	nodePropertiesWritten
1 0.0	0.6849111897241275	148.00097656249997	8807

```
// Paso 5: Probamos
MATCH(m:Movie)
RETURN m.name AS producto,
    m.degreeSameActor AS Grado
ORDER BY Grado DESC;
```

CLOSENESS CENTRALITY:

```
// CLOSENESS CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphCC',
    'Movie',
    'WATCH_TOG_SAME_ACTOR'
);
```

```
1 CALL gds.graph.project(
2     'myGraphCC',
3     'Movie',
4     'WATCH_TOG_SAME_ACTOR'
5 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_ACTOR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_ACTOR" } }	"myGraphCC"	8807	3016	60

```
// PASO 2: calcular la memoria
CALL gds.closeness.write.estimate(
    'myGraphCC', { writeProperty: 'closenessSameActor' }
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.closeness.stream(
    'myGraphCC'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.closeness.write('myGraphCC', {writeProperty:
'closenessSameActor'})
YIELD centralityDistribution, nodePropertiesWritten
```

```
RETURN centralityDistribution.min AS minimumScore,
        centralityDistribution.max AS maxScore,
        centralityDistribution.mean AS meanScore,
        nodePropertiesWritten;
```

```
1 CALL gds.closeness.write('myGraphCC', {writeProperty: 'closenessSameActor'})
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimumScore,
4        centralityDistribution.max AS maxScore,
5        centralityDistribution.mean AS meanScore,
6        nodePropertiesWritten;
```

Table | Text

	minimumScore	maxScore	meanScore	nodePropertiesWritten
1	0.0	1.000007629394531	0.06891265261439902	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.closenessSameActor AS closenessSameActor
ORDER BY closenessSameActor DESC;
```

BETWEENNESS CENTRALITY:

```
// BETWEENNESS CENTRALITY:
// PASO 1:
CALL gds.graph.project(
    'myGraphBC',
    'Movie',
    'WATCH_TOG_SAME_ACTOR'
);
```

```
1 CALL gds.graph.project(
2     'myGraphBC',
3     'Movie',
4     'WATCH_TOG_SAME_ACTOR'
5 );
```

Table | Text | Code

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1	{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_ACTOR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_ACTOR" } }	"myGraphBC"	8807	3016	57

```
// PASO 2: calcular la memoria
CALL gds.betweenness.write.estimate(
    'myGraphBC', {writeProperty: 'betweennessSameActor'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.betweenness.stream('myGraphBC')
YIELD nodeId, score
```

```
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC;

// PASO 4: Escribimos el resultado como un atributo extra en cada nodo
CALL gds.betweenness.write('myGraphBC', {writeProperty:
"betweennessSameActor"})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimunScore,
       centralityDistribution.mean AS meanScore,
       centralityDistribution.max AS maxScore,
       nodePropertiesWritten;
```

```
1 CALL gds.betweenness.write('myGraphBC', {writeProperty: "betweennessSameActor"})
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimunScore,
4        centralityDistribution.mean AS meanScore,
5        centralityDistribution.max AS maxScore,
6        nodePropertiesWritten;
```

	minimunScore	meanScore	maxScore	nodePropertiesWritten
1	0.0	234.19645133981697	145766.99999999997	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.betweennessSameActor AS betweennessSameActor
ORDER BY betweennessSameActor DESC;
```

PAGERANK:

```
// PAGERANK:
//PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphPR',
    'Movie',
    'WATCH_TOG_SAME_ACTOR'
);
```

```
1 CALL gds.graph.project(
2     'myGraphPR',
3     'Movie',
4     'WATCH_TOG_SAME_ACTOR'
5 );
```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1	{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_ACTOR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_ACTOR" } }	"myGraphPR"	8807	3016	85

```
// PASO 2: calcular la memoria
```

```
// Aca estamos calculando el pageRank
CALL gds.pageRank.write.estimate(
    'myGraphPR', {
        writeProperty: 'pagerankSameActor',
        maxIterations: 20,
        dampingFactor: 0.85}
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.pageRank.stream(
    'myGraphPR'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
        writeProperty: 'pagerankSameActor'
    }
)
YIELD nodePropertiesWritten, ranIterations;
```

```
1 CALL gds.pageRank.write(
2     'myGraphPR',
3     {
4         maxIterations: 20,
5         dampingFactor: 0.85,
6         writeProperty: 'pagerankSameActor'
7     }
8 )
9 YIELD nodePropertiesWritten, ranIterations;
```

	nodePropertiesWritten	ranIterations
1	8807	20

```
// Paso 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.pagerankSameActor AS pageRank
ORDER BY pageRank DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie)

DEGREE:

```
// DEGREE CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphDEG',
    'Movie',
    {WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphDEG',
    'Movie',
    {WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_DIRECTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_DIRECTOR" } }	"myGraphDEG"	8807	120	47

```
// Paso 2: Calculo la memoria necesaria:
CALL gds.degree.write.estimate(
    'myGraphDEG', {writeProperty: 'degreeSameDirector'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo:
CALL gds.degree.stream(
    'myGraphDEG'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS id, gds.util.asNode(nodeId).name
AS Movie, score
ORDER BY score DESC, Movie ASC;

// Paso 4: Implementamos
```

```
CALL gds.degree.write('myGraphDEG', { writeProperty: 'degreeSameDirector' })
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore,
centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

```
1 CALL gds.degree.write('myGraphDEG', { writeProperty: 'degreeSameDirector' })
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore,
4 centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

minimumScore	meanScore	maxScore	nodePropertiesWritten
0.0	0.013625540743624567	6.000030517578124	8807

```
// Paso 5: Probamos
MATCH (m:Movie)
RETURN m.name AS producto,
    m.degreeSameDirector AS Grado
ORDER BY Grado DESC;
```

CLOSENESS CENTRALITY:

```
// CLOSENESS CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphCC',
    'Movie',
    'WATCH_TOG_SAME_DIRECTOR'
);
```

```
1 CALL gds.graph.project(
2     'myGraphCC',
3     'Movie',
4     'WATCH_TOG_SAME_DIRECTOR'
5 );
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_DIRECTOR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_DIRECTOR" } }	"myGraphCC"	8807	60	62

```
// PASO 2: calcular la memoria
CALL gds.closeness.write.estimate(
    'myGraphCC', { writeProperty: 'closenessSameDirector' }
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;
```

```
// PASO 3: Aplicamos el algoritmo
CALL gds.closeness.stream(
    'myGraphCC'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.closeness.write('myGraphCC', {writeProperty:
'closenessSameDirector'})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
       centralityDistribution.max AS maxScore,
       centralityDistribution.mean AS meanScore,
       nodePropertiesWritten;
```

```
CALL gds.closeness.write('myGraphCC', {writeProperty: 'closenessSameDirector'})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
       centralityDistribution.max AS maxScore,
       centralityDistribution.mean AS meanScore,
       nodePropertiesWritten;
```

	minimumScore	maxScore	meanScore	nodePropertiesWritten
1	0.0	1.000007629394531	0.004969547376116166	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.closenessSameDirector AS closenessSameDirector
ORDER BY closenessSameDirector DESC;
```

BETWEENNESS CENTRALITY:

```
// BETWEENNESS CENTRALITY:
// PASO 1:
CALL gds.graph.project(
    'myGraphBC',
    'Movie',
    'WATCH_TOG_SAME_DIRECTOR'
);
```

```
CALL gds.graph.project(
    'myGraphBC',
    'Movie',
    'WATCH_TOG_SAME_DIRECTOR'
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
`{ "Movie": { "label": "Movie", "properties": { } } }`	`{ "WATCH_TOG_SAME_DIRECTOR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_DIRECTOR" } }`	"myGraphBC"	8807	60	79

```
// PASO 2: calcular la memoria
CALL gds.betweenness.write.estimate(
    'myGraphBC', {writeProperty: 'betweennessSameDirector'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.betweenness.stream('myGraphBC')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC;

// PASO 4: Escribimos el resultado como un atributo extra en cada nodo
CALL gds.betweenness.write('myGraphBC', {writeProperty:
"betweennessSameDirector"})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimunScore,
       centralityDistribution.mean AS meanScore,
       centralityDistribution.max AS maxScore,
       nodePropertiesWritten;
```

```
CALL gds.betweenness.write('myGraphBC', {writeProperty: "betweennessSameDirector"})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimunScore,
       centralityDistribution.mean AS meanScore,
       centralityDistribution.max AS maxScore,
       nodePropertiesWritten;
```

minimunScore	meanScore	maxScore	nodePropertiesWritten
0.0	0.0009083700759383516	4.000030517578124	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.betweennessSameDirector AS betweennessSameDirector
ORDER BY betweennessSameDirector DESC;
```

PAGERANK:

```
// PAGERANK:
//PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphPR',
    'Movie',
    'WATCH_TOG_SAME_DIRECTOR'
);
```

```
1 CALL gds.graph.project(
2     'myGraphPR',
3     'Movie',
4     'WATCH_TOG_SAME_DIRECTOR'
5 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_DIRECTOR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_DIRECTOR" } }	"myGraphPR"	8807	60	81

```
// PASO 2: calcular la memoria
// Aca estamos calculando el pageRank
CALL gds.pageRank.write.estimate(
    'myGraphPR', {
        writeProperty: 'pagerankSameDirector',
        maxIterations: 20,
        dampingFactor: 0.85}
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.pageRank.stream(
    'myGraphPR'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
```

```
        writeProperty: 'pagerankSameDirector'
    }
)
YIELD nodePropertiesWritten, ranIterations;
```

```
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
        writeProperty: 'pagerankSameDirector'
    }
)
YIELD nodePropertiesWritten, ranIterations;
```

	nodePropertiesWritten	ranIterations
1	8807	20

```
// Paso 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.pagerankSameDirector AS pageRank
ORDER BY pageRank DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie)

DEGREE:

```
// DEGREE CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphDEG',
    'Movie',
    {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphDEG',
    'Movie',
    {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_RELEASE_YEAR" } }	"myGraphDEG"	8807	105316	167

```
// Paso 2: Calculo la memoria necesaria:
CALL gds.degree.write.estimate(
    'myGraphDEG', {writeProperty: 'degreeSameReleaseYear'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo:
CALL gds.degree.stream(
    'myGraphDEG'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).id AS id, gds.util.asNode(nodeId).name
AS Movie, score
ORDER BY score DESC, Movie ASC;

// Paso 4: Implementamos
CALL gds.degree.write('myGraphDEG', { writeProperty:
'degreeSameReleaseYear' })
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
centralityDistribution.mean AS meanScore,
centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

```
CALL gds.degree.write('myGraphDEG', { writeProperty: 'degreeSameReleaseYear' })
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore, centralityDistribution.mean AS meanScore,
centralityDistribution.max AS maxScore, nodePropertiesWritten;
```

minimumScore	meanScore	maxScore	nodePropertiesWritten
1 0.0	11.958250794032246	1332.0078124999998	8807

```
// Paso 5: Probamos
MATCH(m:Movie)
RETURN m.name AS producto,
```

```
    m.degreeSameReleaseYear AS Grado
ORDER BY Grado DESC;
```

CLOSENESS CENTRALITY:

```
// CLOSENESS CENTRALITY
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
    'myGraphCC',
    'Movie',
    'WATCH_TOG_SAME_RELEASE_YEAR'
);
```

```
1 CALL gds.graph.project(
2     'myGraphCC',
3     'Movie',
4     'WATCH_TOG_SAME_RELEASE_YEAR'
5 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_RELEASE_YEAR" } }	"myGraphCC"	8807	52658	84

```
// PASO 2: calcular la memoria
CALL gds.closeness.write.estimate(
    'myGraphCC', { writeProperty: 'closenessSameReleaseYear' }
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.closeness.stream(
    'myGraphCC'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.closeness.write('myGraphCC', {writeProperty:
'closenessSameReleaseYear'})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
       centralityDistribution.max AS maxScore,
       centralityDistribution.mean AS meanScore,
```

```
        nodePropertiesWritten;
```

```
CALL gds.closeness.write('myGraphCC', {writeProperty: 'closenessSameReleaseYear'})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimumScore,
       centralityDistribution.max AS maxScore,
       centralityDistribution.mean AS meanScore,
       nodePropertiesWritten;
```

minimumScore	maxScore	meanScore	nodePropertiesWritten
0.0	1.000007629394531	0.3678184183120078	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.closenessSameReleaseYear AS closenessSameReleaseYear
ORDER BY closenessSameReleaseYear DESC;
```

BETWEENNESS CENTRALITY:

```
// BETWEENNESS CENTRALITY:
// PASO 1:
CALL gds.graph.project(
    'myGraphBC',
    'Movie',
    'WATCH_TOG_SAME_RELEASE_YEAR'
);
```

```
CALL gds.graph.project(
    'myGraphBC',
    'Movie',
    'WATCH_TOG_SAME_RELEASE_YEAR'
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_RELEASE_YEAR" } }	"myGraphBC"	8807	52658	51

```
// PASO 2: calcular la memoria
CALL gds.betweenness.write.estimate(
    'myGraphBC', {writeProperty: 'betweennessSameReleaseYear'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.betweenness.stream('myGraphBC')
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).name AS name, score
ORDER BY score DESC, name ASC;
```

```
// PASO 4: Escribimos el resultado como un atributo extra en cada nodo
CALL gds.betweenness.write('myGraphBC', {writeProperty: "betweennessSameReleaseYear"})
YIELD centralityDistribution, nodePropertiesWritten
RETURN centralityDistribution.min AS minimunScore,
       centralityDistribution.mean AS meanScore,
       centralityDistribution.max AS maxScore,
       nodePropertiesWritten;
```

```
1 CALL gds.betweenness.write('myGraphBC', {writeProperty: "betweennessSameReleaseYear"})
2 YIELD centralityDistribution, nodePropertiesWritten
3 RETURN centralityDistribution.min AS minimunScore,
4        centralityDistribution.mean AS meanScore,
5        centralityDistribution.max AS maxScore,
6        nodePropertiesWritten;
```

minimunScore	meanScore	maxScore	nodePropertiesWritten
0.0	1078.0955362195418	360339.99999999994	8807

```
// PASO 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.betweennessSameReleaseYear AS betweennessSameReleaseYear
ORDER BY betweennessSameReleaseYear DESC;
```

PAGERANK:

```
// PAGERANK:
//PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphPR',
    'Movie',
    'WATCH_TOG_SAME_RELEASE_YEAR'
);
```

```
1 CALL gds.graph.project(
2     'myGraphPR',
3     'Movie',
4     'WATCH_TOG_SAME_RELEASE_YEAR'
5 );
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_RELEASE_YEAR" } }	"myGraphPR"	8807	52658	50

```
// PASO 2: calcular la memoria
// Aca estamos calculando el pageRank
CALL gds.pageRank.write.estimate(
```

```
    'myGraphPR', {
        writeProperty: 'pagerankSameReleaseYear',
        maxIterations: 20,
        dampingFactor: 0.85}
)
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.pageRank.stream(
    'myGraphPR'
)
YIELD nodeId, score
RETURN gds.util.asNode(nodeId).Name AS Name, score
ORDER BY score DESC, Name ASC;

// PASO 4: añadir el atributo
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
        writeProperty: 'pagerankSameReleaseYear'
    }
)
YIELD nodePropertiesWritten, ranIterations;
```

```
CALL gds.pageRank.write(
    'myGraphPR',
    {
        maxIterations: 20,
        dampingFactor: 0.85,
        writeProperty: 'pagerankSameReleaseYear'
    }
)
YIELD nodePropertiesWritten, ranIterations;
```

	nodePropertiesWritten	ranIterations
1	8807	20

```
// Paso 5: Verificar
MATCH(m:Movie)
RETURN m.name AS producto,
    m.pagerankSameReleaseYear AS pageRank
ORDER BY pageRank DESC;
```

Aplicación de los algoritmos de comunidad

En este apartado, establecemos todos los elementos necesarios para aplicar los algoritmos de detección de comunidades de manera eficiente y efectiva.

(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)

TRIANGLE COUNT:

```
// TRIANGLE COUNT
// PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphTr',
    'Movie',
    {WATCH_TOG_SAME_CATEGORY:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
1 CALL gds.graph.project(
2     'myGraphTr',
3     'Movie',
4     {WATCH_TOG_SAME_CATEGORY:
5         {
6             orientation: 'UNDIRECTED'
7         }
8     }
9 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_CATEGORY" } }	"myGraphTr"	8807	319848	288

```
// PASO 2: calcular la memoria
CALL gds.triangleCount.write.estimate(
    'myGraphTr', {writeProperty: 'trianglesSameCategory'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.triangleCount.stream('myGraphTr')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).name AS name, triangleCount
ORDER BY triangleCount DESC, name ASC;
```

```
// PASO 4: veamos que nodos pertenecen a un triangulo
CALL gds.triangleCount.write('myGraphTr', {
        writeProperty: 'trianglesSameCategory'
    }
)
YIELD globalTriangleCount, nodeCount;
```

```
1 CALL gds.triangleCount.write('myGraphTr', {
2         writeProperty: 'trianglesSameCategory'
3     }
4 )
5 YIELD globalTriangleCount, nodeCount;
```

	globalTriangleCount	nodeCount
1	285829	8807

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.id AS ID,
    m.trianglesSameCategory as numero_de_triagulos
ORDER BY numero_de_triagulos DESC;
```

LOCAL CLUSTERING COEFFICIENT:

```
// LOCAL CLUSTERING COEFFICIENT
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphLCC',
  'Movie',
  {WATCH_TOG_SAME_CATEGORY:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
1 CALL gds.graph.project(
2   'myGraphLCC',
3   'Movie',
4   {WATCH_TOG_SAME_CATEGORY:
5         {
6             orientation: 'UNDIRECTED'
7         }
8     }
9 );
```

	nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1	{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_CATEGORY" } }	"myGraphLCC"	8807	319848	64

```
// PASO 2: calcular la memoria
CALL gds.localClusteringCoefficient.write.estimate('myGraphLCC', {
  writeProperty: 'clusteringCoefficientSameCategory'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.localClusteringCoefficient.stream('myGraphLCC')
YIELD nodeId, localClusteringCoefficient
RETURN gds.util.asNode(nodeId).name AS name, localClusteringCoefficient
ORDER BY localClusteringCoefficient DESC, name;

// PASO 4: escribir atributo
CALL gds.localClusteringCoefficient.write(
    'myGraphLCC',
    {
        writeProperty: 'clusteringCoefficientSameCategory'
    }
)
YIELD nodeCount, nodePropertiesWritten;
```

```
1 CALL gds.localClusteringCoefficient.write(
2     'myGraphLCC',
3     {
4         writeProperty: 'clusteringCoefficientSameCategory'
5     }
6 )
7 YIELD nodeCount, nodePropertiesWritten;
```

Table

	nodeCount	nodePropertiesWritten
1	8807	8807

Text

```
// PASO 5: comprobar el atributo clusteringCoeficient con el mayor y
sus vecinos
MATCH (m:Movie)
ORDER BY m.clusteringCoefficientSameCategory
MATCH
(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]-(v:Movie)-[r2:WATCH_TOG_SAME_CATE
GORY]-(v2:Movie)
RETURN m, r, v, r2, v2;
```

WEAKLY CONNECTED COMPONENTS:

```
// WEAKLY CONNECTED COMPONENTS
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphWCC',
  'Movie',
  {WATCH_TOG_SAME_CATEGORY:
```

```
            {
                orientation: 'UNDIRECTED',
                Properties: 'quantity'
            }
        }
);
```

```
CALL gds.graph.project(
  'myGraphWCC',
  'Movie',
  {WATCH_TOG_SAME_CATEGORY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
`{ "Movie": { "label": "Movie", "properties": { } } }`	`{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_CATEGORY" } }`	"myGraphWCC"	8807	319848	443

```
// PASO 2: calcular la memoria
CALL gds.wcc.write.estimate('myGraphWCC', {writeProperty: 'communityWCCSameCategory' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.wcc.stream('myGraphWCC')
YIELD nodeId, componentId
RETURN gds.util.asNode(nodeId).name AS name, componentId
ORDER BY componentId, name;

// PASO 4: escribir atributo
CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCCSameCategory'})
YIELD componentCount, componentDistribution;
```

```
CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCCSameCategory'})
YIELD componentCount, componentDistribution;
```

Table | Text | Code

	componentCount	componentDistribution
1	477	{ "min": 1, "p5": 1, "max": 5647, "p999": 5647, "p99": 11, "p1": 1, "p10": 1, "p90": 2, "p50": 1, "p25": 1, "p75": 1, "p95": 3, "mean": 18.463312368972748 }

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.communityWCCSameCategory as numero_de_WCC,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_WCC DESC;
```

LABEL PROPAGATION:

```
// LABEL PROPAGATION (General)
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_CATEGORY:
        {
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_CATEGORY:
        {
            Properties: 'quantity'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1		"myGraphLP"	8807	159924	220

```
{
  "Movie": {
    "label": "Movie",
    "properties": {

    }
  }
}
```

```
{
  "WATCH_TOG_SAME_CATEGORY": {
    "aggregation": "DEFAULT",
    "orientation": "NATURAL",
    "indexInverse": false,
    "properties": {
      "quantity": {
        "aggregation": "DEFAULT",
        "property": "quantity",
        "defaultValue": null
      }
    },
    "type": "WATCH_TOG_SAME_CATEGORY"
  }
}
```

```
// PASO 2: calcular la memoria
CALL gds.labelPropagation.write.estimate('myGraphLP', { writeProperty: 'labelPropagationSameCategory' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.labelPropagation.stream('myGraphLP')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).name AS Name, Community
ORDER BY Community, Name;

// PASO 4: escribir atributo
CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagationSameCategory'})
YIELD communityCount, ranIterations, didConverge;
```

```
CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagationSameCategory'})
YIELD communityCount, ranIterations, didConverge;
```

communityCount	ranIterations	didConverge
494	10	false

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.labelPropagationSameCategory as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

LOUVAIN:

```
// LOUVAIN MODULARITY
```

```
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphLM',
  'Movie',
  {WATCH_TOG_SAME_CATEGORY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
1  CALL gds.graph.project(
2    'myGraphLM',
3    'Movie',
4    {WATCH_TOG_SAME_CATEGORY:
5          {
6              orientation: 'UNDIRECTED',
7              Properties: 'quantity'
8          }
9      }
10 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_CATEGORY" } }	"myGraphLM"	8807	319848	149

```
// PASO 2: calcular la memoria
CALL gds.louvain.write.estimate('myGraphLM', {writeProperty:
'louvainSameCategory' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.louvain.stream('myGraphLM')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name AS name, communityId
ORDER BY name ASC;

// PASO 4: escribir atributo
CALL gds.louvain.write('myGraphLM', {writeProperty:
'louvainSameCategory'})
YIELD communityCount, modularity, modularities;
```

```
CALL gds.louvain.write('myGraphLM', {writeProperty: 'louvainSameCategory'})
YIELD communityCount, modularity, modularities;
```

communityCount	modularity	modularities
498	0.49110430539409067	[0.47964267169373126, 0.48995888781832847, 0.49085250483412096, 0.49110430539409067]

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.louvainSameCategory as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie)

TRIANGLE COUNT:

```
// RELACION (m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie)

// TRIANGLE COUNT
// PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphTr',
    'Movie',
    {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphTr',
    'Movie',
    {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_COUNTRY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_COUNTRY" } }	"myGraphTr"	8807	342472	679

```
// PASO 2: calcular la memoria
CALL gds.triangleCount.write.estimate(
```

```
    'myGraphTr', {writeProperty: 'trianglesSameCountry'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.triangleCount.stream('myGraphTr')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).name AS name, triangleCount
ORDER BY triangleCount DESC, name ASC;

// PASO 4: veamos que nodos pertenecen a un triangulo
CALL gds.triangleCount.write('myGraphTr', {
        writeProperty: 'trianglesSameCountry'
    }
)
YIELD globalTriangleCount, nodeCount;
```

```
CALL gds.triangleCount.write('myGraphTr', {
        writeProperty: 'trianglesSameCountry'
    }
)
YIELD globalTriangleCount, nodeCount;
```

	globalTriangleCount	nodeCount
1	913612	8807

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.id AS ID,
    m.trianglesSameCountry as numero_de_triagulos
ORDER BY numero_de_triagulos DESC;
```

LOCAL CLUSTERING COEFFICIENT:

```
// LOCAL CLUSTERING COEFFICIENT
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphLCC',
  'Movie',
  {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
  'myGraphLCC',
  'Movie',
  {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_COUNTRY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_COUNTRY" } }	"myGraphLCC"	8807	342472	262

```
// PASO 2: calcular la memoria
CALL gds.localClusteringCoefficient.write.estimate('myGraphLCC', {
  writeProperty: 'clusteringCoefficientSameCountry'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.localClusteringCoefficient.stream('myGraphLCC')
YIELD nodeId, localClusteringCoefficient
RETURN gds.util.asNode(nodeId).name AS name, localClusteringCoefficient
ORDER BY localClusteringCoefficient DESC, name;

// PASO 4: escribir atributo
CALL gds.localClusteringCoefficient.write(
    'myGraphLCC',
    {
        writeProperty: 'clusteringCoefficientSameCountry'
    }
)
YIELD nodeCount, nodePropertiesWritten;
```

```
CALL gds.localClusteringCoefficient.write(
    'myGraphLCC',
    {
        writeProperty: 'clusteringCoefficientSameCountry'
    }
)
YIELD nodeCount, nodePropertiesWritten;
```

nodeCount	nodePropertiesWritten
1 8807	8807

```
// PASO 5: comprobar el atributo clusteringCoeficient con el mayor y
sus vecinos
MATCH (m:Movie)
```

```
ORDER BY m.clusteringCoefficientSameCountry
MATCH
(m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]-(v:Movie)-[r2:WATCH_TOG_SAME_COUNT
RY]-(v2:Movie)
RETURN m, r, v, r2, v2;
```

WEAKLY CONNECTED COMPONENTS:

```
// WEAKLY CONNECTED COMPONENTS
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphWCC',
  'Movie',
  {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
  'myGraphWCC',
  'Movie',
  {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1		"myGraphWCC"	8807	342472	465

```
{
  "Movie": {
    "label": "Movie",
    "properties": {

    }
  }
}
```

```
{
  "WATCH_TOG_SAME_COUNTRY": {
    "aggregation": "DEFAULT",
    "orientation": "UNDIRECTED",
    "indexInverse": false,
    "properties": {
      "quantity": {
        "aggregation": "DEFAULT",
        "property": "quantity",
        "defaultValue": null
      }
    },
    "type": "WATCH_TOG_SAME_COUNTRY"
  }
}
```

```
// PASO 2: calcular la memoria
CALL gds.wcc.write.estimate('myGraphWCC', { writeProperty:
'communityWCCSameCountry' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.wcc.stream('myGraphWCC')
YIELD nodeId, componentId
RETURN gds.util.asNode(nodeId).name AS name, componentId
```

```
ORDER BY componentId, name;

// PASO 4: escribir atributo
CALL gds.wcc.write('myGraphWCC', {writeProperty:
'communityWCCSameCountry'})
YIELD componentCount, componentDistribution;
```

```
1 CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCCSameCountry'})
2 YIELD componentCount, componentDistribution;
```

Table | Text | Code

	componentCount	componentDistribution
1	2625	{ "min": 1, "p5": 1, "max": 5920, "p999": 8, "p99": 3, "p1": 1, "p10": 1, "p90": 1, "p50": 1, "p25": 1, "p75": 1, "p95": 2, "mean": 3.355047619047619 }

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.communityWCCSameCountry as numero_de_WCC,
    COUNT(m.title) AS Miembros
ORDER BY Miembros DESC;
```

LABEL PROPAGATION:

```
// LABEL PROPAGATION (General)
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_COUNTRY:
        {
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_COUNTRY:
        {
            Properties: 'quantity'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_COUNTRY": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_COUNTRY" } }	"myGraphLP"	8807	171236	115

```
// PASO 2: calcular la memoria
CALL gds.labelPropagation.write.estimate('myGraphLP', { writeProperty: 'labelPropagationSameCountry' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.labelPropagation.stream('myGraphLP')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).name AS Name, Community
ORDER BY Community, Name;

// PASO 4: escribir atributo
CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagationSameCountry'})
YIELD communityCount, ranIterations, didConverge;
```

```
CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagationSameCountry'})
YIELD communityCount, ranIterations, didConverge;
```

communityCount	ranIterations	didConverge
2722	8	true

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.labelPropagationSameCountry as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

LOUVAIN:

```
// LOUVAIN MODULARITY
// PASO 1: subgrafo
```

```
CALL gds.graph.project(
  'myGraphLM',
  'Movie',
  {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
1  CALL gds.graph.project(
2    'myGraphLM',
3    'Movie',
4    {WATCH_TOG_SAME_COUNTRY:
5        {
6            orientation: 'UNDIRECTED',
7            Properties: 'quantity'
8        }
9    }
10 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_COUNTRY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_COUNTRY" } }	"myGraphLM"	8807	342472	179

```
// PASO 2: calcular la memoria
CALL gds.louvain.write.estimate('myGraphLM', {writeProperty: 'louvainSameCountry' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.louvain.stream('myGraphLM')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name AS name, communityId
ORDER BY name ASC;

// PASO 4: escribir atributo
CALL gds.louvain.write('myGraphLM', {writeProperty: 'louvainSameCountry'})
YIELD communityCount, modularity, modularities;
```

```
CALL gds.louvain.write('myGraphLM', {writeProperty: 'louvainSameCountry'})
YIELD communityCount, modularity, modularities;
```

communityCount	modularity	modularities
2637	0.1740922328736807	[0.1443310313306017, 0.16640187933368042, 0.1740922328736807]

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.louvainSameCountry as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie)

TRIANGLE COUNT:

```
// TRIANGLE COUNT
// PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphTr',
    'Movie',
    {WATCH_TOG_SAME_ACTOR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphTr',
    'Movie',
    {WATCH_TOG_SAME_ACTOR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_ACTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_ACTOR" } }	"myGraphTr"	8807	6032	382

```
// PASO 2: calcular la memoria
CALL gds.triangleCount.write.estimate(
    'myGraphTr', {writeProperty: 'trianglesSameActor'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;
```

```
// PASO 3: Aplicamos el algoritmo
CALL gds.triangleCount.stream('myGraphTr')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).name AS name, triangleCount
ORDER BY triangleCount DESC, name ASC;

// PASO 4: veamos que nodos pertenecen a un triangulo
CALL gds.triangleCount.write('myGraphTr', {
        writeProperty: 'trianglesSameActor'
    }
)
YIELD globalTriangleCount, nodeCount;
```

```
1 CALL gds.triangleCount.write('myGraphTr', {
2         writeProperty: 'trianglesSameActor'
3     }
4 )
5 YIELD globalTriangleCount, nodeCount;
```

Table | Text

	globalTriangleCount	nodeCount
1	232	8807

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.id AS ID,
    m.trianglesSameActor as numero_de_triagulos
ORDER BY numero_de_triagulos DESC;
```

LOCAL CLUSTERING COEFFICIENT:

```
// LOCAL CLUSTERING COEFFICIENT
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphLCC',
  'Movie',
  {WATCH_TOG_SAME_ACTOR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
  'myGraphLCC',
  'Movie',
  {WATCH_TOG_SAME_ACTOR:
      {
          orientation: 'UNDIRECTED'
      }
  }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_ACTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_ACTOR" } }	"myGraphLCC"	8807	6032	130

```
// PASO 2: calcular la memoria
CALL gds.localClusteringCoefficient.write.estimate('myGraphLCC', {
  writeProperty: 'clusteringCoefficientSameActor'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.localClusteringCoefficient.stream('myGraphLCC')
YIELD nodeId, localClusteringCoefficient
RETURN gds.util.asNode(nodeId).name AS name, localClusteringCoefficient
ORDER BY localClusteringCoefficient DESC, name;

// PASO 4: escribir atributo
CALL gds.localClusteringCoefficient.write(
    'myGraphLCC',
    {
        writeProperty: 'clusteringCoefficientSameActor'
    }
)
YIELD nodeCount, nodePropertiesWritten;
```

```
CALL gds.localClusteringCoefficient.write(
    'myGraphLCC',
    {
        writeProperty: 'clusteringCoefficientSameActor'
    }
)
YIELD nodeCount, nodePropertiesWritten;
```

nodeCount	nodePropertiesWritten
1 8807	8807

```
// PASO 5: comprobar el atributo clusteringCoeficient con el mayor y
sus vecinos
MATCH (m:Movie)
```

```
ORDER BY m.clusteringCoefficientSameActor
MATCH
(m:Movie)-[r:WATCH_TOG_SAME_ACTOR]-(v:Movie)-[r2:WATCH_TOG_SAME_ACTOR]-
(v2:Movie)
RETURN m, r, v, r2, v2;
```

WEAKLY CONNECTED COMPONENTS:

```
// WEAKLY CONNECTED COMPONENTS
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphWCC',
  'Movie',
  {WATCH_TOG_SAME_ACTOR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
1  CALL gds.graph.project(
2    'myGraphWCC',
3    'Movie',
4    {WATCH_TOG_SAME_ACTOR:
5        {
6            orientation: 'UNDIRECTED',
7            Properties: 'quantity'
8        }
9    }
10 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
`{ "Movie": { "label": "Movie", "properties": { } } }`	`{ "WATCH_TOG_SAME_ACTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_ACTOR" } }`	"myGraphWCC"	8807	6032	161

```
// PASO 2: calcular la memoria
CALL gds.wcc.write.estimate('myGraphWCC', { writeProperty:
'communityWCCSameActor' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.wcc.stream('myGraphWCC')
YIELD nodeId, componentId
RETURN gds.util.asNode(nodeId).name AS name, componentId
```

```
ORDER BY componentId, name;

// PASO 4: escribir atributo
CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCCSameActor'})
YIELD componentCount, componentDistribution;
```

```
1 CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCCSameActor'})
2 YIELD componentCount, componentDistribution;
```

Table | Text | Code

	componentCount	componentDistribution
1	7654	{ "min": 1, "p5": 1, "max": 807, "p999": 6, "p99": 2, "p1": 1, "p10": 1, "p90": 1, "p50": 1, "p25": 1, "p75": 1, "p95": 1, "mean": 1.150640188136922 }

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.communityWCCSameActor as numero_de_WCC,
    COUNT(m.title) AS Miembros
ORDER BY Miembros DESC;
```

LABEL PROPAGATION:

```
// LABEL PROPAGATION (General)
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_ACTOR:
        {
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_ACTOR:
        {
            Properties: 'quantity'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_ACTOR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_ACTOR" } }	"myGraphLP"	8807	3016	77

```
// PASO 2: calcular la memoria
CALL gds.labelPropagation.write.estimate('myGraphLP', { writeProperty: 'labelPropagationSameActor' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.labelPropagation.stream('myGraphLP')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).name AS Name, Community
ORDER BY Community, Name;

// PASO 4: escribir atributo
CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagationSameActor'})
YIELD communityCount, ranIterations, didConverge;
```

```
CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagationSameActor'})
YIELD communityCount, ranIterations, didConverge;
```

communityCount	ranIterations	didConverge
1 7695	4	true

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.labelPropagationSameActor as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

LOUVAIN:

```
// LOUVAIN MODULARITY
// PASO 1: subgrafo
```

```
CALL gds.graph.project(
  'myGraphLM',
  'Movie',
  {WATCH_TOG_SAME_ACTOR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
1  CALL gds.graph.project(
2    'myGraphLM',
3    'Movie',
4    {WATCH_TOG_SAME_ACTOR:
5          {
6              orientation: 'UNDIRECTED',
7              Properties: 'quantity'
8          }
9      }
10 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_ACTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_ACTOR" } }	"myGraphLM"	8807	6032	62

```
// PASO 2: calcular la memoria
CALL gds.louvain.write.estimate('myGraphLM', {writeProperty: 'louvainSameActor' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.louvain.stream('myGraphLM')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name AS name, communityId
ORDER BY name ASC;

// PASO 4: escribir atributo
CALL gds.louvain.write('myGraphLM', {writeProperty: 'louvainSameActor'})
YIELD communityCount, modularity, modularities;
```

```
CALL gds.louvain.write('myGraphLM', {writeProperty: 'louvainSameActor'})
YIELD communityCount, modularity, modularities;
```

communityCount	modularity	modularities
7668	0.8037797089263986	[0.6730243739841973, 0.79867651042363, 0.8037797089263986]

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.louvainSameActor as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie)

TRIANGLE COUNT:

```
// TRIANGLE COUNT
// PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphTr',
    'Movie',
    {WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphTr',
    'Movie',
    {WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
`{ "Movie": { "label": "Movie", "properties": { } } }`	`{ "WATCH_TOG_SAME_DIRECTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_DIRECTOR" } }`	"myGraphTr"	8807	120	94

```
// PASO 2: calcular la memoria
CALL gds.triangleCount.write.estimate(
    'myGraphTr', {writeProperty: 'trianglesSameDirector'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;
```

```
// PASO 3: Aplicamos el algoritmo
CALL gds.triangleCount.stream('myGraphTr')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).name AS name, triangleCount
ORDER BY triangleCount DESC, name ASC;

// PASO 4: veamos que nodos pertenecen a un triangulo
CALL gds.triangleCount.write('myGraphTr', {
        writeProperty: 'trianglesSameDirector'
    }
)
YIELD globalTriangleCount, nodeCount;
```

```
1 CALL gds.triangleCount.write('myGraphTr', {
2         writeProperty: 'trianglesSameDirector'
3     }
4 )
5 YIELD globalTriangleCount, nodeCount;
```

Table | Text

	globalTriangleCount	nodeCount
1	5	8807

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.id AS ID,
    m.trianglesSameDirector as numero_de_triagulos
ORDER BY numero_de_triagulos DESC;
```

LOCAL CLUSTERING COEFFICIENT:

```
// LOCAL CLUSTERING COEFFICIENT
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphLCC',
  'Movie',
  {WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
  'myGraphLCC',
  'Movie',
  {WATCH_TOG_SAME_DIRECTOR:
      {
          orientation: 'UNDIRECTED'
      }
  }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_DIRECTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_DIRECTOR" } }	"myGraphLCC"	8807	120	45

```
// PASO 2: calcular la memoria
CALL gds.localClusteringCoefficient.write.estimate('myGraphLCC', {
  writeProperty: 'clusteringCoefficientSameDirector'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.localClusteringCoefficient.stream('myGraphLCC')
YIELD nodeId, localClusteringCoefficient
RETURN gds.util.asNode(nodeId).name AS name, localClusteringCoefficient
ORDER BY localClusteringCoefficient DESC, name;

// PASO 4: escribir atributo
CALL gds.localClusteringCoefficient.write(
    'myGraphLCC',
    {
        writeProperty: 'clusteringCoefficientSameDirector'
    }
)
YIELD nodeCount, nodePropertiesWritten;
```

```
CALL gds.localClusteringCoefficient.write(
    'myGraphLCC',
    {
        writeProperty: 'clusteringCoefficientSameDirector'
    }
)
YIELD nodeCount, nodePropertiesWritten;
```

nodeCount	nodePropertiesWritten
1 8807	8807

```
// PASO 5: comprobar el atributo clusteringCoeficient con el mayor y
sus vecinos
MATCH (m:Movie)
```

```
ORDER BY m.clusteringCoefficientSameDirector
MATCH
(m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]-(v:Movie)-[r2:WATCH_TOG_SAME_DIRE
CTOR]-(v2:Movie)
RETURN m, r, v, r2, v2;
```

WEAKLY CONNECTED COMPONENTS:

```
// WEAKLY CONNECTED COMPONENTS
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphWCC',
  'Movie',
  {WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
1  CALL gds.graph.project(
2    'myGraphWCC',
3    'Movie',
4    {WATCH_TOG_SAME_DIRECTOR:
5        {
6            orientation: 'UNDIRECTED',
7            Properties: 'quantity'
8        }
9    }
10 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_DIRECTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_DIRECTOR" } }	"myGraphWCC"	8807	120	81

```
// PASO 2: calcular la memoria
CALL gds.wcc.write.estimate('myGraphWCC', { writeProperty:
'communityWCCSameDirector' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.wcc.stream('myGraphWCC')
YIELD nodeId, componentId
RETURN gds.util.asNode(nodeId).name AS name, componentId
```

```
ORDER BY componentId, name;

// PASO 4: escribir atributo
CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCCSameDirector'})
YIELD componentCount, componentDistribution;
```

```
1 CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCCSameDirector'})
2 YIELD componentCount, componentDistribution;
```

Table | Text | Code

	componentCount	componentDistribution
1	8781	{ "min": 1, "p5": 1, "max": 4, "p999": 2, "p99": 1, "p1": 1, "p10": 1, "p90": 1, "p50": 1, "p25": 1, "p75": 1, "p95": 1, "mean": 1.0029609383897051 }

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.communityWCCSameDirector as numero_de_WCC,
    COUNT(m.title) AS Miembros
ORDER BY Miembros DESC;
```

LABEL PROPAGATION:

```
// LABEL PROPAGATION (General)
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_DIRECTOR:
        {
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_DIRECTOR:
        {
            Properties: 'quantity'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_DIRECTOR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_DIRECTOR" } }	"myGraphLP"	8807	60	61

```
// PASO 2: calcular la memoria
CALL gds.labelPropagation.write.estimate('myGraphLP', { writeProperty: 'labelPropagationSameDirector' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.labelPropagation.stream('myGraphLP')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).name AS Name, Community
ORDER BY Community, Name;

// PASO 4: escribir atributo
CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagationSameDirector'})
YIELD communityCount, ranIterations, didConverge;
```

```
CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagationSameDirector'})
YIELD communityCount, ranIterations, didConverge;
```

communityCount	ranIterations	didConverge
8781	2	true

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.labelPropagationSameDirector as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

LOUVAIN:

```
// LOUVAIN MODULARITY
```

```
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphLM',
  'Movie',
  {WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
1  CALL gds.graph.project(
2    'myGraphLM',
3    'Movie',
4    {WATCH_TOG_SAME_DIRECTOR:
5        {
6            orientation: 'UNDIRECTED',
7            Properties: 'quantity'
8        }
9    }
10 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1		"myGraphLM"	8807	120	82

```
{
  "Movie": {
    "label": "Movie",
    "properties": {

    }
  }
}
```

```
{
  "WATCH_TOG_SAME_DIRECTOR": {
    "aggregation": "DEFAULT",
    "orientation": "UNDIRECTED",
    "indexInverse": false,
    "properties": {
      "quantity": {
        "aggregation": "DEFAULT",
        "property": "quantity",
        "defaultValue": null
      }
    },
    "type": "WATCH_TOG_SAME_DIRECTOR"
  }
}
```

```
// PASO 2: calcular la memoria
CALL gds.louvain.write.estimate('myGraphLM', {writeProperty:
'louvainSameDirector' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.louvain.stream('myGraphLM')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name AS name, communityId
ORDER BY name ASC;

// PASO 4: escribir atributo
CALL gds.louvain.write('myGraphLM', {writeProperty:
'louvainSameDirector'})
YIELD communityCount, modularity, modularities;
```

```
CALL gds.louvain.write('myGraphLM', {writeProperty: 'louvainSameDirector'})
YIELD communityCount, modularity, modularities;
```

communityCount	modularity	modularities
8781	0.9155555555555556	[0.9155555555555556]

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.louvainSameDirector as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie)

TRIANGLE COUNT:

```
// TRIANGLE COUNT
// PASO 1: Creamos el subgrafo:
CALL gds.graph.project(
    'myGraphTr',
    'Movie',
    {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphTr',
    'Movie',
    {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_RELEASE_YEAR" } }	"myGraphTr"	8807	105316	45

```
// PASO 2: calcular la memoria
CALL gds.triangleCount.write.estimate(
    'myGraphTr', {writeProperty: 'trianglesSameReleaseYear'})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;
```

```
// PASO 3: Aplicamos el algoritmo
CALL gds.triangleCount.stream('myGraphTr')
YIELD nodeId, triangleCount
RETURN gds.util.asNode(nodeId).name AS name, triangleCount
ORDER BY triangleCount DESC, name ASC;

// PASO 4: veamos que nodos pertenecen a un triangulo
CALL gds.triangleCount.write('myGraphTr', {
        writeProperty: 'trianglesSameReleaseYear'
    }
)
YIELD globalTriangleCount, nodeCount;
```

```
1 CALL gds.triangleCount.write('myGraphTr', {
2         writeProperty: 'trianglesSameReleaseYear'
3     }
4 )
5 YIELD globalTriangleCount, nodeCount;
```

	globalTriangleCount	nodeCount
1	40812	8807

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.id AS ID,
    m.trianglesSameReleaseYear as numero_de_triagulos
ORDER BY numero_de_triagulos DESC;
```

LOCAL CLUSTERING COEFFICIENT:

```
// LOCAL CLUSTERING COEFFICIENT
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphLCC',
  'Movie',
  {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED'
        }
    }
);
```

```
CALL gds.graph.project(
  'myGraphLCC',
  'Movie',
  {WATCH_TOG_SAME_RELEASE_YEAR:
      {
          orientation: 'UNDIRECTED'
      }
  }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { }, "type": "WATCH_TOG_SAME_RELEASE_YEAR" } }	"myGraphLCC"	8807	105316	62

```
// PASO 2: calcular la memoria
CALL gds.localClusteringCoefficient.write.estimate('myGraphLCC', {
  writeProperty: 'clusteringCoefficientSameReleaseYear'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.localClusteringCoefficient.stream('myGraphLCC')
YIELD nodeId, localClusteringCoefficient
RETURN gds.util.asNode(nodeId).name AS name, localClusteringCoefficient
ORDER BY localClusteringCoefficient DESC, name;

// PASO 4: escribir atributo
CALL gds.localClusteringCoefficient.write(
    'myGraphLCC',
    {
        writeProperty: 'clusteringCoefficientSameReleaseYear'
    }
)
YIELD nodeCount, nodePropertiesWritten;
```

```
CALL gds.localClusteringCoefficient.write(
    'myGraphLCC',
    {
        writeProperty: 'clusteringCoefficientSameReleaseYear'
    }
)
YIELD nodeCount, nodePropertiesWritten;
```

nodeCount	nodePropertiesWritten
1 8807	8807

```
// PASO 5: comprobar el atributo clusteringCoeficient con el mayor y
sus vecinos
```

```
MATCH (m:Movie)
ORDER BY m.clusteringCoefficientSameReleaseYear
MATCH
(m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]-(v:Movie)-[r2:WATCH_TOG_SAME_
RELEASE_YEAR]-(v2:Movie)
RETURN m, r, v, r2, v2;
```

WEAKLY CONNECTED COMPONENTS:

```
// WEAKLY CONNECTED COMPONENTS
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphWCC',
  'Movie',
  {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
1  CALL gds.graph.project(
2    'myGraphWCC',
3    'Movie',
4    {WATCH_TOG_SAME_RELEASE_YEAR:
5        {
6            orientation: 'UNDIRECTED',
7            Properties: 'quantity'
8        }
9    }
10 );
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_RELEASE_YEAR" } }	"myGraphWCC"	8807	105316	184

```
// PASO 2: calcular la memoria
CALL gds.wcc.write.estimate('myGraphWCC', { writeProperty:
'communityWCCSameReleaseYear' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.wcc.stream('myGraphWCC')
YIELD nodeId, componentId
```

```
RETURN gds.util.asNode(nodeId).name AS name, componentId
ORDER BY componentId, name;

// PASO 4: escribir atributo
CALL gds.wcc.write('myGraphWCC', {writeProperty:
'communityWCCSameReleaseYear'})
YIELD componentCount, componentDistribution;
```

```
1 CALL gds.wcc.write('myGraphWCC', {writeProperty: 'communityWCCSameReleaseYear'})
2 YIELD componentCount, componentDistribution;
```

Table | Text | Code

	componentCount	componentDistribution
1	1325	{ "min": 1, "p5": 1, "max": 1135, "p999": 1017, "p99": 105, "p1": 1, "p10": 1, "p90": 1, "p50": 1, "p25": 1, "p75": 1, "p95": 2, "mean": 6.646792452830189 }

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.communityWCCSameReleaseYear as numero_de_WCC,
    COUNT(m.title) AS Miembros
ORDER BY Miembros DESC;
```

LABEL PROPAGATION:

```
// LABEL PROPAGATION (General)
// PASO 1: subgrafo
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
    'myGraphLP',
    'Movie',
    {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            Properties: 'quantity'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT", "orientation": "NATURAL", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_RELEASE_YEAR" } }	"myGraphLP"	8807	52658	86

```
// PASO 2: calcular la memoria
CALL gds.labelPropagation.write.estimate('myGraphLP', { writeProperty:
'labelPropagationSameReleaseYear' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.labelPropagation.stream('myGraphLP')
YIELD nodeId, communityId AS Community
RETURN gds.util.asNode(nodeId).name AS Name, Community
ORDER BY Community, Name;

// PASO 4: escribir atributo
CALL gds.labelPropagation.write('myGraphLP', {writeProperty:
'labelPropagationSameReleaseYear'})
YIELD communityCount, ranIterations, didConverge;
```

```
CALL gds.labelPropagation.write('myGraphLP', {writeProperty: 'labelPropagationSameReleaseYear'})
YIELD communityCount, ranIterations, didConverge;
```

communityCount	ranIterations	didConverge
1451	9	true

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.labelPropagationSameReleaseYear as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

LOUVAIN:

```
// LOUVAIN MODULARITY
```

```
// PASO 1: subgrafo
CALL gds.graph.project(
  'myGraphLM',
  'Movie',
  {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
1  CALL gds.graph.project(
2    'myGraphLM',
3    'Movie',
4    {WATCH_TOG_SAME_RELEASE_YEAR:
5        {
6            orientation: 'UNDIRECTED',
7            Properties: 'quantity'
8        }
9    }
10 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1					

```
{
  "Movie": {
    "label": "Movie",
    "properties": {

    }
  }
}
```

```
{
  "WATCH_TOG_SAME_RELEASE_YEAR": {
    "aggregation": "DEFAULT",
    "orientation": "UNDIRECTED",
    "indexInverse": false,
    "properties": {
      "quantity": {
        "aggregation": "DEFAULT",
        "property": "quantity",
        "defaultValue": null
      }
    },
    "type": "WATCH_TOG_SAME_RELEASE_YEAR"
  }
}
```

graphName	nodeCount	relationshipCount	projectMillis
"myGraphLM"	8807	105316	105

```
// PASO 2: calcular la memoria
CALL gds.louvain.write.estimate('myGraphLM', {writeProperty: 'louvainSameReleaseYear' })
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo
CALL gds.louvain.stream('myGraphLM')
YIELD nodeId, communityId, intermediateCommunityIds
RETURN gds.util.asNode(nodeId).name AS name, communityId
ORDER BY name ASC;

// PASO 4: escribir atributo
CALL gds.louvain.write('myGraphLM', {writeProperty: 'louvainSameReleaseYear'})
YIELD communityCount, modularity, modularities;
```

```
CALL gds.louvain.write('myGraphLM', {writeProperty: 'louvainSameReleaseYear'})
YIELD communityCount, modularity, modularities;
```

communityCount	modularity	modularities
1325	0.857085141489061	[0.7223643019061927, 0.8446492831087611, 0.857085141489061]

```
// PASO 5: comprobar el atributo
MATCH (m:Movie)
RETURN m.louvainSameReleaseYear as numero_de_comunidad,
    COLLECT(m.title) AS Miembros
ORDER BY numero_de_comunidad DESC;
```

Aplicación de los algoritmos de similaridad

(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie):

```
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
  'myGraphSameCategory',
  'Movie',
  {WATCH_TOG_SAME_CATEGORY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
  'myGraphSameCategory',
  'Movie',
  {WATCH_TOG_SAME_CATEGORY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_CATEGORY" } }	"myGraphSameCategory"	8807	319848	280

```
// Paso 2: Calculo la memoria necesaria:
CALL  gds.nodeSimilarity.filtered.write.estimate('myGraphSameCategory',
{
    writeRelationshipType: 'SIMILAR_CATEGORY',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// Paso 3: Aplicamos el algoritmo:
CALL          gds.nodeSimilarity.filtered.stream('myGraphSameCategory',
{sourceNodeFilter:'Movie' , targetNodeFilter:'Movie' } )
YIELD node1, node2, similarity
RETURN          gds.util.asNode(node1).title          AS          Movie1,
gds.util.asNode(node2).title AS Movie2, similarity
ORDER BY similarity DESCENDING, Movie1, Movie2;

// Paso 4: Implementamos
CALL gds.nodeSimilarity.filtered.write('myGraphSameCategory',{
    writeRelationshipType: 'SIMILAR_CATEGORY',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodesCompared, relationshipsWritten;
```

```
1 CALL gds.nodeSimilarity.filtered.write('myGraphSameCategory',{
2     writeRelationshipType: 'SIMILAR_CATEGORY',
3     writeProperty: 'score',
4     sourceNodeFilter:'Movie',
5     targetNodeFilter:'Movie'
6 })
7 YIELD nodesCompared, relationshipsWritten;
```

	nodesCompared	relationshipsWritten
1	8400	80865

```
// Paso 5: Probamos
MATCH (m:Movie)-[r:SIMILAR_CATEGORY]->(m2:Movie)
RETURN   m.title   AS   Movie1,   m2.title   AS   Movie2,   r.score   AS
SimilarityScore
ORDER BY r.score DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie):

```
// PASO 1: Creamos el subgrafo
```

```
CALL gds.graph.project(
  'myGraphSameCountry',
  'Movie',
  {WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
1  CALL gds.graph.project(
2    'myGraphSameCountry',
3    'Movie',
4    {WATCH_TOG_SAME_COUNTRY:
5          {
6              orientation: 'UNDIRECTED',
7              Properties: 'quantity'
8          }
9      }
10 );
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1 { "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_COUNTRY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_COUNTRY" } }	"myGraphSameCountry"	8807	342472	286

```
// Paso 2: Calculo la memoria necesaria:
CALL gds.nodeSimilarity.filtered.write.estimate('myGraphSameCountry', {
    writeRelationshipType: 'SIMILAR_COUNTRY',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// Paso 3: Aplicamos el algoritmo:
CALL gds.nodeSimilarity.filtered.stream('myGraphSameCountry',
{sourceNodeFilter:'Movie' , targetNodeFilter:'Movie' } )
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).title AS Movie1,
gds.util.asNode(node2).title AS Movie2, similarity
ORDER BY similarity DESCENDING, Movie1, Movie2;

// Paso 4: Implementamos
```

```
CALL gds.nodeSimilarity.filtered.write('myGraphSameCountry',{
    writeRelationshipType: 'SIMILAR_COUNTRY',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodesCompared, relationshipsWritten;
```

```
1 CALL gds.nodeSimilarity.filtered.write('myGraphSameCountry',{
2     writeRelationshipType: 'SIMILAR_COUNTRY',
3     writeProperty: 'score',
4     sourceNodeFilter:'Movie',
5     targetNodeFilter:'Movie'
6 })
7 YIELD nodesCompared, relationshipsWritten;
```

Table | Text

	nodesCompared	relationshipsWritten
1	6353	56349

```
// Paso 5: Probamos
MATCH (m:Movie)-[r:SIMILAR_COUNTRY]->(m2:Movie)
RETURN m.title AS Movie1, m2.title AS Movie2, r.score AS
SimilarityScore
ORDER BY r.score DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie):

```
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
  'myGraphSameActor',
  'Movie',
  {WATCH_TOG_SAME_ACTOR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
  'myGraphSameActor',
  'Movie',
  {WATCH_TOG_SAME_ACTOR:
      {
          orientation: 'UNDIRECTED',
          Properties: 'quantity'
      }
    }
);
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
1					

```
{
  "Movie": {
    "label": "Movie",
    "properties": {

    }
  }
}
```

```
{
  "WATCH_TOG_SAME_ACTOR": {
    "aggregation": "DEFAULT",
    "orientation": "UNDIRECTED",
    "indexInverse": false,
    "properties": {
      "quantity": {
        "aggregation": "DEFAULT",
        "property": "quantity",
        "defaultValue": null
      }
    },
    "type": "WATCH_TOG_SAME_ACTOR"
  }
}
```

graphName	nodeCount	relationshipCount	projectMillis
"myGraphSameActor"	8807	6032	75

```
// Paso 2: Calculo la memoria necesaria:
CALL gds.nodeSimilarity.filtered.write.estimate('myGraphSameActor', {
    writeRelationshipType: 'SIMILAR_ACTOR',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo:
CALL gds.nodeSimilarity.filtered.stream('myGraphSameActor',
{sourceNodeFilter:'Movie' , targetNodeFilter:'Movie' } )
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).title AS Movie1,
gds.util.asNode(node2).title AS Movie2, similarity
ORDER BY similarity DESCENDING, Movie1, Movie2;

// Paso 4: Implementamos
CALL gds.nodeSimilarity.filtered.write('myGraphSameActor',{
    writeRelationshipType: 'SIMILAR_ACTOR',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodesCompared, relationshipsWritten;
```

```
CALL gds.nodeSimilarity.filtered.write('myGraphSameActor',{
    writeRelationshipType: 'SIMILAR_ACTOR',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodesCompared, relationshipsWritten;
```

	nodesCompared	relationshipsWritten
1	1324	8924

```
// Paso 5: Probamos
MATCH (m:Movie)-[r:SIMILAR_ACTOR]->(m2:Movie)
RETURN m.title AS Movie1, m2.title AS Movie2, r.score AS
SimilarityScore
ORDER BY r.score DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie):

```
// RELACIÓN (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie)
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
  'myGraphSameDirector',
  'Movie',
  {WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
  'myGraphSameDirector',
  'Movie',
  {WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

Table | Text | Code

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_DIRECTOR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_DIRECTOR" } }	"myGraphSameDirector"	8807	120	62

```
// Paso 2: Calculo la memoria necesaria:
CALL   gds.nodeSimilarity.filtered.write.estimate('myGraphSameDirector',
{
    writeRelationshipType: 'SIMILAR_DIRECTOR',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// Paso 3: Aplicamos el algoritmo:
CALL           gds.nodeSimilarity.filtered.stream('myGraphSameDirector',
{sourceNodeFilter:'Movie' , targetNodeFilter:'Movie' } )
YIELD node1, node2, similarity
RETURN          gds.util.asNode(node1).title          AS          Movie1,
gds.util.asNode(node2).title AS Movie2, similarity
ORDER BY similarity DESCENDING, Movie1, Movie2;

// Paso 4: Implementamos
CALL gds.nodeSimilarity.filtered.write('myGraphSameDirector',{
    writeRelationshipType: 'SIMILAR_DIRECTOR',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodesCompared, relationshipsWritten;
```

```
CALL gds.nodeSimilarity.filtered.write('myGraphSameDirector',{
    writeRelationshipType: 'SIMILAR_DIRECTOR',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodesCompared, relationshipsWritten;
```

nodesCompared	relationshipsWritten
46	26

```
// Paso 5: Probamos
MATCH (m:Movie)-[r:SIMILAR_DIRECTOR]->(m2:Movie)
RETURN m.title AS Movie1, m2.title AS Movie2, r.score AS SimilarityScore
ORDER BY r.score DESC;
```

(m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie):

```
// PASO 1: Creamos el subgrafo
CALL gds.graph.project(
  'myGraphSameReleaseYear',
  'Movie',
  {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
CALL gds.graph.project(
  'myGraphSameReleaseYear',
  'Movie',
  {WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
`{ "Movie": { "label": "Movie", "properties": { } } }`	`{ "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_RELEASE_YEAR" } }`	"myGraphSameReleaseYear"	8807	105316	153

```
// Paso 2: Calculo la memoria necesaria:
CALL
gds.nodeSimilarity.filtered.write.estimate('myGraphSameReleaseYear', {
    writeRelationshipType: 'SIMILAR_RELEASE_YEAR',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// Paso 3: Aplicamos el algoritmo:
CALL gds.nodeSimilarity.filtered.stream('myGraphSameReleaseYear',
{sourceNodeFilter:'Movie' , targetNodeFilter:'Movie' } )
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).title AS Movie1,
gds.util.asNode(node2).title AS Movie2, similarity
ORDER BY similarity DESCENDING, Movie1, Movie2;

// Paso 4: Implementamos
CALL gds.nodeSimilarity.filtered.write('myGraphSameReleaseYear',{
    writeRelationshipType: 'SIMILAR_RELEASE_YEAR',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodesCompared, relationshipsWritten;
```

```
1 CALL gds.nodeSimilarity.filtered.write('myGraphSameReleaseYear',{
2     writeRelationshipType: 'SIMILAR_RELEASE_YEAR',
3     writeProperty: 'score',
4     sourceNodeFilter:'Movie',
5     targetNodeFilter:'Movie'
6 })
7 YIELD nodesCompared, relationshipsWritten;
```

Table

	nodesCompared	relationshipsWritten
1	7568	69751

Text

```
// Paso 5: Probamos
MATCH (m:Movie)-[r:SIMILAR_RELEASE_YEAR]->(m2:Movie)
RETURN m.title AS Movie1, m2.title AS Movie2, r.score AS
SimilarityScore
ORDER BY r.score DESC;
```

COMBINACIÒN DE RELACIONES:

```
// COMBINACIÒN DE RELACIONES
```

```
// (m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
// (m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie)
// (m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie)
// (m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie)
// (m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie)

CALL gds.graph.project(
  'myGraphAll',
  'Movie',
  {WATCH_TOG_SAME_CATEGORY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        },
    WATCH_TOG_SAME_COUNTRY:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        },
    WATCH_TOG_SAME_ACTOR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        },
    WATCH_TOG_SAME_DIRECTOR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        },
    WATCH_TOG_SAME_RELEASE_YEAR:
        {
            orientation: 'UNDIRECTED',
            Properties: 'quantity'
        }
    }
);
```

```
neo4j$ CALL gds.graph.project( 'myGraphAll', 'Movie', {WATCH_TOG_SAME_CATEGORY: { orientation: 'UNDIRECTED', Properties: 'quan…
```

nodeProjection	relationshipProjection	graphName	nodeCount	relationshipCount	projectMillis
{ "Movie": { "label": "Movie", "properties": { } } }	{ "WATCH_TOG_SAME_CATEGORY": { "aggregation": "DEFAULT", "orientation": "UNDIRECTED", "indexInverse": false, "properties": { "quantity": { "aggregation": "DEFAULT", "property": "quantity", "defaultValue": null } }, "type": "WATCH_TOG_SAME_CATEGORY" }, "WATCH_TOG_SAME_RELEASE_YEAR": { "aggregation": "DEFAULT",	"myGraphAll"	8807	773788	743

```
// Paso 2: Calculo la memoria necesaria:
CALL gds.nodeSimilarity.filtered.write.estimate('myGraphAll', {
    writeRelationshipType: 'SIMILAR_ALL',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodeCount, relationshipCount, bytesMin, bytesMax, requiredMemory;

// PASO 3: Aplicamos el algoritmo:
CALL gds.nodeSimilarity.filtered.stream('myGraphAll',
{sourceNodeFilter:'Movie' , targetNodeFilter:'Movie' } )
YIELD node1, node2, similarity
RETURN gds.util.asNode(node1).title AS Movie1,
gds.util.asNode(node2).title AS Movie2, similarity
ORDER BY similarity DESCENDING, Movie1, Movie2;

// Paso 4: Implementamos
CALL gds.nodeSimilarity.filtered.write('myGraphAll',{
    writeRelationshipType: 'SIMILAR_ALL',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodesCompared, relationshipsWritten;
```

```
CALL gds.nodeSimilarity.filtered.write('myGraphAll',{
    writeRelationshipType: 'SIMILAR_ALL',
    writeProperty: 'score',
    sourceNodeFilter:'Movie',
    targetNodeFilter:'Movie'
})
YIELD nodesCompared, relationshipsWritten;
```

nodesCompared	relationshipsWritten
8790	87900

```
// Paso 5: Probamos
MATCH (m:Movie)-[r:SIMILAR_ALL]->(m2:Movie)
RETURN m.title AS Movie1, m2.title AS Movie2, r.score AS
SimilarityScore
ORDER BY r.score DESC;
```

Creación de Tablero de Recomendaciones complejas

Codigo para tablero de recomendaciones Complejas (general)

```
//
:param neodash_movie_title => 'Black Mirror'

// top 5 same category con mayor degree
MATCH (m: Movie)
WHERE m.title = $neodash_movie_title
MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
WITH m2.degreeSameCategory AS sameCatDegree, m2.title as title
RETURN title, sameCatDegree
ORDER BY sameCatDegree DESC LIMIT 5;

// top 5 same category con mayor betweenes
MATCH (m: Movie)
WHERE m.title = $neodash_movie_title
MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
WITH m2.betweennessSameCategory AS sameCatBetweeness, m2.title as title
RETURN title, sameCatBetweeness
ORDER BY sameCatBetweeness DESC LIMIT 5;

// top 5 same category con mayor closeness
MATCH (m: Movie)
WHERE m.title = $neodash_movie_title
MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
WITH m2.closenessSameCategory AS sameCatCloseness, m2.title as title
RETURN title, sameCatCloseness
ORDER BY sameCatCloseness DESC LIMIT 5;

// top 5 same category con mayor page rank
MATCH (m: Movie)
```

```
  WHERE m.title = $neodash_movie_title
  MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
  WITH m2.pagerankSameCategory AS sameCatPageRank, m2.title as title
  RETURN title, sameCatPageRank
  ORDER BY sameCatPageRank DESC LIMIT 5;

  // top 5 same category con mayor triangle count
  MATCH (m: Movie)
  WHERE m.title = $neodash_movie_title
  MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
  WITH m2.trianglesSameCategory AS sameCatTriangle, m2.title as title
  RETURN title, sameCatTriangle
  ORDER BY sameCatTriangle DESC LIMIT 5;

  // top 5 same category con mayor clustering coefficient

  MATCH (m: Movie)
  WHERE m.title = $neodash_movie_title
  MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
  WITH m2.clusteringCoefficientSameCategory AS sameCatClusCoeff, m2.title as
title
  RETURN title, sameCatClusCoeff
  ORDER BY sameCatClusCoeff DESC LIMIT 5;

  // top 5 same category con mismo louvain y mayor degree

  MATCH (m: Movie)
  WHERE m.title = $neodash_movie_title
  MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
  WHERE m2.louvainSameCategory = m.louvainSameCategory
  WITH m2.degreeSameCategory AS sameCatDegree, m2.title as title
  RETURN title
  ORDER BY sameCatDegree DESC LIMIT 5;

  // top 5 same category con mismo louvain y mayor betweennes

  MATCH (m: Movie)
  WHERE m.title = $neodash_movie_title
  MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
  WHERE m2.louvainSameCategory = m.louvainSameCategory
  WITH m2.betweennessSameCategory AS sameCatBetweeness, m2.title as title
  RETURN title
```

```
  ORDER BY sameCatBetweeness DESC LIMIT 5;

  // top 5 same category con misma comunidad WCC y mayor clustering
coefficient

  MATCH (m: Movie)
  WHERE m.title = $neodash_movie_title
  MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
  WHERE m2.communityWCCSameCategory = m.communityWCCSameCategory
  WITH m2.clusteringCoefficientSameCategory AS sameCatClusCoeff, m2.title as
title
  RETURN title
  ORDER BY sameCatClusCoeff DESC LIMIT 5;

  // top 5 same category con misma comunidad WCC y mayor triangle count

  MATCH (m: Movie)
  WHERE m.title = $neodash_movie_title
  MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
  WHERE m2.communityWCCSameCategory = m.communityWCCSameCategory
  WITH m2.trianglesSameCategory AS sameCatTriangle, m2.title as title
  RETURN title
  ORDER BY sameCatTriangle DESC LIMIT 5;

  // top 5 same category con misma labelPropagation y mayor page rank

  MATCH (m: Movie)
  WHERE m.title = $neodash_movie_title
  MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
  WHERE m2.labelPropagationSameCategory = m.labelPropagationSameCategory
  WITH m2.pagerankSameCategory AS sameCatPageRank, m2.title as title
  RETURN title
  ORDER BY sameCatPageRank DESC LIMIT 5;

  // top 5 same category con misma labelPropagation y mayor closeness

  MATCH (m: Movie)
  WHERE m.title = $neodash_movie_title
  MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
  WHERE m2.labelPropagationSameCategory = m.labelPropagationSameCategory
  WITH m2.closenessSameCategory AS sameCatCloseness, m2.title as title
  RETURN title
```

```
ORDER BY sameCatCloseness DESC LIMIT 5;

// top 5 same category con misma labelPropagation y Louvain por degree

MATCH (m: Movie)
WHERE m.title = $neodash_movie_title
MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
WHERE m2.labelPropagationSameCategory = m.labelPropagationSameCategory
AND m2.louvainSameCategory = m.louvainSameCategory
WITH m2.degreeSameCategory AS sameCatDegree, m2.title as title
RETURN title
ORDER BY sameCatDegree DESC LIMIT 5;

// top 5 same category con misma comuniadWCC y Louvain por betweenes

MATCH (m: Movie)
WHERE m.title = $neodash_movie_title
MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
WHERE m2.communityWCCSameCategory = m.communityWCCSameCategory
AND m2.louvainSameCategory = m.louvainSameCategory
WITH m2.betweennessSameCategory AS sameCatBetweeness, m2.title as title
RETURN title
ORDER BY sameCatBetweeness DESC LIMIT 5;

// top 5 same category con misma comuniadWCC y labelPropagation por
clustering coeff

MATCH (m: Movie)
WHERE m.title = $neodash_movie_title
MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
WHERE m2.labelPropagationSameCategory = m.labelPropagationSameCategory
AND m2.communityWCCSameCategory = m.communityWCCSameCategory
WITH m2.clusteringCoefficientSameCategory AS sameCatClusCoeff, m2.title as
title
RETURN title
ORDER BY sameCatClusCoeff DESC LIMIT 5;

// top 5 same category con misma louvain, comuniadWCC y labelPropagation
por page rank

MATCH (m: Movie)
WHERE m.title = $neodash_movie_title
```

```
MATCH (m) -[r:WATCH_TOG_SAME_CATEGORY]-> (m2: Movie)
WHERE m2.labelPropagationSameCategory = m.labelPropagationSameCategory
AND m2.communityWCCSameCategory = m.communityWCCSameCategory
AND m2.louvainSameCategory = m.louvainSameCategory
WITH m2.pagerankSameCategory AS sameCatPageRank, m2.title as title
RETURN title
ORDER BY sameCatPageRank DESC LIMIT 5;
```

Tablero de recomendaciones Complejas (general)

Tablero de recomendaciones Complejas (por relaciones específicas)

```
// Archivo para el segundo tablero de CQL:

// Recomendaciones para la Relación:
(m:Movie)-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)

// Centralidad:
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WITH ParamTITLE, m2 ORDER BY m2.closenessSameCategory DESC
WITH ParamTITLE, collect(m2)[0] AS m_closeness

WITH ParamTITLE, m_closeness
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title
WITH ParamTITLE, m_closeness, m2 ORDER BY m2.betweennessSameCategory
DESC
WITH ParamTITLE, m_closeness, collect(m2)[0] AS m_between

WITH ParamTITLE, m_closeness, m_between
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title AND m2.title <> m_between.title
WITH m_closeness, m_between, m2 ORDER BY m2.pagerankSameCategory DESC
WITH m_closeness, m_between, collect(m2)[0] AS m_pagerank

WITH [
```

```
  {
    title: m_closeness.title,
    Parametro: "closenessSameCategory"
  },
  {
    title: m_between.title,
    Parametro: "betweennessSameCategory"
  },
  {
    title: m_pagerank.title,
    Parametro: "pagerankSameCategory"
  }
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;


// Comunidad
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m.communityWCCSameCategory = m2.communityWCCSameCategory
WITH ParamTITLE, m2 ORDER BY m2.trianglesSameCategory DESC
WITH ParamTITLE, collect(m2)[0] AS m_trianglesWCC

WITH ParamTITLE, m_trianglesWCC
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_trianglesWCC.title AND m.louvainSameCategory =
m2.louvainSameCategory
WITH ParamTITLE, m_trianglesWCC, m2 ORDER BY m2.trianglesSameCategory
DESC
WITH ParamTITLE, m_trianglesWCC, collect(m2)[0] AS m_louvainTriangle

WITH ParamTITLE, m_trianglesWCC, m_louvainTriangle
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
```

```
WHERE m2.title <> m_trianglesWCC.title AND m2.title <>
m_louvainTriangle.title
                        AND m.labelPropagationSameCategory =
m2.labelPropagationSameCategory
WITH m_trianglesWCC, m_louvainTriangle, m2 ORDER BY
m2.trianglesSameCategory DESC
WITH m_trianglesWCC, m_louvainTriangle, collect(m2)[0] AS
m_labeltriangles

WITH [
  {
    title: m_trianglesWCC.title,
    Parametro: "CommunitySameCategory"
  },
  {
    title: m_louvainTriangle.title,
    Parametro: "CommunitySameCategory"
  },
  {
    title: m_labeltriangles.title,
    Parametro: "CommunitySameCategory"
  }
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;

// Similaridad:
MATCH
(m:Movie{title:$neodash_movie_title})-[r:SIMILAR_CATEGORY]->(m2:Movie)
ORDER BY r.score DESC
LIMIT 3
RETURN m2.title AS title,
    "SimilarSameCategory" AS Parametro;


// Recomendaciones para la Relación:
(m:Movie)-[r:WATCH_TOG_SAME_COUNTRY]->(m2:Movie)

// Centralidad:
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
```

```
MATCH (m:Movie {title: ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WITH ParamTITLE, m2 ORDER BY m2.closenessSameCountry DESC
WITH ParamTITLE, collect(m2)[0] AS m_closeness

WITH ParamTITLE, m_closeness
WHERE ParamTITLE <> ""
MATCH (m:Movie {title: ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title
WITH ParamTITLE, m_closeness, m2 ORDER BY m2.betweennessSameCountry DESC
WITH ParamTITLE, m_closeness, collect(m2)[0] AS m_between

WITH ParamTITLE, m_closeness, m_between
WHERE ParamTITLE <> ""
MATCH (m:Movie {title: ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title AND m2.title <> m_between.title
WITH m_closeness, m_between, m2 ORDER BY m2.pagerankSameCountry DESC
WITH m_closeness, m_between, collect(m2)[0] AS m_pagerank

WITH [
  {
    title: m_closeness.title,
    Parametro: "closenessSameCountry"
  },
  {
    title: m_between.title,
    Parametro: "betweennessSameCountry"
  },
  {
    title: m_pagerank.title,
    Parametro: "pagerankSameCountry"
  }
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;


// Comunidad
```

```
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m.communityWCCSameCountry = m2.communityWCCSameCountry
WITH ParamTITLE, m2 ORDER BY m2.trianglesSameCountry DESC
WITH ParamTITLE, collect(m2)[0] AS m_trianglesWCC

WITH ParamTITLE, m_trianglesWCC
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_trianglesWCC.title AND m.louvainSameCountry =
m2.louvainSameCountry
WITH ParamTITLE, m_trianglesWCC, m2 ORDER BY m2.trianglesSameCountry
DESC
WITH ParamTITLE, m_trianglesWCC, collect(m2)[0] AS m_louvainTriangle

WITH ParamTITLE, m_trianglesWCC, m_louvainTriangle
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_trianglesWCC.title AND m2.title <>
m_louvainTriangle.title
    AND m.labelPropagationSameCountry = m2.labelPropagationSameCountry
WITH m_trianglesWCC, m_louvainTriangle, m2 ORDER BY
m2.trianglesSameCountry DESC
WITH m_trianglesWCC, m_louvainTriangle, collect(m2)[0] AS
m_labeltriangles

WITH [
  {
    title: m_trianglesWCC.title,
    Parametro: "CommunitySameCountry"
  },
  {
    title: m_louvainTriangle.title,
    Parametro: "CommunitySameCountry"
  },
  {
    title: m_labeltriangles.title,
    Parametro: "CommunitySameCountry"
  }
```

```
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;

// Similaridad:
MATCH
(m:Movie{title:$neodash_movie_title})-[r:SIMILAR_COUNTRY]->(m2:Movie)
ORDER BY r.score DESC
LIMIT 3
RETURN m2.title AS title,
    "SimilarSameCountry" AS Parametro;



//          Recomendaciones          para          la          Relación:
(m:Movie)-[r:WATCH_TOG_SAME_ACTOR]->(m2:Movie)

// Centralidad:
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
MATCH                          (m:Movie                          {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WITH ParamTITLE, m2 ORDER BY m2.closenessSameActor DESC
WITH ParamTITLE, collect(m2)[0] AS m_closeness

WITH ParamTITLE, m_closeness
WHERE ParamTITLE <> ""
MATCH                          (m:Movie                          {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title
WITH ParamTITLE, m_closeness, m2 ORDER BY m2.betweennessSameActor DESC
WITH ParamTITLE, m_closeness, collect(m2)[0] AS m_between

WITH ParamTITLE, m_closeness, m_between
WHERE ParamTITLE <> ""
MATCH                          (m:Movie                          {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title AND m2.title <> m_between.title
WITH m_closeness, m_between, m2 ORDER BY m2.pagerankSameActor DESC
WITH m_closeness, m_between, collect(m2)[0] AS m_pagerank
```

```
WITH [
  {
    title: m_closeness.title,
    Parametro: "closenessSameActor"
  },
  {
    title: m_between.title,
    Parametro: "betweennessSameActor"
  },
  {
    title: m_pagerank.title,
    Parametro: "pagerankSameActor"
  }
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;


// Comunidad
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m.communityWCCSameActor = m2.communityWCCSameActor
WITH ParamTITLE, m2 ORDER BY m2.trianglesSameActor DESC
WITH ParamTITLE, collect(m2)[0] AS m_trianglesWCC

WITH ParamTITLE, m_trianglesWCC
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_trianglesWCC.title AND m.louvainSameActor =
m2.louvainSameActor
WITH ParamTITLE, m_trianglesWCC, m2 ORDER BY m2.trianglesSameActor DESC
WITH ParamTITLE, m_trianglesWCC, collect(m2)[0] AS m_louvainTriangle

WITH ParamTITLE, m_trianglesWCC, m_louvainTriangle
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
```

```
WHERE m2.title <> m_trianglesWCC.title AND m2.title <>
m_louvainTriangle.title
    AND m.labelPropagationSameActor = m2.labelPropagationSameActor
WITH m_trianglesWCC, m_louvainTriangle, m2 ORDER BY
m2.trianglesSameActor DESC
WITH m_trianglesWCC, m_louvainTriangle, collect(m2)[0] AS
m_labeltriangles

WITH [
  {
    title: m_trianglesWCC.title,
    Parametro: "CommunitySameActor"
  },
  {
    title: m_louvainTriangle.title,
    Parametro: "CommunitySameActor"
  },
  {
    title: m_labeltriangles.title,
    Parametro: "CommunitySameActor"
  }
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;

// Similaridad:
MATCH
(m:Movie{title:$neodash_movie_title})-[r:SIMILAR_ACTOR]->(m2:Movie)
ORDER BY r.score DESC
LIMIT 3
RETURN m2.title AS title,
    "SimilarSameActor" AS Parametro;


// Recomendaciones para la Relación:
(m:Movie)-[r:WATCH_TOG_SAME_DIRECTOR]->(m2:Movie)

// Centralidad:
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
```

```
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WITH ParamTITLE, m2 ORDER BY m2.closenessSameDirector DESC
WITH ParamTITLE, collect(m2)[0] AS m_closeness

WITH ParamTITLE, m_closeness
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title
WITH ParamTITLE, m_closeness, m2 ORDER BY m2.betweennessSameDirector
DESC
WITH ParamTITLE, m_closeness, collect(m2)[0] AS m_between

WITH ParamTITLE, m_closeness, m_between
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title AND m2.title <> m_between.title
WITH m_closeness, m_between, m2 ORDER BY m2.pagerankSameDirector DESC
WITH m_closeness, m_between, collect(m2)[0] AS m_pagerank

WITH [
  {
    title: m_closeness.title,
    Parametro: "closenessSameDirector"
  },
  {
    title: m_between.title,
    Parametro: "betweennessSameDirector"
  },
  {
    title: m_pagerank.title,
    Parametro: "pagerankSameDirector"
  }
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;


// Comunidad
```

```
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m.communityWCCSameDirector = m2.communityWCCSameDirector
WITH ParamTITLE, m2 ORDER BY m2.trianglesSameDirector DESC
WITH ParamTITLE, collect(m2)[0] AS m_trianglesWCC

WITH ParamTITLE, m_trianglesWCC
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_trianglesWCC.title AND m.louvainSameDirector =
m2.louvainSameDirector
WITH ParamTITLE, m_trianglesWCC, m2 ORDER BY m2.trianglesSameDirector
DESC
WITH ParamTITLE, m_trianglesWCC, collect(m2)[0] AS m_louvainTriangle

WITH ParamTITLE, m_trianglesWCC, m_louvainTriangle
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_trianglesWCC.title AND m2.title <>
m_louvainTriangle.title
AND m.labelPropagationSameDirector =
m2.labelPropagationSameDirector
WITH m_trianglesWCC, m_louvainTriangle, m2 ORDER BY
m2.trianglesSameDirector DESC
WITH m_trianglesWCC, m_louvainTriangle, collect(m2)[0] AS
m_labeltriangles

WITH [
  {
    title: m_trianglesWCC.title,
    Parametro: "CommunitySameDirector"
  },
  {
    title: m_louvainTriangle.title,
    Parametro: "CommunitySameDirector"
  },
  {
    title: m_labeltriangles.title,
    Parametro: "CommunitySameDirector"
```

```
  }
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;

// Similaridad:
MATCH
(m:Movie{title:$neodash_movie_title})-[r:SIMILAR_DIRECTOR]->(m2:Movie)
ORDER BY r.score DESC
LIMIT 3
RETURN m2.title AS title,
    "SimilarSameDirector" AS Parametro;


//          Recomendaciones          para          la          Relación:
(m:Movie)-[r:WATCH_TOG_SAME_RELEASE_YEAR]->(m2:Movie)

// Centralidad:
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
MATCH                          (m:Movie                          {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WITH ParamTITLE, m2 ORDER BY m2.closenessSameReleaseYear DESC
WITH ParamTITLE, collect(m2)[0] AS m_closeness

WITH ParamTITLE, m_closeness
WHERE ParamTITLE <> ""
MATCH                          (m:Movie                          {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title
WITH ParamTITLE, m_closeness, m2 ORDER BY m2.betweennessSameReleaseYear
DESC
WITH ParamTITLE, m_closeness, collect(m2)[0] AS m_between

WITH ParamTITLE, m_closeness, m_between
WHERE ParamTITLE <> ""
MATCH                          (m:Movie                          {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_closeness.title AND m2.title <> m_between.title
WITH  m_closeness,  m_between,  m2  ORDER  BY  m2.pagerankSameReleaseYear
DESC
```

```
WITH m_closeness, m_between, collect(m2)[0] AS m_pagerank

WITH [
  {
    title: m_closeness.title,
    Parametro: "closenessSameReleaseYear"
  },
  {
    title: m_between.title,
    Parametro: "betweennessSameReleaseYear"
  },
  {
    title: m_pagerank.title,
    Parametro: "pagerankSameReleaseYear"
  }
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;

// Comunidad
WITH $neodash_movie_title AS ParamTITLE
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m.communityWCCSameReleaseYear = m2.communityWCCSameReleaseYear
WITH ParamTITLE, m2 ORDER BY m2.trianglesSameReleaseYear DESC
WITH ParamTITLE, collect(m2)[0] AS m_trianglesWCC

WITH ParamTITLE, m_trianglesWCC
WHERE ParamTITLE <> ""
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_trianglesWCC.title AND m.louvainSameReleaseYear =
m2.louvainSameReleaseYear
WITH ParamTITLE, m_trianglesWCC, m2 ORDER BY
m2.trianglesSameReleaseYear DESC
WITH ParamTITLE, m_trianglesWCC, collect(m2)[0] AS m_louvainTriangle

WITH ParamTITLE, m_trianglesWCC, m_louvainTriangle
WHERE ParamTITLE <> ""
```

```
MATCH (m:Movie {title:
ParamTITLE})-[r:WATCH_TOG_SAME_CATEGORY]->(m2:Movie)
WHERE m2.title <> m_trianglesWCC.title AND m2.title <>
m_louvainTriangle.title
AND m.labelPropagationSameReleaseYear =
m2.labelPropagationSameReleaseYear
WITH m_trianglesWCC, m_louvainTriangle, m2 ORDER BY
m2.trianglesSameReleaseYear DESC
WITH m_trianglesWCC, m_louvainTriangle, collect(m2)[0] AS
m_labeltriangles

WITH [
  {
    title: m_trianglesWCC.title,
    Parametro: "CommunitySameReleaseYear"
  },
  {
    title: m_louvainTriangle.title,
    Parametro: "CommunitySameReleaseYear"
  },
  {
    title: m_labeltriangles.title,
    Parametro: "CommunitySameReleaseYear"
  }
] AS recomendaciones

UNWIND recomendaciones AS rec
RETURN rec.title AS title,
    rec.Parametro AS Parametro;

// Similaridad:
MATCH
(m:Movie{title:$neodash_movie_title})-[r:SIMILAR_RELEASE_YEAR]->(m2:Mov
ie)
ORDER BY r.score DESC
LIMIT 3
RETURN m2.title AS title,
    "SimilarSameReleaseYear" AS Parametro;



// Recomendación General:
```

```
MATCH (m:Movie{title:$neodash_movie_title})-[r:SIMILAR_ALL]->(m2:Movie)
ORDER BY r.score DESC
LIMIT 3
RETURN m2.title AS title,
    "Similar_ALL" AS Parametro;
```

Seleccionar el título de la película

Movie title: How to Train Your Dragon 2

Recomendación por Centralidad (Categoria)

title	Parametro
The Karate Kid Part II	closenessSameCategory
The Karate Kid Part III	betweennessSameCategory
Madagascar: Escape 2 Africa	pagerankSameCategory

Rows per page: 5 1–3 of 3

Recomendación por Comunidades (Categoria)

title	Parametro
The Karate Kid Part II	CommunitySameCategory
Madagascar: Escape 2 Africa	CommunitySameCategory
Hotel Transylvania 3: Summer Vacation	CommunitySameCategory

Recomendación por Similaridad (Categoria)

title	Parametro
The Karate Kid	SimilarSameCategory
Hotel Transylvania 3: Summer Vacation	SimilarSameCategory
Monster House	SimilarSameCategory

Recomendación por Centralidad (Pais)

title	Parametro
Garuda in My Heart 2	closenessSameCountry
Kung Fu Panda 2	betweennessSameCountry
Hotel for Dogs	pagerankSameCountry

Rows per page: 5 1–3 of 3

Recomendación por Comunidades (Pais)

title	Parametro
Hotel Transylvania 3: Summer Vacation	CommunitySameCountry
The Karate Kid	CommunitySameCountry
Madagascar: Escape 2 Africa	CommunitySameCountry

Rows per page: 5 1–3 of 3

Recomendación por Similaridad (Pais)

title	Parametro
You	SimilarSameCountry
Hotel Transylvania 3: Summer Vacation	SimilarSameCountry
Avengers: Infinity War	SimilarSameCountry

Recomendación por Centralidad (Mismo Actor)

title	Parametro
Barbie in Rock 'N Royals	closenessSameActor
Hotel Transylvania 3: Summer Vacation	betweennessSameActor
Kung Fu Panda 2	pagerankSameActor

Recomendación por Comunidades (Actor)

title	Parametro
Hotel Transylvania 3: Summer Vacation	CommunitySameActor
ParaNorman	CommunitySameActor
Mowgli: Legend of the Jungle	CommunitySameActor

Rows per page: 5 1–3 of 3

Recomendación por Similaridad (Actor)

title	Parametro
Avengers: Infinity War	SimilarSameActor
Thor: Ragnarok	SimilarSameActor
Black Panther	SimilarSameActor

Rows per page: 5 1–3 of 3

Recomendación por Centralidad (Mismo Director)

title	Parametro
Motu Patlu Kung Fu Kings 4 The Challenge of Kun...	closenessSameDirector
Penguins of Madagascar: The Movie	betweennessSameDirector
The Karate Kid Part III	pagerankSameDirector

Recomendación por Comunidades (Director)

```
Query returned no data.
```

Recomendación por Similaridad (Mismo Director)

```
Query returned no data.
```

Recomendación por Centralidad (Año de Lanzamiento)

title	Parametro
Chhota Bheem - Neeli Pahaadi	closenessSameReleaseYear
Hotel Transylvania 3: Summer Vacation	betweennessSameReleaseYear
Madagascar 3: Europe's Most Wanted	pagerankSameReleaseYear

Rows per page: 5 1–3 of 3

Recomendación por Comunidades (Año de Lanzamiento)

title	Parametro
Penguins of Madagascar: The Movie	CommunitySameReleaseYear
Santa Claws	CommunitySameReleaseYear
Yellowbird	CommunitySameReleaseYear

Recomendación por Similaridad (Año de Lanzamiento)

title	Parametro
Transcendence	SimilarSameReleaseYear
Hunter X Hunter (2011)	SimilarSameReleaseYear
Penguins of Madagascar: The Movie	SimilarSameReleaseYear

Recomendación por Similaridad (TODAS LAS RELACIONES)

title	Parametro
The Karate Kid	Similar_ALL
Penguins of Madagascar: The Movie	Similar_ALL
Monster House	Similar_ALL

Rows per page: 5 1–3 of 3

Conclusiones y recomendaciones:

Se recolectaron datos de la base de datos de Netflix. Sin embargo, a esa información se le agregó mediante Python la generación de perfiles ficticios y relaciones “ADD” randoms.

Gracias a la notación intuitiva de nodos y relaciones, el lenguaje Cypher es mucho más sencillo que sql; más aún si se aumenta el número de joins o un patrón de búsqueda. Además, el lenguaje SQL resulta ser menos eficiente, al tener que recorrer todas las tablas mediante joins [3].

Además de la justificación anterior hay que agregar que nuestras recomendaciones no están basadas en las relaciones “ADD” y tampoco en aquellas películas que fueron vistas por el mismo perfil. En su lugar, nuestras recomendaciones están basadas en casos más específicos que parten de aquellas películas que fueron ADD por el mismo perfil, y que además cumplen otras condiciones, cómo puede ser: Tener la misma categoría, tener el mismo director, el año de lanzamiento, etc.

Conclusiones en referencia a los hallazgos:

En base a todo el análisis realizado y documentado en este escrito, es que podemos establecer y obtener las siguientes conclusiones:

- Si bien aunque se hicieron recomendaciones aleatorias hubo una selección de películas “elegidas” las cuales elegimos para representar las más populares de la plataforma:
    - Black Mirror
    - Avengers: Infinity War
    - How to Train Your Dragon 2
    - Madagascar: Escape 2 Africa
    - Once Upon a Time
    - The Karate Kid
    - Hotel Transylvania 3: Summer Vacation
    - You
    - Penguins of Madagascar: The Movie
    - Spider-Man 3
    - The Devil All The Time
    - Twilight
    - School of Rock
    - Kung Fu Panda 2
    - A Family
    - Demon Slayer: Kimetsu no Yaiba
    - Jurassic World Camp Cretaceous
    - ICarly
    - Men in Black
    - Sex Education
    - Men in Black II
    - Merlí
    - Grown Ups
    - 13 Reasons Why
    - Transcendence
    - Cobra Kai
    - The Pianist
    - The 100
    - Thor: Ragnarok

- ➢ The Seven Deadly Sins
- ➢ Lucifer
- ➢ The Help
- ➢ Hotel for Dogs
- ➢ Spider-Man: Into the Spider-Verse
- ➢ Black Panther
- ➢ The Karate Kid Part II
- ➢ Luis Miguel - The Series
- ➢ Hotel Transylvania
- ➢ Orange Is the New Black
- ➢ Never Have I Ever
- ➢ Control Z
- ➢ Madagascar 3: Europe's Most Wanted
- ➢ Hunter X Hunter (2011)
- ➢ Attack on Titan
- ➢ The Fosters
- ➢ Squid Game
- ➢ The Walking Dead
- ➢ Monster House
- ➢ The Karate Kid Part III
- ➢ Good Girls

❖ Resulta esperable que al pedir el top de recomendaciones de películas salga dos veces la misma relación pero con los nodos puestos en sentidos opuestos, la razón por la cual lo dejamos establecido así es para no perder la información de que relación es la que genera ese efecto.

❖ Dadas las siguientes consultas podemos establecer que hay ciertas películas que no aplican para hacer recomendaciones, y que lo más óptimo sería aplicar otro criterio que no sea el de comunidades:

- ➢ 407 películas para Películas vistas en conjunto que tienen la misma categoría.

```
MATCH (m:Movie)
WITH m.communityWCCSameCategory as numero_de_WCC,
    COUNT(m.title) AS Miembros
WHERE Miembros = 1
RETURN COUNT(Miembros) AS Total_Peliculas_Aisladas;
```

Total_Peliculas_Aisladas
407

- ➢ 2454 Películas para películas vistas por películas pertenecientes al mismo país.

```
MATCH (m:Movie)
WITH m.communityWCCSameCountry as numero_de_WCC,
    COUNT(m.title) AS Miembros
WHERE Miembros = 1
RETURN COUNT(Miembros) AS Total_Peliculas_Aisladas;
```

Total_Peliculas_Aisladas
2454

- ➢ 7483 películas para: Películas vistas en conjunto en los que actúa el mismo actor.

```
MATCH (m:Movie)
WITH m.communityWCCSameActor as numero_de_WCC,
    COUNT(m.title) AS Miembros
WHERE Miembros = 1
RETURN COUNT(Miembros) AS Total_Peliculas_Aisladas;
```

Total_Peliculas_Aisladas
7483

- ➢ 6761 películas para: Películas vistas en conjunto que son dirigidas por el mismo director.

```
MATCH (m:Movie)
WITH m.communityWCCSameDirector as numero_de_WCC,
    COUNT(m.title) AS Miembros
WHERE Miembros = 1
RETURN COUNT(Miembros) AS Total_Peliculas_Aisladas;
```

Total_Peliculas_Aisladas
8761

- ➢ 1239 películas para: Películas vistas en conjunto con otras películas estrenadas el mismo año.

```
MATCH (m:Movie)
WITH m.communityWCCSameReleaseYear as numero_de_WCC,
    COUNT(m.title) AS Miembros
WHERE Miembros = 1
RETURN COUNT(Miembros) AS Total_Peliculas_Aisladas;
```

Total_Peliculas_Aisladas
1239

- ❖ Aunque hayamos simulado las relaciones de la plataforma de Netflix, es interesante ver cómo aún así los algoritmos de Similaridad si recomiendan cosas con sentido en la vida real.
  - ➢ Probablemente se deba a que más allá de la cantidad de relaciones esas películas o series si cuentan con estructuras muy parecidas.
- ❖ Creamos 2 tableros de recomendaciones complejas para poder realizar distintas perspectivas sobre las recomendaciones:
  - ➢ Una más aplicada a la relación más importante: WATCH_TOG_SAME_CATEGORY
  - ➢ Otra aplicada a cada relación con su respectiva similaridad, centralidad y comunidad.

Recomendaciones:

- ❖ Sería conveniente poder completar las relaciones “IS_CITIZEN_OF” Dado que esto nos permitiría ver si los ADD por países corresponden a películas o series realizadas por actores del mismo lugar, lo cual puede jugar con un factor clave en la industria del arte: la nostalgia.
- ❖ Sería muy fructífero agregar idiomas y poder hacer un top con eso, dado que muchas películas o series están condicionadas por el lenguaje en el cual se escuchan.
  - ➢ Principalmente hay casos en dónde las traducciones son muy malas, pero en los lenguajes originales son muy buenas películas.
    - ■ Por consiguiente se podría llevar a cabo un análisis de este fenómeno.

Referencias bibliográficas:

1. Ortega Guzmán, V. Gutiérrez Preciado, L. Cervantes, F. Alcaraz Mejia, M. (2024). *A Methodology for Knowledge Discovery in Labeled and Heterogeneous Graphs*. Electronic, Systems and Informatics Department, ITESO - The Jesuit University of Guadalajara.
2. Partida Ochoa, S. (2021). *Sistema de recomendación basado en grafos: Escenario de recomendación de productos* [Tesis de maestría, Instituto Tecnológico y de Estudios Superiores de Occidente]. REI ITESO.
3. Gutiérrez, J., & Flórez, C. (2021). *Analítica de grafos para identificar entidades relevantes y comunidades en Mercado Libre: un estudio de caso*. Revista Mutis, 11(1), 77-95.
4. Needham, M., & Hodler, A. E. (2019). *Graph algorithms: Practical examples in Apache Spark and Neo4j*. O'Reilly Media.
5. Neo4j. (s.f.). Degree centrality. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/degree-centrality/
6. Neo4j. (s.f.). Closeness centrality. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/closeness-centrality/
7. Neo4j. (s.f.). Betweenness centrality. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/betweenness-centrality/
8. Neo4j. (s.f.). Page Rank. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/page-rank/
9. Neo4j. (s.f.). Articulation Points. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/articulation-points/
10. Neo4j. (s.f.). Bridges. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/bridges/
11. Neo4j. (s.f.). Triangle count. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/triangle-count/
12. Neo4j. (s.f.). Local clustering coefficient. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/local-clustering-coefficient/
13. Neo4j. (s.f.). Strongly connected components. Neo4j Graph Data Science Documentation.https://neo4j.com/docs/graph-data-science/2.12/algorithms/strongly-connected-components/
14. Neo4j. (s.f.). Weakly connected components. Neo4j Graph Data Science Documentation.https://neo4j.com/docs/graph-data-science/2.12/algorithms/wcc/
15. Neo4j. (s.f.). Label propagation. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/label-propagation/
16. Neo4j. (s.f.). K-1 Coloring. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/k1coloring/
17. Neo4j. (s.f.). Modularity Metric. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/modularity/
18. Neo4j. (s.f.). Louvain. Neo4j Graph Data Science Documentation. https://neo4j.com/docs/graph-data-science/2.12/algorithms/louvain/