Universidad Da Vinci De Guatemala

Facultad de Ingeniería

Carrera: Licenciatura en Ingeniera en Sistemas

“Examen Parcial – Bases de Datos II”

Hugo Ronaldo Breganza Rodas

202301727

Curso: Bases de Datos 2 - Grupo 1 - LISCC-L

Guatemala, octubre 2025

SECCIÓN I – CONCEPTOS FUNDAMENTALES

1. ¿Qué es una base de datos basada en grafos y dos ventajas sobre una base relacional tradicional?

Una base de datos basada en grafos es un sistema que almacena información usando nodos (entidades), relaciones (conexiones) y propiedades (atributos). En lugar de usar tablas y llaves foráneas como en una base relacional, un grafo guarda directamente cómo están conectadas las cosas.

Dos ventajas frente a una base relacional tradicional son:

- **Rendimiento en relaciones complejas:** Cuando necesito responder preguntas como "¿quién conoce a quién a través de quién?" o "amigos de mis amigos", un grafo lo resuelve muy rápido porque las relaciones están almacenadas como primer nivel de datos. En SQL normal, esto implicaría muchos JOIN que se vuelven lentos cuando hay mucho volumen.
- **Modelo más natural para redes:** En una red social, lo más importante son las conexiones entre usuarios, gustos, grupos, etc. En un grafo ese modelo es directo y fácil de entender visualmente. En cambio, en una base relacional hay que partirlo en muchas tablas intermedias y es más difícil mantener y escalar ese tipo de consultas.

2. ¿Qué tipo de problemas resuelve Redis y por qué se clasifica como base de datos clave-valor?

Redis está pensado para almacenar y acceder datos muy rápido en memoria (RAM). Se usa para información que cambia rápido o que necesito consultar miles de veces por segundo, como sesiones de usuario, caché de resultados, contadores, rankings, últimos vistos, etc.

Se llama base de datos clave-valor porque todo en Redis se accede usando una "clave" única (key), y esa clave apunta a un "valor". Ese valor puede ser de distintos tipos nativos como STRING, LIST, SET, HASH, etc. Pero la idea central es: yo pregunto por una clave exacta y Redis me devuelve el valor asociado sin tener que hacer búsquedas complejas.

Resumiendo, podemos decir que Redis resuelve problemas de velocidad, disponibilidad inmediata y datos temporales / de alta rotación, por eso es tan usado junto con otros motores.

3. ¿Cuál es la diferencia entre un nodo y una relación en Neo4j?

- **Nodo:** representa una entidad/objeto del mundo real. Ejemplo: un Estudiante, un GrupoDeEstudio, un Profesor, un Curso.
- **Relación:** representa cómo dos nodos están conectados y de qué forma se relacionan. Ejemplo: (Estudiante)-[:SIGUE]->(Estudiante), o (GrupoDeEstudio)-[:ASOCIADO_A]->(Curso).

Importante: en Neo4j las relaciones también son datos de primera clase. O sea, no son "solo un enlace", también pueden tener propiedades. Eso hace que no solo sepamos que A conoce a B, sino desde cuándo, con qué peso, nivel de afinidad, etc.

4. ¿Qué son las propiedades en Neo4j y cómo se aplican a un modelo de red social?

Las propiedades son pares campo = valor que se guardan dentro de nodos y también dentro de relaciones. Sirven para describir más a fondo.

Por ejemplo:

- Nodo Estudiante podría tener propiedades: {nombre: "Juan Pérez", carrera: "Ingeniería en Sistemas", semestre: 5}.
- Relación [:SIGUE] podría tener propiedad {desde: "2025-02-10"} para saber desde cuándo un estudiante sigue a otro.

En una red social, las propiedades permiten personalizar la experiencia. Por ejemplo, en un grupo de estudio puedo guardar {tema: "Bases de Datos II", horario: "Noches", modalidad: "Virtual"}, y en una relación [:RECOMIENDA] entre un Estudiante y un Profesor puedo guardar {comentario: "Explica claro", rating: 5}. Eso hace posible filtrar, recomendar y puntuar.

SECCIÓN II – APLICACIÓN PRÁCTICA

Parte A – Modelado en Neo4j

Condiciones del caso ConnectU:

- Cada Estudiante puede [:SIGUE] a otros Estudiantes.
- Los Estudiantes pueden [:PERTENECE_A] uno o varios GruposDeEstudio.
- Los GruposDeEstudio están relacionados con un Curso mediante [:ASOCIADO_A].
- Los Estudiantes pueden [:RECOMIENDA] a un Profesor.

a) Describir el grafo con pseudonodos y relaciones

Podemos representarlo así (formato tipo diagrama textual):

(Estudiante)-[:SIGUE]->(Estudiante)

(Estudiante)-[:PERTENECE_A]->(GrupoDeEstudio)

(GrupoDeEstudio)-[:ASOCIADO_A]->(Curso)

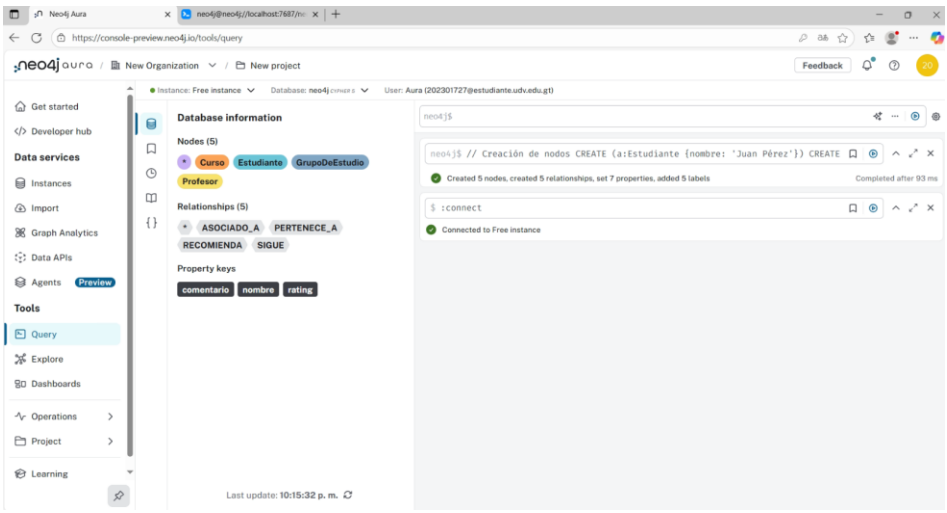
(Estudiante)-[:RECOMIENDA {comentario, rating}]->(Profesor)

Ejemplo concreto con nombres ficticios:
(Estudiante {nombre:"Juan Pérez"})-[:SIGUE]->(Estudiante {nombre:"Ana López"})

(Estudiante {nombre:"Juan Pérez"})-[:PERTENECE_A]->(GrupoDeEstudio {nombre:"BD2 - Sección B"})

(GrupoDeEstudio {nombre:"BD2 - Sección B"})-[:ASOCIADO_A]->(Curso {nombre:"Bases de Datos II"})

(Estudiante {nombre:"Ana López"})-[:RECOMIENDA {rating:5, nota:"Excelente"}]->(Profesor {nombre:"Carlos García"})



b) Consultas Cypher

i) Encontrar los compañeros de grupo de "Juan Pérez".

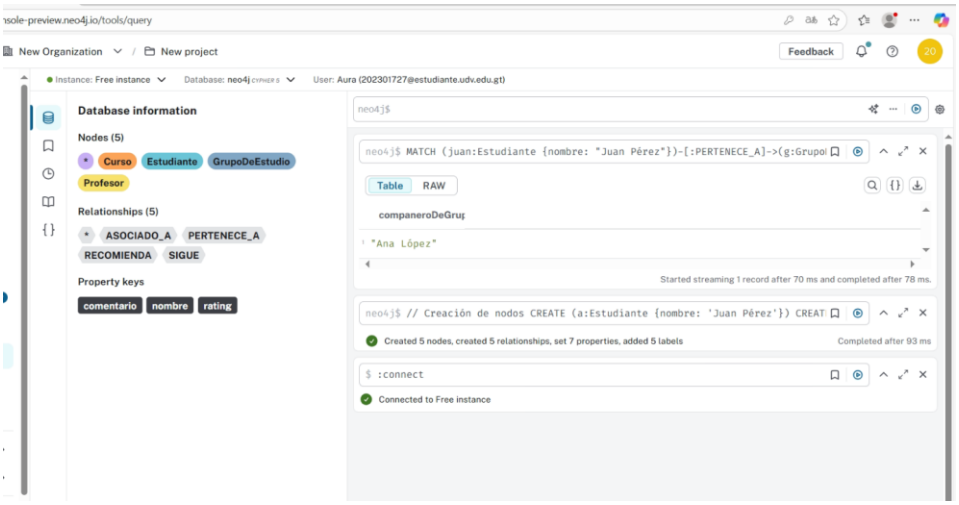
Objetivo: ver otros estudiantes que pertenecen a los mismos grupos que Juan Pérez.

MATCH (juan:Estudiante {nombre: "Juan Pérez"})-[:PERTENECE_A]->(g:GrupoDeEstudio)<-[:PERTENECE_A]-(compa:Estudiante)

WHERE compa <> juan

RETURN DISTINCT compa.nombre AS companeroDeGrupo;

Explicación rápida: buscamos los grupos de Juan, luego vemos quién más está en esos mismos grupos, excluyendo al propio Juan.

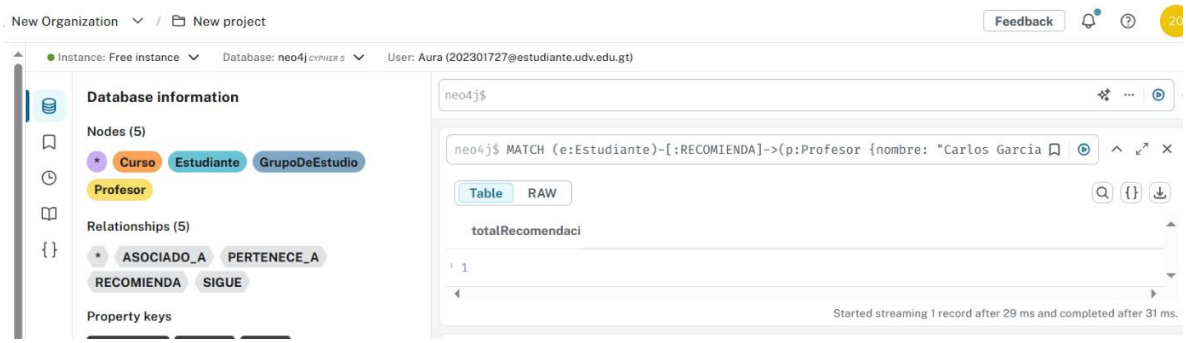


ii) Contar cuántos estudiantes recomiendan al profesor "Carlos García".

MATCH (e:Estudiante)-[:RECOMIENDA]->(p:Profesor {nombre: "Carlos García"})

RETURN COUNT(DISTINCT e) AS totalRecomendaciones;

Esto devuelve cuántos estudiantes únicos han creado una relación RECOMIENDA hacia ese profesor.

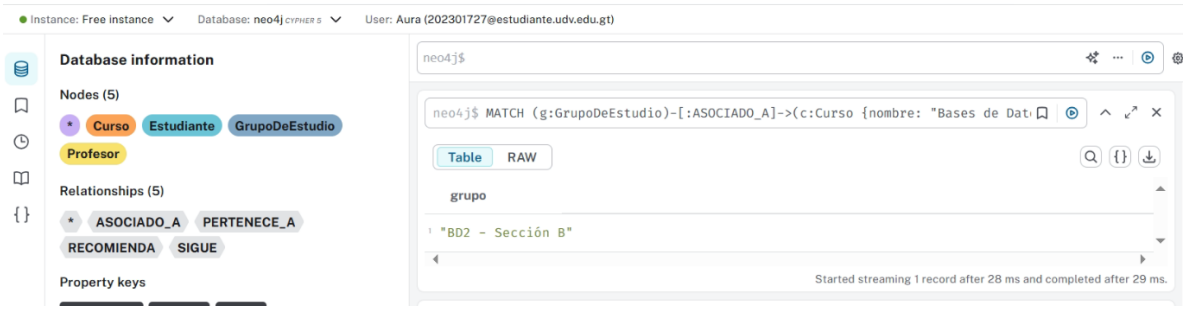


iii) Obtener los grupos asociados al curso "Bases de Datos II".

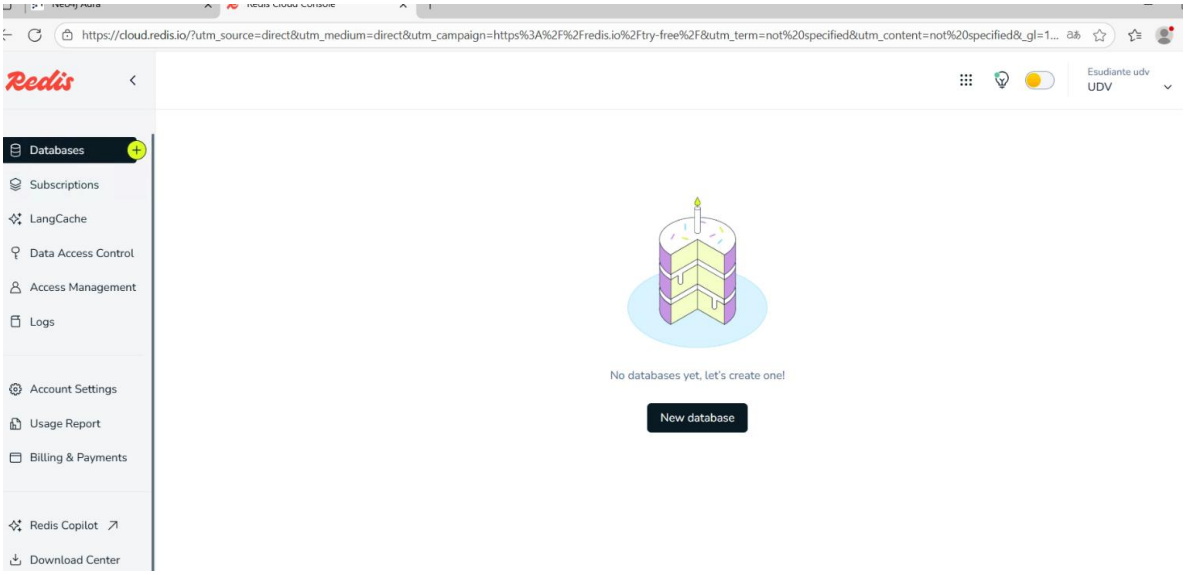
MATCH (g:GrupoDeEstudio)-[:ASOCIADO_A]->(c:Curso {nombre: "Bases de Datos II"})

RETURN DISTINCT g.nombre AS grupo, g

Aquí listamos cuáles grupos de estudio están vinculados con ese curso.

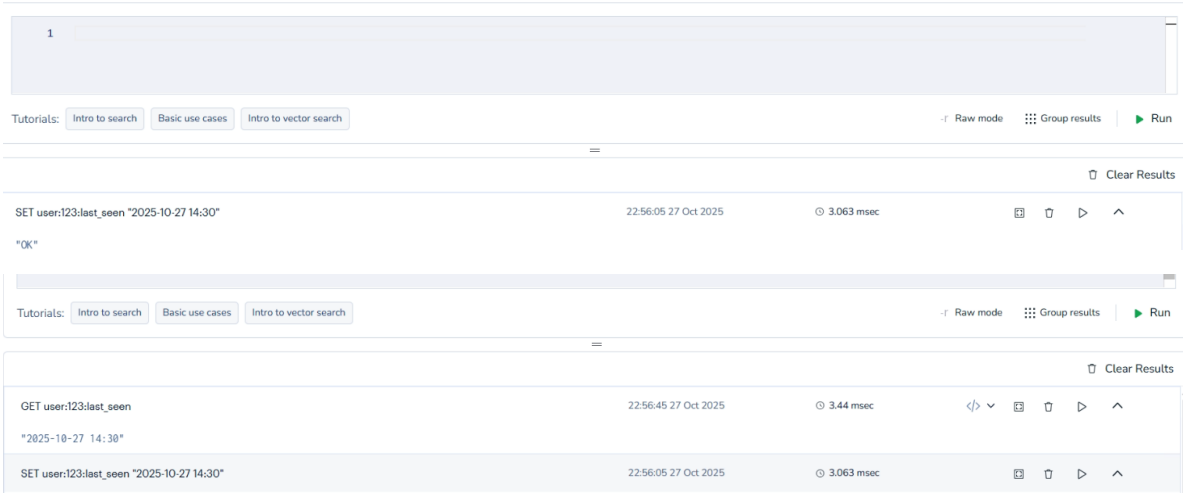


Parte B – Redis

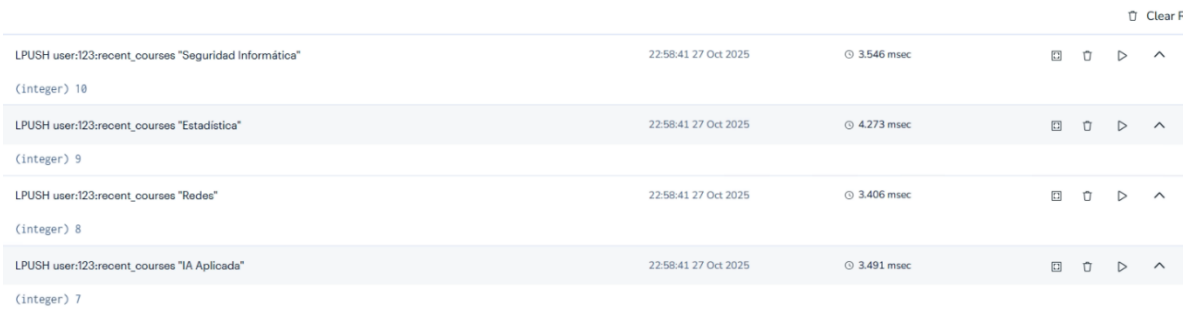


1. Propón tres claves Redis que podrían existir en ConnectU y explica el tipo de valor que guardarían.

Ejemplo 1
Clave: user:123:last_seen
Tipo: STRING
Contenido: una marca de tiempo (timestamp) o una fecha/hora de la última vez que el usuario 123 inició sesión.
Uso: saber si el usuario está “activo recientemente” para mostrar “en línea hace 2 min”.



Ejemplo 2
Clave: user:123:recent_courses
Tipo: LIST
Contenido: lista ordenada de los cursos que el usuario 123 visitó recientemente, el más reciente al inicio de la lista.
Uso: historial de navegación académica rápida.



Ejemplo 3
Clave: top_groups
Tipo: SET (sorted set) o SET con conteo aparte
Contenido: grupos de estudio con una puntuación que indica popularidad (por ejemplo cantidad de miembros activos en las últimas 24h).
Uso: ranking dinámico de “grupos más populares ahora”.

2. **Mostrar cómo almacenar y recuperar:**

a) Los últimos 5 cursos visitados por un estudiante.

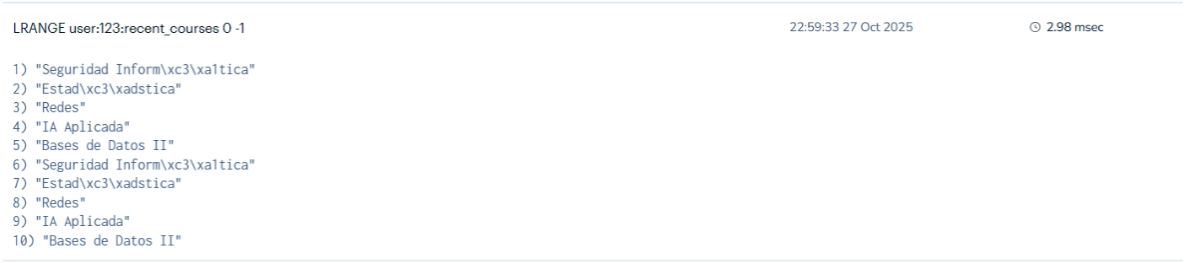
Supongamos el estudiante con id 123. Usamos LIST porque necesitamos orden.

Guardar cuando visita un curso nuevo (por ejemplo "Bases de Datos II"):

Comando para insertar al inicio:
LPUSH user:123:recent_courses "Bases de Datos II"

Para limitar a 5 elementos (cortar la lista a tamaño 5):
LTRIM user:123:recent_courses 0 4

Para leer los cursos recientes:
LRANGE user:123:recent_courses 0 4



Eso devuelve hasta 5 cursos más recientes.

b) Los 3 grupos más populares del momento.

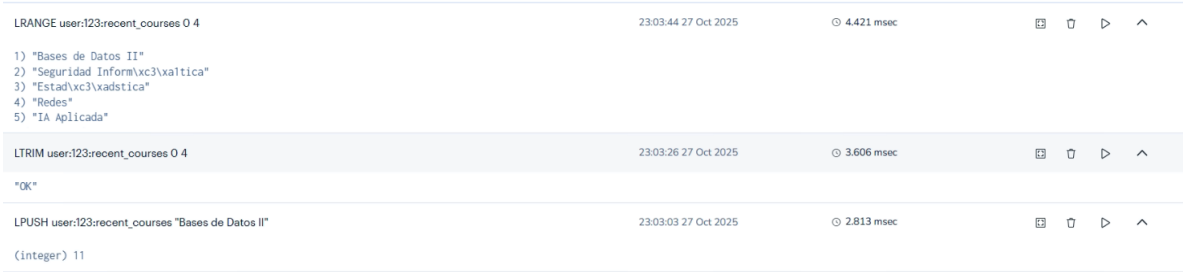
Podemos mantener un ranking en una lista fija o un SET ordenado manualmente.

Ejemplo simple con LIST:

Guardar (sobrescribir una lista global con los top 3):

DEL trending_groups
RPUSH trending_groups "Grupo Data Science" "Grupo BD2 Sección B" "Grupo IA Aplicada"

Leer los top 3:
LRANGE trending_groups 0 2



3. **¿Por qué Redis es útil en conjunto con Neo4j en este caso?**

Porque Neo4j es excelente para entender la estructura social (quién está conectado con quién, qué grupos son relevantes para ti según tus amigos, etc.), pero algunas respuestas se repiten mucho y cambian rápido. Ejemplos: “grupos más populares ahora”, “últimos cursos que viste”, “usuarios activos en la última hora”.

Si calculáramos todo eso en Neo4j cada vez, sería más caro. En cambio, podemos usar Neo4j para calcular una recomendación “inteligente”, guardarla temporalmente en Redis, y luego Redis responde rapidísimo cuando el usuario abre la app. Eso da tanto relevancia (Neo4j) como velocidad (Redis).

SECCIÓN III – ANÁLISIS Y DISEÑO

El equipo quiere agregar recomendaciones personalizadas de grupos según intereses y conexiones de un estudiante.

1. **¿Cómo se podría modelar esta funcionalidad usando relaciones en Neo4j?**

Podemos extender el modelo agregando más tipos de relaciones y propiedades para capturar afinidades e intereses.

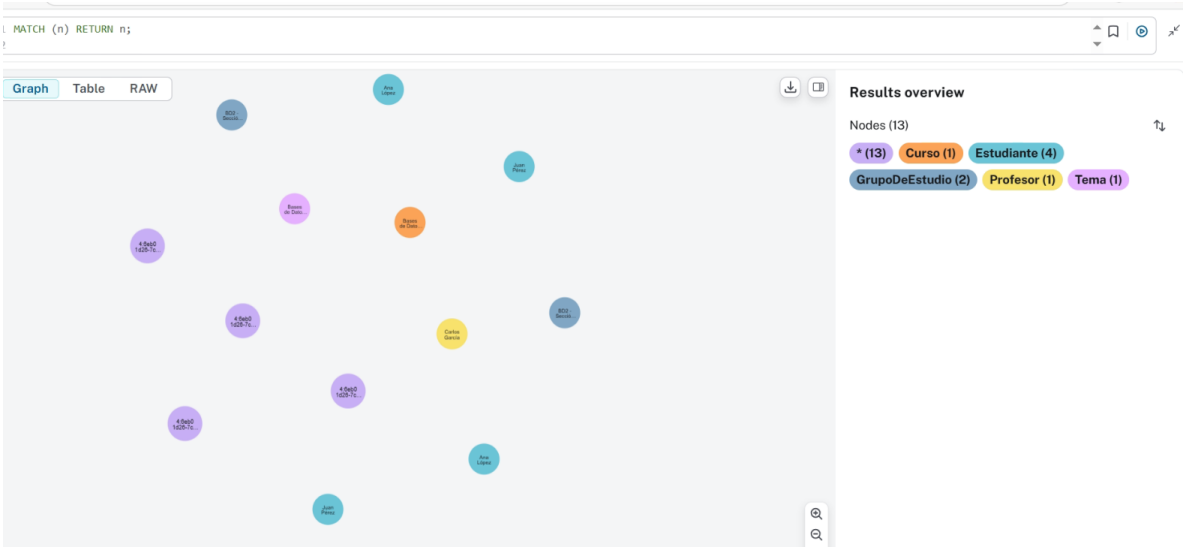
Ejemplo:

- (Estudiante)-[:INTERES_EN {nivel: "alto"}]->(Tema)
- (GrupoDeEstudio)-[:TRATA_SOBRE]->(Tema)
- (Estudiante)-[:AMIGO_DE {afinidad:0.8}]->(Estudiante)

Con esto, Neo4j puede buscar grupos que:
a) tratan temas que me interesan, y
b) donde también están amigos míos o gente con afinidad alta conmigo.

Una posible consulta conceptual sería:
“Dame los grupos que tocan temas que me interesan y donde ya participa alguien con quien tengo relación cercana.”

la recomendación sale de caminar las relaciones (hops) en el grafo, no de hacer JOINs cruzando tablas manualmente.



2. **Propón una estrategia que combine Neo4j y Redis para mejorar el rendimiento.**

Estrategia propuesta:

Paso 1 (cálculo profundo):
Neo4j ejecuta una consulta pesada de recomendación:

- Mira los intereses del estudiante.
- Busca grupos relacionados con esos intereses.
- Les da una puntuación considerando cuántos amigos cercanos ya pertenecen a ese grupo.

Paso 2 (cacheo rápido):
El resultado de esa consulta (por ejemplo, “te recomendamos Grupo BD2 Sección B, Grupo IA Aplicada, Grupo Redes Seguras”) se guarda en Redis en una clave tipo user:123:recommended_groups como una LIST.

Ejemplo en Redis:
RPUSH user:123:recommended_groups "Grupo BD2 Sección B" "Grupo IA Aplicada" "Grupo Redes Seguras"

Cuando el usuario abra la app, en vez de volver a correr toda la lógica compleja en Neo4j cada vez, primero leo Redis:
LRANGE user:123:recommended_groups 0 -1

Así la app responde casi instantáneo con sugerencias personalizadas que vienen originalmente de un análisis de grafo pero ahora están cacheadas.

También se puede agregar un TTL (tiempo de expiración) en Redis para que esas recomendaciones se refresquen cada cierto tiempo y no queden viejas.

3. **Justifica por qué una base relacional no sería óptima para esta funcionalidad.**

Una base relacional tradicional (tablas + JOINs) sí podría guardar todo, pero no es la mejor opción para este tipo de recomendación social dinámica por varias razones:

- Consultas muy relacionales: Recomendar grupos implica analizar “amigos de mis amigos”, “temas en común”, “grado de afinidad”, “cuántos amigos están en cada grupo”. En SQL esto termina siendo múltiples JOIN anidados y subconsultas recursivas. Ese tipo de consulta se vuelve costoso y difícil de mantener cuando la red crece.
- Profundidad variable: En una red social real no solo te importan tus amigos directos, te pueden interesar amigos de amigos, compañeros de curso, gente del mismo semestre, etc. En Neo4j es natural caminar saltos (hops) en el grafo; en SQL relacional hay que simular eso con autorrelaciones y CTE recursivas que no siempre escalan bien.
- Personalización en tiempo casi real: La recomendación cambia según quién se unió a qué grupo hace un minuto. Para dar una experiencia fluida, necesitamos tanto flexibilidad de grafo (Neo4j) como respuesta rápida cacheada (Redis). Una base relacional pura tendría que recalcular desde cero muchas veces y eso es más lento y más caro computacionalmente conforme la cantidad de usuarios y relaciones crece.

En resumen:

- Neo4j = modelo natural de conexiones + análisis de grafos.
- Redis = rapidez y caché de resultados efímeros.
- SQL relacional puro = más rígido y con alto costo en consultas profundas de relaciones sociales.

CONCLUSIONES FINALES

1. Neo4j es ideal para modelar redes sociales académicas como ConnectU porque trata las relaciones humanas (quién sigue a quién, qué grupo comparte qué curso, quién recomienda a qué profesor) como datos de primera clase y permite analizarlas de forma directa y eficiente. Esto hace posible construir funciones como “compañeros de grupo”, “profesor más recomendado” o “grupos recomendados según mis intereses” sin caer en consultas de 8 JOINs.

2. Redis complementa este enfoque guardando información volátil y de acceso rápido, como los últimos cursos vistos o los grupos de estudio más populares en este momento. Además, Redis puede almacenar resultados precalculados de recomendaciones personalizadas para que el usuario reciba sugerencias inmediatas sin recalcular todo el grafo cada vez.

3. Una base relacional tradicional sigue siendo excelente para muchos casos transaccionales, pero no es la mejor cuando la lógica depende de conexiones sociales profundas y dinámicas. En ese contexto, la combinación Neo4j + Redis resuelve tanto la parte inteligente (relaciones y afinidades) como la parte de rendimiento (respuesta inmediata en la aplicación).