

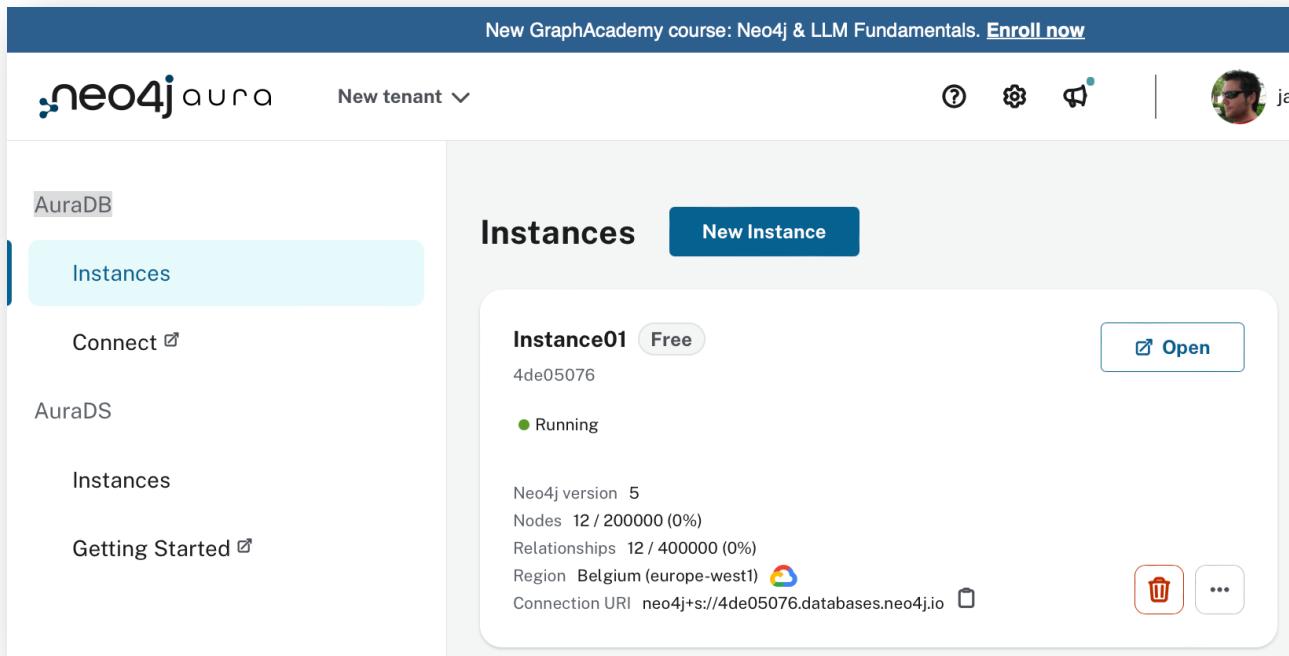


# Neo4J

## Aura Neo4j y Sandbox

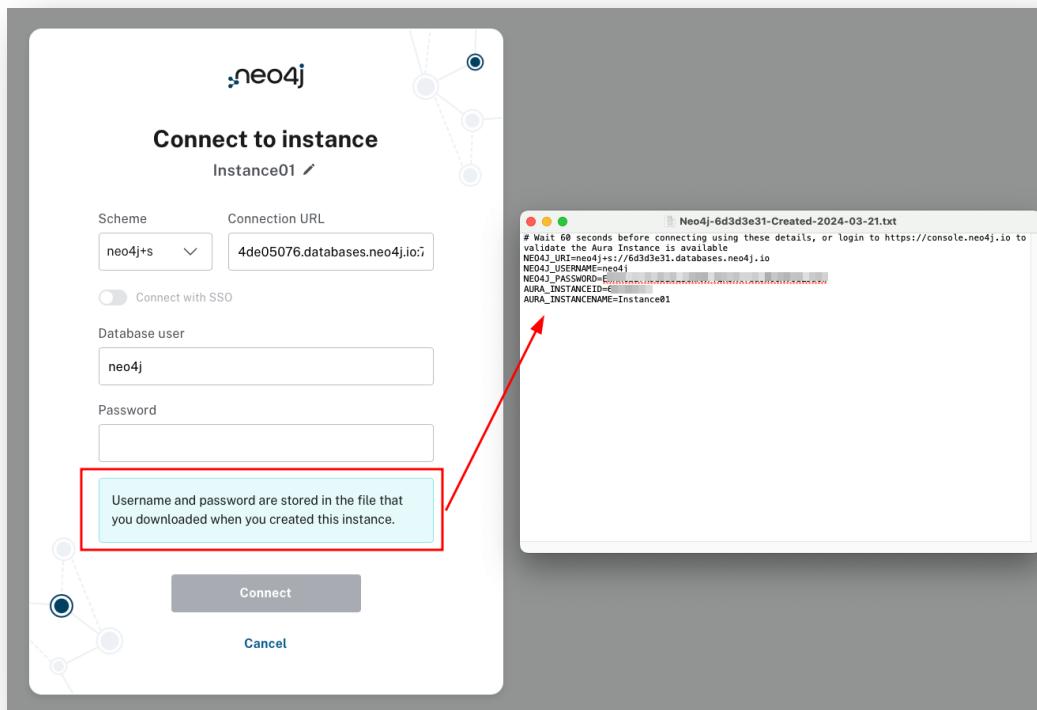
Neo4J ofrece su uso como SaaS (Software as a Service) mediante su plataforma AuraDB en la que podremos alojar nuestras bases de datos de grafos en una capa gratuita donde podremos guardar hasta 200 mil nodos y 400 mil relaciones.

Como es habitual, durante el proceso de registro nos preguntarán sobre qué proveedor queremos ejecutar y almacenar nuestra instancia. Para instancias de pago podremos elegir entre Google GCP, Amazon AWS y Microsoft Azure. Por ahora, en la capa gratuita solamente está disponible la plataforma de Google y además la base de datos se pausará automáticamente pasados 3 días de inactividad.



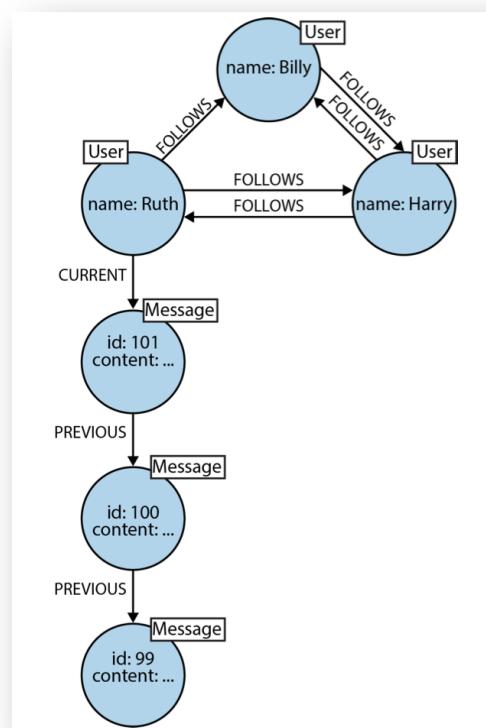
The screenshot shows the neo4j aura web interface. At the top, there's a banner for a GraphAcademy course: "New GraphAcademy course: Neo4j & LLM Fundamentals. [Enroll now](#)". Below the banner, the header includes the neo4j aura logo, a "New tenant" dropdown, and user profile icons. The main area has a sidebar with links for "AuraDB", "Instances", "Connect", "AuraDS", "Instances", and "Getting Started". The main content area is titled "Instances" and shows a list with one item: "Instance01" (Free), status "Running", Neo4j version 5, 12 nodes (0%), 12 relationships (0%), Region Belgium (europe-west1), and a Connection URI. There are "Open", delete, and more options buttons for the instance.

Una vez registrados y con acceso a AuraDB podremos crear instancia de nuevas bases de datos. Durante el proceso de creado tendremos que indicar que tipo de credenciales usaremos para acceder a esta instancia en el futuro. En esa primera configuración se descargará un archivo de texto con esa información, debemos custodiar con seguridad esos datos, serán necesarios para acceder.



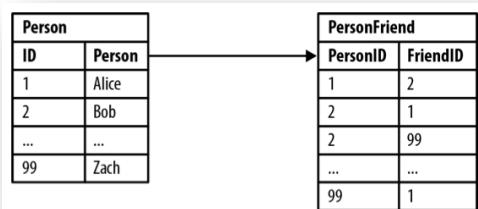
## ¿Qué es un grafo?

Podemos ver un grafo como una colección de nodos y relaciones. Los nodos son todas las entidades del problema como producto, usuario o servicio. Las relaciones son las maneras en que se relacionan unas entidades con otras. Normalmente es fácil distinguir entidades y relaciones porque en el enunciado del problema, las entidades son los nombres y las relaciones los verbos. Tanto los nodos como las relaciones tienen un tipo que llamaremos etiqueta y pueden contener propiedades en forma parejas de clave-valor.



## ¿Cuándo seleccionar una base de datos de grafos?

Siempre que el problema fundamental consiste en las relaciones entre entidades. Un caso típico es el de relaciones en redes sociales. Incluso en el diseño más sencillo, cuando los datos empiezan a crecer, las bases de datos relacionales quedan por detrás.



Example 2-3. Alice's friends-of-friends

```

SELECT p1.Person AS PERSON, p2.Person AS FRIEND_OF_FRIEND
FROM PersonFriend pf1 JOIN Person p1
    ON pf1.PersonID = p1.ID
JOIN PersonFriend pf2
    ON pf2.PersonID = pf1.FriendID
JOIN Person p2
    ON pf2.FriendID = p2.ID
WHERE p1.Person = 'Alice' AND pf2.FriendID <> p1.ID
    
```



En el experimento de Partner y Vukotic se buscó la relación de "amigo de amigo" con una profundidad de 5 grados de separación en una base de datos de 1 millón de personas con aproximadamente 50 amigos cada uno.

Table 2-1. Finding extended friends in a relational database versus efficient finding in Neo4j

Depth	RDBMS execution time(s)	Neo4j execution time(s)	Records returned
2	0.016	0.01	~2500
3	30.267	0.168	~110,000
4	1543.505	1.359	~600,000
5	Unfinished	2.132	~800,000

## Cypher

El lenguaje de consulta usado por Neo4J se llama Cypher. Por ahora es exclusivo de Neo4J pero se está usando como base para un nuevo estándar llamado GQL equivalente al SQL que conocemos.

Cypher está diseñado para ser fácilmente leído y entendido porque recuerda mucho al grafo con el que representamos el problema. De una manera muy informal podemos decir que las sentencias en Cypher describen algo como "busca algo como esto".

La consulta más sencilla tiene una cláusula MATCH y otra cláusula RETURN.

## Match y Return

La cláusula MATCH es donde se indica el patrón que queremos buscar. Dibujaremos nodos entre paréntesis y relaciones con corchetes cuadrados. Para indicar que una relación está unida a un par de nodos usamos el doble guion acabado en un signo de menor o mayor para indicar dirección.

En la cláusula MATCH escribimos el patrón de búsqueda con nodos y relaciones. Ambos elementos pueden estar identificados, etiquetados y también podemos especificar alguna de sus propiedades entre llaves.

Al buscar patrones es probable que estos se repitan muchas veces dentro del grafo y que nosotros solo estemos interesados en algunos de ellos. Para esto podemos establecer restricciones indicando las propiedades que nodos y/o relaciones deben cumplir. Como alternativa también podemos identificar nodos y/o relaciones y posteriormente aplicar una cláusula WHERE con las mismas restricciones.



La cláusula RETURN especifica los nodos, relaciones y propiedades de la etapa MATCH que se devuelven

## Insertar

Para crear un nodo o una relación entre nodos usamos la cláusula CREATE.

```

CREATE (shakespeare:Author {firstname:'William', lastname:'Shakespeare'}),
(juliusCaesar:Play {title:'Julius Caesar'}),
(shakespeare)-[:WROTE_PLAY {year:1599}]->(juliusCaesar),
(theTempest:Play {title:'The Tempest'}),
(shakespeare)-[:WROTE_PLAY {year:1610}]->(theTempest),
(rsc:Company {name:'RSC'}),
(production1:Production {name:'Julius Caesar'}),
(rsc)-[:PRODUCED]->(production1),
(production1)-[:PRODUCTION_OF]->(juliusCaesar),
(performance1:Performance {date:20120729}),
(performance1)-[:PERFORMANCE_OF]->(production1),
(production2:Production {name:'The Tempest'}),
(rsc)-[:PRODUCED]->(production2),
(production2)-[:PRODUCTION_OF]->(theTempest),
(performance2:Performance {date:20061121}),
(performance2)-[:PERFORMANCE_OF]->(production2),
(performance3:Performance {date:20120730}),
(performance3)-[:PERFORMANCE_OF]->(production1),
(billy:User {name:'Billy'}),
(review:Review {rating:5, review:'This was awesome!'}),
(billy)-[:WROTE REVIEW]->(review),
(review)-[:RATED]->(performance1),
(theatreRoyal:Venue {name:'Theatre Royal'}),
(performance1)-[:VENUE]->(theatreRoyal),
(performance2)-[:VENUE]->(theatreRoyal),
(performance3)-[:VENUE]->(theatreRoyal),
(greyStreet:Street {name:'Grey Street'}),
(theatreRoyal)-[:STREET]->(greyStreet),
(newcastle:City {name:'Newcastle'}),
(greyStreet)-[:CITY]->(newcastle),
(tyneAndwear:County {name:'Tyne and Wear'}),
(newcastle)-[:COUNTY]->(tyneAndwear),
(england:Country {name:'England'}),
(tyneAndwear)-[:COUNTRY]->(england),
(stratford:City {name:'Stratford upon Avon'}),
(stratford)-[:COUNTRY]->(england),
(rsc)-[:BASED_IN]->(stratford),
(shakespeare)-[:BORN_IN]->stratford

```

Aunque Neo4J es una base de datos NoSQL y por tanto no tiene un esquema obligatorio, sí es posible indicar restricciones.

To ensure that all country names are unique, we can add a uniqueness constraint:

```
CREATE CONSTRAINT ON (c:Country) ASSERT c.name IS UNIQUE
```



## Entorno de pruebas

Para esta guía usaré el entorno de pruebas Sandbox que proporciona Neo4J donde ya hay bases de datos con datasets incorporados que nos permitirán enfocar las consultas y no preocuparnos por rellenas datos de entrada con nodos y relaciones.

### Neo4j Sandbox

To explore a wide variety of datasets in an [online setup](#) without a local installation, you can use the [Neo4j Sandbox ↗](#).

Each sandbox is available for at least three days after creation and can also be remotely accessed from applications using any Neo4j driver.

Except for the "blank" sandbox, all other sandboxes come prepopulated with the domain data and focus on use case specific queries.

All sandboxes provide access to Neo4j Browser, Neo4j Bloom, APOC, Graph Data Science, neosemantics (n10s) and a GraphQL integration.

En el Sandbox disponemos de variados datasets para nuestras pruebas. Para esta guía usaré el dataset de "Movies" y "Contact Tracing" donde simularemos una base de datos de películas y actores además del escenario de los rastreos que había que realizar durante los meses de COVID-19.

## Películas y actores

Esta base de datos que proporciona Neo4j tiene pocos datos pero eso la hace ideal para probar consultas y poder comprobar y confirmar que los resultados son correctos.

Básicamente se guardan dos entidades etiquetadas como "Person" y "Movie". Para las personas se guardan el nombre y el año de nacimiento., para las películas apenas el nombre y el año de lanzamiento.

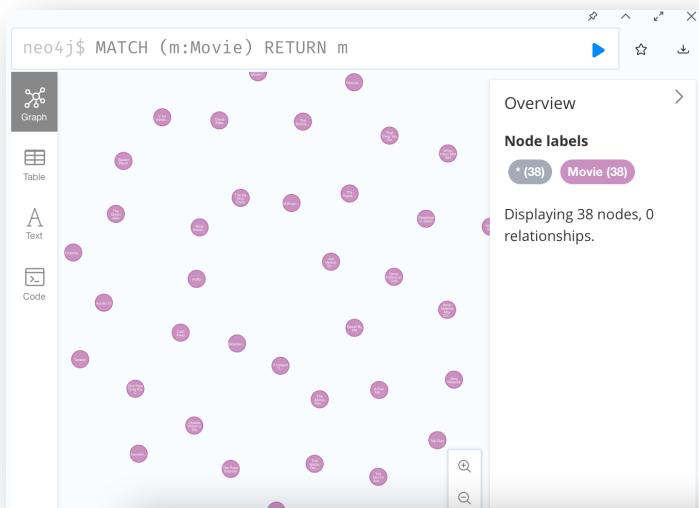


Estas dos entidades se unen mediante distintas relaciones etiquetadas como "Directed", "Wrote", "Produced", "Reviewed", "Acted\_in" y "Follow". Para ese primer ejemplo solo usaremos la relación "Acted\_in".

Vamos a suponer que has actuado en la película "The Matrix" y tenemos que añadirte a la base de datos. Para empezar listemos todo el catálogo de películas de la base de datos.

```
MATCH ( m : Movie ) RETURN m
```

Con este comando tiene dos fases, en el MATCH buscamos el patrón de nodos etiquetados como "Movie" y les asignamos la variable "m". En la fase RETURN indicamos que queremos proyectar esa variable "m" que hemos ido poblando a medida que el motor encontraba una coincidencia del patrón.



El resultado son 38 nodos, cada uno con la información de una película. Esto es el parecido a los registros de una base de datos relacional en la que queremos ver el contenido de la tabla películas. Lo cierto es que solo estamos interesados en saber si está "The Matrix" entre los resultados así que podemos poner alguna restricción en la consulta.

```
MATCH ( m : Movie { title : "The Matrix" } ) RETURN m
MATCH ( m : Movie ) WHERE m.title="The Matrix" RETURN m
```

Estas dos consultas devuelven el mismo resultado. Si nos fijamos veremos que en la primera se define un patrón de nodo tipo "Movie" que tenga como propiedad "title" el valor de "The Matrix". En la segunda consulta, el patrón solo busca nodos de tipo "Movie" y cuando ya los tiene filtra dentro de la cláusula WHERE aquello que cumplen la condición de que sus propiedades "title" son igual a "The Matrix".



Fijémonos en la parte de "The Matrix". En la primera consulta tiene dos puntos porque se está definiendo un patrón de nodo. En la segunda consulta tiene un igual porque se está definiendo la condición que tienen que cumplir los resultados.

Ahora que ya sabemos que la película existe vamos a crear nuestro propio nodo de tipo "Person"

```
CREATE ( p : Person { name : "Javi" } )
```

Con este comando damos la orden de instanciar un nuevo nodo etiquetado "Person". Queremos que ese dato contenga nuestro nombre así que le pasamos un JSON actualizando la propiedad "name".

```
MATCH ( p : Person { name : "Javi" } ) SET p.born = 1980 RETURN *
```

Con este comando actualizamos valores de un nodo ya creado. Fijémonos en cómo se combina la cláusula MATCH para devolver un nodo en concreto y la cláusula SET para actualizar o añadir (o eliminar) una propiedad.

Ya estaba la película y ya tenemos nuestro nodo, ahora toca relacionarlos. Para ello debemos buscar los nodos que vamos a relacionar y posteriormente crear la relación.

```
MATCH ( m : Movie { title : "The Matrix" } ) , ( p : Person { name : "Javi" } ) CREATE (p)-[:ACTED_IN]->(m)
```

Si analizamos esta sentencia Podemos distinguir dos partes, la etapa MATCH y la etapa CREATE. En la etapa MATCH se escriben los patrones que se van a buscar. En este caso buscaremos dos patrones, los nodos tipo "Movie" con valor de título "The Matrix" que guardaremos en la variable "m" y por otro lado, los nodos de tipo "Person" con valor de nombre "Javi" o lo que corresponda. En la etapa CREATE usamos las variables anteriores para escribir el patrón en el que unimos estos dos nodos mediante la relación "ACTED\_IN" entre corchetes. Fijémonos es que es una relación con dirección por lo que pondremos una punta de flecha marcando el sentido.

```
MATCH (yo : Person { name:"Javi" } ) - [:ACTED_IN]->(m:Movie)<-[ :ACTED_IN]-(compis : Person) RETURN compis
```

En esta consulta queremos ver a otros actores y actrices que también participaron en el rodaje de la película y que, por tanto somos compañeros de rodaje.

The screenshot shows the Neo4j interface with a search query: `neo4j$ MATCH (yo:Person {name:"Javi"})-[:ACTED_IN]->(m:Movie)<-[ :ACTED_IN]-(compis:Person) RETURN compis`. On the left, there are four navigation tabs: Graph (selected), Table, Text, and Code. The main area displays five purple circular nodes representing actors: Keanu Reeves, Hugo Weaving, Laurence Fishburne, Emil Eifrem, and Carrie-Anne Moss. Each node has its name printed on it.



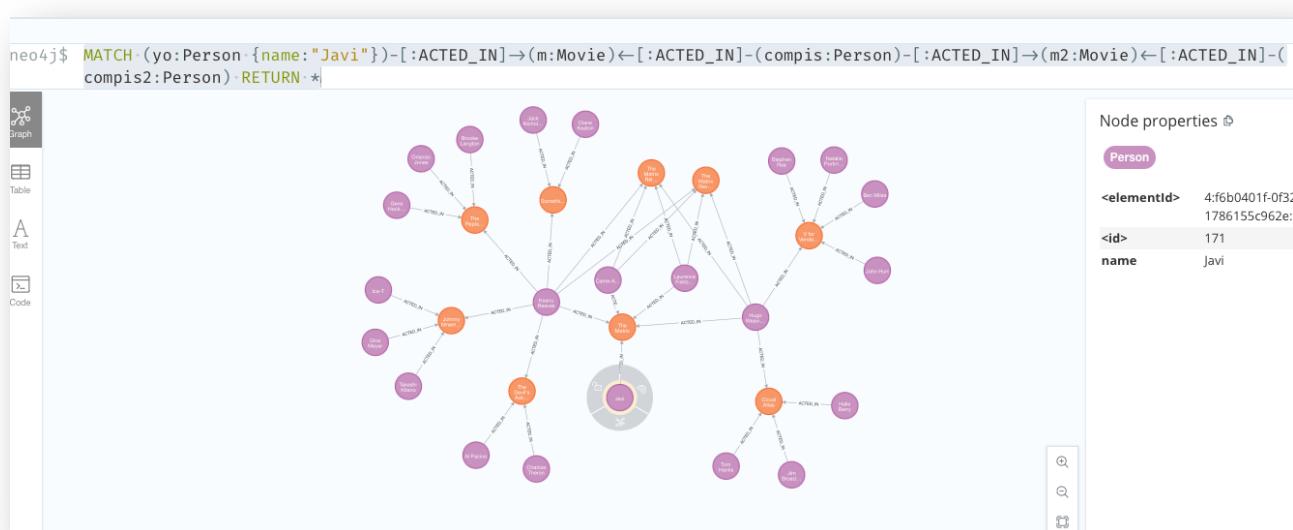
En la siguiente consulta vamos a buscar a los amigos de nuestros amigos. En otras palabras, queremos ver un grafo en el que se vean los actores y actrices con los que han trabajado en otras películas nuestros compañeros de "The Matrix". No hemos trabajado directamente con ellos pero tenemos un contacto en común que ha trabajado con ambos.

```
MATCH (yo:Person {name:"Javi"})-[:ACTED_IN]->(m:Movie)<-[ :ACTED_IN]-(compis:Person)-[:ACTED_IN]->(m2:Movie)<-[ :ACTED_IN]-(compis2:Person) RETURN *
```

Si analizamos la sentencia veremos que empieza por mi nodo y busca un patrón en el que pase por actuar en una película (m) en la que, a su vez, hayan actuado otros nodos (compis). Hasta aquí la consulta idéntica a la anterior. La novedad consiste en que podemos seguir complicando el patrón. Lo hacemos añadiendo al patrón que los nodos "compis" actúen en una segunda película (m2) en la que, a su vez hayan actuado otros actores (compis2)

Si nos fijamos en la captura de abajo, veremos que he señalado mi nodo y que sobre él está mi única película, "The Matrix". De ella salen también relaciones al resto de actores y actrices que participaron en ella. De cada uno salen tantas otras relaciones como otras películas en las que haya actuado y de cada una de ellas, los actores y actrices que participaron.

En general está bien pero en la parte central se puede ver como el patrón ha dado por bueno nodos de actores que participaron en otras películas pero que ya conozco porque rodaron conmigo en "The Matrix"



En bases de datos de grafos, este tipo de patrones representa la superioridad contra las bases de datos relacionales. Pensemos que para hacer esta consulta en relacional tendríamos que hacer JOINs entre

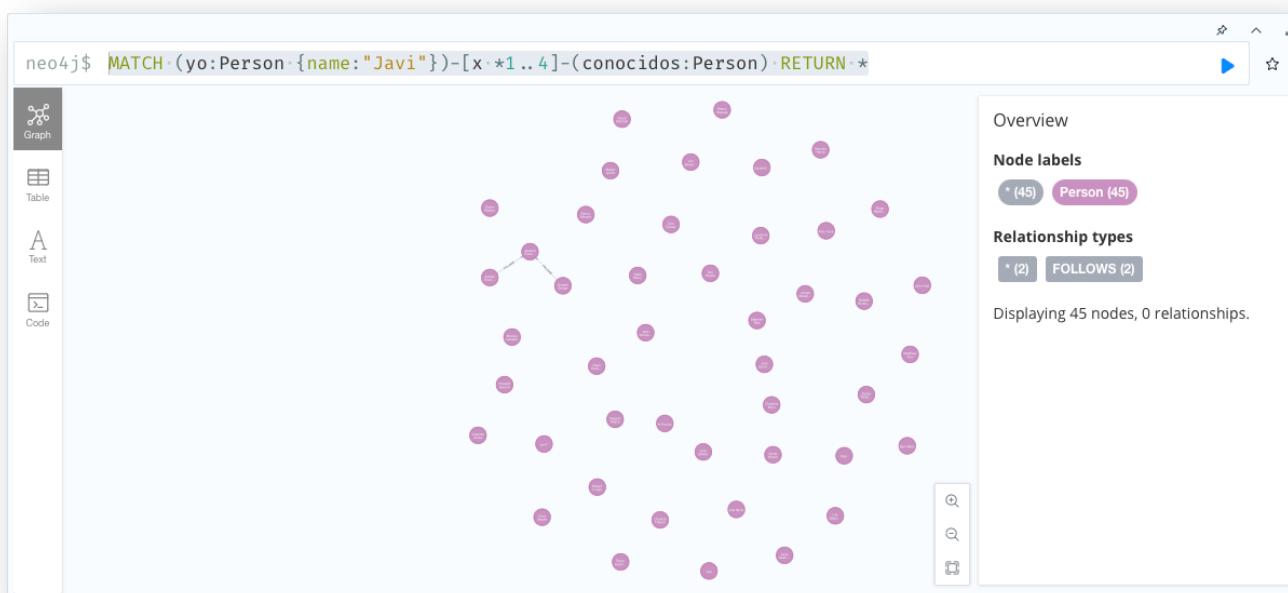


tablas mientras que en esta base de datos las relaciones simplemente son punteros que se recorren sin gasto computacional.

Además, los lenguajes de consulta de bases de datos de grafos permiten realizar consultas con estos patrones especificando exactamente el número de saltos o un rango de valores.

```
MATCH (yo:Person {name:"Javi"})-[x *1..4]-(conocidos:Person) RETURN conocidos
```

Esta consulta muestra los nodos de tipo persona que se encuentran a entre 1 y 4 saltos de mí. Si nos fijamos en la estructura del patrón veremos que el nodo inicial soy yo y que después le pedimos que busque saltos a través de entre 1 y 4 relaciones de cualquier tipo que finalicen en un nodo de tipo persona.



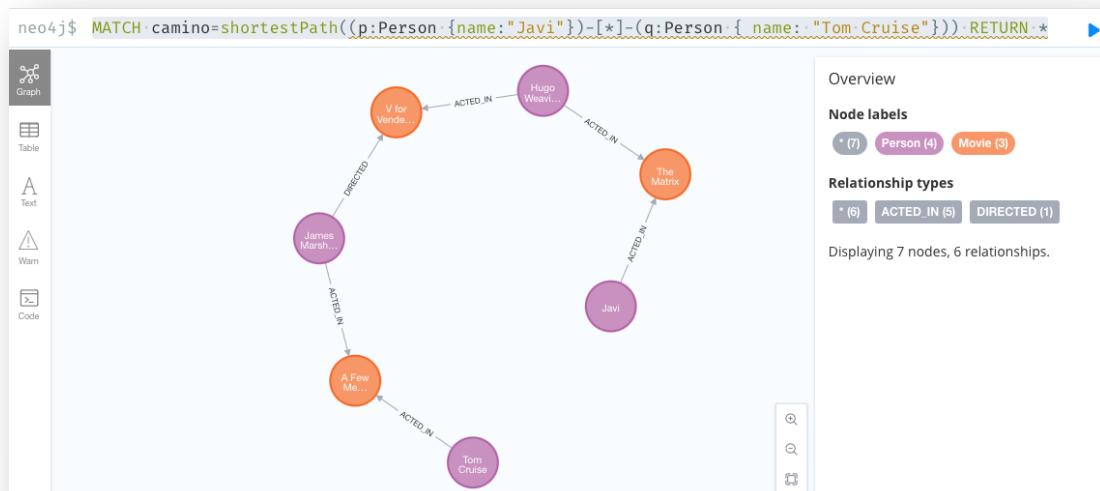
Como estamos viendo, las bases de datos de grafos proporcionan muchas facilidades y herramientas para hacer consultas donde lo principal esté en el camino y las relaciones.

La última herramienta que veremos es la búsqueda del camino más corto.

```
MATCH camino=shortestPath((p:Person {name:"Javi"})-[*]-(q:Person { name: "Tom Cruise"})) RETURN *
```



Esta consulta devuelve el camino más corto entre mi nodo y el nodo que representa a "Tom Cruise". Para ello indico mi nodo en el patrón, el nodo de "Tom Cruise" y le indico que siga el camino de cualquier tipo de relación, en cualquier sentido y con cualquier número de saltos. Lo único importante es encontrar el camino más corto que nos une.



Al parecer, con los datos de esta base de datos el camino más corto que nos une es a través de Hugo Weaving que actuó en "The Matrix" y también en "V de vendetta". Esta película fue dirigida por James Marshall que, a su vez actuó en "Unos pocos hombres buenos" junto a "Tom Cruise"

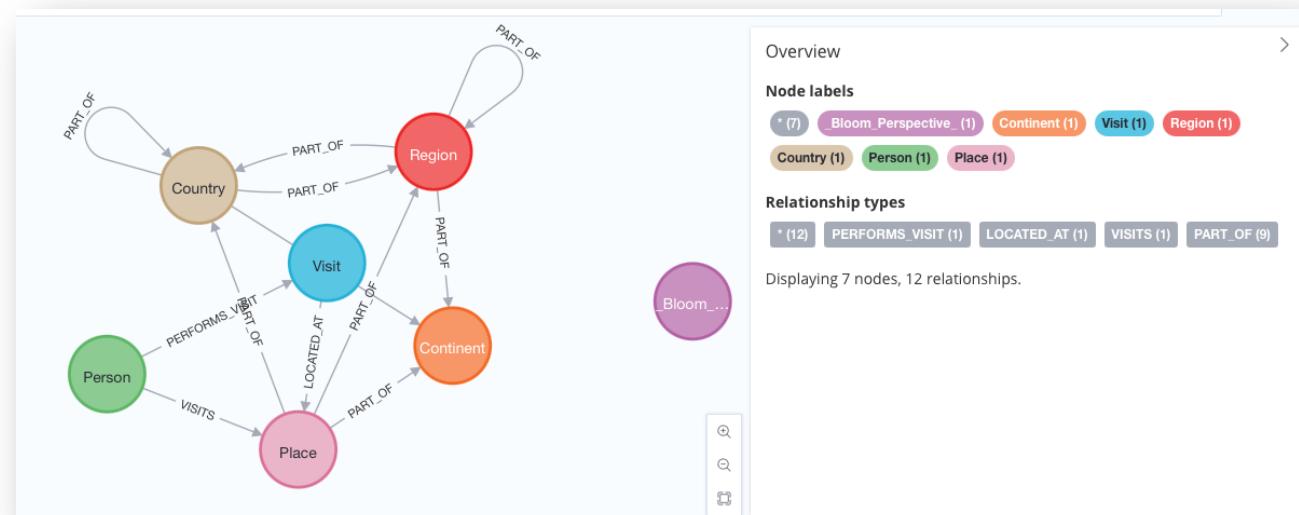
Fijémonos en que en este camino se han usado relaciones de dos tipos (actuar y dirigir) porque no hemos indicado ninguna restricción.

## Contact Tracing

Para esta parte de la guía usaremos la base de datos de pruebas de "Contact Tracing". Debemos seleccionarla de entre las disponibles dentro del SandBox de Noe4j.

```
CALL db.schema.visualization()
```

Con esta línea obtenemos la representación gráfica del esquema. La base de datos mostrará todos los tipos de nodos según su etiqueta y las relaciones que ha encontrado entre parejas de nodos.



```
MATCH (p:Person) RETURN * LIMIT 1
```

Con esta sentencia seleccionaremos uno cualquiera de los nodos etiquetados como "Person" y podremos ver sus propiedades. En el caso de "Person" tenemos su nombre, su estado de salud, su dirección y la fecha de última actualización.

neo4j\$ match (p:Person) return \* limit 1

Graph

Table

Text

Code

Landyn Greer

Node properties

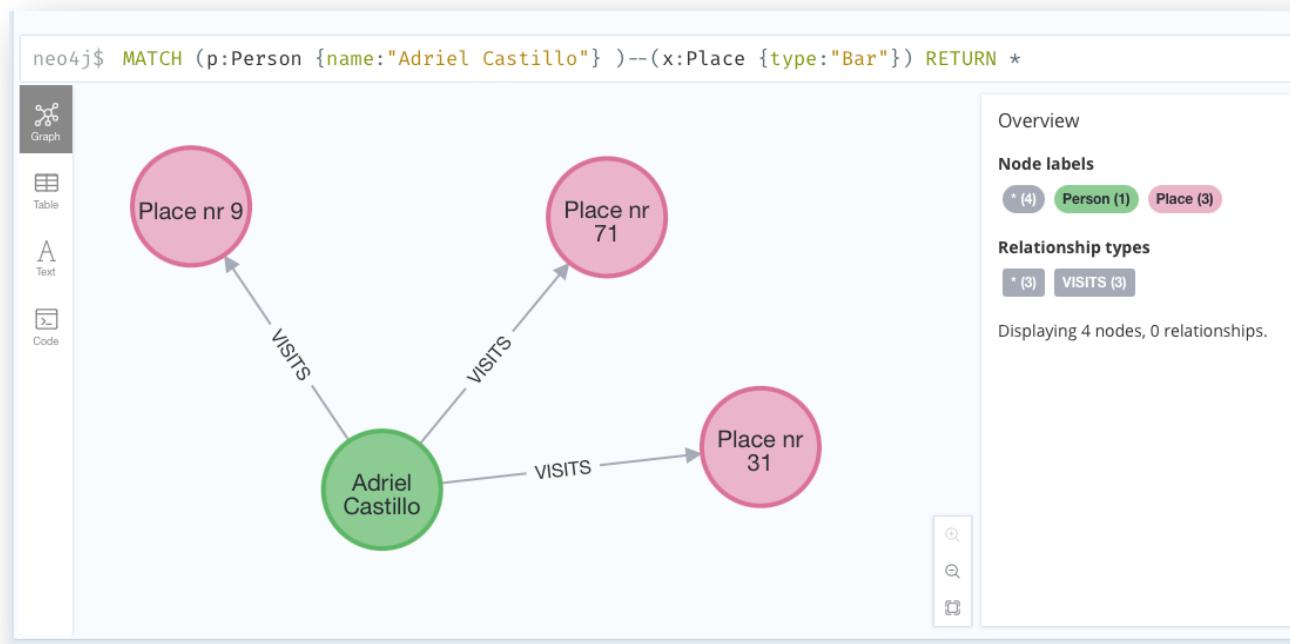
Person	
<elementId>	4:3fd047e2-8f70-48d6-b902-bf798534aa92:0
<id>	0
addresslocation	point({srid:7203, x:51.21236082, y:4.410299302})
confirmedtime	"2020-05-04T21:54:12Z"
healthstatus	Healthy
id	1
name	Landyn Greer



En este punto podemos repetir el comando para ver las propiedades del resto de nodos y relaciones y entender lo que se representa en esta base de datos

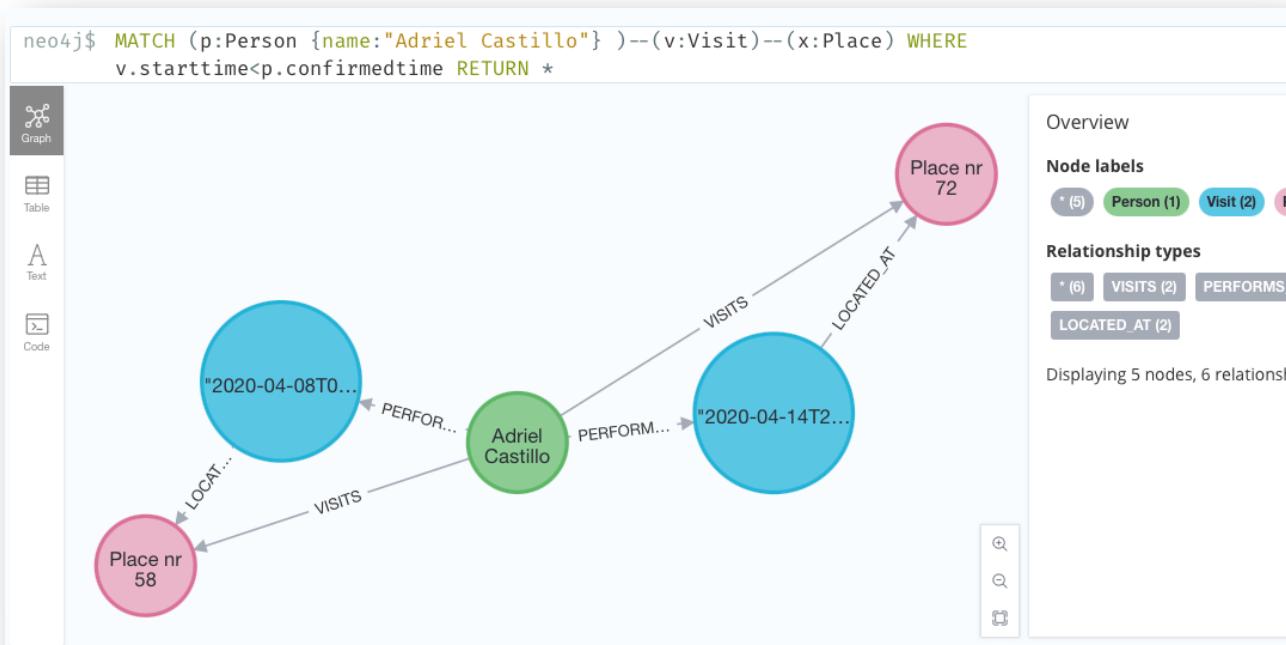
```
MATCH (p:Person {name:"Adriel Castillo"}) - (x:Place {type:"Bar"}) RETURN *
```

Sabemos que Adriel Castillo está enfermo y con esta consulta queremos averiguar todos los bares en los que ha estado.



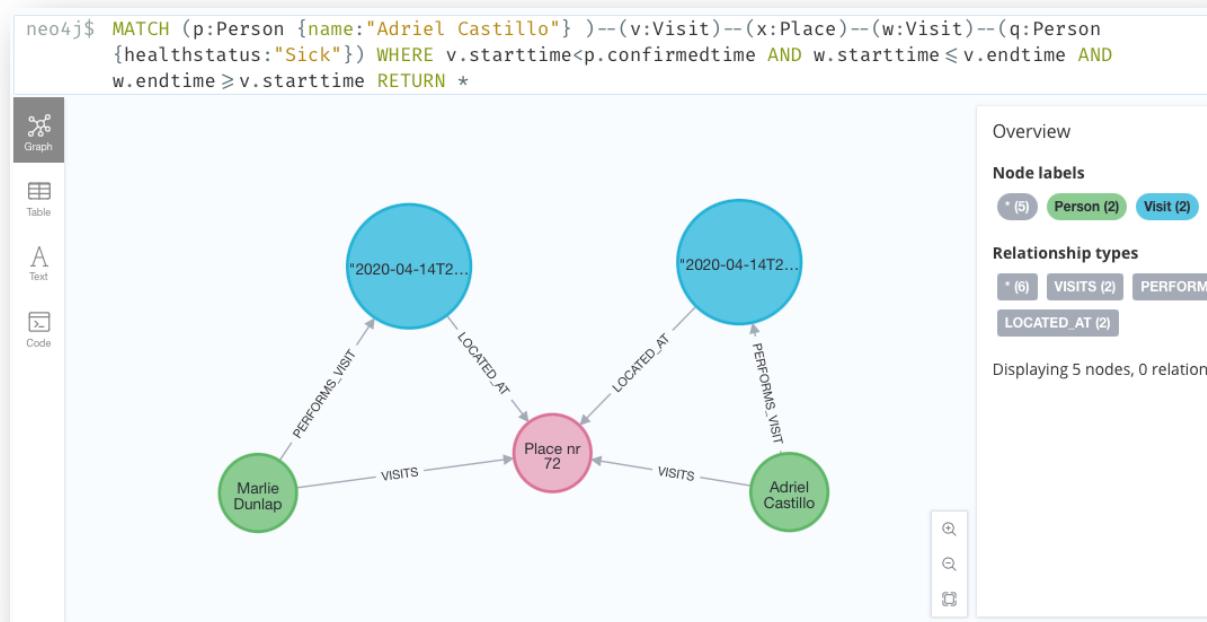
```
MATCH (p:Person {name:"Adriel Castillo"})-(v:Visit)-(p:Place) WHERE vstarttime < p.confirmedtime RETURN *
```

Con esta consulta buscamos todos aquellos lugares que Adriel visitó antes de estar enfermo. Usando las fechas de las visitas podemos ver de manera gráfica el orden en el que se visitaron estos lugares que, en principio fueron el origen del contagio.



```
MATCH (p:Person {name:"Adriel Castillo"}) -(v:Visit)-(x:Place)-(y:Visit)-(q:Person {healthstatus:"Sick"}) WHERE vstarttime < p.confirmedtime AND y starttime <= v.endtime AND y endtime >= vstarttime RETURN *
```

Con esta consulta conseguimos las personas que coincidieron al mismo tiempo en los lugares que visitó Adriel antes de estar enfermo



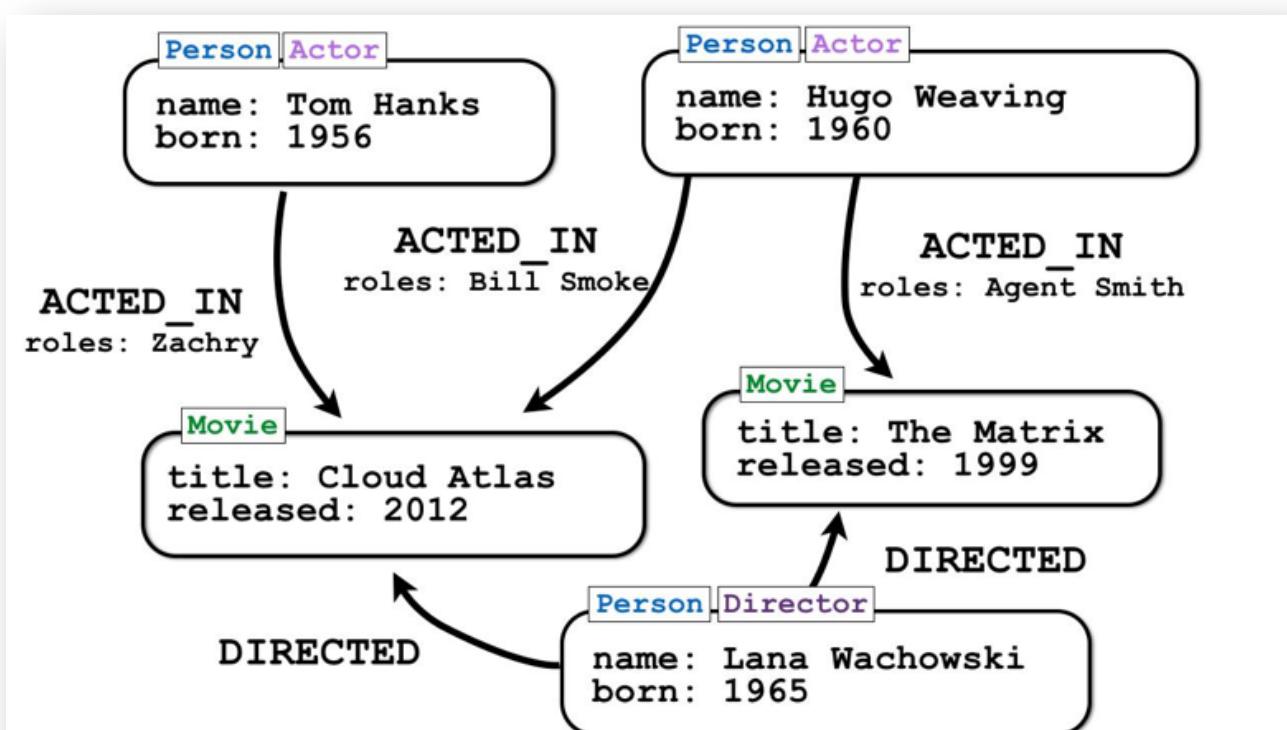


Como vemos en la imagen obtenemos que Marie, estando enferma, coincidió en el mismo momento con Adriel en un parque.

Como hemos visto en estos ejemplos, las consultas que se hacen en un entorno de grafos consisten en la búsqueda de patrones uniendo nodos mediante relaciones.

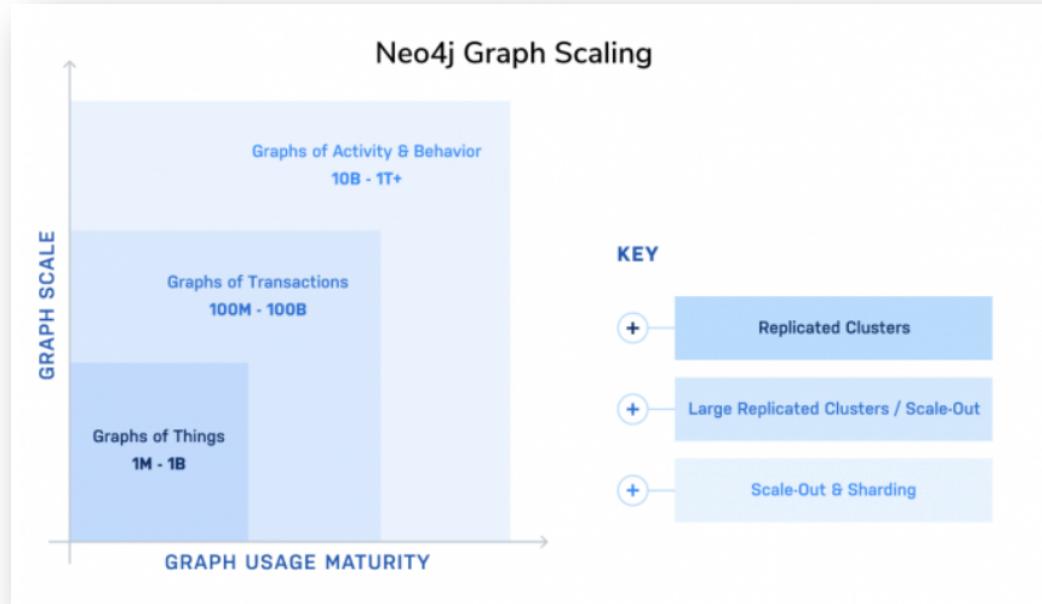
## Conclusiones

Las bases de datos de grafos forman parte de las bases de datos llamadas NoSQL porque usan un modelo de datos distinto al tabular. En este caso se usa un modelo de datos de grafos en la que los nodos representan las entidades y las aristas representan las relaciones entre ellos. Tanto nodos como relaciones pueden contener propiedades y etiquetas.





Como es habitual en las bases NoSQL, Neo4j tiene capacidades nativas para realizar réplicas y sharding lo que permite que puedan escalar horizontalmente.



Otra de las características de las bases de datos NoSQL es la ausencia de esquema obligatorio. En el caso de Neo4j podemos establecer restricciones al esquema de manera opcional.

Neo4j usa un lenguaje de consulta llamado Cypher apropiado para el modelo de datos de grafos. El lenguaje de consulta propio para las bases de datos NoSQL de grafos ya tiene un estándar, el ISO/IEC 39075:2024 desde abril del 2.024. Es una noticia importante porque el anterior lenguaje de consulta certificado fue SQL en 1.987.

Este motor y tipo de bases de datos hemos visto que es el adecuado para los problemas que se representan con grafos y en todos aquellos donde las relaciones entre nodos son importantes. Aquellos problemas en los que, de manera tradicional con bases de datos relacionales, había que hacer JOINs con tablas grandes también son buenos candidatos para usar una base de datos de tipo grafo.