



# Proyecto final

## Bases de Datos No Relacionales

*Autores:*

Juliana Caballero - C.I.: 4.958.048-1

Augusto Alonso - C.I.: 4.866.351-3

GRUPO 17

*Docente:*

Lorena Etcheverry

**Julio 2021**

# 1. Introducción

El objetivo de este proyecto es estudiar los métodos actuales de migración de bases de datos relacionales a bases de datos de grafos. Además, se pretende experimentar con alguno de los métodos estudiados, y compararlo con la implementación disponible por parte de la aplicación *Neo4j*[7] para hacerlo. Con el conocimiento de que el enfoque de *Neo4j* se basa en la traducción directa de tuplas a elementos del grafo, se espera encontrar una implementación con otro enfoque para poder compararlas.

En el resto del documento se presentan las siguientes secciones. Primero se presentan en la sección 2 los artículos y trabajos que se utilizaron para desarrollar este trabajo. En las secciones 3 y 4 se plantea el estudio y resumen de los documentos mencionados, entre ellos el trabajo inspirador de este proyecto, *R2G: a Tool for Migrating Relations to Graphs*. Estos informes se organizan de acuerdo a la disponibilidad de implementaciones que se podrían testear en este trabajo. Luego en la sección 5 se presentan las conclusiones del trabajo, exponiendo tanto una comparación entre los modelos estudiados así como también una conclusión personal sobre la transformación de bases de datos relacionales a bases de datos de grafos. Finalmente, en la sección 6 se exponen posibles trabajos a futuro.

# 2. Trabajos relacionados

Para el desarrollo de este trabajo se utilizan en total seis artículos publicados entre los años 2013 y 2020 que enfrentan desde distintos enfoques el problema de migración de Bases de Datos Relacionales (BDR) a Bases de Datos de Grafos (BDG). Los artículos utilizados son [1], en el cual se describe un algoritmo genérico de transformación de esquemas y consultas de BDR a BDG; [6] presenta el uso de un algoritmo de conversión de BDR a BDG como herramienta para el filtrado colaborativo de recomendaciones; de manera similar [4] refleja la utilidad de la transformación de BDR a BDG en las relaciones existentes entre documentos legales; [3] presenta un algoritmo de conversión de BDR a BDG enfocado en el uso del esquema resultado para *graph mining*; el siguiente artículo trabajado [2] propone un algoritmo genérico para la unión de numerosas BDR a través de la transformación de las bases a una única BDG; finalmente [5] presenta un algoritmo de mapeo directo de BDR a BDG, una base sencilla para transformar cualquier tipo de BDR a una BDG.

# 3. Parte central

En esta sección se expone primero el estudio de los artículos [1], [6], [4], [3] y [2]. Estos trabajos no cuentan con una implementación disponible, publicada y abierta al público, pero se consideran relevantes por demostrar que distintos enfoques y realidades dan paso a distintos métodos de transformación de bases de datos relacionales a bases de datos de grafos. Finalmente se expone el artículo [5] cuya implementación si se encuentra actualmente disponible pero la misma no será utilizada para este trabajo debido a la falta de disponibilidad otras herramientas para realizar una comparación.

## 3.1. R2G: a Tool for Migrating Relations to Graphs

En el artículo [1], los autores mencionan que en la actualidad existen muchas herramientas y métodos que plantean transformaciones de bases de datos relacionales a bases de datos de grafos, pero dichos métodos o herramientas fueron implementados para modelos específicos y se basan en técnicas un tanto ingenuas, como por ejemplo, mapeando tuplas a nodos y claves foráneas a aristas.

Además mencionan que dichas estrategias y herramientas dejan de lado una parte muy importante para la transformación; las consultas que típicamente se realizan en la base de datos de origen. Dicho aspecto es muy importante para poder adaptar la capa de aplicación a la nueva capa de persistencia de manera efectiva y eficiente.

Para solucionar dicho problema, los autores proponen una herramienta que transforma cualquier base de datos relacional **r** a una base de datos de grafos **g**, y a su vez, cualquier consulta en la base de datos **r** a un recorrido de grafos en la base de datos **g**.

La estrategia planteada aprovecha las restricciones en la base de origen para minimizar los accesos que son necesarios en la base de salida para poder resolver una consulta. Para ello, en cada nodo de la base de salida se guarda la información que tiene más probabilidad de ser obtenida en conjunto en una consulta en la base de origen.

### 3.1.1. Transformación de los datos

Dado un esquema relacional **R**, el esquema de grafos **RG** es un grafo dirigido $\langle N, E \rangle$ tal que:

- Hay un nodo $A \in N$ por cada atributo $A$ de una tupla en **R**
- Existe una arista $(A_i, A_j) \in E$ si se cumple alguna de las siguientes:
  - $A_i$ es una clave en una relación $R$ en **R** y $A_j$ es un atributo que no es clave de dicha relación
  - $A_i$,$A_j$ son clave en una relación $R$ en **R**
  - $A_i$,$A_j$ son claves en las relaciones $R_i$ y $R_j$ respectivamente en **R**

Los autores mencionan que debido a la gran variedad de gestores de bases de datos de grafos (GDBMS), no hay una definición estandarizada de un modelo de base de datos de grafos, por lo cuál presentan la siguiente definición: Una base de datos de grafos es un multigrafo etiquetado $\mathbf{g} = (N, E)$ donde todo nodo $n \in N$ está asociado a un conjunto de pares $\langle clave, valor \rangle$.

Esto permite aplicar la metodología planteada a bases de datos RDF y PG.

### 3.1.2. Arquitectura de la herramienta

La herramienta implementada por los autores tiene cuatro componentes principales:

- Metadata Analyzer: Inspecciona el esquema de la base de datos relacional y construye el esquema de la base de datos de grafos utilizando la metodología planteada en el punto anterior.
- Data Mapper: Genera el mapeo entre los datos de la base de datos relacional y la base de datos de grafos.
- Query Mapper: Mapea las consultas realizadas para la base de datos de grafos a un lenguaje apto para la base de datos de grafos.
- Graph Manager: Se encarga de realizar la migración de datos y la ejecución de consultas en la base de datos obtenida.

Los manejadores de bases de datos relacionales aceptados por la herramienta para las bases de datos de entrada son PostgreSQL y JDBC, mientras que Neo4J y OrientDB son los manejadores de baeses de datos de grafo aceptados para las bases de datos de salida.

### 3.1.3. Transformación de los datos

Como se mencionó anteriormente, **R2G** sigue una estrategia donde se agrupan los atributos que tienen más probabilidades de aparecer juntos en un join en un mismo nodo, lo cuál puede llevar a acumular demasiada información en un solo nodo.

Por ello, los autores adoptan una estrategia a la que llaman *infiabilidad*, donde dos valores $v_1 = t_1[A]$ y $v_2 = t_2[B]$ son *infiables* si se cumple alguna de las siguientes premisas:

- $t_1 = t_2$ y $A$ y $B$ no pertenecen a una clave compuesta
- $t_1$ y $t_2$ son joineables y $A$ pertenece a una clave compuesta
- $t_1$ y $t_2$ son joineables, $A$ y $B$ no pertenecen a una clave compuesta, y no hay ningún otra tupla $t_3$ que sea joineable con $t_2$

Esta noción garantiza una distribución eficiente entre de los datos en los nodos así como también una buena eficiencia en la ejecución de consultas en la base de datos de salida.

### 3.1.4. Transformación de las consultas

La herramienta planteada por los autores permite la transformación de consultas que incluyen selección, proyecciones y joins donde se crea un "Query Template"que tiene los subgrafos de la base de datos de salida que tienen los resultados de la consulta.

Los pasos para la transformación de una consulta Q son los siguientes:

1. Se identifica un set mínimo $SP$ donde para cada join $R_i.A_i = R_j.A_j$ en la consulta Q hay una arísta $(R_i.A_i, R_j.A_j)$ en al menos un $sp \in SP$.
2. Si en el **WHERE** de la consulta existe un condición sobre un atributo $R.A$ que no ocurre en ningún path de $SP$, se agrega un nuevo path $sp$ a $SP$ que contenga a $A$ y a otro atributo de otro path $sp' \in SP$
3. Se construye una base de datos relacional $r_{\mathbf{Q}}$ que contenga:
   a) Un set de tablas $R(A)$ cuyas instancias tengan el valor $c$ para cada condición de selección $R.A = c$ en el **WHERE** de la consulta.
   b) Un set de tablas $R(A)$ cuyas instancias tengan el caracter especial ? por cada atributo R.A en el **SELECT** de la consulta.
   c) El "Query Template"mencionado en la sección de mapeo de consultas se obtiene aplicando la conversión de datos a $SP$ y $r_Q$ mencionada en la sección de mapeo de datos.

### 3.1.5. Notas finales y conclusiones

Para probar la herramienta implementada, los autores utilizaron datasets generados específicamente para dichas pruebas así como también datasets reales. Las pruebas fueron divididas en los siguientes cuatro escenarios.

1. El primer escenario muestra cómo se realiza la migración de los datos permitiendole al usuario visualizar cómo se transforman las bases relacionales en bases de datos de grafos mediante el uso de herramientas visuales.
2. El segundo escenario muestra cómo se mapean las consultas SQL en consultas de bases de datos de grafo, permitiendole al usuario ver paso por paso cómo se transforman dichas consultas. En la figura 1 se puede ver una consulta SQL y su respectiva transformación al lenguaje Gremlin en la figura 2.
3. Para el tercer escenario, el usuario puede ingresar consultas SQL y obtener su trasnformación para Neo4J, OrientDB o Gremlin.

4. Este escenario muestra que la herramienta desarrollada por los autores es mejor que sus competidores, ya que Neo4J y OrientDb no tienen capacidad para transformar consultas.

   Los autores no brindan números en cuanto a rendimiento, y dado a que la herramienta no se encuentra disponible al momento de realizado este informe, no se pudo corroborar que la metodología presentada por los autores sea más performante que otras.

```
1        SELECT US.uname FROM User US, Blog BG
2        WHERE (BG.admin = US.uid) AND (BG.bname = 'Inf. Systems')
```

Figura 1: Consulta SQL para obtener los usuarios que escribieron el artículo 'Inf. Systems'

```
1        g.V.filter\{it.BGbname=='Inf. Systems'\}
2        .outE.filter\{it.label=='BLOG\_ADMIN'\}
3        .inV.'USuname'
```

Figura 2: Consulta en Gremlin para obtener los usuarios que escribieron el artículo 'Inf. Systems'

## 3.2. Automated Generation of Graphs from Relational Sources to Optimise Queries for Collaborative Filtering

Este artículo [6] está orientado a un problema en particular que es el *filtro colaborativo* para motores de recomendaciones. Esto es utilizado para realizar sugerencias a usuarios en base a ciertas preferencias y es muy utilizado en muchos tipos de negocios, por lo que es de interés poder encontrar una forma óptima en transformar bases de datos relacionales que contengan datos utilizados para dichos motores de recomendación, en bases de datos de grafos para una mayor eficiencia. Los autores mencionan que los procesos de ETL donde se transforman bases de datos relacionales a bases de datos de grafos muchas veces son hechos a medida, lo que se torna costoso. Es por esto que proponen un método para automatizar dicha transformación.

### 3.2.1. Definiciones previas

Definimos un esquema relacional $S(R_1, R_2, ..., R_n)$ como un set de relaciones $R_i$, donde cada $R_i$ consiste de un set de atributos $A_i(A_{i1}, A_{i2}, ..., A_{im})$ que puede ser identificado por un set de claves primarias $PK_i(PK_{i1}, PK_{i2}, ..., PK_{in})$. Un atributo de clave primaria $Pk_{ik} in Pk_i$ puede ser usado para referenciar otra relación $R_j$, también conocido como clave foránea. Un set de claves foráneas para una relación $R_j$ se define como $FK_j(FK_{j1}, FK_{j2}, ..., FK_{jn})$ y $FK_j \subset PK_j$.

Un grafo G está definido como $G = (V, E, T_V, T_E)$, donde $V$ es un set finito de nodos, $E \subseteq VxV$ es un multiset finito de arístas, $T_V$ es un set finito de tipos de nodos, y $T_E$ es un set finito de tipos de arístas.

Cada nodo está mapeado a un tipo de nodo mediante una función de mapeo $\phi V : V \to T_V$ y cada arísta se mapea a un tipo de arísta mediante una función de mapeo $\phi E : E \to T_E$.

Un nodo $v_i \in V$ o una arista $e_k(v_i, v_j) \in E$ tiene un set de tuplas $< attribute, value >$. El esquema de un property graph G está definido como un grafo dirigido $GS = (T_V, T_E)$ donde $T_V$ es un set finito de tipos de nodos y $T_E \subseteq T_V x T_V$ es un set finito de tipos de arístas.

### 3.2.2. Transformación de los datos

Sea $K$ el número total de cláves foráneas en una relación $R_i$. Por cada relación $R_i$ con un conjunto de cláves primarias $PK_i$, dado un atributo $A_{ij}$ de dicha relación, se le llama $|CA_{ij}|$ a la cardinalidad de ese atributo, es decir, la cantidad de valores distintos que existen para el mismo. Por lo tanto, $|PK_i|$ es el número total de tuplas en la relación. El radio de valores a tuplas es:

$$\lambda = |CA_{ij}|/|PK_i|$$

Sea T un umbral para un dominio dado. Las siguientes reglas se aplican para la transformación de un esquema relacional S a un esquema de grafos GS:

1. Si $K = 1$ en $R_i$, para cada $r_i \in R_i$ referenciando a $r_j \in R_j$, se crean dos vértices para $r_i$ y $r_j$ unidos por una arísta con la propiedad $FK_{i1}$.

2. Si $K = 2$ en $R_i$, para cada $r_i \in R_i$ referenciando a $r_j \in R_j$ y $r_k \in R_k$, crear dos vértices para $r_j$ y $r_i$ unidos por una arísta con propiedades equivalentes a la de todos los atributos de $r_i$.

3. Si $K \geq 3$ en $R_i$, para cara $r_i \in R_i$ referenciando a $r_j \in R_j$, $r_k \in R_k$, ..., $r_n \in R_n$ crear $n+1$ vértices $r_i, r_j, ..., r_n$ con vértices con propiedades $FK_{i1}...Fk_{in}$ respectivamente.

4. Para cada atributo $A_i$ en algún nodo, si $\lambda$ es menor que el umbral T, se crea un nuevo nodo que contiene al atributo, un nuevo id se autogenera para dicho nodo y se relaciona al nodo padre al cual el atributo solía pertenecer.

5. Si un nodo $r_i$ ya existe, no será creado de nuevo, sino que será utilizado con otros vértices creados utilizando las tres primeras reglas

El método asume que el esquema está en la 5ta forma normal y está fundado por la aplicación de un conjunto de reglas. La razón para esto es que para un grado menor de normalización, podría existir una relación en la que todas las columnas pueden formar una clave primaria compuesta, lo que significa que no tendría columnas que no sean claves. Esto significa que en la migración, cada tupla se transformará en un nuevo nodo lo que disminuiría la velocidad de ejecución de las consultas.

#### 3.2.3. Notas finales y conclusiones

Para las pruebas de rendimiento, los autores utilizaron la base de datos de Movielens que incluye una gran cantidad de reseñas de películas.

Se utilizó un motor de recomendaciones implementado en Scala para realizar recomendaciones en base a información provista por los usuarios, y se utilizó five-fold cross validation para medir el porcentaje de aciertos del motor de recomendaciones utilizando la metodología planteada por los autores y la metodología planteada por Neo4j que se basa en la utilización de solamente las primeras tres reglas del método propuesto por los autores.

Ambas metodologías arrojaron resultados correctos en el 67 % de los casos pero la metodología propuesta por los autores, a la que llamaremos Optimised Collaborative Filtering (OCF), fue sustancialmente más eficiente.

En las figuras 3 y 4 se puede observar la diferencia entre la eficiencia de OCF y la de la herramienta ETL de Neo4j.

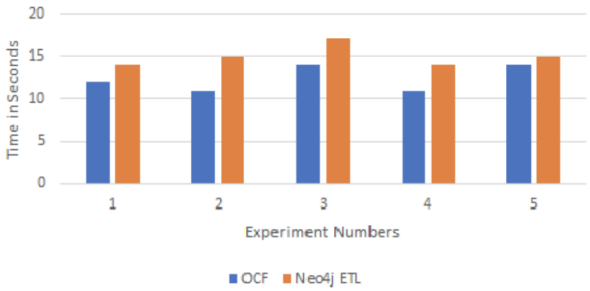


Figura 3: Resultados de five-fold cross validation para cualquier tipo de usuarios utilizando OCF y Neo4j ETL

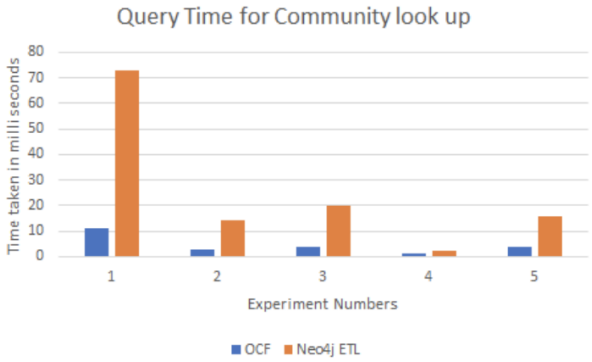


Figura 4: Resultados de cinco consultas para usuarios nuevos utilizando OCF y Neo4j ETL

## 3.3. Migration of Data from Relational Database to Graph Database

El artículo [4] es introducido mencionando la complejidad de los documentos legales y como un documento está relacionado a un artículo de otro documento que a la vez hace referencia a un párrafo de otro artículo, y otros ejemplos que dejan en evidencia la semejanza entre los documentos legales de una jurisdicción con un grafo.

Quizás este artículo no es uno de los más interesantes y complejos que han sido seleccionados para este trabajo, pero es de interés ver cómo el hecho de tomar un enfoque bastante inocente a una realidad modelada utilizando bases de datos relacionades puede llevar a una mejora significativa en el rendimiento.

### 3.3.1. Transformación de los datos

Las reglas utilizadas para la migración de la base de datos relacional a la base de datos de grafos son las siguientes:

1. Cada tabla de entidad es representada por una etiqueta en los nodos.
2. Cada tupla en la tabla de entidad es representada como un nodo.
3. Las columnas de dichas tablas se convierten en propiedades del nodo.
4. Se eliminan todas las claves primarias y se mantienen las claves primarias de dominio.
5. Se crean restricciones de unicidad para claves primarias de negocio, y se agregan índices para atributos que sean frecuentemente utilizados para búsquedas.
6. Reemplazar claves foráneas con relaciones a las otras tablas, y luego eliminar dichas claves.
7. Remover valores por defecto.
8. La información que esté duplicada y en tablas que no estén normalizadas tienen que ser extraída a nodos separados para obtener un modelo más limpio.
9. Tablas utilizadas para joins (Join tables) se transforman en relaciones y las columnas de dichas tablas se agregan como atributos a la relación.

### 3.3.2. Notas finales y conclusiones

Se escribieron consultas para los modelos relacionales y de grafos para poder realizar una comparación entre ambos modelos.

La consulta SQL de la figura 5 realiza un select con dos joins de tablas que tienen una enorme cantidad de datos. Los autores mencionan que, además de que la consulta implementada en Cypher de la figura 6 es más fácil de leer, demora 10 veces menos que su equivalente en el modelo relacional.

Mencionan también que obtener 1000 tuplas en la base de datos de grafos es 6 veces más rápido que obtenerlas en la base de datos relacional, y obtener 10000 tuplas es 30 veces más rápido, lo que demuestra que el rendimiento de la base de datos de grafos no se degrada con el crecimiento de la misma, contrario a lo que sucede en la base de datos relacional.

Los autores aclaran que las bases de datos de grafos no siempre pueden ser apropiadas y que, por lo tanto, se debe investigar los datos antes de definir que tipo de base de datos usar.

Mencionan que los datos utilizados en el estudio realizado tienen una estructura arborescente, y realizar las consultas en la base relacional no es muy eficiente al momento de tener grandes volúmenes de datos. Sin embargo, al transformar el esquema en un esquema de grafo, el rendimiento mejora drásticamente.

```sql
SELECT law.first_name FROM law
INNER JOIN legislation
ON law.legistation_id = legistation.id
INNER JOIN legislation_set
ON legislation_set.legislation_set_id = legislation.legislation_set_id
WHERE legislation_set.legislation_set_first_name = 'Vergi Mevzuat Seti';
```

Figura 5: Obtener las leyes del sistema cuyo padre es 'Vergi_Mevzuat_Seti' (SQL)

```
match(law:law)-[*]->(legislation_set:legislation_set) where
legislation_set.legislation_set_first_name='Vergi Mevzuat Seti'
return kanun;
```

Figura 6: Obtener las leyes del sistema cuyo padre es 'Vergi_Mevzuat_Seti' (Cypher)

## 3.4. Property Oriented Relational-to-Graph Database Conversion

El artículo [3] comienza planteando que a partir del surgimiento de las bases de datos NoSQL y con el paso del tiempo, tanto desarrolladores como investigadores han notado que mucha información utilizada en variadas áreas de conocimiento pueden representarse más naturalmente a través de modelos de grafos. Expone que la creciente necesidad de análisis de big data ha afectado el almacenamiento de información en grafos, especialmente para *graph mining*, es decir el estudio de algoritmos orientados a encontrar patrones dentro de un grafo normalmente relacionado a métodos de *mining* frecuente en subgrafos.

Se presenta además el enfoque del artículo: el diseño e implementación de algoritmos universales para conversión de datos de bases relacionales a bases de grafos teniendo en mente métodos de *graph mining*. Asegura que se presentan en el trabajo logrado características de uso de las propiedades de nodos y aristas en el property graph, independencia de la semántica de los datos y conversión sin pérdida de información. La base de datos de grafos resultante provee una buena fuente de información para algoritmos de *graph mining*.

### 3.4.1. Transformación de los datos

Los puntos clave del algoritmo propuesto son:

1. El grafo no dirigido creado en el proceso no será necesariamente conexo ni acíclico, estos aspectos dependerán de la estructura de la base de datos y los datos contenidos.
2. Cada nodo representa una tupla de la base de datos relacional. Algunas aristas representan una tupla, otras representan una clave foránea.
3. Las etiquetas se usan para nombrar los diferentes tipos de nodos y aristas. Cada nodo y cada arista tienen exactamente una etiqueta, que ayuda a determinar el tipo de un nodo o una arista y es crucial para el procedimiento.
4. La etiqueta de un nodo es el nombre de la tabla de la base de datos que contiene la tupla que representa el nodo. Lo mismo sucede con una arista si ésta representa una tupla. Si la arista representa una clave foránea entonces el nombre de dicha clave foránea es la etiqueta de la arista.
5. Las propiedades de nodos y aristas son pares clave-valor que representan los atributos y valores de la tupla representada. El atributo es la clave y el valor del atributo en la tupla es el valor del par. Las aristas que representan claves foráneas no tienen propiedades.
6. Cada nodo o arista que representa una tupla tiene la propiedad id con el valor de la clave primaria de la tupla. Si la clave primaria tiene más de un atributo se usa la concatenación de los valores de los atributos como clave. Todos los elementos de una misma tabla del esquema relacional tienen

la misma etiqueta y un id único, es decir el id no es único en todo el grafo pero la combinación etiqueta-id si es única.

7. Las tablas con exactamente dos claves foráneas cuya clave primaria no es referenciada por ninguna otra tabla representan en el esquema relacional las relaciones M:N. Las tuplas de estas tablas se representan como aristas que conectan los nodos que representan a las dos tuplas referenciadas en las claves foráneas.

8. Las tuplas de todas las tablas que no cumplen con las características del punto (7) se representan con nodos.

9. En caso de relaciones paralelas no cíclicas entre dos tablas (una tabla tiene más de una clave foránea que referencian a la misma tabla), los nodos que representan tuplas de esas tablas tendrán más de una arista entre ellos, cada una con la etiqueta correspondiente a la clave foránea que representan.

10. Los atributos de las tuplas que son parte de claves foráneas no forman parte de las propiedades. Los valores de la clave primaria quedan contenidos en la propiedad id, los valores de clave foránea quedan contenidos en la relación con otro elemento del grafo, y solo los atributos dependientes se escriben como propiedades de los elementos del grafo.

**Análisis de Metadata** Dado que el objetivo es definir un algoritmo eficiente para exportar datos de una base relacional e importarlos a una base de grafos, es importante que los datos de cada tabla sean considerados la mínima cantidad de veces posible. Por esto, antes de generar la conversión, se estudia la metadata del esquema, reconociendo las siguientes categorías:

- Tablas cuyas tuplas serán aristas
- Tablas cuyas tuplas serán nodos
- Relaciones cíclicas formadas por claves foráneas que generan ciclos en el grafo
- Orden de creación de nodos y aristas
- Atributos de cada tabla que forman la clave primaria, que conforman el id
- Atributos de cada tabla que son parte de claves foráneas, que conforman aristas entre nodos
- Atributos de cada tabla que no son parte de ninguna clave y conforman propiedades
- Para lograr esto se utilizan algoritmos, algunos presentados en los próximos puntos, cuyo rendimiento depende de la complejidad del esquema relacional.

**Orden de conversión** El algoritmo utilizado considera primero las tablas que tienen exactamente dos claves foráneas y cuya clave primaria no es referenciada por ninguna otra tabla, y las coloca en un set *ToEdges*. Las tuplas de estas tablas serán convertidas en aristas luego de crear los nodos del grafo. El resto de las tablas, que se representaran con nodos, se ordenan en una lista *ToNodes*, donde aparecen

primero las tablas que no tienen claves foráneas. Luego se repite el proceso de agregar tablas a la lista que cumplan que todas sus claves foráneas referencian tablas que ya están en la lista hasta que no haya más tablas para agregar. Cada tupla de estas tablas se convierte en un nodo y cada clave foránea de la tupla en una arista al nodo de la tupla referenciada. El orden asegura que el nodo de la tupla referenciada siempre exista al momento de crear la arista.

**Referencias cíclicas** El algoritmo que chequea si una tabla tiene alguna referencia cíclica es un algoritmo recursivo que recorre todos los caminos posibles a través de claves foráneas que parten de la tabla analizada. El paso base es que la tabla considerada en el algoritmo sea la misma que la referenciada por la clave foránea considerada. En ese caso se agrega la clave foránea al conjunto *CyclicFK*, cuyos elementos se considerarán últimos en la conversión. Se mantiene además referencia a las claves ya analizadas en cada tabla para asegurar la salida de la recursión. Este es el único momento del proceso en el que se vuelven a visitar los datos del esquema relacional, para determinar qué nodos conectan las nuevas aristas.

### 3.4.2. Notas finales y conclusiones

Para analizar la efectividad del método presentado, se lo compara con otros dos métodos conocidos: *Zero-property model* y *Node-property model*. En el primero cada tupla es un nodo, cada atributo de la tupla es un nodo conectado al nodo de la tupla y cada clave foránea es una arista. El segundo modelo convierte cada tupla en un nodo, a cada atributo en una propiedad y a cada clave foránea en una arista. Se utilizan como datos de entrada parte de sitios web de Stack Exchange orientados a varios profesionales. Los datos consisten de ocho tablas, de las cuales se omite una para simplificar, quedando un total de siete tablas con 903.298 tuplas, 57 atributos, y un promedio de 1,57 claves foráneas por tabla. Los resultados de la conversión muestran significativa menor cantidad de nodos que *Zero-property* pero con más propiedades por nodo (*Zero-property* no genera propiedades), y alrededor de 20 % menos nodos y 15 % menos propiedades respecto a *Node-property*. Para analizar la velocidad de ejecución de las consultas en los tres métodos, se ejecutaron las siguientes tres consultas:

- q1: Usuarios y su radio de aceptación de respuestas. Obtiene los usuarios, numero de respuestas, numero de respuestas aceptadas. Se tiene en cuenta solo aquellos usuarios que tengan al menos 10 respuestas posteadas y al menos una aceptada.
- q2: Usuarios y su puntaje de comentarios. Obtiene los usuarios, su numero de comentarios y el puntaje promedio de sus comentarios. Solo se tienen en cuenta usuarios que tengan al menos 20 comentarios.
- q3: Posts duplicados. Obtiene posts duplicados. Solo se tienen en cuenta posts duplicados hasta el cuarto nivel de duplicación (la cuarta copia del original).

En la figura 7 se muestran los tiempos de ejecución de las consultas utilizando cada uno de los métodos mencionados.

Al analizar el comportamiento de las bases convertidas frente a distintas consultas, se comprobó que el método planteado es ampliamente más eficiente que *Zero-property* y levemente más eficiente que *Node-property*. Las consultas ejecutadas analizan gran cantidad de nodos, y lógicamente a mayor cantidad de datos, más lenta será la ejecución de la consulta.

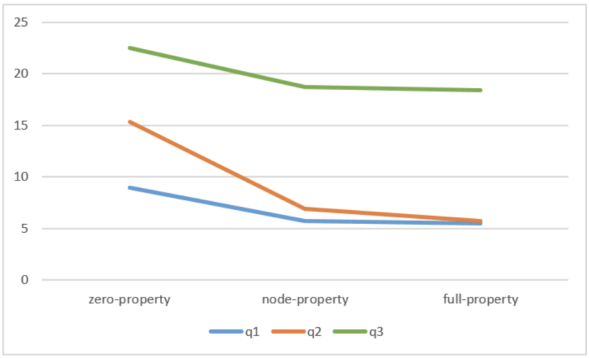


Figura 7: Resultados de la ejecución de las consultas q1, q2 y q3

## 3.5. Table2Graph: A scalable Graph Construction from Relational Tables using Map Reduce

Este artículo [2] presenta un enfoque diferente a los anteriores, ya que enfrenta el problema de integrar datos de distintas bases de datos relacionales para obtener una base de datos de grafos. Esto implica colocar los datos en en diferentes estructuras para luego unificarlas en un único modelo.

Los autores plantean que aunque existen soluciones a este problema que funcionan correctamente, estas soluciones tienen la desventaja de no ser escalables ni flexibles a modificaciones en los modelos de origen de los datos. Proponen entonces Table2Graph, una implementación escalable y reconfigurable de un proceso de extracción, transformación y carga (ETL por sus siglas en inglés). Con esta implementación los usuarios podrán definir los mapeos necesarios entre las bases origen y la base de grafos y estos datos se convierten automáticamente en código de MapReduce para instanciar el grafo. La configuración del usuario quedará guardada en un archivo XML para que puedan hacerse cambios futuros directo en el archivo.

### 3.5.1. Transformación de los datos

Table2Graph opera en dos pasos, *Schema Mapping* y *Data Conversion*. En el primero los usuarios crean un mapeo de una base de datos relacional origen a un esquema de base de datos de grafos, y en la segunda se transforman las tuplas de las tablas origen a nodos y aristas de un grafo de acuerdo a lo especificado en el primer paso.

#### Schema Mapping

Este paso esta diseñado para permitir configuración y edición flexible, de manera que el mapeo creado pueda ser reutilizado a partir de modificaciones simples al archivo XML generado. El mapeo se genera independientemente para nodos y aristas. Para los nodos, este paso implica identificar uno o varios atributos del esquema relacional que constituyan el nodo objetivo (*clave* del nodo), otros atributos que se incluirán como *propiedades* del nodo objetivo, y una *etiqueta* para el nodo objetivo. De manera similar, para las aristas se identifica la *clave*, las *propiedades* y la *etiqueta*, además de un par *desde* y *hasta* que consiste de las *claves* de los nodos conectados por la arista.

#### Data Conversion

El segundo paso del método ejecuta una serie de tareas de map-reduce en dos fases. Durante la primer fase, muchas tareas de map-reduce crean colectivamente nodos y artistas temporales que llamamos *componentes* de nodos y aristas. Un identificador de recursos uniforme (URI) se usa para identificar un nodo o arista durante esta fase. En la segunda fase los componentes con el mismo URI se unen para producir los nodos y aristas del grafo final.

Las tareas en *Data Conversion* se dividen en cuatro tipos:

- **Constructor de componentes de nodo:** Toma como entrada cinco parámetros R, U, P, $\psi$ y l. R es una relación de la base de datos origen, U es in conjunto de atributos URI, P un conjunto de atributos que serán propiedades del nodo, $\psi$ es una fórmula proposicional lógica y l es la etiqueta de tipo de los nodos a generar. Devuelve un conjunto de nodos de tipo l resultantes de convertir la relación R. Las URI de los nuevos nodos se generan a partir de los valores de los atributos en U, y las propiedades se corresponden con los atributos en P.
- **Constructor de componentes de arista:** Toma como entrada los mismos parámetros que el constructor anterior, en este caso l una etiqueta de tipo de arista. Además recibe la información de los nodos que unen la arista, es decir $U_from$ y $U_to$ que son conjuntos de los atributos URI del nodo origen y destino, y $l_from$ y $l_to$ que son las etiquetas de tipo de los nodos origen y destino. Devuelve un conjunto de aristas de tipo l resultante de convertir la relación R dada de manera similar al constructor de componentes de nodos.
- **Unificador de conjuntos de nodos:** Recibe los componentes de nodo de los constructores ejecutados y unifica aquellos que tienen el mismo URI para generar los nodos definitivos del grafo.
- **Unificador de conjuntos de aristas:** Recibe los componentes de arista de los constructores ejecutados y unifica aquellos que tienen el mismo URI para generar las aristas definitivas del grafo.

### 3.5.2. Notas finales y conclusiones

El enfoque del artículo es generar un método de conversión de bases de datos relacionales a bases de datos de grafos que sea escalable y fácil de reutilizar. Para probar esto se utilizó parte de los datos de Amazon Web Service utilizando distintos niveles de recursos informáticos. Como estaba previsto, el rendimiento fue notablemente mejorado con más recursos y por lo tanto capaz de procesar mayor cantidad de datos en paralelo. Los autores concluyen que lograron el objetivo planteado pero pretenden continuar investigando para volver el proceso más automático y adaptar el modelo a usos con sistemas de procesamiento de datos en memoria como Apache Spark.

## 3.6. R2PG-DM - a direct mapping from relational databases to property graphs

Este artículo [5] se introduce planteando la realidad de que, a pesar de que la mayoría de los datos hallados en la práctica hoy por hoy se encuentran expresados en formatos de Bases de Datos Relacionales (BDR), se ha comprobado que las estructuras de bases de datos de grafos son extremadamente útiles para reconocer patrones complejos de relación entre datos. Se propone que, en particular, los Property Graphs (PG) se han vuelto una herramienta muy importante en el análisis de datos.
Los autores plantean entonces que es fundamental entender las relaciones entre BDRs y PGs, empezando por el estudio de mapeo directo de RDBs a PGs, definiendo un mapeo directo como una transformación que debe ser independiente del esquema y el dominio de los datos, además de transformar el contenido de una instancia en el formato original a una instancia en el formato objetivo, en este caso una instancia de BDR debe generar una instancia de PG.
Se asegura que el mapeo directo presentado (R2PG-DM) transforma tanto la instancia como el esquema sin perder información y preservando la estructura original de la BDR ingresada sin necesidad de la interacción del usuario. Esto simplifica la tarea de transformación ya que no se necesita un conocimiento específico del modelo del resultado final deseado para utilizar la herramienta.

### 3.6.1. Transformación de los datos

**Transformación de instancias**

- Cada tupla t de una relación r en los datos ingresados se representa en el nuevo grafo con un vértice vt, identificado por el id de t y con etiqueta r. Además, por cada atributo a de la tupla t se agrega un par clave-valor (a,t(a)).
- Para cada relación de clave foránea f en la BDR, se agrega una arista entre cada par de nodos vt y vt´ correspondientes a las tuplas t y t´ que se relacionan por f. La arista se etiqueta r-s, donde r y s son las relaciones a las que pertenecen t y t´ respectivamente.

**Transformación de esquema**

En cuanto al esquema de la BDR, cada relación y cada atributo de la relación se representa también con un vértice y las claves foráneas entre atributos se representan con aristas. Sea **R** la BDR que se desea transformar, el esquema resultado tiene cuatro tipos de nodos según su etiqueta que pueden tener propiedades y estar conectados por aristas:

1. Nodos *Rel* que representan el nombre de una relación $r$ en **R**.
2. Nodos *Att* que representan un atributo a de una relación $r$ en **R**.
3. Nodos *fk* que representan una clave foránea entre dos atributos $ar$ de una relación $r$ y $as$ de una relación $s$, donde $r$,$s \in$ **R**.

4. Nodos *Fk* que representan una clave foránea entre los atributos $a_{r_1}, \ldots, a_{r_n}$ de la relación $r$ y $a_{s_1}, \ldots, a_{s_n}$ de la relación $s$, donde $r$,$s \in \mathbf{R}$.

Existen tres tipos de aristas según su etiqueta: aristas *Rel-Att*, *Att-fk* y *fk-Fk*, cada una representando una arista entre dos nodos con las respectivas etiquetas que forman el nombre de la etiqueta de la arista.

El esquema contiene propiedades para nodos de tipo *Rel* y tipo *Att*. Los nodos *Rel* tienen una sola propiedad *"nombre"* que contiene el nombre de la relación representada, mientras que los nodos *Att* tienen la propiedad *"nombre"* que contiene el nombre del atributo representado y la propiedad *"pk"* que contiene un valor booleano especificando se el atributo forma parte de la clave primaria de una relación $r$ en $\mathbf{R}$.

- Generación de nodos:

  Se genera un nodo *Rel* por cada relación $r$ en $\mathbf{R}$, y un nodo *Att* por cada atributo $a$ de cada relación $r$ en $\mathbf{R}$. Luego, para cada clave foránea *fKey* entre dos relaciones $r$,$s \in \mathbf{R}$ se genera un nodo *fk* por cada atributo que compone *fKey*. Finalmente se genera un nodo *Fk* por cada clave foránea entre dos relaciones $r$,$s \in \mathbf{R}$.

- Generación de aristas:

  Sea $r$ una relación en R y $a$ un atributo de $r$, se genera una arista entre los nodos que representan a $r$ y $a$ en el esquema, con la etiqueta *Rel-Att*.

  Dada una clave foránea *fKey* entre el atributo $a_s$ de la relación $s$ y el atributo $a_r$ de la relación $r$, donde $a_s \subseteq a_r$, se genera una arista del nodo de tipo *Att* que representa a $a_s$ a un nodo de tipo *fk* $f$ y una arista del nodo $f$ al nodo de tipo *Att* que representa a $a_r$, ambos con etiqueta *Att-fk*.

  Finalmente se generan aristas de los nodos *fk* a los nodos *Fk* que representan las claves foráneas formadas por los atributos asociados a cada *fk*.

### 3.6.2. Arquitectura de la herramienta

El mapeo se implementa como una aplicación Java usando la herramienta Maven. Se utiliza la biblioteca JDBC para obtener el esquema de la base de datos, que soporta MySQL 5.1, Java DB 10.5.3.0, Oracle 11, PostgreSQL 8.4, DB2 9.7, Sybase ASE 15 y Microsoft SQL Server 2008.

La aplicación Java toma como entrada una instancia de BDR de la cual extrae metadata como nombres de relaciones y claves foráneas. Esta información se guarda en otra instancia de BDR con tres tablas: nodo, propiedad y arista. Además, se utilizan tres clases en la implementación que representan elementos de un PG. La clase nodo se utiliza para guardar información de un nodo (id, etiqueta), la clase propiedad se utiliza para guardar información de una propiedad (id, clave, valor) y la clase arista se utiliza para guardar información de una arista (id, idOrigen, idDestino, etiqueta). La implementación comienza extrayendo toda la información de la base de entrada, y luego se divide en dos pasos: primero crear nodos y propiedades y luego crear las aristas.

La implementación de nodos y propiedades se basa en iterar sobre las tablas y por cada nombre de relación $r$ se obtiene toda la información relacionada, incluyendo el id de la relación *rId*. Se crea un objeto de clase nodo para el nombre de la relación $r$ y un objeto de clase propiedad para cada atributo de la relación de nombre $r$. Luego se inserta la información en la base de salida.

Para implementar las aristas se necesita conseguir el id de los nodos de origen y destino. Para lograr esto se utilizan las claves foráneas como parámetro. Es decir por cada clave foránea (compuesta o no) entre una relación $r$ y una relación $s$ se hace un *join* entre las relaciones basado en los atributos de la clave foránea para obtener los valores de los nodos relacionados de la base de entrada. Estos valores se usan en una segunda consulta a la parte ya generada de la base de salida para encontrar el id de los nodos. Finalmente se crea la arista y se inserta la información en la base de salida.

Para que la implementación sea escalable se limita el alcance de las operaciones a una tabla por vez, de manera que no se ocupe demasiado espacio en memoria en un mismo momento. Cuando todas las operaciones sobre una tabla terminan, se libera la memoria ocupada por la misma.

Al finalizar el mapeo se exporta el PG a archivos *.scv* que luego pueden importarse a Neo4j usando un archivo cypher personalizado. El código de implementación de la herramienta se encuentra disponible en el repositorio `https://github.com/radualex/R2PG-DM` y es open-source.

### 3.6.3. Notas finales y conclusiones

Para probar el funcionamiento de la herramienta los autores utilizan distintas bases de datos tomadas de distintos orígenes, entre ellas un conjunto de datos reales de círculos de amigos de Facebook, el conjunto de datos reales de películas IMDb y un conjunto de datos sintéticos de un vendedor de modelos a escala de autos clásicos. Para probar la precisión se compara la cantidad de nodos con la cantidad de filas en las tablas originales, la cantidad de propiedades con el producto cartesiano de la cantidad de atributos por la cantidad de filas donde el atributo no es null, y la cantidad de aristas con la suma de la cantidad de valores distintos que componen cada clave foránea en las tablas. Se comprueba que todas son iguales.

Se observa que las bases con pocas tablas y muchos atributos son más lentas que las que tienen más tablas con menos atributos, y que las bases con cantidad de atributos balanceados entre las tablas tienen mejor tiempo de ejecución.

# 4. Conclusiones y trabajo futuro

Como primera observación, es claro que generar un algoritmo universal que sea óptimo para cualquier realidad es muy difícil, por no decir imposible.

Esto se debe a que las bases de datos de grafos se construyen teniendo en cuenta las consultas que se van a realizar y no los datos que contendrán, a diferencia de las bases de datos relacionales.

Cuatro de los seis artículos estudiados fueron realizados en base a realidades puntuales. El artículo [6] fue ideado teniendo en cuenta el problema de filtro colaborativo para recomendaciones; el artículo [4] tiene como fin mejorar la eficiencia de las consultas en una base de datos de documentos legales con muchos ciclos; en el caso de el artículo [3] se centra el estudio en la recopilación de datos o graph mining; y el artículo [2] busca integrar datos de distintas bases de datos relacionales en una sola base de datos de grafos. Por otro lado, los artículos [1] y [5] buscan encontrar un algoritmo definitivo para la automatización de transformaciones para cualquier realidad.

Dicho esto, creemos que ninguno de los métodos planteados son mejores o peores sino que cada uno puede funcionar mejor o peor dada cierta realidad. Sin embargo, en caso de que la eficiencia de las consultas no sea un requisito mayor, cualquiera de los métodos es válido, es decir, sirven para transformar una base de datos relacional a una base de datos de grafos.

Dentro de los objetivos iniciales de este trabajo, se planteaba realizar una comparación de todos los métodos investigados utilizando distintos tipos de bases de datos relacionales, más específicamente, bases de datos con forma de grafos, árboles, etc. Sin embargo, debido a que solo una de las herramientas se encontraba disponible al momento de realizar el trabajo, no fue posible cumplir dicho objetivo.

Por lo tanto, se considera postergar este objetivo a futuro esperando obtener dichas herramientas, aprender a utilizarlas y lograr compararlas.

# Referencias


[1] Riccardo Torlone Roberto De Virgilio Antonio Maccioni. *Converting relational to graph databases*. 2013. URL: https://openproceedings.org/EDBT/2014/edbtdemo2014_submission_2.pdf.

[2] Sangkeun Lee Byung H. Park Seung-Hwan Lim Mallikarjun Shankar. *Table2Graph: A Scalable Graph Construction From Relational Tables using Map-Reduce*. 2015. URL: https://ieeexplore.ieee.org/document/7184893.

[3] Mirta Baranović Ognjen Orel Slaven Zakošek. *Property Oriented Relational-To-Graph Database Conversion*. 2016. URL: https://hrcak.srce.hr/180725.

[4] Halit Oguztuzun Yelda Unal. *Migration of Data from Relational Database to Graph Database*. 2018. URL: https://www.researchgate.net/publication/325095439_Migration_of_data_from_relational_database_to_graph_database.

[5] Dr. Juan F. Sequda Radu-Alexandru Stoica Dr. George H.L. Fletcher. *A direct mapping from relational databases to property graphs*. 2019. URL: https://pure.tue.nl/ws/portalfiles/portal/139492780/R2PG_DM_Final.R.A.Stoica.pdf.

[6] Frans Coenen Ahmad Shahzad. *Automated Generation of Graphs from Relational Sources to Optimise Queries for Collaborative Filtering*. 2020. URL: http://www.thinkmind.org/articles/dbkda_2020_1_40_50020.pdf.

[7] *Sitio web oficial de Neo4j*. URL: https://neo4j.com/.