ReGraph: Bridging Relational and Graph Databases

Patrícia Cavoto[1], André Santanchè[1]

[1]Laboratory of Information Systems (LIS)
Institute of Computing (IC)
Univeristy of Campinas (UNICAMP)
Campinas – SP – Brazil

patricia.cavoto@gmail.com, santanche@ic.unicamp.br

Abstract. *In this paper, we present ReGraph[1], a framework to map data from a relational to a graph database, managing a dynamic coexistence and evolution of both, not supported by related work. ReGraph has minimal impact in the existing infrastructure, providing a flexible and tailored graph model for each relational schema. It uses an initial ETL (Extract, Transform and Load) process to replicate the existing data in the graph database. A scheduled service is responsible for reflecting changes in the relational data into the graph, keeping both synchronized. ReGraph also provides an annotation functionality that allows users to add new information in the mapped graph, providing the materialization of inferences and data enrichment.*

1. Introduction

In the data analysis context, links have been increasingly considered as important as the data elements that they connect. It is possible to analyze these links, and the existing latent semantics, through the analysis of the network topology formed by them. Graph databases are very effective in this type of analysis due to their flexible structure, which helps in defining the fitted model to each required analysis. Moreover, the OLAP (OnLine Analytical Processing) concept of creating an exclusive database for data analysis is a consolidated approach. This database is designed and tuned for the respective operations – over graphs, in our case – allowing users to manage data without impact in the original databases and to dynamically create information over the existing data.

Combining the graph database and the OLAP concept, we propose ReGraph, a framework that generates a graph database from a relational database, providing extra functionalities for analysis. Using a tailored mapping, an ETL process generates the graph database linked to the relational one, preserving both databases in their native forms. Beyond synchronizing the dynamic evolution of the relational database with the graph, our framework supports adding new data in the graph, annotating the mapped one. Annotations enrich the semantics in the existing data and can improve the network analysis performed in the graph database. As part of the ReGraph work, we have been implemented FishGraph, a project in which we generate a graph database from FishBase, a relational database and information system for biological data storage of fish species. Through FishGraph, the relational and the graph databases interact with each other, maintaining

[1] http://patricia.cavoto.com.br/regraph/

their native forms. In [Cavoto et al. 2015] we present two scenarios in which a graph database can be a better choice for network analysis in the FishBase context and how annotations are helpful in improving this analysis.

Concerning the related work, there are several approaches for representing data as a graph structure starting from a relational database. The Database Graph Views [Gutiérrez et al. 1994] proposes an abstraction layer as a mechanism for creating views to manipulate graphs, independent of the physical arrangement where the original data is stored. The D2RQ Map [Bizer 2003] proposes the integration of ontologies in RDF with relational databases through a module that creates a virtual RDF graph. As both approaches do not materialize the graph database, they neither take fully benefit of the advantages in the network analysis offered by a database tuned for graph operations, nor the possibility of adding new data in the existing data. The Virtuoso RDF View [Blakeley 2007] proposes mapping relational data to RDF. It operates exclusively over the RDF model and the Virtuoso relational database, restricting the use of other graph models and database management systems.

In this paper, we present the ReGraph framework, detailing the generation of the graph database starting from a relational database, keeping both in their native forms and interacting with each other, dividing tasks of management and analysis according to their specialties. Moreover, ReGraph allows the creation of new information by annotating the existing data and provides a dynamic evolution of the graph database, reflecting the changes executed in the relational database.

2. The ReGraph Framework

ReGraph is a framework that departs from a relational database, allowing to create and maintain within a graph database two connected subgraphs: mapped and annotation. It mainly addresses a Property Graph model, even though it can also work with the RDF model. Figure 1 illustrates an overview of the ReGraph framework that has three main modules: Mapping, Graph and Sync.

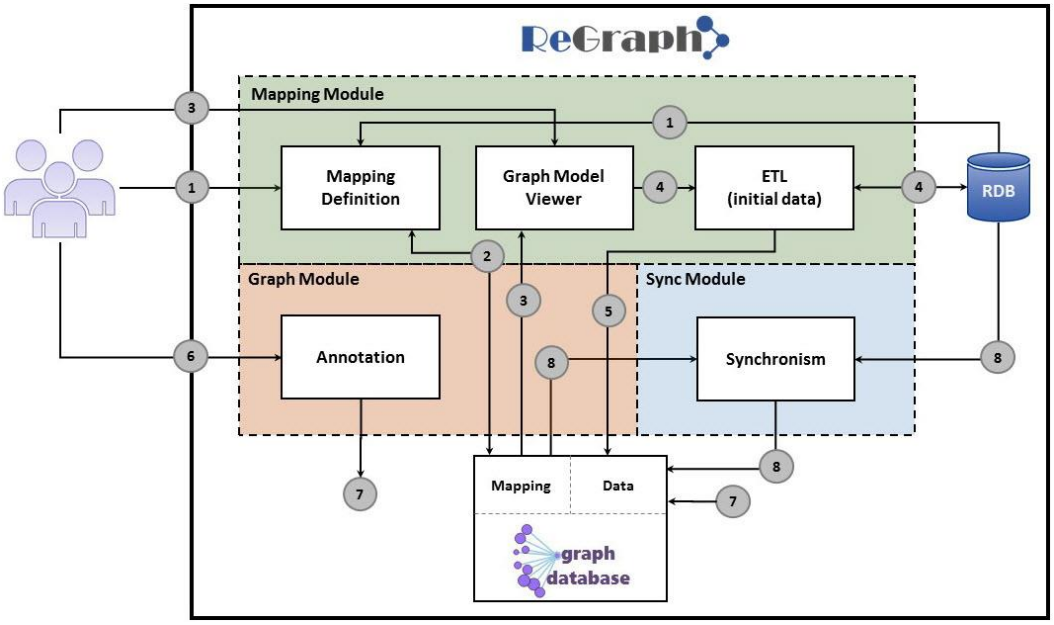

Figure 1 - The ReGraph Framework

ReGraph keeps both databases in their native forms and does not require changes in the relational schema, which will generate a minimal impact on the existing infrastructure. Therefore, it allows keep using the existing systems that operate over the relational database and using the dynamically synchronized graph database for analysis, focusing on the relations among data elements.

To the best of our knowledge, ReGraph is the first framework that migrates data from the relational database to a property graph database, Neo4J, keeping the databases in their native forms and providing a dynamic evolution of both.

2.1. The Mapping Module

In the Mapping Module, through the Mapping Definition process, flow ① in Figure 1, it is possible to map data from a relational database schema to a graph database. There are two mapping options:

a. **Automatic:** maps the relational database schema automatically to a graph database model. Tim Berners-Lee [Berners-Lee 1998] discussed a set of mapping principles when converting relational databases to RDF (Resource Framework Description). The principles are as follows: a record in the relational database becomes an RDF node; a column name becomes an RDF predicate (a labeled edge); and a table cell becomes a value. As the target of ReGraph is a property graph, and the Tim Berners-Lee mapping considers an RDF graph, we have adapted the rules to our model as follows:
   - a table in the relational model becomes a node type, also called label;
   - each record becomes a unique node;
   - each table cell (except the foreign keys) of the record becomes a node's property;
   - each foreign key becomes an edge connecting the nodes generated by the involved records.

b. **Manual:** allows defining tailored mappings, according to their analysis requirements. For each table and column of a given relational database, it is possible to define a mapping to graph labels, nodes, properties and edges. Moreover, one can also map a relational database to an RDF graph model by keeping properties unselected – in this case, we are only considering the RDF graph model, since fully RDF mapping would require other RDF semantic representation concerns.

After the mapping definition, ReGraph saves this mapping in the graph database, creating it as a representation of the graph model, as shown in Figure 1, flow ②. The graph model viewer presents a representation of the mapping and provides a previous data organization in the graph database, flow ③ in Figure 1.

The last feature of this module is the ETL process, responsible for retrieving data from the relational database and inserting data into the graph database, generating the mapped subgraph, as specified in the mapping definition, Figure 1, flows ④ and ⑤. Each table and column mapped from the relational database receives a trigger that registers any insert, update or delete in a notification table, in order to perform the synchronism process in the future. All nodes and relationships generated from the relational database have a special property called "DataSource" defined as "Mapped", which forbids changes in their data directly in the graph database. Any required change in the mapped subgraph data must

occur in the relational database and the synchronism process performs the respective changes in the graph database.

2.2. The Graph Module

The Graph Module allows the interaction with the graph database as annotations over the mapped subgraph, as shown in Figure 1, flow ⑥. While it is not allowed to change the mapped data in the graph, in order to maintain the consistency, it is possible to create new nodes and edges over this data as annotations, Figure 1, flow ⑦. The new annotation nodes and edges have the "DataSource" property, mentioned previously, defined as "Annotation". The Annotation Graph aims to facilitate the network analysis, adding details over the existing data and enabling to materialize intermediary nodes and relations inferred from existing data.

In the end, there are two main distinct, but interconnected subgraphs: the mapped and the annotation. One cannot create nodes and edges in the annotation subgraph using the existing labels defined in the mapped subgraph. This rule aims to prevent the creation of redundant and conflicting data between the mapped and the annotation subgraphs.

2.3. The Sync Module

The Sync Module is responsible for maintaining the graph database synchronized with the relational database, providing a dynamic evolution of both, as shown in Figure 1, flow ⑧. It also keeps the graph database updated with the last changes to perform the network analysis. A scheduled service periodically reads the data registered in the notification tables, populated by the triggers created in the end of the ETL process, and runs the synchronism process, updating the graph database with the new information. There are three main rules applied in synchronism process:

a. **Delete:** there are two distinct policies that can be configured for deleted records in the relational database. The first one is the "Delete" policy, in which, when a mapped record is deleted in the relational database, the synchronism process reflects it in the graph by deleting the respective nodes, properties or edges. In this case, all the related annotations exclusively linked with the deleted nodes are also deleted. The second is the "Keep" policy, in which, when a mapped data is deleted in the relational database, the synchronism process performs only an update in the "DataSource" property, changing it from "Mapped" to "Deleted" and keeping the respective nodes and edges in the graph database. Moreover, it is not possible to perform changes in the "Deleted" nodes and edges. The "Keep" policy has direct impact in the Insert rule;

b. **Insert:** when a new record is inserted into a mapped table in the relational database, the synchronism process will check if this data already exists in the graph database as "Deleted" data. If this data does not exist, the synchronism process maps it onto the graph database, according to the mapping configuration. Otherwise, the synchronism process will return the "DataSource" property to "Mapped" again, and update all the related data;

c. **Update:** when a mapped record in the relational database is updated, the synchronism process will update the respective data in the graph, according to the mapping definition.

3. Software Overview

The ReGraph framework has a web interface, providing easy access to the user through any browser. Figure 2 presents an overview of the interface, with an emphasis in the mapping and the graph model viewer screen.

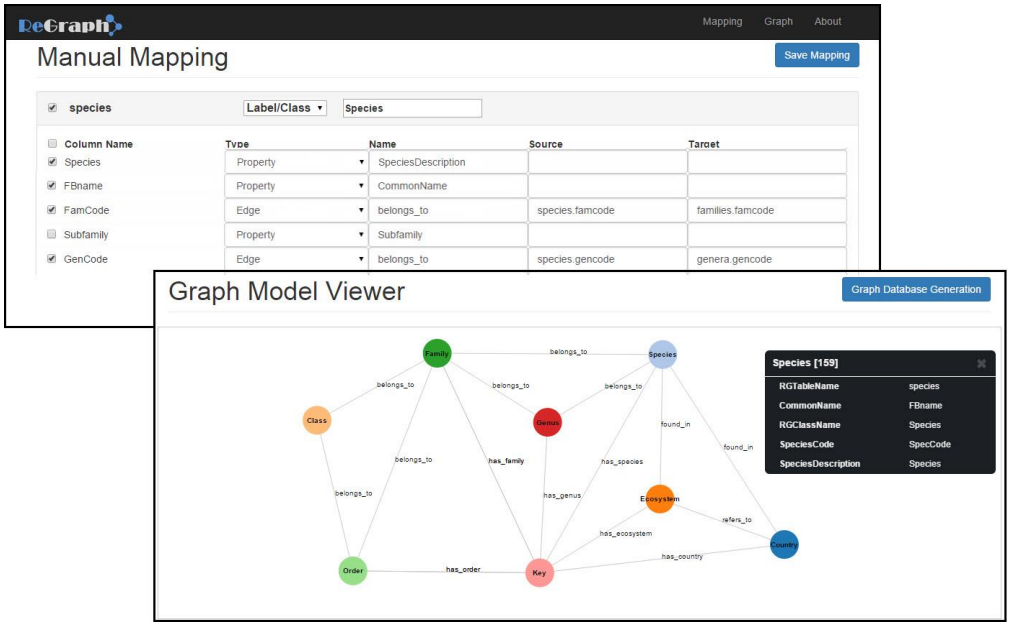

Figure 2 – ReGraph Interface: Manual Mapping and Graph Model Viewer

In the first step, the user defines the general configuration, including information about the relational and the graph databases (server, instance, user and password), the mapping type from the relational to the graph database (automatic or manual) and the time interval among synchronism cycles.

The automatic mapping option lists all tables and columns of the relational database, and users can filter their interests removing tables or columns from the mapping, unchecking the respective tables/columns (except the primary and foreign keys). It helps in migrating only the necessary data to the graph database, e.g., fields containing detailed descriptions and big notes could be irrelevant in a network analysis. As in the automatic mapping, the manual mapping also lists all tables and columns from the relational database. The user checks each table and column to be mapped, having the duty of defining the appropriated destination type, i.e., a label, node, property or edge.

After concluding the mapping, the "Save Mapping" button saves the mapping defined by the user and creates a graph database model that can be visualized by the user. It represents an abstract model of the graph database, presenting to the user a review of the mapping, as shown in Figure 2. At this point, it is possible to change the mapping in order to find the better model to fit an analysis. Whenever the user is satisfied with the mapping and model, it is possible to perform the first data load, carried by an ETL static process. After the mapping and the ETL processes, it is only possible to include tables and/or columns – it is not possible to remove tables and/or columns from the mapping. This rule aims to avoid inconsistencies in the graph database, considering the annotation subgraph. The ETL process also creates the triggers associated with the mapped data to perform the synchronism.

The synchronism process is transparent to the user. It is operated by a scheduled service running automatically on the server. This process is responsible for maintaining a dynamical evolution of the graph database, considering the changes executed in the relational database and following the rules explained in the previous section.

The Graph Module encompasses routines to compute new graph elements inferred from the existing ones or to enrich existing data with external resources, creating new labels, nodes, edges and properties. One example using the Graph Module is an interface with DBpedia (http://www.dbpedia.org/), in which the user can compare a specific subgraph of the database with the DBpedia RDF content. It produces three kinds of annotation: (a) *equal* – data that is the same in the graph database and in DBpedia; (b) *not found* – data that exists in the graph database but does not exists in DBpedia; and (c) *inconsistent* – data in the graph database that has any difference compared to DBpedia. These annotations point data to be reviewed, which are presented in a report to the user.

4. Conclusions

In this paper, we present ReGraph, a framework bridging relational and graph databases. It offers to the user operations of automatic or manual mapping from a relational to a graph database, dynamically updating data in the graph according to the changes performed in the relational database. Moreover, the ReGraph infrastructure offers mechanisms to interconnect data available in the graph database with third party resources, as data available in DBpedia, allowing the enrichment of existing data. We are working to connect the graph database with other data sources on the Web. To the best of our knowledge, ReGraph is the first framework that maps data from a relational database to a property graph database, keeping the databases in their native forms and providing a dynamic synchronized evolution of both.

Acknowledgment

Research partially funded by projects NAVSCALES (FAPESP 2011/52070-7), the Center for Computational Engineering and Sciences (FAPESP CEPID 2013/08293-7), CNPq-FAPESP INCT in eScience (FAPESP 2011/50761-2), INCT in Web Science (CNPq 557.128/2009-9) and individual grants from CNPq and CAPES.

References

Berners-Lee, T. (1998). Relational Databases on the Semantic Web, http://www.w3.org/DesignIssues/RDB-RDF.html, June, 2015.

Bizer, C. (2003). D2R Map - A database to rdf mapping language. In: *12th Int. World Wide Web Conference*, pages 4–6, 2003.

Blakeley, C. (2007). Virtuoso RDF Views Getting Started Guide. http://www.openlinksw.co.uk/virtuoso/Whitepapers/pdf/Virtuoso_SQL_to_RDF_Mapping.pdf, June 2015.

Cavoto, P., Cardoso, V., Vignes Lebbe and Santanchè, A. (2015). FishGraph: A Network-Driven Data Analysis. In: *11th IEEE Int. Conference on e-Science*, pages 1–10.

Gutiérrez, A., Pucheral, P., Steffen, H. and Thévenin, J. (1994). Database Graph Views: A Practical Model to Manage Persistent Graphs. In: *Proceedings of the 20th Int. Conference on Very Large Data Bases (VLDB)*, pages 1–20.