A Survey of Approaches for Supporting Data Interoperability between RDF and Property Graph Databases

Harsh Thakkar
University of Bonn, Germany
s6hthak@uni-bonn.de

Abstract. RDF and Graph databases are two approaches for data management that are based on modeling, storing, and querying graph-like data. In this report, we present a short study about state of the art in data interoperability between these approaches, i.e., data exchange problem. We review the current solutions to this question, identify their features, and discuss the inherent issues.

Keywords: RDF, Property Graph, Graph Data Model

1 Introduction

RDF [17] and Graph databases [25] are two approaches for data management that are based on modeling, storing, and querying graph-like data. Several database systems based on these models are gaining relevance in the industry due to their use in several domains where graphs and network analytics are required [4].

Both RDF and graph database systems are tightly connected as they are based on graph-oriented database models. RDF databases are based on the RDF data model [17], their standard query language is SPARQL [13], and RDF Schema [9] allows to describe classes of resources and properties (i.e., the data schema). On the other hand, most graph databases are based on the Property Graph (PG) data model, there is no standard query language (although there are several proposals [2]), and the notions of integrity constraints are reduced [24]. Therefore, RDF and PG databases (in particular the latter) are dissimilar in the data model, schema, query language, meaning, and content. Database interoperability is relevant for several reasons: promotes data exchange and data integration [23,21]; facilitates the reuse of available systems and tools [19,26]; enables a fair comparison of database systems by using benchmarks [3,30,28]; and supports the success of emergent systems and technologies [26].

The problem. Given the heterogeneity between RDF and graph databases, it is necessary to develop methods to allow interoperability among these systems. Several methods and approaches for database interoperability have been proposed [10] in recent years. Database interoperability can be divided into semantic interoperability (i.e., data exchange via schema and instance mappings)
and query interoperability (i.e., transformations among different query languages or data accessing methods). In the context of RDF and PG databases, semantic interoperability implies the definition of mappings to transform RDF graphs into PGs (and vice versa), and query interoperability presupposes the description of methods to convert SPARQL queries into PG query languages (and vice versa).

**Objectives & Contributions.** The main objective\(^1\) of this short paper is to analyze the current status of the approaches for semantic (data) interoperability between RDF, and PG approaches. This report aims to provide a basis and general direction for further research in this domain.

**Organization.** The remainder of this article is organized as follows: In Section (2) we introduce the notion of both the RDF and Property graph databases and their data models; Section (3) discusses the current state-of-the-art for supporting data interoperability between RDF and Property graph databases; we analyze their features, characterize the approaches, and isolate the main issues related to the problem in Section(4).

## 2 RDF Databases and Property Graph Databases

Here we formally introduce the main elements concerning RDF databases and PG databases. Specifically, we define the concepts of the RDF graph, RDF graph Schema, Property graph, and Property graph Schema as edge-labeled graphs. Additionally, we introduce the concept of a valid RDF and Property graph, respectively.

### 2.1 RDF Database

The Resource Description Framework (RDF) is a well-known W3C standard, which is used for data modeling and encoding machine readable content on the Web\(^2\) and within intranets. RDF defines a data model that can be viewed as a directed labeled multigraph.

Assume that \(I\) and \(L\) are two disjoint infinite sets, where \(I\) corresponds to IRIs (identifiers used to identify resources) and \(L\) to Literals (simple atomic values, e.g., strings, numbers or dates). An RDF term is an element in the set \(T = I \cup L.\)

**RDF Graph.** Informally, an RDF graph is a set of RDF triples. An RDF triple is a tuple \((v_1, v_2, v_3)\) where \(v_1\) is called the *subject*, \(v_2\) is the *predicate* and

---

\(^1\) This report has been compiled as a requirement of the partial fulfillment of the MA-INF 4313 - Seminar Semantic Data Web Technologies seminar of the University of Bonn.

\(^2\) In addition to IRIs and Literals, the RDF data model considers a domain of anonymous resources called Blank Nodes. Based on the work of Hogan et al. [16], we avoid the use of Blank Nodes as that their absence doesn’t affect the results presented in this paper.
v3 is the object. Here, the subject represents a resource identified by an IRI, the predicate represents either a property (when the object is a Literal) or a relationship (when the object is another resource), and the object represents the value of the property. Figure shows a graphical representation of an RDF graph.

**RDF Graph Schema.** RDF Schema (RDFS) [9] defines a standard vocabulary (i.e., a set of terms, each having a well-defined meaning), which enables the description of resource classes and property classes. From a database perspective, RDF Schema allows us to define the structure of the data in an RDF graph, i.e., a schema for RDF data.

In order to describe classes of resources and properties, the RDF Schema vocabulary defines the following terms: rdfs:Class, rdf:Property and rdfs:Literal represent the classes of resources, properties and literals respectively; rdf:type can be used (as property) to state that a resource is an instance of a class; rdfs:domain and rdfs:range allow to define the domain and range of a property, respectively; the term rdfs:label allows to describe the name of a class or property. It is important to note that rdf: and rdfs: are the prefixes for RDF and RDF Schema, respectively.

Figure 2 shows a graphical representation of an RDF schema description.

### 2.2 Property Graph Database

A Property Graph is a labeled directed multigraph whose main characteristic is that nodes and edges can contain a set (possibly empty) of name-value pairs referred to as properties. From the point of view of data modeling, each node represents an entity, each edge represents a relationship (between two entities), and each property represents a specific characteristic (of an entity or a relationship).

Figure 3 presents a graphical representation of a Property Graph. The circles represent nodes, the arrows represent edges, and the boxes contain the properties for nodes and edges. Next, we introduce a formal definition for this graph-based structure.
Fig. 2. The Schema of the RDF graph shown in Figure 1.

Fig. 3. A graphical representation of a Property Graph.
Property Graph. Assume that $L$ is an infinite set of labels (for nodes, edges and properties), $V$ is an infinite set of (atomic or complex) values, and $T$ is a finite set of data types (e.g. string, integer, date, etc.). Given a value $v \in V$, the function $\text{type}(v)$ returns the datatype of $v$. The values in $V$ will be distinguished as quoted literal. Given a set $X$, $\mathcal{P}(X)$ denotes the power set of $X$, i.e., the set of all the subsets of $X$, including the empty set $\emptyset$ and $X$ itself.

Definition 1 (Property Graph). A Property Graph is defined as a tuple: $G^P = (N, E, P, \delta, \lambda, \sigma, \rho)$ where:

- $N$ is a finite set of nodes, $E$ is a finite set of edges, $P$ is the finite set of properties, such that $N, E, P$ are mutually disjoint sets;
- $\delta : E \rightarrow (N \times N)$ is a total function that associates each edge in $E$ with a pair of nodes in $N$;
- $\lambda : (N \cup E) \rightarrow \mathcal{P}(L)$ is a total function that associates a node/edge with a set of labels from $L$ (i.e., $\lambda$ is a labeling function for nodes and edges);
- $\sigma : (N \cup E) \rightarrow \mathcal{P}(P)$ is a total function that associates nodes and edges to a set of properties, such that $\sigma(o_1) \cap \sigma(o_2) = \emptyset$ for each pair of objects $o_1, o_2 \in \text{dom}(\sigma)$, respectively;
- $\rho : P \rightarrow (L \times V)$ is a total function that assigns a label-value pair to each property.

Property Graph Schema. A PG schema defines the types of nodes, edges, and properties allowed in a given PG database. As with relational databases, the schema allows us to define and validate the structure of a property graph. For instance, Figure 4 shows a graphical representation of a property graph schema.

![Property Graph Schema Diagram]

Fig. 4. The Schema of the Property Graph shown in Fig. 3.
3 Data Interoperability between RDF and Property Graphs

In this section we present a review of the approaches to obtain data interoperability between RDF and Property graph databases, i.e. mappings, methods, or tools that allow to transform RDF graphs into property graphs (i.e. RDF-PG transformations) and vice versa (i.e. PG-RDF transformations). First, let us present a short summary of both data models.

The RDF data model [17] is based on labeled directed multigraphs allowing three types of nodes (web resources, anonymous resources called Blank Nodes, and simple values called Literals), two types of edges (data properties that relate resource nodes with literals, and object properties that relate two resource nodes). A tuple node-edge-node is called an RDF triple, and a collection of triples is called an RDF graph. RDF reification is a feature that means to create triples about triples, here metadata (e.g., time-stamps). The schema of an RDF database is a collection of resource classes, property classes, which can be described by using terms defined by the RDF Schema vocabulary [9]. The classes could be hierarchically organized by using subclass and subproperty relationships. RDF and RDF Schema provide semantic interpretations that allow us to infer additional RDF triples. This feature is called RDF(S) entailment.

Whereas, the Property Graph data model is based on labeled directed multigraphs where nodes and edges can contain a collection of key-value pairs called properties. The notion of a schema for a property graph database has not been developed, but some systems use the notions of node types and edge types. Next, we present some approaches that transform data and schema between these models. Despite there being a clear connection and motivation for addressing the interoperability between the RDF and Property graphs, there exists little literature addressing the problem, which are discussed below. These approaches have been compiled from the condensed surveys presented in the literature [5,6],

3.1 From RDF to Property Graphs

Hartig [14] proposes two transformations between RDF* and property graphs. RDF* is a syntactic extension of RDF based on reification. The first transformation maps any RDF triple as an edge in the resulting property graph. Each node has the “kind” attribute to describe the type of a node (e.g., IRI). This implies that all nodes in the graph can serve as the traversal source, thus thwarting the system performance at run time. The second transformation distinguishes data and object properties. The former are transformed into node properties and latter into edges of a Property graph. The limitation of the second transformations is that metadata triples cannot be transformed. The shortcoming of this approach is that RDF* isn’t supported by RDF stores and requires the conversion of existing RDF data beforehand.

Schätzle et al. [27] propose an RDF-PG transformation which is native to GraphX (a parallel processing system implemented on Apache Spark). The proposed graph model is an extension of the regular graph, but lacking the concept
of attributes. The mapping uses attribute label to store the node and edge identifiers, i.e. each triple \( t = (s, p, o) \) is represented using two vertices \( v_s, v_o \), an edge \((v_s, v_o)\) and properties \(v_s.label = s, v_o.label = o\), \((v_s, v_o).label = p\). The proposed method does not address blank nodes or RDF entailments.

Nyugen et al. [20], propose a graph model for RDF data, namely – LDM-3N (labeled directed multigraph-three nodes). It is an extension of the regular graph, without the concept of attributes, and represents each triple element as separate nodes, thus three nodes (3N). The LDM-3N graph model is used to address the Singleton Property (SP) based on reified RDF data. Due to this and the 3N modelling, we can infer that: (a) it requires at least \( 2 \times n \) extra edges and almost \( \frac{2}{3} \times n \) extra nodes as compared to the PG data model (modelling-wise); (b) the SP-based reification adds an extra computation step (and extra \( n \) triples); and (c) LDM-3N does not address blank nodes.

Hernandez et al. [15] present a transformation of RDF data (Wikidata triples) to a reified graph, in their empirical performance analysis of various databases. This was due to the limitation in Neo4J’s version used, which did not support queries over edges (labels, IDs, and attributes). Furthermore, [15] mention that the reification used is similar to the Standard Reification for RDF, thus implying the creation of at least additional \( 4 \times 2n \) nodes and \( 4 \times n \) edges (i.e. reifying \( n \) triples using SR implies additional \( 4 \times n \) triples, and each triple can be represented using two nodes and one edge respectively). Their approach does not address blank nodes, as the graph model is transformed directly from RDF JSON dumps, which were blank node free.

Tomaszuk [33], presents an approach that uses the YARS serialization for transforming RDF data into PGs. This approach basically performs a transformation between encoding schemes and does not consider the RDF schema, its entailments, or blank nodes.

Brandizi et al. [8] propose rdf2neo, a tool that can be used to map any RDF schema to the desired PG schema. This hybrid architecture facilitates access to knowledge networks based on shared data models. However, the disadvantage of this solution is that it maintains a more complex infrastructure that works well in the paper use case, but not for more general applications.

Another approach is presented in [12]. In this paper, the author presents a proposal for converting an RDF data store to a graph database by exploiting the ontology and the constraints of the source.

In [6], the authors have proposed a novel approach, which consists of three direct mappings, to transform RDF databases into PG databases. They empirically prove that the proposed mappings have an efficient implementation to process large datasets. Furthermore, they formally prove that two of the proposed mappings satisfy the property of information preservation, i.e., there exist inverse mappings that allow recovering the original databases without losing information.
### 3.2 From Property Graphs to RDF

There exist very few proposals for the PG-to-RDF transformation, such as Das et al. [11] and Hartig [14], that mainly use RDF reification methods (including Blank Nodes) to convert nodes and edge properties in a PG to RDF data. While [14] propose an in-direct mapping that requires converting to the RDF* model (as mentioned earlier), [11] lacks a formal foundation. Both approaches do not consider the presence of a PG schema.

Another approach is Unified Relational Storage (URS) [35]. It focuses on interchangeably managing RDF and PGs, and this is not a strict transformation method.

Barrasa [7] proposes NSMNTX, a plugin that enables the use of RDF in Neo4j. This plugin allows the import and export of both schema and data. The problem with this approach is that NSMNTX is not formally defined, and the mappings do not satisfy the property of information preservation.

In [34], the authors present an ontology-based approach to transform (automatically) property graphs into RDF graphs called PGO, which defines a set of terms that allows describing the elements of a property graph. The authors formally discuss the transformation methods along with some desired properties (such as complexity, data preservation, and monotonicity).

Table 1, presents a summary of the related work and the features they address. It should be mentioned that some works have studied the problem of mapping RDF to PGs in the scope of specific use cases, e.g., disease networks [18], protein structure exploration [1], and Wikidata reification [15].

<table>
<thead>
<tr>
<th>Literature</th>
<th>Target</th>
<th>DM</th>
<th>Formally Defined</th>
<th>IM</th>
<th>SM</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDF* [14]</td>
<td>RDF ↔ PG</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>?</td>
</tr>
<tr>
<td>URS [35]</td>
<td>RDF ↔ PG</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>NSMNTX [7]</td>
<td>RDF ↔ PG</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RDF2PG [6]</td>
<td>RDF ↔ PG</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>S2X [27]</td>
<td>RDF → PG</td>
<td>?</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>LDM-3N [20]</td>
<td>RDF → PG</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>rdf2neo [8]</td>
<td>RDF → PG</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SRDS [12]</td>
<td>RDF → PG</td>
<td>?</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Das et al. [11]</td>
<td>PG → RDF</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>YARS [33]</td>
<td>PG → RDF</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGO [34]</td>
<td>PG → RDF</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1. A consolidated summary of the approaches supporting data interoperability between RDF and Property graphs. The terms **DM, IM, SM** represent support for direct mapping, instance mapping and schema mapping in the respective work. The term **IP** represents whether or not an approach is information preserving in nature. The lack of information or evidence is denoted as “?”. 
4 Discussion and Conclusions

The methods to transform RDF into PGs can be classified according to two features: the use of the RDF schema in the transformation, and the support for Blank nodes. Reification is used to represent triple-level meta-data such as trust and provenance, information regarding the data source, time-stamps, etc. RDF reification can be performed using various techniques, of which Standard Reification (SR) [9], and N-ary Relations (NR) [22] are part of the W3C RDF standard.

Among the most critical issues for transforming RDF data into PGs are: (1) RDF schema information is usually merged with the data. It can imply a pre-processing of the data to extract the schema and then apply an RDF-PG transformation. (2) RDF schema information could be incomplete. So it could be necessary the “discover” the schema, and then transform the data. (3) RDF introduces particular features like blank nodes, reification (relations between relations), and entailment, PG databases cannot directly support that. (4) Reified RDF data introduce syntactic and semantic issues. A syntactic issue is the restriction to represent multi-level metadata in PGs. A semantic issue is a proper representation of a reified triple in a PG. (5) Reification leads to an explosion in the size of the resulting graph. This can be avoided by implementing a “smart” transformation that can recognize a set of triples describing a reification and map them to a single node in the property graph. It is evident that the state-of-the-art doesn’t cover all issues addressing the data interoperability between RDF and Property graphs. Hence, there is a substantial opportunity for further research. Furthermore, there is on-going work addressing the query interoperability between RDF and Property graphs such as as [29,31,32]; however, there is still scope for improvement.

References


