Converting Property Graphs to RDF: A Preliminary Study of the Practical Impact of Different Mappings

Shahrzad Khayatbashi
shahrzad.khayatbashi@liu.se
Linköping University
Linköping, Sweden

Sebastián Ferrada
sebastian.ferrada@liu.se
Linköping University
Linköping, Sweden

Olaf Hartig
olaf.hartig@liu.se
Linköping University
Linköping, Sweden

ABSTRACT

Today's space of graph database solutions is characterized by two main technology stacks that have evolved separate from one another: on one hand, there are systems that focus on supporting the RDF family of standards; on the other hand, there is the Property Graph category of systems. As a basis for bringing these stacks together and, in particular, to facilitate data exchange between the different types of systems, different direct mappings between the underlying graph data models have been introduced in the literature. While fundamental properties are well-documented for most of these mappings, the same cannot be said about the practical implications of choosing one mapping over another. Our research aims to contribute towards closing this gap. In this paper we report on a preliminary study for which we have selected two direct mappings from (Labeled) Property Graphs to RDF, where one of them uses features of the RDF-star extension to RDF. We compare these mappings in terms of the query performance achieved by two popular commercial RDF stores, GraphDB and Stardog, in which the converted data is imported. While we find that, for both of these systems, none of the mappings is a clear winner in terms of guaranteeing better query performance, we also identify types of queries that are problematic for the systems when using one mapping but not the other.

CCS CONCEPTS

• **Information systems → Graph-based database models**; *Data model extensions*; *Data exchange*; **Database performance evaluation**.

KEYWORDS

Graph Databases, Property Graph, RDF, RDF-star

ACM Reference Format:
Shahrzad Khayatbashi, Sebastián Ferrada, and Olaf Hartig. 2022. Converting Property Graphs to RDF: A Preliminary Study of the Practical Impact of Different Mappings. In *Joint Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA) (GRADES & NDA'22), June 12, 2022, Philadelphia, PA, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3534540.3534695

GRADES & NDA'22, June 12, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9384-3/22/06.
https://doi.org/10.1145/3534540.3534695

1 INTRODUCTION

While the usage of graph database systems is becoming more widespread and popular [9, 16, 24], users typically have to choose between two classes of such systems: On the one hand, there are systems such as Neo4j, TigerGraph, JanusGraph, RedisGraph, and SAP HANA that support some form of so-called Property Graphs [20]. On the other hand, there are systems such as OpenLink's Virtuoso, Ontotext GraphDB, Stardog, and AllegroGraph that support the Resource Description Framework (RDF) [21], which is a family of standards developed by the World Wide Web Consortium (W3C).

A decision for a system of one of these two classes implies adopting the respective type of graph databases supported by this system (i.e., Property Graphs or RDF graphs). Yet, users may still want to incorporate graph data of the respective other type into their applications without having to maintain two systems, or they may want to leverage another system of the respective other class for specific processing tasks. Examples for the latter may be to query an RDF dataset using an expressive graph traversal language or graph analytics algorithms available in a Property Graph system, or to apply semantics-based reasoning to a Property Graph by using an RDF system with the relevant reasoning capabilities, which are typically not available in Property Graph systems.

Achieving interoperability and such a form of data exchange between the two classes of systems has been a topic of research in recent years [2]. One of the challenges in this context are differences in the underlying data models [2, 17]. Property Graphs can be characterized as directed multigraphs in which both the vertices and the edges can be annotated with so-called properties in the form of key-value pairs. Additionally, in many systems, the vertices and the edges may have a label, in which case we speak of Labeled Property Graphs (LPGs). Figure 1 illustrates such an LPG with two nodes and two edges. An RDF graph, on the other hand, is a set of triples of the form *(subject, predicate, object)* which can be considered as edges, where the subject and the object are the vertices and the predicate captures the type of such an edge (similar to an edge label in an LPG); see Figure 2 for such a graph representation of an RDF graph with five triples. Due to the set-based definition, RDF graphs are not true multigraphs; that is, they cannot contain multiple edges of the same type between the same vertices (which is possible in LPGs). Another feature of LPGs for which there is no direct counterpart in RDF graphs are edge properties. However, RDF also has features that are not available natively in LPGs. For instance, IRIs as globally unique identifiers [8] are a fundamental building block of RDF triples, which is particularly relevant for data exchange and data integration; in contrast, the building blocks of LPGs (vertices, edges, labels, etc.) have an identity only within the

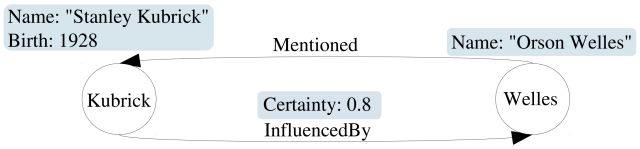

Figure 1: Illustration of an LPG (adapted from [11]).

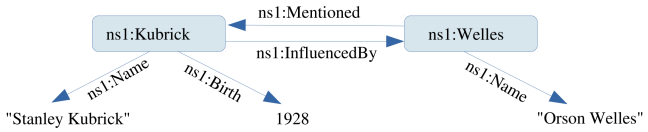

Figure 2: Illustration of an RDF graph with five triples.

scope of the LPG that they belong to. Also, RDF has standardized serialization formats whereas formats for LPGs are vendor specific.

A recent extension of RDF that aims to bridge the gap between the two data models is called *RDF-star* [12, 14], which is already supported by several RDF systems[1], including GraphDB[2], Stardog[3], Apache Jena[4] and Oxigraph[5], and is currently proposed for standardization by the W3C. The feature that RDF-star adds to RDF is that it permits the subject and the object of a triple to be another triple [12, 14]. This idea of embedding a triple into another one resembles the notion of edge properties in LPGs and may be leveraged for converting LPGs to be stored or processed by RDF systems.

Indeed, as a foundation of such conversions, several authors have introduced direct mappings from LPGs to RDF-star [5, 11, 13]. Other authors have proposed direct mappings from LPGs to pure RDF graphs [7, 18, 23] (i.e., without relying on the features of RDF-star). Similarly, there exist direct mappings for the reverse direction, from RDF to LPGs [3, 11, 22].

While defining such mappings is important as a foundation for achieving interoperability of the different classes of graph database systems, an understanding of the practical implications of these mappings is equally important. To the best of our knowledge, such an understanding is still largely missing in the literature. To begin closing this gap, in this paper we report on a preliminary study that compares two such mappings (one of them in two variations) from LPGs to RDF for which we answer the following research question:

How does the choice of mapping affect the performance that RDF systems achieve for queries over the resulting data?

The mappings that we have selected for this study are the direct LPG-to-RDF-star mapping defined by Hartig [13] and a direct LPG-to-RDF mapping described by Nguyen et al. [18]. The latter transforms edges of the given LPG into vertices that, in the resulting RDF graphs, are captured as blank nodes; the properties of LPG edges are then captured as RDF triples with these blank nodes as subject. As an example, consider the LPG in Figure 1, which would be mapped to the RDF graph in Figure 3. In contrast, Hartig's mapping leverages RDF-star, which makes it possible to capture the edges of an LPG as triples and the edge properties become nested (RDF-star) triples with the triple for the corresponding edge as their subject. In this case, the LPG in Figure 1 would be mapped to an RDF-star graph with triples as illustrated in Figure 2, plus a nested triple of the form (t, ns1:Certainty, 0.8) where t is the ns1:InfluencedBy triple in Figure 2.

[1]https://w3c.github.io/rdf-star/reports/index.html
[2]https://www.ontotext.com/fundamentals/what-is-rdf-star/
[3]https://www.stardog.com/blog/property-graphs-meet-stardog/
[4]https://jena.apache.org/documentation/rdf-star/
[5]https://github.com/oxigraph/oxigraph/releases/tag/v0.3.0

To compare the impact that these mappings have on query performance achieved by systems in which the resulting data is imported, we have designed an experiment based on the Panama Papers dataset, which is a real-world LPG. For this preliminary study, we have selected two commercial RDF systems that support RDF-star; namely, GraphDB and Stardog. Additionally, as a baseline, we also include Neo4j as a native LPG system in our experiments.

The main findings of our study are the following: (1) there is no clear best mapping in terms of query execution times; in some cases, queries over the data resulting from the RDF-star-based mappings perform better, and in others worse, compared with their equivalent counterparts for the pure RDF representations; (2) Stardog produces sub-optimal query plans for the RDF-star case if queries contain nested triple patterns; and (3) for the RDF-based approach, using separate IRIs for expressing node and edge labels reduces query execution times, mainly because the queries become simpler.

Before we describe the experiment setup and the results in detail (Sections 5–6), we define the mappings (Sections 2–3) and discuss how they differ in terms of the way queries over the resulting data need to be formulated (Section 4). All material related to work in this paper can be found in a corresponding github repository.[6]

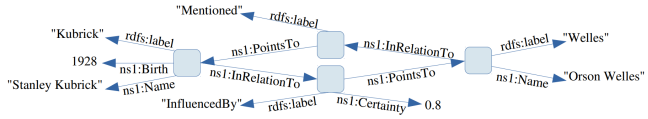

Figure 3: RDF graph representation of the LPG of Figure 1 when using the RDF-based mapping approach 1.

2 PRELIMINARIES

As a basis for providing a more formal definition of the mappings considered in this paper, we first need to define the relevant notions of LPGs, RDF, and RDF-star.

Labeled Property Graphs (LPGs) have originally been defined only informally [20] and their implementation in different systems varies slightly in terms of features (e.g., some support multi-valued properties or multiple labels, others not). As a consequence, formal definitions that have appeared in the literature also differ slightly. In this paper we adopt the following definition of Angles et al. [1], which has also been used by Hartig for the LPG-to-RDF-star mapping [12] that we consider in this paper.

The definition assumes three infinite countable sets: *Labels* (labels), *Props* (property names), and *Values* (property values).

Definition 2.1. An *LPG* is a tuple $(V, E, \rho, \lambda, \sigma)$ where:

[6]https://github.com/LiUSemWeb/GRADES2022-paper

- V is a finite set of *vertices* (or *nodes*);
- E is a finite set of *edges* such that $V \cap E = \emptyset$;
- $\rho : E \to (V \times V)$ is a total function;
- $\lambda : (V \cup E) \to \mathit{Labels}$ is a total function;
- $\sigma : (V \cup E) \times \mathit{Props} \nrightarrow \mathit{Values}$ is a partial function.

Example 2.2. The LPG illustrated in Figure 1 consists of the following elements:

$$V = \{n_1, n_2\},\ \lambda(n_1) = \texttt{"Kubrick"},\quad \lambda(n_2) = \texttt{"Welles"},$$
$$E = \{e_1, e_2\},\ \lambda(e_1) = \texttt{"Mentioned"},\ \lambda(e_2) = \texttt{"InfluencedBy"},$$
$$\rho(e_1) = (n_2, n_1),\qquad \rho(e_2) = (n_1, n_2),$$
$$\sigma(n_1, \texttt{"Name"}) = \texttt{"Stanley Kubrick"},\ \sigma(n_1, \texttt{"Birth"}) = \texttt{1928},$$
$$\sigma(n_2, \texttt{"Name"}) = \texttt{"Orson Welles"},\ \sigma(e_2, \texttt{"Certainty"}) = \texttt{0.8}.$$

RDF is a family of W3C standards with a well-defined data model at its core. The main concepts of this data model are RDF triples and RDF graphs [6]. To define these concepts formally we assume three countably infinite and pairwise disjoint sets: $\mathcal{I}$ (IRIs), $\mathcal{B}$ (blank nodes), and $\mathcal{L}$ (literals).

Definition 2.3. An *RDF triple* is a 3-tuple (s, p, o) with $s \in (\mathcal{I} \cup \mathcal{B})$, $p \in \mathcal{I}$, and $o \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$. An *RDF graph* is a finite set of such triples.

All RDF-related examples in this paper, such as Figure 2, represent IRIs in a typical compact form [4]; we omit the corresponding prefix declarations as they are not relevant here.

RDF-star extends RDF with the possibility to use triples as the subject and the object of other triples [14]. The following definition captures this idea formally.

Definition 2.4. An *RDF-star triple* is a 3-tuple that is defined recursively as follows:

- every RDF triple is an RDF-star triple;
- if t and t' are RDF-star triples, $s \in (\mathcal{I} \cup \mathcal{B})$, $p \in \mathcal{I}$, and $o \in (\mathcal{I} \cup \mathcal{B} \cup \mathcal{L})$, then (t, p, o), (s, p, t), and (t, p, t') are RDF-star triples, for which we say they are *nested* and, in the context of such a nested triple, t and t' are *quoted triples* [14].

An *RDF-star graph* is a finite set of RDF-star triples.

Notice that, by definition, every RDF triple is an RDF-star triple, and every RDF graph is an RDF-star graph, but not vice versa. Notice also that the definition allows for an arbitrarily deep nesting of triples. However, for the LPG-to-RDF-star mapping that we consider in this paper, only one level of nesting is needed.

Example 2.5. Let t = (ns1:Kubrick, ns1:InfluencedBy, ns1:Welles) be an ordinary RDF triple—which, of course, is also an RDF-star triple—the triple (t, ns1:Certainty, 0.8) is a nested RDF-star triple with t as a quoted triple in its subject position.

3 DEFINITION OF THE MAPPINGS

We are now ready to define the three mappings considered in our study. The first two map LPGs to RDF graphs in two slightly different ways, and the third maps to RDF-star graphs.

Algorithm 1 RDF-Based Approach 1

1: **INPUT:** an LPG $g = (V, E, \rho, \lambda, \sigma)$
2: **RESULT:** an RDF graph G
3: $G \leftarrow \emptyset$
4: **for** $n \in V$, with $\lambda(n) = \ell$ **do**
5: add $\big(\mathit{idm}(n), u_{\mathsf{label}}, \mathit{lm}(\ell)\big)$ to G
6: **for** $p \in \mathit{Props}$ such that $(n, p) \in \mathrm{dom}(\sigma)$ **do**
7: add $\big(\mathit{idm}(n), \mathit{pnm}(p), \mathit{pvm}(v)\big)$ to G, with $\sigma(n, p) = v$
8: **end for**
9: **end for**
10: **for** $e \in E$, with $\rho(e) = (n, n')$ and $\lambda(e) = \ell$ **do**
11: add $\big(\mathit{idm}(n), u_{\mathsf{in}}, \mathit{idm}(e)\big)$ to G
12: add $\big(\mathit{idm}(e), u_{\mathsf{out}}, \mathit{idm}(n')\big)$ to G
13: add $\big(\mathit{idm}(e), u_{\mathsf{label}}, \mathit{lm}(\ell)\big)$ to G
14: **for** $p \in \mathit{Props}$ such that $(e, p) \in \mathrm{dom}(\sigma)$ **do**
15: add $\big(\mathit{idm}(e), \mathit{pnm}(p), \mathit{pvm}(v)\big)$ to G, with $\sigma(n, p) = v$
16: **end for**
17: **end for**
18: **return** G

3.1 RDF-Based Approach 1

The first mapping is adapted from Nguyen et al. who introduce an approach to convert arbitrary LPGs into a specific class of LPGs that, then, can be mapped directly into RDF [18]. The idea of this approach is to transform every edge into a vertex which gets connected to the two vertices incident to the transformed edge by adding two new edges, labeled "in" and "out". Notice that these new edges are the only types of edges after this transformation step, and there are no more edge properties (they have become vertex properties). In the RDF representation, all vertices—the original ones and the ones created from the original edges—are mapped to distinct blank nodes, and the new "in" and "out" edges become triples. Each vertex property and vertex label is also captured as a triple.

While Nguyen et al. describe this mapping only informally, we now define the mapping formally in terms of a conversion algorithm (cf. Algorithm 1). That is, for every LPG $g = (V, E, \rho, \lambda, \sigma)$, this algorithm returns the RDF graph that the LPG is mapped to according to the mapping. For this definition we assume three special IRIs, denoted in Algorithm 1 by u_{in}, u_{out} and u_{label}, and we assume the following four abstract mapping functions that define how the individual LPG elements are mapped into the relevant RDF terms (IRIs, blank nodes, and literals):

- $\mathit{pnm} : \mathit{Props} \to \mathcal{I}$ maps property names to IRIs;
- $\mathit{pvm} : \mathit{Values} \to \mathcal{L}$ maps property values to literals;
- $\mathit{lm} : \mathit{Labels} \to \mathcal{L}$ maps labels to literals;
- $\mathit{idm} : V \cup E \to \mathcal{I} \cup \mathcal{B}$ maps every vertex and edge of g to an IRI or a blank node, respectively.

How these mapping functions are actually implemented is irrelevant for the mapping and may differ for different application scenarios. However, for the mapping to be information preserving (i.e., lossless), these functions must be injective and the three special IRIs (u_{in}, u_{out} and u_{label}) must not be in the image of pnm (i.e., no property name should be mapped to them).

Given such four mapping functions and the three IRIs, in a first phase (lines 4–9 in Algorithm 1), the algorithm iterates over the

vertices to create triples that capture the vertex labels, where the special IRI u_{label} is used as the predicate of these triples (line 5). Additionally, for every property of a vertex, a corresponding triple is created (line 7). Thereafter, in a second phase (lines 10–18), the edges are considered. Each edge is transformed into a vertex, represented by a blank node or IRI, with two triples that represent the aforementioned new "in" and "out" edges, for which the IRIs u_{in} and u_{out} are used as predicates, respectively. Furthermore, the edge labels are captured as triples, using again the IRI u_{label} (line 13), and so are the edge properties (line 15).

Example 3.1. The result of applying Algorithm 1 to the LPG of Example 2.2 (cf. Figure 1) is the set of RDF triples illustrated in Figure 3 and serialized in RDF Turtle syntax [19] as follows:

```
_:x1    rdfs:label       "Kubrick".
_:x1    ns1:Name         "Stanley Kubrick".
_:x1    ns1:Birth        1928.
_:x2    rdfs:label       "Welles".
_:x2    ns1:Name         "Orson Welles".
_:x1    ns1:InRelationTo _:x3.
_:x3    ns1:PointsTo     _:x2.
_:x3    rdfs:label       "InfluencedBy".
_:x3    ns1:Certainty    0.8.
_:x2    ns1:InRelationTo _:x4.
_:x4    ns1:PointsTo     _:x1.
_:x4    rdfs:label       "Mentioned".
```

In this serialization, _:x1, _:x2, _:x3 and _:x4 represent the blank nodes assigned to the vertices n_1 and n_2 and to the edges e_1 and e_2 of the LPG, respectively. For the abstract special IRIs u_{in}, u_{out} and u_{label}, the example uses the concrete IRIs ns1:InRelationTo, ns1:PointsTo and rdfs:label, respectively.

3.2 RDF-Based Approach 2

The second LPG-to-RDF mapping is a variation of the first one, with the following difference. Instead of using the same IRI u_{label} as predicate of the triples that capture vertex labels and the triples that capture edge labels, two separate IRIs are used: u_{nlabel} for vertexes and u_{elabel} for edges. Hence, the algorithm that defines this mapping is as Algorithm 1 with lines 5 and 13 changed as follows.

5: add $\big(idm(n), u_{\text{nlabel}}, lm(\ell)\big)$ to G
13: add $\big(idm(e), u_{\text{elabel}}, lm(\ell)\big)$ to G

Example 3.2. The set of RDF triples resulting from applying the second LPG-to-RDF mapping to the LPG from Example 2.2 is basically the same as in Example 3.1, except for the triples about labels. For instance, (_:x1, rdfs:label, "Kubrick") becomes (_:x1, node:label, "Kubrick"), and (_:x3, rdfs:label, "InfluencedBy") becomes (_:x3, edge:label, "InfluencedBy"). The complete result in RDF Turtle syntax:

```
_:x1    node:label       "Kubrick".
_:x1    ns1:Name         "Stanley Kubrick".
_:x1    ns1:Birth        1928.
_:x2    node:label       "Welles".
_:x2    ns1:Name         "Orson Welles".
_:x1    ns1:InRelationTo _:x3.
_:x3    ns1:PointsTo     _:x2.
_:x3    edge:label       "InfluencedBy".
_:x3    ns1:Certainty    0.8.
_:x2    ns1:InRelationTo _:x4.
_:x4    ns1:PointsTo     _:x1.
_:x4    edge:label       "Mentioned".
```

Algorithm 2 RDF-star-Based Approach

1: **INPUT:** an LPG $g = (V, E, \rho, \lambda, \sigma)$
2: **RESULT:** an RDF-star graph G
3: $G \leftarrow \emptyset$
4: **for** $n \in V$, with $\lambda(n) = \ell$ **do**
5: add $\big(idm(n), u_{\text{label}}, lm(\ell)\big)$ to G
6: **for** $p \in Props$ such that $(n, p) \in \text{dom}(\sigma)$ **do**
7: add $\big(idm(n), pnm(p), pvm(v)\big)$ to G, with $\sigma(n, p) = v$
8: **end for**
9: **end for**
10: **for** $e \in E$, with $\rho(e) = (n, n')$ and $\lambda(e) = \ell$ **do**
11: $t \leftarrow \big(idm(n), elm(\ell), idm(n')\big)$
12: add t to G
13: **for** $p \in Props$ such that $(e, p) \in \text{dom}(\sigma)$ **do**
14: add $\big(t, pnm(p), pvm(v)\big)$ to G, with $\sigma(n, p) = v$
15: **end for**
16: **end for**

3.3 RDF-star-Based Approach

As defined by Hartig, this mapping transforms LPGs into RDF-star graphs [13]. For vertices, this approach works exactly as the RDF-based approach 1. Each LPG edge, however, is captured as a triple; the predicate of this triple is an IRI that depends on the label of the edge. The edge properties are then captured as nested triples in which the subject is the triple that captures the edge.

For the formal definition of this mapping, we need an additional abstract mapping function (in addition to *pnm*, *pvm*, *lm* and *idm*):

- $elm : Labels \rightarrow \mathcal{I}$ that maps (edge) labels to IRIs

Then, Algorithm 2 defines the RDF-star-based mapping. The algorithm has two phases: The first phase (lines 4–9) covers the vertices and is equivalent to the first phase of Algorithm 1. The second phase starts by mapping each edge to its corresponding RDF triple (lines 11 and 12); then, for each property defined for the edge, the algorithm adds a nested triple containing the triple representing the edge as a quoted triple in the subject, and the property and value mappings in the predicate and object positions (line 14).

It should be noted that RDF-star cannot capture multiple edges with the same label that connect the same pair of vertices. In this case, the mapping as defined by Algorithm 2 collapses all such multiple edges into just one.

Example 3.3. The result of applying Algorithm 2 to the LPG of Example 2.2 is the following set of RDF-star triples, serialized in Turtle-star format [14], which is an RDF-star extension of Turtle.

```
_:x1    rdfs:label              "Kubrick".
_:x1    ns1:Name                "Stanley Kubrick".
_:x1    ns1:Birth               1928.
_:x2    rdfs:label              "Welles".
_:x2    ns1:Name                "Orson Welles".
_:x2    ns1:Mentioned           _:x1.
_:x1    ns1:InfluencedBy        _:x2.
<<_:x1 ns1:InfluencedBy _:x2>> ns1:Certainty  0.8.
```

The last line in this serialization represents a nested triple that captures the certainty property of edge e_2 of the LPG. Notice that this example also illustrates that, in terms of translating the vertices of an LPG, the RDF-star-based mapping and the RDF-based approach 1

are equivalent (the first five triples in this example are exactly the same as in Example 3.1). For edges, on the other hand, the mappings are different; by using nested triples, the RDF-star-based mapping does not require the extra blank nodes (_:x3 and _:x4 in Examples 3.1 and 3.2) and the ns1:InRelationTo and ns1:PointsTo triples with which the RDF-based mappings capture the edges of the translated LPG.

4 QUERYING THE CONVERTED DATA

While the RDF and RDF-star graphs resulting from the mappings can be queried using the RDF query language SPARQL [10] and its RDF-star-related extension SPARQL-star [14], queries may need to be formulated differently, depending on which of the three mappings is used. In this section, we identify and discuss key differences that need to be considered in this context. To this end, we consider several Cypher queries for LPGs and show how to reformulate them as semantically-equivalent SPARQL and SPARQL-star queries for the RDF and RDF-star graphs obtained from mapping the LPG.

To mimic a Cypher query that matches all vertices and edges in an LPG (i.e., `MATCH (n1)-[e]->(n2) RETURN *`), the following SPARQL query may be used for the RDF graphs produced by either of the two RDF-based approaches:

```
SELECT * WHERE{ ?n1 u_in ?e . ?e u_out ?n2 . }
```

For the RDF-star-based approach, in contrast, we only need a single triple pattern but we have to add a FILTER to avoid obtaining property names and values in addition to the vertices and edges:

```
SELECT * WHERE{ ?n1 ?e ?n2 . FILTER (!IsLiteral(?n2))}
```

The difference between the two RDF-based approaches becomes relevant when querying for vertex labels: To mimic the Cypher query `MATCH (n) RETURN labels(n) AS l`, for the RDF-based approach 1, we use the following SPARQL query:

```
SELECT ?l WHERE{ ?n u_label ?l .
                 FILTER NOT EXISTS { ?n u_in ?e . } }
```

Here, we need to use `FILTER NOT EXISTS` to obtain only vertex labels and no edge labels. Notice that the filter could contain the triple pattern (?e, u_{out}, ?n) and achieve the same effect.

For the RDF-based approach 2, we can, instead, use:

```
SELECT ?l WHERE{ ?n u_nlabel ?l . }
```

Similarly, for the RDF-star-based approach, we can use the latter query after replacing u_{nlabel} by u_{label}.

Cypher produces null values when a property is not defined on a given vertex or edge. This behavior can be replicated in SPARQL by using OPTIONAL patterns. As an example, consider the Cypher query `MATCH (n) RETURN n.height AS h`, which can be reformulated in SPARQL as follows, for the RDF-based approaches:

```
SELECT ?h WHERE{ OPTIONAL{?n pnm(height) ?h.
                          FILTER NOT EXISTS {?n u_in ?e}}}
```

In this case, FILTER NOT EXISTS is used to filter out edges that have a height property. For the RDF-star-based approach, the query has to be formulated as follows:

```
SELECT ?h WHERE{ OPTIONAL{ ?n pnm(height) ?h.
                           FILTER ! IsTriple(?n) }  }
```

The built-in function IsTriple in this query is a SPARQL-star-specific feature that we use here in the FILTER to keep only the bindings for variable ?n that are not quoted triples (which the RDF-star-based mapping approach creates for edges).

Finally, we consider queries that retrieve edge properties such as the Cypher query `MATCH ()-[e]->() RETURN e.weight AS w .` For both RDF-based approaches, the SPARQL equivalent is:

```
SELECT ?w WHERE{ OPTIONAL{?e pnm(weight) ?w.
                          FILTER EXISTS{?e u_out ?n}} }
```

Here, the purpose of the FILTER EXISTS pattern is to keep only the edges. For the RDF-star-based approach we can use the following SPARQL-star query:

```
SELECT ?w WHERE{OPTIONAL{ <<?n1 ?e ?n2>> pnm(weight) ?w}}
```

To conclude, the examples in this section illustrate that the queries for the data resulting from the three mappings need to be different; for some cases, even different SPARQL operators need to be used in these queries. These differences may have an effect on the performance that RDF systems achieve for such queries

5 EXPERIMENT DESIGN AND SETUP

The goal of our experiment is to compare the three mappings in terms of the query performance that RDF systems achieve for queries over the particular form of RDF/RDF-star graphs produced by the mappings. To this end, we have selected a real-world LPG, used the three mappings to convert this LPG into two RDF graphs and one RDF-star graph, and then loaded these three graphs into both GraphDB (v.9.10.0) and Stardog (v.7.9.0). These two systems are commercial RDF stores that already support RDF-star and SPARQL-star. Additionally, we have also loaded the original LPG into Neo4j (community edition 4.3.7). To query the data we have created a collection of twelve benchmark queries (see below), where each query is defined in four semantically equivalent versions: a Cypher version for the LPG, two SPARQL versions for the two RDF representations, and a SPARQL-star version for the RDF-star-based approach. For each graph representation in each system, we have executed each of the corresponding twelve queries ten times and, for each such execution, we have measured the execution time. We shall report on the averages of these ten measurements.

In terms of technical setup, we have performed the experiment on a server machine with two 8-core Intel Xeon E5-2667 v3@3.20GHz CPUs, 256 GB of RAM, and a 1 TB HDD. The machine runs a 64-bit Debian GNU/Linux 10 server operation system. On this machine, we use Docker (v.9.03.6) to run all systems of the experiment setup in a separate, virtual environment (i.e., Neo4j, GraphDB, and Stardog).

The remainder of this section focuses on the dataset and the queries, and Section 6 presents the experiment results.

5.1 Dataset

For the experiment we use a real-world LPG with data about the Panama Papers.[7] What makes this dataset interesting for our experiment is that it was created natively as an LPG and it uses all features of LPGs; namely, it has multiple types of vertices and edges, with multiple types of vertex and edge properties, as well as vertex and edge labels. Figure 4 depicts the schema of this LPG. There are four types of vertices: officer, entity, address, and intermediary, and three types of edges: officer_of (to connect each officer to either an entity, intermediary, or another officer), intermediary_of (to connect an intermediary to an entity), and registered_address (to connect each

[7] https://offshoreleaks.icij.org/pages/database

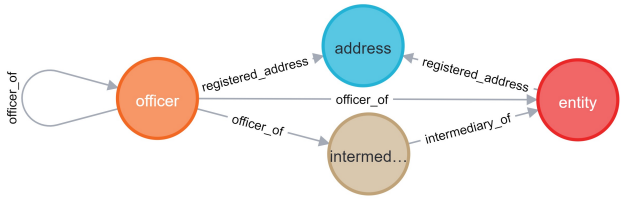

Figure 4: Schema of the Panama Papers LPG.

officer and entity to an address). Additionally, vertices and edges have properties. While, for vertices, the types of properties vary depending on the type of vertex, the edges all have properties of the same types, namely: start_date, end_date, sourceID, link, and valid_until.

In terms of size, the original LPG version of this dataset consists of 559,600 vertices and 674,102 edges. When converting it using any of the two RDF-based approaches, we obtain an RDF graph consisting of 9,678,501 triples. In contrast, applying the RDF-star-based mapping results in an RDF-star graph with 8,261,110 triples. We have made available these resulting RDF and RDF-star representations of the dataset for other researchers and future experiments.[8]

5.2 Benchmark Queries

This section introduces our twelve benchmark queries; we describe what they are intended for and what features of LPGs they access. Given that the RDF-star-based approach collapses edges with the same label between a given pair of vertices, we chose queries that do not traverse such multi-edges in order to have the same number of results when evaluating the queries in the different graph representations. The actual Cypher, SPARQL, and SPARQL-star representations of these queries can be found in the aforementioned github repository for this paper.

Q1 obtains IDs of vertices with value "XU JIE" on property name.

Q2 obtains information of vertices that have name "EL PORTADOR". The main aim of this query is to measure the influence of the FILTER NOT EXISTS pattern on the execution time on the RDF-based approach 1 when the number of results is large.

Q3 retrieves the distinct vertex labels of the dataset. The main purpose of this query is to observe how the execution time may differ when it comes to scanning the whole graph.

Q4 obtains all the distinct edge labels.

Q5 is the first to use path patterns. In this case, the pattern searches for vertices that target a vertex with name "ORION HOUSE SERVICES (HK) LIMITED", through an intermediary_of edge.

Q6 obtains the vertex that have a relationship with a set of vertices that have name "BEARER" or "The Bearer". The main goal of the query is to see how the approaches perform in terms of the execution time when querying edge properties.

Q7 obtains vertex and edge properties, such that the source vertex has name "THE BEARER", the edge has property end-date with value "11-APR-2014" and the target vertex has property status with value "active", or the edge has property end-date with value "29-DEC-2009" and the target vertex has property status with value "defaulted". The aim of this query is to measure the impact of filtering by edge property values.

Q8 obtains labels and properties of vertices and edges that participate in registered_address relationships.

Q9 obtains vertex and edge properties, such that a given vertex has two registered_address relationships with two different vertices.

Q10 retrieves edge properties of edges of type registered_address that connect an officer vertex and an entity vertex to a same target vertex. The main aim of the query is to observe how the increased number of triple patterns in a query would affect its performance.

Q11 obtains edge properties of edges that connect an intermediary vertex to an entity vertex that is connected to the same address vertex as an officer vertex. In contrast to Q9 and Q10, the edge labels in this query are not specified. Therefore, the primary goal of this query is to see how the identification of the edge affects its execution time.

Q12 obtains various properties of vertices and edges that participate in a given path, where each vertex has a given label and each edge has a given type. The aim of this query is to produce rather large BGPs and measure their execution time.

6 EXPERIMENT RESULTS

Figure 5 presents the average execution time of the benchmark queries over the RDF/RDF-star graphs produced by the three described approaches on GraphDB and Stardog, compared with their equivalent version in Neo4j. The query execution times on Neo4j, GraphDB, and Stardog are depicted in dark gray, white, and light gray bars, respectively. Moreover, the bars with the dotted pattern represent the queries for the RDF-based approach 1, the slashed pattern is used for the RDF-based approach 2, and the circle pattern for the RDF-star-based approach. The vertical axis has a logarithmic scale. In general, it can be seen that GraphDB outperforms both Stardog and Neo4j for the majority of the queries. Neo4j performs better than GraphDB and Stardog in some specific cases. Stardog performs best on queries retrieving vertex and edge labels.

In Q1, Q2 and Q3, which aim to obtain vertex properties and vertex labels, the queries over the data from the first RDF-based approach take longer than the queries for RDF-based approach 2, both in GraphDB and Stardog. This behavior is expected since these three queries for the RDF-based approach 1 require a FILTER NOT EXISTS operation, which is very costly. However, this operation does not have a significant effect on the execution time of Q1 because of the low amount of matches for the triple pattern (?n, ns1:name, 'XU JIE') (6 results) versus the 9,325 results of (?n, ns1:name, 'EL PORTADOR') in Q2, and all nodes in Q3. For Q2, Q3 and Q4, we see that Stardog performs significantly better than GraphDB in the RDF-based approach 2 and the RDF-star approach. It can be seen in the query plans that Stardog has a better cardinality estimation.[9] Also, Stardog scans a POC index, thus not producing bindings for the variable in the subject of the triple pattern.

Q4, that aims to get all distinct edge labels, follows a similar pattern, being the RDF-based approach 2 the one that performs best, which is expected since the query uses just one triple pattern,

[8] https://doi.org/10.5281/zenodo.6524085

[9] The query plans for all the queries can be found in the github repository of this paper.

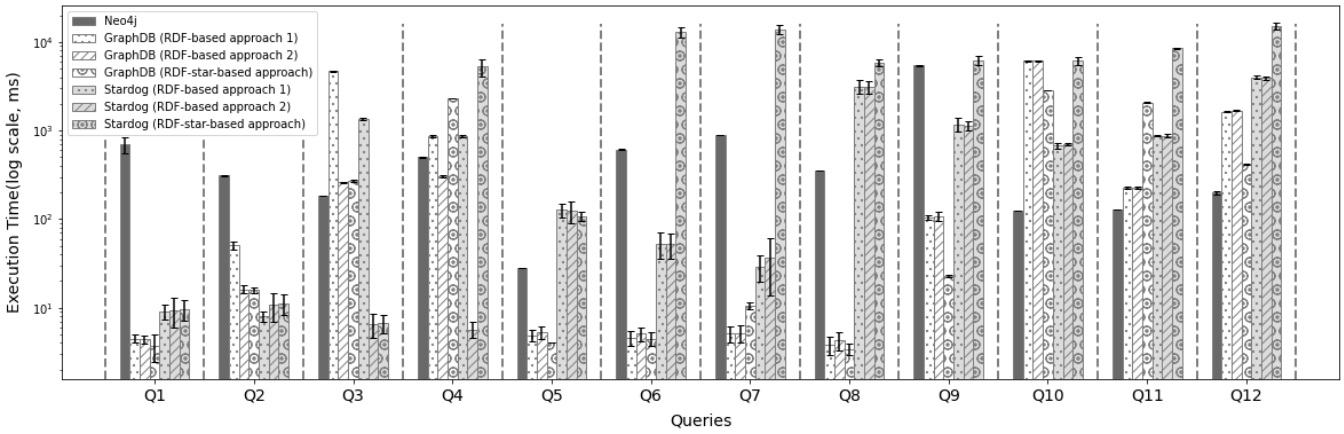

Figure 5: Execution time comparison of queries over data produced based on the three mappings, as achieved by GraphDB and by Stardog; execution times of corresponding Cypher queries over the original LPG in Neo4j are included as a baseline.

versus the 2 that the RDF-based approach 1 uses. The RDF-star approach uses a FILTER. Stardog presents the best performance.

Q5 is the first of the queries to include graph patterns to be matched over the data. In this case, obtaining property values of vertices connected to a given vertex. Here the three approaches behave similarly in terms of execution time, across both systems. This behavior makes sense, because the queries for the three approaches are only basic graph patterns and one OPTIONAL pattern.

Edge properties are the main focus of Q6–Q12. As it can be seen, queries for the RDF-based approaches behave similarly. It is an expected occurrence since both approaches have almost the same SPARQL query patterns differing only in some IRIs for labels. On the other hand, the queries for the RDF-star-based approach in Stardog behave inadequately when compared to GraphDB. In GraphDB, queries for the RDF-star-based approach are sometimes faster and sometimes slower than those for the RDF-based approaches.

In Q6, GraphDB presents the best performance across all mapping approaches, since it resolves the OPTIONAL patterns with less operations than Stardog. For the RDF-star-based approach, Stardog has a poor performance, presenting an execution time two orders of magnitude larger than the other alternatives in the same system.

Q7 and Q11 are queries containing nested triple patters with three variables in the RDF-star approach. In Stardog, the queries for the RDF-star-based approach take a significantly longer time which is not the case in GraphDB. However, in GraphDB they take a longer time than the other two RDF-based approaches. This performance in Stardog can be explained by having a look at the query plans and see how the system deals with the evaluation of nested triple patterns all in variables. For instance, for Q7 in the RDF-star based approach, Stardog first evaluates the embedded triple pattern that has three variables (?n1, ?r1, ?n2) (which matches with the entire RDF-star graph), then binds each solution mapping to a fresh variable, and then sorts the values and looks for matches in the edge property. This plan is very costly and most likely the reason for taking a long execution time. On the other hand, GraphDB first matches the edge property, and then performs a join with the rest of the triple patterns. Other queries with nested triple patterns, like Q9 and Q12, do not contain embedded triple patterns with three variables, however RDF-star-based queries tend to take longer than the RDF-based ones in Stardog, whereas in GraphDB, RDF-star queries perform best than the RDF-based ones in Q9, Q10 and Q12.

Q8 and Q10 present similar structures, but in different sizes. Both queries have a basic graph pattern and then obtain edge properties. However, this difference in query size produces a large variance in query execution time. In GraphDB, it takes three orders of magnitude more time to evaluate Q8 than Q10 for all three approaches. Stardog presents a similar performance on both queries, since it is mainly determined by the evaluation of the OPTIONAL patterns. Neo4j presents a lower execution time for Q10 that for Q8.

7 RELATED WORK

Existing works related to ours can be grouped into work on mappings between LPGs and RDF, and on studies that compare query performance over LPGs and RDF graphs.

In terms of mappings, there are several approaches to map RDF graphs to LPGs. For instance, Tomaszuk has introduced a serialization format called YARS which serializes RDF data in property graph style by using three declaration directives: one for prefixes, one for vertices, and one for edges [22]. Angles et al. have introduced a semantic interoperability approach between RDF and LPGs which includes three different mappings: simple database mapping, generic database mapping, and complete database mapping [3].

More relevant for our work in this paper, however, are mappings in the reverse direction, from LPGs to RDF or to RDF-star graphs. While we have already introduced the approaches of Nguyen et al. [18] and Hartig [12], there exist several other proposals, which we plan to experiment with as part of our future work.

In particular, Tomaszuk et al. have suggested a mapping that describes LPGs in RDF by using an ontology named PGO [23]. The ontology defines five classes for graph elements (e.g., pgo:Node), and for making the connections between instances of these classes, eight properties are defined (e.g., pgo:hasEdge). The number of triples for mapping a vertex, edge, and properties is equal to two, four, and four, respectively. Due to the large number of triples produced

by this mapping, querying might be challenging. Furthermore, a larger space for managing the data is required.

Das et al. study LPG-to-RDF transformation techniques for three methods of translating LPGs to RDF: reification based (RF), subproperty based (SP), and named graph based (NG) [7]. The number of triples produced for each edge in RF, SP, and NG is equal to four, three, and one, respectively. A large number of joins leads to challenges in querying the data. Moreover, for RDF stores that do not support named graphs, the NG approach can not be implemented.

Finally, Bruyat et al. introduce a tool for converting an LPG into an RDF-star graph based on a direct mapping that is encoded in the tool [5]. This mapping, however, is essentially the same as the one defined by Hartig [12] which we consider in this paper.

In terms of experimentation to compare query performance over LPGs and RDF graphs, Hernández et al. transform Wikidata to a relational database, to an LPG, and to four forms of RDF reification [15]. The main goal of the work is testing the engines from different families of data models in terms of query execution time. Four systems—Neo4j, Blazegraph, Virtuoso, and PostgreSQL–are used in the experiments. The authors have performed two types of experiments: atomic-lookup and basic graph pattern (BGPs). In the former one, Neo4j and Blazegraph behave the worst and even time out in some of the queries, and PostgreSQL performs the best. Whereas with BGPs, Virtuoso performs the best. Moreover, among four different RDF reification representing Wikidata, Named Graph and Standard Reification perform the best.

Das et al. [7], in their aforementioned study of three methods for translating LPGs to RDF, consider four types of queries, namely: node-centric, edge-centric, aggregate, and graph traversal queries. The named graph based method performs the best in the queries accessing edge properties because of using less joins due to the fewer number of triples needed for mapping an edge. A major difference between their work and ours is that we use a real world LPG dataset while they generate an LPG from a social network with only one type of relationships and no actual edge properties.

Similarly, instead of using graph data that has actually been created as an LPG, Nguyen et al. (from which we have adopted the RDF-based approaches) have used RDF data when experimenting with their model for representing LPGs in an RDF-compatible way [18]. Then, they have used a SPARQL-to-Gremlin plugin to query the generated LPG, instead of querying the RDF data directly.

8 CONCLUSIONS

In this paper we present results on an experiment comparing query performance over RDF and RDF-star graphs produced by applying different LPG mapping approaches. We define 12 benchmark queries that are executed over the different graphs by two RDF systems, GraphDB and Stardog, while we use corresponding Cypher queries over the original LPG loaded into Neo4j as a baseline.

From our experiments, we conclude that none of the three mapping approaches is a clear winner: Cypher queries are the fastest for three of the twelve cases, SPARQL queries for the RDF-based approaches 1 and 2 for one and for three cases, respectively, and the queries for the RDF-star-based approach for five cases.

Nonetheless, we can draw two main conclusions: (1) introducing two separate IRIs for vertex and edge labels tends to perform better than using a single IRI because it results in simpler queries (less triple patterns or FILTERs); and (2) Stardog produces sub-optimal query plans for SPARQL-star queries that contain nested triple patterns, which causes high execution times, whereas GraphDB produces better plans, which is reflected in lower execution times. It is worth noting that Neo4j is also not a clear winner in this scenario; it tends to perform better for queries that have longer and more complex path patterns.

As part of our future work, we plan to expand our experiment by considering additional mapping approaches, datasets, and RDF systems. Additionally, we plan a similar study for the direct mappings in the reverse direction (i.e., RDF or RDF-star to LPG).

ACKNOWLEDGMENTS

This work was funded by the Swedish Research Council (Vetenskapsrådet, project reg. no. 2019-05655) and by the CENIIT program at Linköping University (project no. 17.05).

REFERENCES

[1] Renzo Angles, Marcelo Arenas, Pablo Barceló, Aidan Hogan, Juan L. Reutter, and Domagoj Vrgoc. 2017. Foundations of Modern Query Languages for Graph Databases. *ACM Comput. Surv.* 50, 5 (2017).

[2] Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. 2019. RDF and Property Graphs Interoperability: Status and Issues. In *Proceedings of the 13th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web (AMW).*

[3] Renzo Angles, Harsh Thakkar, and Dominik Tomaszuk. 2020. Mapping RDF Databases to Property Graph Databases. *IEEE Access* 8 (2020).

[4] Mark Birbeck and Shane McCarron. 2010. CURIE Syntax 1.0. W3C Working Group Note, Online at https://www.w3.org/TR/curie/.

[5] Julian Bruyat, Pierre-Antoine Champin, Lionel Médini, and Frédérique Laforest. 2021. PREC: Semantic Translation of Property Graphs. *CoRR* abs/2110.12996 (2021). https://arxiv.org/abs/2110.12996

[6] Richard Cyganiak, David Wood, Markus Lanthaler, Graham Klyne, Jeremy J Carroll, and Brian McBride. 2014. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, Online at http://www.w3.org/TR/rdf11-concepts/.

[7] Souripriya Das, Jagannathan Srinivasan, Matthew Perry, Eugene Inseok Chong, and Jayanta Banerjee. 2014. A Tale of Two Graphs: Property Graphs as RDF in Oracle. In *Proceedings of the 17th International Conference on Extending Database Technology, EDBT 2014, Athens, Greece, March 24-28, 2014.*

[8] M. Duerst and M. Suignard. 2005. Internationalized Resource Identifiers (IRIs). RFC 3987, Online at http://www.ietf.org/rfc/rfc3987.txt.

[9] Laurence Goasduff. 2021. Gartner Identifies Top 10 Data and Analytics Technology Trends for 2021. https://www.gartner.com/en/newsroom/press-releases/2021-03-16-gartner-identifies-top-10-data-and-analytics-technologies-trends-for-2021.

[10] Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. 2013. SPARQL 1.1 Query Language. W3C Recommendation, Online at https://www.w3.org/TR/sparql11-query.

[11] Olaf Hartig. 2014. Reconciliation of RDF* and Property Graphs. *CoRR* abs/1409.3288 (2014). arXiv:1409.3288 http://arxiv.org/abs/1409.3288

[12] Olaf Hartig. 2017. Foundations of RDF* and SPARQL*: An Alternative Approach to Statement-Level Metadata in RDF. In *Proceedings of the 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web (AMW).*

[13] Olaf Hartig. 2019. Foundations to Query Labeled Property Graphs using SPARQL. In *Joint Proceedings of the 1st International Workshop On Semantics For Transport and the 1st International Workshop on Approaches for Making Data Interoperable co-located with 15th Semantics Conference (CEUR Workshop Proceedings, Vol. 2447).*

[14] Olaf Hartig, Pierre-Antoine Champin, Gregg Kellogg, and Andy Seaborne. 2021. RDF-star and SPARQL-star. W3C Community Group Report, Online at https://www.w3.org/2021/12/rdf-star.html.

[15] Daniel Hernández, Aidan Hogan, Cristian Riveros, Carlos Rojas, and Enzo Zerega. 2016. Querying Wikidata: Comparing SPARQL, Relational and Graph Databases. In *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 9982).*

[16] Philip Howard and Daniel Howard. 2019. Graph Database Market Update 2019. https://www.bloorresearch.com/research/graph-database-market-update-2019/.

[17] Ora Lassila, Michael Schmidt, Brad Bebee, Dave Bechberger, Willem Broekema, Ankesh Khandelwal, Kelvin Lawrence, Ronak Sharda, and Bryan B. Thompson. 2021. Graph? Yes! Which one? Help! *CoRR* abs/2110.13348 (2021). https://arxiv.org/abs/2110.13348

[18] Vinh Nguyen, Hong Yung Yip, Harsh Thakkar, Qingliang Li, Evan Bolton, and Olivier Bodenreider. 2019. Singleton Property Graph: Adding A Semantic Web Abstraction Layer to Graph Databases. In *Proceedings of the Blockchain enabled Semantic Web Workshop (BlockSW) and Contextualized Knowledge Graphs (CKG) Workshop.*

[19] Eric Prud'hommeaux and Gavin Carothers. 2014. RDF 1.1 Turtle. W3C Recommendation, Online at https://www.w3.org/TR/turtle/.

[20] Ian Robinson, Jim Webber, and Emil Eifrém. 2015. *Graph Databases: New Opportunities for Connected Data* (2nd ed.). O'Reilly Media, Inc.

[21] Guus Schreiber and Yves Raimond. 2014. RDF 1.1 Primer. W3C Working Group Note, Online at https://www.w3.org/TR/rdf11-primer/.

[22] Dominik Tomaszuk. 2016. RDF Data in Property Graph Model. In *Metadata and Semantics Research - 10th International Conference, MTSR 2016, Göttingen, Germany, November 22-25, 2016, Proceedings (Communications in Computer and Information Science, Vol. 672).*

[23] Dominik Tomaszuk, Renzo Angles, and Harsh Thakkar. 2020. PGO: Describing Property Graphs in RDF. *IEEE Access* 8 (2020).

[24] Noel Yuhanna. 2017. Vendor Landscape: Graph Databases. https://www.forrester.com/report/Vendor+Landscape+Graph+Databases/-/E-RES121473.