A Unified Relational Storage Scheme for RDF and Property Graphs

Ran Zhang[1,2], Pengkai Liu[1,2], Xiefan Guo[1,2], Sizhuo Li[1,2], and Xin Wang[1,2](✉)

[1] College of Intelligence and Computing, Tianjin University, Tianjin, China
{gxf_1998,wangx}@tju.edu.cn

[2] Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin University, Tianjin, China

Abstract. With the advance of Semantic Web and development of Linked Data, the scale of knowledge graphs has surged dramatically. On the one hand, RDF graph is a mainstream data model of the knowledge graph. On the other hand, property graphs are widely accepted in graph databases. How to manage large-scale RDF and property graphs in an interchangeable way has become popular in both academic and industrial communities. Thus we present an effective unified relational storage scheme, that can seamlessly accommodate both RDF and property graphs. Furthermore, we have implemented the storage schema on an open-source graph database to verify its effectiveness. Ultimately, our experimental results show that the proposed unified storage schema for both RDF and property graphs can effectively manage large-scale knowledge graphs, efficiently avoid data redundancy, and achieve high-performance queries.

Keywords: Knowledge graph · RDF · Property graph · Efficient storage

1 Introduction

Knowledge graphs have become the cornerstone of artificial intelligence. The construction and publishing of large-scale knowledge graphs in various domains have posed new challenges on the management of those graphs.

Currently, there are two mainstream data models of knowledge graphs, namely the RDF [8] (Resource Description Framework) model and the property graph model. The former has been standardized by the W3C (World Wide Web Consortium), and the latter has been widely accepted in industrial communities of graph databases. Unlike the relational database communities, however, the two models of knowledge graphs and their query languages have not yet been unified. For RDF graphs, their model has a profound mathematical foundation and relatively complete model characteristics, and with the Linked Data [11] initiative, an increasingly large number of RDF data have been published on

W. Ni et al. (Eds.): WISA 2019, LNCS 11817, pp. 418–429, 2019.
https://doi.org/10.1007/978-3-030-30952-7_41

the Semantic Web; whereas for property graphs, their model has built-in support for properties and several query languages, including Cypher [9], Gremlin [10], and PGQL [12]. Property graphs, which have not been standardized yet, have been widely recognized in industrial communities with the application of graph databases. Due to the hypergraph structure of RDF graphs, it has been demonstrated that RDF graphs are more expressive than property graphs. How to effectively manage both RDF and property graphs in a unified storage schema has become an urgent problem. In this paper, we firstly focus on the integration of RDF and property graphs at the storage layer.

The relational model has increasingly turned mature over several decades. It has concise and universal relational structures, and expresses the operations and constraints of relationships using relational algebra with strict mathematical definitions. Therefore, it can provide a solid theoretical foundation to store RDF and property graphs.

Our contributions can be summarized as follows:

(1) We propose a unified relational storage schema, that can seamlessly accommodate both RDF and property graphs.
(2) We then implement the storage schema on an open-source database, AgensGraph, to verify its effectiveness and efficiency.
(3) To some extent, we manage RDF and property graphs in an interchangeable way and realize the interoperability between the two models.

The remainder of this paper is organized as follows: Sect. 2 introduces the related work and the formal definitions of RDF and property graphs are given in Sect. 3. The unified storage schema we proposed is illustrated in Sect. 4, the subsequent Sect. 5 describes the implementation on an open-source graph database with the experimental results. Finally, we conclude the paper by discussing future research directions in Sect. 6.

2 Related Work

The knowledge graph data model is based on the graph structure, with vertices representing entities and edges representing the relationships between those entities. This kind of general data representation can naturally depict the extensive connections between things in the real world.

2.1 The RDF Storage Schema

There are two typical approaches to designing RDF data management systems: relational approaches and graph-based approaches [18]. The relational approaches map RDF data to a tabular representation and then execute SPARQL queries on it while the latter approach is graph-based, which model both RDF and the SPARQL query as graphs and execute the query by subgraph matching using homomorphism [16].

Relationship-Based Knowledge Graphs Storage Management. Relational databases are still the most widely used database management system at present, and the storage scheme based on relational database is a main storage method of knowledge graphs data currently [1]. The triple table storage scheme directly stores RDF data; the horizontal table storage scheme [3,7,15] records all predicates and objects of a subject in each row; the property table storage scheme is a subdivision of the horizontal table, and the same subject will be stored in a table, which solves the problem of too many columns in the table; the vertical partitioning storage scheme creates a two-column table for each predicate [2]; the sextuple indexing storage scheme divides all six permutations of a triple into six tables [14]. Last but not least, DB2RDF [6] has been used to improve query performance recent years by creating entity-oriented storage structures that reduce the Cartesian product operations in queries.

Graph-Based Knowledge Graphs Storage Management. The advantage of graph-based approach is that it maintains the original representation of the RDF data as well as it enforces the intended semantics of SPARQL. The disadvantage, however, is that the cost of subgraph matching by graph homomorphism is NP-complete [18]. Systems such as that proposed by Bönström et al. [5], gStore [15,17], and chameleon-db [4] follow this approach.

2.2 The Property Graph Storage Scheme

A property graph is a directed, labeled, and attributed multi-graph. It means that the edges of a property graphs are directed, and both vertices and edges can be labeled and can have any number of properties, and there can be multiple edges between any two nodes [13]. Neo4j[1] is a native graph database that supports transactional applications and graph analytics, and it is currently the most popular property graphs database. Neo4j is also based on a network-oriented model where relations are first-class objects.

At present, the knowledge graph data model and the query language are not unified. The main reason for the surge of relational databases is that it has a precisely defined relational data model and a unified query language SQL. The unified data model and query language not only reduce the development and maintenance costs of the database management system, but also reduce the learning difficulty of users. Therefore, based on the existing work, we propose a unified relational storage scheme for RDF and property graph model.

3 Preliminaries

In this section, we provide the formal definitions of RDF triple, RDF graph, and property graph, which can be the basis for the transformations to relational tables in the document.

[1] https://neo4j.com/.

Definition 1 (RDF triple). *Let U, B and L be disjoint sets of URIs, blank nodes and literals, respectively. An RDF triple $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ states the fact that the resource s has the relationship p to the resource $o \in U$, or the resource s has the property p with the value $o \in L$, where s is called the subject, p the predicate (or property), and o the object.*

Definition 2 (RDF graph). *A finite set of RDF triples is called an RDF graph. Given an RDF graph T, we use $S(T)$, $P(T)$, and $O(T)$ to denote the set of subjects, predicates, and objects in T, respectively. For a certain subject $s_i \in S(T)$, we refer to the triples with the same subject s_i collectively as the entity s_i, denoted by $\mathsf{Ent}(s_i) = \{t \in T \mid \exists p, o \ s.t. \ t = (s_i, p, o)\}$.*

We can use RDF Schema (RDFS) to define classes of entities and the relationships between these classes. For example, $(s, \mathsf{rdf{:}type}, C)$ declares that the entity s is an instance of the class C. Given an RDF graph T, we assume that for each subject $s \in S(T)$ there exists at least a triple $(s, \mathsf{rdf{:}type}, C) \in \mathsf{Ent}(s)$, denoted by $s \in C$. We believe that this assumption is reasonable since every entity should belong to at least one type in the real world.

Definition 3 (Property graph). *Let L and T be countable sets of node labels and relationship types, respectively [16]. A property graph is a tuple $G = (N, R, src, tgt, l, \lambda, \tau)$ where:*

- *N is a finite subset of N, whose elements are referred to as the nodes of G.*
- *R is a finite subset of R, whose elements are referred to as the relationships of G.*
- *src: $R \rightarrow N$is a function that maps each relationship to its source node.*
- *tgt: $R \rightarrow NN$ is a function that maps each relationship to its target node.*
- *l: $(N \cup R) \times K \rightarrow V$ is a finite partial function that maps a (node or relationship) identifier and a property key to a value.*
- *λ: $N \rightarrow 2L$ is a function that maps each node id to a finite (possibly empty) set of labels.*
- *τ: $R \rightarrow T$ is a function that maps each relationship identifier to a relationship type.*

4 The Unified Relational Storage Schema

Originally, we propose a unified relational storage schema for both RDF and property graphs. Then we elaborate on the specific rules for transforming RDF and property graphs into relational tables to effectively realize the storage integration.

4.1 Integration of RDF and Property Graphs in Relational Tables

As the representations of knowledge graph models, RDF and property graphs are relatively independent with expressivity difference, increasing the difficulty of the direct mapping. As shown in Fig. 1, we select the mature relational model as the physical storage model to realize the integration of RDF and property graphs.

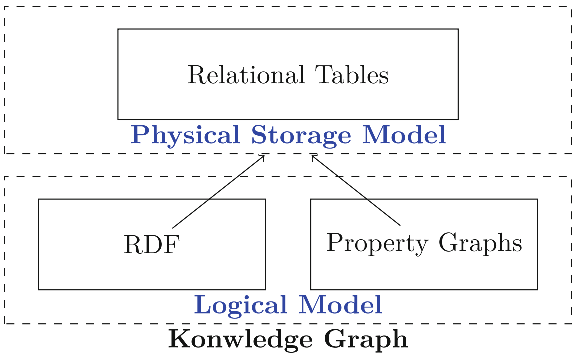


Fig. 1. The unified relational storage schema

4.2 Transforming RDF Graphs into Relational Tables

Since an RDF graph is defined as a finite set of triples, an RDF graph can be mapped into multiple relational tables. Mapping rules for an RDF graph to relational tables will be defined as follows.

RDF triples, by definition, will be formalized as $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$. For simplicity, the namespace prefix of the resource and predicate URI names will be omitted in this paper (RDF built-in names is not omitted, such as rdf:type). Since the introduction of blank nodes will not make a fundamental change to the RDF data management method, the blank node in the RDF graph will be equated to the URI in this paper.

For three different forms of RDF triples, we define the basic mapping rules for RDF to relational tables as follows:

Rule 1. An RDF triple in the form of $\langle U_1 \rangle$ $\langle$rdf:type$\rangle$ $\langle U_2 \rangle$, that the predicate of the RDF triple is $\langle$rdf:type$\rangle$, then it can be expressed as a row with *id* (primary key) and *properties* in relational table U_2.

Rule 2. An RDF triple in the form of $\langle U_1 \rangle \langle U_2 \rangle \langle L \rangle$, that the object of the RDF triple is *literal*, then it can be expressed as a property $\{U_2 : L\}$ in properties of U_1.

Rule 3. An RDF triple in the form of $\langle U_1 \rangle \langle U_2 \rangle \langle U_3 \rangle$, that the subject, the predicate, and the object of the RDF triple are all URI, then it can be expressed as a row with *id* (primary key), *start* that is the foreign key referencing the *id* of U_1, *end* that is the foreign key referencing the *id* of U_3, and *properties* in relational table $U2$.

As shown in Fig. 2, most RDF graphs can be mapped to relational schemata according to the above basic rules.

In particular, the intersection of vertices and edges is not empty in RDF graphs. Specifically, the predicate can also act as the subject or the object of another RDF triple. We then propose a solution to implementation of RDF reification. In the relational schema, we artificially create a relational table

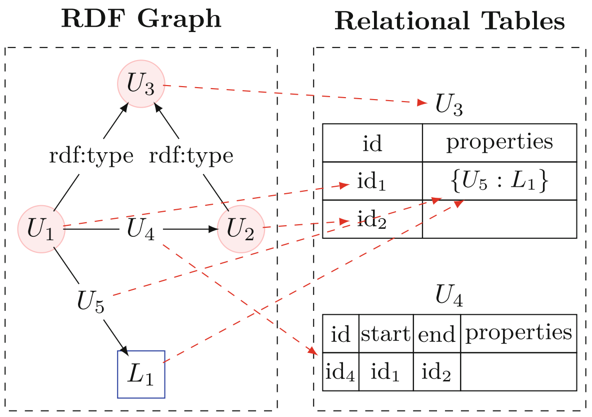


Fig. 2. The basic mapping from RDF graphs to relational tables

called "Edge_Vertex" with column *Vertexid* (primary key), column *Edgeid* that is the foreign key referencing the id of the edge, and column *properties*. The Edge_Vertex table stores edges as vertices to realize following relationships between edges and vertices or between edges and edges. Namely, as presented in Fig. 3, we use the dual storage to reserve the complete information of RDF in the relational model.

4.3 Transforming Property Graphs into Relational Tables

Property graphs also play a considerable role in knowledge graphs. In property graphs, an entity is represented as a vertex. Vertices and edges can have an arbitrary number of properties and can be categorized with labels. Labels are used to gather vertices and edges that have the same category. Furthermore, edges are directionally connected between two vertices, a start vertex and an end vertex.

We explore the transformation from property graphs to relational tables. For vertices and edges, we define the mapping rules for property graphs to relational tables as follows:

Rule 1. Labels can be represented as relational tables within vertices and edges of the same category.

Rule 2. Vertex tables have two columns, namely *id* (primary key) and *properties*.

Rule 3. Edge tables have four columns, namely *id* (primary key), *start* and *end* that are both the foreign keys referencing the *id* of vertex tables, and *properties*.

Rule 4. A vertex or an edge can be expressed as a row of the relational table.

According to the above rules, Fig. 4 visually shows the mapping from property graphs to relational tables.

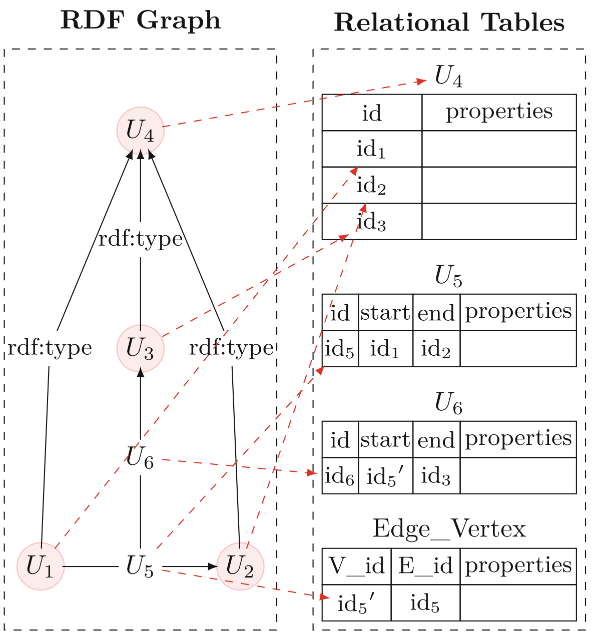


Fig. 3. The complete mapping from RDF graphs to relational tables

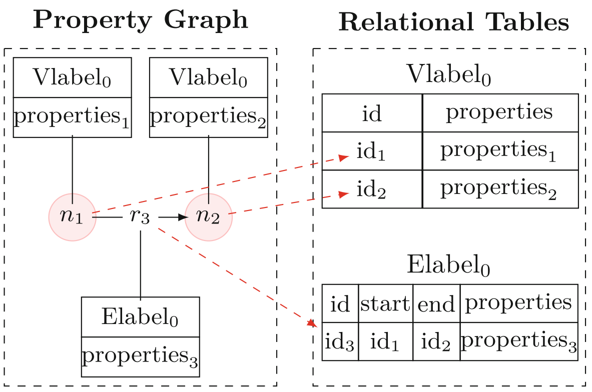


Fig. 4. The mapping from property graphs to relational tables

5 Experiments

We have conducted experiments on synthetic RDF datasets to verify the effectiveness and efficiency of our method. The database is deployed on a desktop computer that has an Intel i54520 CPU with 2 cores of 2.31 GHz, 8 GB memory, 512 GB disk, and 64-bit Centos7.0 as the OS.

We implemented the storage schema on AgensGraph v2.1.1[2]. AgensGraph is a new generation multi-model graph database for the modern complex data environment, that is very robust, fully-featured and ready for enterprise use. AgensGraph both supports relational tables and property graphs, and it has already realized the mapping from property graphs to relational tables. Consequently, RDF graphs are required to be imported into AgensGraph as relational tables.

As shown in Fig. 5, based on the existing storage mechanism, we extended the storage schema to accommodate RDF storage for AgensGraph with no effect to the original storage of relational tables and property graphs. According to the extension, RDF and property graphs can be stored and managed independently and compatibly in AgensGraph.

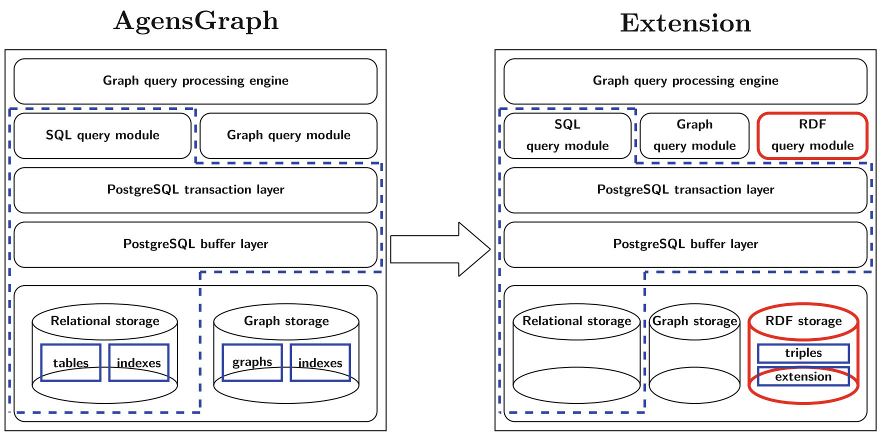


Fig. 5. The complete mapping from RDF graphs to relational tables

We generated five synthetic datasets using the LUBM (Lehigh University Benchmark), which is developed to facilitate the evaluation of Semantic Web repositories in a standard and systematic way, as a test sample imported into AgensGraph. LUBM consists of a university domain ontology, customizable and repeatable synthetic data, a set of test queries, and several performance metrics. The characteristics of each dataset are shown in Table 1.

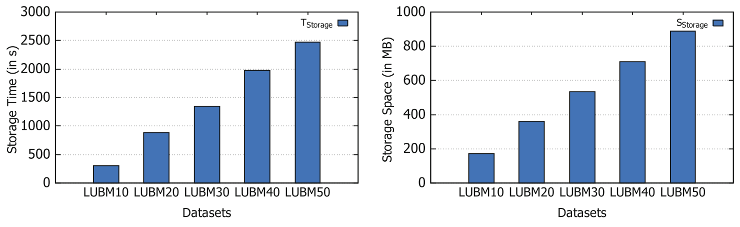
[2] https://bitnine.net/.

Table 1. Characteristics of experimental datasets

Datasets	Sizes	Triple #
LUBM10	168.4M	1,316,700
LUBM20	358.0M	2,781,322
LUBM30	529.6M	4,107,812
LUBM40	709.0M	5,494,144
LUBM50	889.5M	6,888,642

Experiment 1: Storage Performance Analysis. We considered two indicators to evaluate the storage performance, namely storage time and storage space.

Storage time overhead is a significant indicator to evaluate the performance of the storage schema for importing RDF triples. Figure 6(a) shows the storage time to store RDF datasets of different sizes.



(a) The storage time of RDF (b) The storage space of RDF

Fig. 6. The storage time and space of RDF

Additionally, storage space overhead is also important to measure storage performance. By importing RDF into AgensGraph, the number of established vertices and edges are shown in Table 2. Figure 6(b) plots the storage space of different RDF datasets in AgensGraph. From Fig. 6(b) we can see that with continued accretion

Table 2. Number of vertex and edge

Datasets	Vertex#	Edge#
LUBM10	219,680	673,602
LUBM20	463,296	1,422,567
LUBM30	684,222	2,101,605
LUBM40	915,519	2,810,798
LUBM50	1,147,136	3,524,142

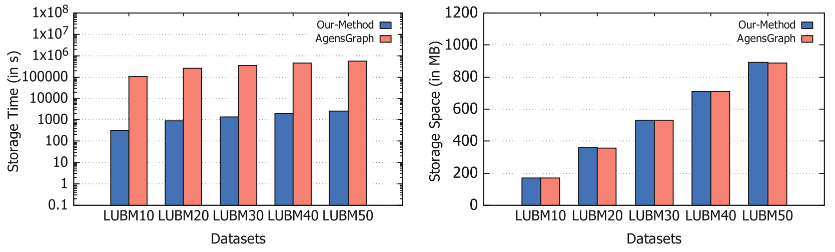
of the size of RDF data, the storage scheme can significantly reduce the spatial storage of knowledge graphs and the redundancy of data storage.

Experiment 2: Interoperability of RDF and Property Graphs. LUBM provides 14 SPARQL query statements to measure the performance. Therefore, we tested them on AgensGraph to realize the interoperability of RDF and property graphs. For instance, Query number, answer, and query time of LUBM50 are shown in Table 3. From Table 3, we found the storage schema can effectively achieve the interoperability.

Table 3. Query Results of LUBM50

QueryNo.	Answer	Time/ms
Q1	4	8.438
Q2	130	9,099.834
Q3	6	32.527
Q4	34	29.932
Q5	678	105.774
Q6	519,842	17,578.083
Q7	67	56.935
Q8	7,790	5,515.660
Q9	13,639	10,193.622
Q10	4	43.472
Q11	0	25.351
Q12	0	0.561
Q13	0	10.739
Q14	393,730	546.531

Experiment 3: Comparison Between Import Methods. To verify the effectiveness of the storage schema, we compared the unified relational storage schema (Our-Method) with importing RDF graphs as property graphs (Agens-Grpah) on storage time. From the experimental results, as shown in Fig. 7, the storage time and storage space are positively correlated with the size of the datasets. The efficiency of the proposed relational storage schema has increased hundreds of times with the roughly equivalent storage space, which is valid for large-scale RDF storage.



(a) The comparision of storage time (b) The comparison of storage space

Fig. 7. The comparison between our method and AgensGraph

6 Conclusion and Outlook

In this paper, we have developed a unified relational storage schema of RDF and property graphs. On the one hand, we have solved the large-scale knowledge graph storage problem to some extent. On the other, the proposal of the unified storage schema promotes the integration of two mainstream data models of knowledge graph, playing an important role in the establishment of dominant knowledge graph databases.

The Unified data model not only lowers the development and maintenance cost of database management system, but also reduces the learning difficulty of users. Based on the unified storage schema, a unified query schema of Cypher and SPARQL needs to be proposed to realize a real sense of RDF to property graph interoperability. Therefore, it is an important research direction in the future to develop a unified knowledge graph query language with precise grammar and semantics. Furthermore, the research and development of distributed storage of large-scale knowledge graph data is still in its infancy, and the efficient algorithm of distributed queries is to be improved.

Acknowledgments. This work is supported by the National Natural Science Foundation of China (61572353), the Natural Science Foundation of Tianjin (17JCYBJC15400), and the National Training Programs of Innovation and Entrepreneurship for Undergraduates (201810056090).

References

1. Wang, X., Zou, L., Wang, C.K., Peng, P.: Research on knowledge graph data management: a survey. Ruan Jian Xue Bao/J. Softw. (2018). (in Chinese)
2. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: SW-store: a vertically partitioned dbms for semantic web data management. VLDB J. **18**(2), 385–406 (2009)
3. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., Tolle, K.: The ICS-forth RDFSuite: managing voluminous RDF description bases. In: SemWeb (2001)

4. Aluc, G.: Workload matters: a robust approach to physical RDF database design (2015)
5. Bonstrom, V., Hinze, A., Schweppe, H.: Storing RDF as a graph. In: Proceedings of the IEEE/LEOS 3rd International Conference on Numerical Simulation of Semiconductor Optoelectronic Devices (IEEE Cat. No. 03EX726), pp. 27–36. IEEE (2003)
6. Bornea, M.A., Broekstra, J., Kampman, A., Van Harmelen, F.: Building an efficient RDF store over a relational database. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 121–132. ACM (2013)
7. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: a generic architecture for storing and querying RDF and RDF schema. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342, pp. 54–68. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-48005-6_7
8. Cyganiak, R., Wood, D., Lanthaler, M., Klyne, G., Carroll, J.J., McBride, B.: RDF 1.1 concepts and abstract syntax. W3C recommendation, 25 February 2014
9. Francis, N., et al.: Cypher: an evolving query language for property graphs. In: Proceedings of the 2018 International Conference on Management of Data, pp. 1433–1445. ACM (2018)
10. Khokha, M.K., Hsu, D., Brunet, L.J., Dionne, M.S., Harland, R.M.: Gremlin is the BMP antagonist required for maintenance of Shh and Fgf signals during limb patterning. Nat. Genet. **34**(3), 303 (2003)
11. Li, W., Chai, L., Yang, C., Wang, X.: An evolutionary analysis of DBpedia datasets. In: Meng, X., Li, R., Wang, K., Niu, B., Wang, X., Zhao, G. (eds.) WISA 2018. LNCS, vol. 11242, pp. 317–329. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02934-0_30
12. van Rest, O., Hong, S., Kim, J., Meng, X., Chafi, H.: PGQL: a property graph query language. In: Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems, p. 7. ACM (2016)
13. Rodriguez, M.A., Neubauer, P.: Constructions from dots and lines. Bull. Am. Soc. Inf. Sci. Technol. **36**(6), 35–41 (2010)
14. Weiss, C., Karras, P., Bernstein, A.: Hexastore: sextuple indexing for semantic web data management. Proc. VLDB Endow. **1**(1), 1008–1019 (2008)
15. Zou, L., Mo, J., Chen, L., Özsu, M.T., Zhao, D.: gStore: answering SPARQL queries via subgraph matching. Proc. VLDB Endow. **4**(8), 482–493 (2011)
16. Zou, L., Özsu, M.T.: Graph-based RDF data management. Data Sci. Eng. **2**(1), 56–70 (2017)
17. Zou, L., Özsu, M.T., Chen, L., Shen, X., Huang, R., Zhao, D.: GStore: a graph-based SPARQL query engine. VLDB J. Int. J. Very Large Data Bases **23**(4), 565–590 (2014)
18. Özsu, M.T.: A survey of RDF data management systems. Front. Comput. Sci. **10**(3), 1–15 (2016)