Semantic Web 0 (2017) 1–0
IOS Press

Evaluation of Metadata Representations in RDF stores

Johannes Frey [a], Kay Müller [a], Sebastian Hellmann [a], Erhard Rahm [b] and Maria-Esther Vidal [c]
[a] *AKSW/KILT Competence Center, InfAI, Leipzig University & DBpedia Association*
`{frey,kay.mueller,hellmann}@informatik.uni-leipzig.de`
http://aksw.org/Groups/KILT.html
[b] *Database group, Leipzig University*
`{rahm}@informatik.uni-leipzig.de`
http://dbs.uni-leipzig.de
[c] *TIB Leibniz Information Centre for Science and Technology & Fraunhofer IAIS, Germany*
`{Maria.Vidal}@tib.eu`
https://www.tib.eu/en/research-development/scientific-data-management/staff/

Editors: Axel-Cyrille Ngonga Ngomo, Institute for Applied Informatics, Leipzig, Germany; Irini Fundulaki, ICS-FORTH, Greece; Anastasia Krithara, National Center for Scientific Research "Demokritos", Greece
Solicited reviews: Gabor Bella, University of Trento, Italy; Pavel Smirnov, AGT International, Germany; Theodore Dalamagas, Athena Research and Innovation Center, Greece; one anonymous reviewer

Abstract. The maintenance and use of metadata such as provenance and time-related information is of increasing importance in the Semantic Web, especially for Big Data applications that work on heterogeneous data from multiple sources and which require high data quality. In an RDF dataset, it is possible to store metadata alongside the actual RDF data and several possible metadata representation models have been proposed. However, there is still no in-depth comparative evaluation of the main representation alternatives on both the conceptual level and the implementation level using different graph backends. In order to help to close this gap, we introduce major use cases and requirements for storing and using diverse kinds of metadata. Based on these requirements, we perform a detailed comparison and benchmark study for different RDF-based metadata representations, including a new approach based on so-called companion properties. The benchmark evaluation considers two datasets and evaluates different representations for three popular RDF stores.

Keywords: Metadata, RDF, Evaluation, Reification

1. Introduction

Within the Semantic Web community, the topic of metadata has been subject of many discussions and works for several years. These works range from the publication of different metadata vocabularies (e.g., PROV-O[1], the Dublin Core Metadata Initiative[2], and the Data Catalog Vocabulary[3]), the application of these vocabularies in datasets, the development of different metadata representation models (MRM), metadata support by graph backends, and much more.

In the context of this paper, we focus on metadata representation models *(MRM)* for *knowledge graphs* and how data and metadata are connected in the same RDF store. Knowledge graphs consist of information collected from different (data) sources, which evolve over time and can reach large dimensions. Usually, heterogeneous sources contain information about *similar* or *equivalent* entities, which represent the same real-world object. Such overlapping entities from different

[1] https://www.w3.org/TR/prov-o/
[2] http://dublincore.org/documents/dcmi-terms/
[3] https://www.w3.org/TR/vocab-dcat/

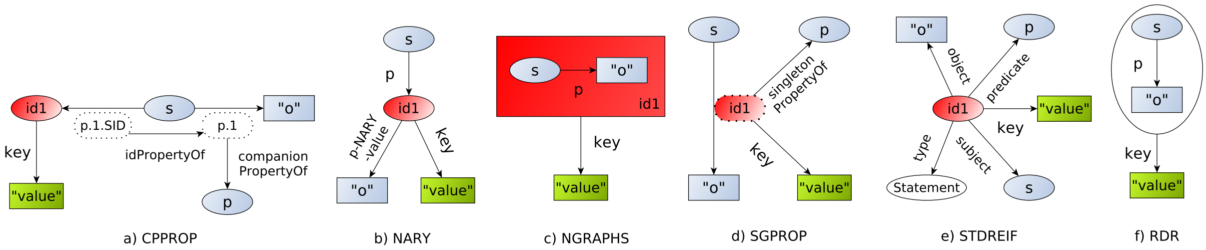

Figure 1. **Structure of different Metadata Representation Models:** Six different ways of describing (or reifying) an RDF triple `s, p, o` with a metadata `key` and `value` pair are studied in this work. Companion property (cpprop), nary relation (naryrel), named graphs (ngraphs), singleton properties (sgprop), standard reification (stdreif), and the Blazegraph-specific Reification Done Right (rdr). Besides rdr all the approaches use an explicit statement identifier (red), which is used to attach metadata (green) to the data (grey). Cpprop and stdreif are based on additional triple handlers (white). Properties which also deal as subjects are drawn with dashed lines.

datasets will have common (e.g., birth year), conflicting (e.g., different heights of a mountain), and complementary property attributes (e.g., fact about an entity only available in one dataset). The process of merging these entity information into one common dataset is called *knowledge fusion* [8] which can include operations such as provenance tracing, conflict detection, conflict resolution, and merge. Resolving conflicting data values can be improved by metadata-based heuristics [5], e.g., prefer newer facts or prefer values from a source, which is known to ensure high data quality. Furthermore, data traceability is an important use case where not only source provenance data is recorded, but also data processing information such as results from normalization and cleaning operations, and information about applied (fusion) algorithms. Traceability can help users and developers to understand the results of Big Data systems and allows them to track erroneous statements back to its data sources and the involved algorithms, by inspecting provenance metadata. This topic has reached a political level and the EU[4] is pushing for new regulations, which will force commercial solutions to add traceability of data to Big Data systems.

As metadata representation model (*MRM*), we define a strategy of splitting an RDF triple t and its set of key-value based metadata facts m into several triples or quads, such that we can store and query metadata, for all triples individually, in an RDF Store.

Handling data and metadata alongside each other can be considered a challenging task. Since more data has to be processed, stored and indexed, a negative impact on the overall system performance might occur. However, solutions which store data and metadata in separate databases or backend types (e.g., data in an RDF store and metadata in a relational database) require complex and time-consuming join, lookup or query federation solutions. Furthermore, such setups are harder to maintain because data and metadata may be out of sync. Hence, this work will evaluate how to store metadata alongside data, using different MRMs and RDF stores. Figure 1 illustrates the main structural differences between the various MRMs, which will be explained in detail in Section 4.

The contributions of this work are as follows: We performed a thorough and comprehensive evaluation of metadata handling in RDF from several angles. As foundation for this evaluation but also as basis for a future RDF metadata performance benchmark, we defined requirements and criteria for an evaluation of metadata representation models, based on an analysis of existing RDF datasets and use cases where metadata is involved. We specified a setup motivated by a knowledge fusion use-case including a novel high degree metadata-dataset and a set of data-only and mixed benchmarking queries from different complexity classes. We validated and revised previous experimental results from the state-of-the-art, and systematically compared and evaluated several MRMs (including vendor-specific approaches and a new companion properties approach) against different RDF stores. To close the gap to previous work, we considered open questions and aspects like the MRM overhead for regular data queries, the impact of dataset specific characteristics, metadata granularity levels and meta-metadata.

The rest of the paper is structured as follows: Section 2 gives an overview about related work. Then the evaluation requirements and criteria are presented in Section 3. In Section 4 we describe different models to represent metadata and introduce companion prop-

[4]http://irishtechnews.net/ITN3/eu-regulations-on-the-traceability-of-your-data-is-looming-on-the-horizon/

erties. The evaluation datasets and the evaluation setup are described in sections 5 and 6. Then the evaluation results are presented in Section 7 and compared to other studies. Finally, we conclude and discuss future work.

2. Related Work

Related work can be separated into three groups. First several SPARQL-benchmarks exist, which try to evaluate the performance of RDF stores. Second, there is related work, which discusses or proposes MRMs, extensions of RDF or systems based on RDF to handle provenance and other metadata. The last group evaluates a set of MRMs against one or more RDF stores.

SPARQL & RDF store benchmarks:

Over the past years several benchmarks have been created with the aim of providing testbeds for the evaluation of the performance of RDF stores. When creating these benchmarks, different strategies have been applied. In the case of synthetic SPARQL benchmarks like LUBM [4], BSBM [18], and SP2Bench [43], the idea for the datasets and search queries are deduced from real-world use cases, but both the dataset and the queries are automatically generated. The Linked Data Benchmark Council (LDBC)[5] offers three benchmarks [2] for different domains, such as semantic publishing, social network and graph analytics [24]. LDBC is using a combination of real and synthetic data while [47] uses real-world datasets and a set of 15 fixed queries. Despite the fact that synthetic benchmarks are inspired by real-world use cases, the characteristics of neither the datasets nor the queries will adequately represent common real-world usage scenarios. Hence, the authors of [32] proposed the DBpedia benchmark, where both real-world data and user query logs are used during a testbed generation. Similarly, Saleem et al. [41] propose a framework that relies on query logs; however, the framework is able to cluster query logs and generate queries that fulfill a set of requirements (e.g., usage of aggregates and GroupBy). In [1], the authors evaluate different RDF benchmarks and show that many of them lack a diverse (with respect to criteria like result cardinality, join degree, and type) testing. They state that varied queries and workloads are necessary to detect performance differences of RDF store implementation details and to drill down performance problems for specific combinations of query features. To close this gap they proposed the *WatDiv* benchmark. Although the authors do not specifically evaluate datasets with metadata, they highlight the need for benchmarks, which cover provenance and temporal data as well. Finally, the semantic web community has been actively proposed testbeds for evaluating: Federated query engines [31, 40, 42], triple/graph or dataset versioning [11, 29, 36], streaming [48], and geospatial[6] querying [15]. However, to the best of our knowledge, there is no benchmark that generates metadata-rich datasets or queries over fine grained meta information.

Metadata handling & extensions in RDF: A first extension of RDF to support efficient storage of provenance for RDF data had been addressed with the proposal of named graphs [6]. Meanwhile, several syntactic and/or semantic metadata or provenance extensions of RDF have been published; however, none of them had been included in Semantic Web W3C standards so far. The need for annotating triples to describe provenance, time, trust or fuzziness of a statement has been outlined in [27], but semantically formalized in [46] with annotated RDF (aRDF). Within aRDF, triples can be described by members of a partially ordered set. Lopes et al. [26] propose basic annotated patterns for enabling query processing over aRDF documents using extended SPARQL based SELECT queries. RDF+ [44] adds a semantic interpretation for metadata to RDF and extends SPARQL with a `WITH META` keyword. While it reuses the named graphs feature in a backward-compatible way, the SPARQL extension is only implemented in a research prototype. Another extension of SPARQL [7], based on a hierarchy of named graphs, uses a new `STATE` keyword to query for the state (or metadata) of a triple. To the best of our knowledge, RDF* and SPARQL* [20] are the only well-founded (but non-standardized) extensions of RDF and SPARQL, which are included in a recent, maintained, and fully SPARQL 1.1 compatible RDF store. Moreover, in order to optimize performance and usability of querying and storage of temporal metadata in RDF (stores), approaches extending RDF and SPARQL have been proposed [25, 30]. While temporal and spatial metadata, as well as triple versioning can be represented with MRMs, approaches optimized and limited with respect to these dimensions are out of the scope of this work.

[5] http://www.ldbcouncil.org/

[6] http://www.opengeospatial.org/standards/geosparql

Since the presented RDF & SPARQL extensions are lacking a widespread adoption in RDF stores, RDF standard compatible MRMs have been published. This includes singleton properties [35] and nanopublications [17] which will be covered in more detail in Section 4.1 and Section 3.1.1.

MRM studies: Despite the fact that the handling of metadata is becoming a more important topic in the Semantic Web community, not many evaluations have been conducted. Work studying different MRMs has been reported in [13, 22, 23]. The storage of redundant provenance per pairwise relation between chemicals, genes, and diseases is evaluated in [13]. Since the same backends have been used, there will be a detailed discussion of the results in Section 7.7. In [22] and [23], the authors apply different MRMs on a Wikidata dataset. In [22], Virtuoso and Blazegraph in combination with different MRMs are compared to the non-RDF backends PostgreSQL and Neo4J using several template queries. In [23], the focus lies on the comparison of the RDF stores 4store, BlazeGraph, GraphDB, Jena TDB and Virtuoso but only using a set of 15 fixed query instances.

This work features a novel DBpedia based dataset including high degree revision metadata, meta-metadata and a set of data-only and mixed query templates from three complexity classes. By revising and reproducing results from [22] we allow for a direct comparison of results and studying of dataset specific impacts between Wikidata and our knowledge fusion use case.

3. Evaluation requirements and criteria

To establish a foundation for our evaluation, we define requirements and criteria for metadata usage, MRMs, and their comparison. In this section we will give a brief rundown of our requirements, and present a set of criteria, which are used to analyze, measure and compare the different MRMs. Moreover, we present an overview of metadata usage in RDF as basis for the requirements.

3.1. Metadata (Requirements) Analysis

In order to establish MRM evaluation requirements, an analysis of metadata usage and datasets with metadata has been conducted.

3.1.1. Metadata usage analysis

Several search strategies have been used to explore metadata usage. First repositories such as datahub.io[7], as well as other CKAN[8] repository instances were searched. Second, with the help of LODVader [3], LodLaundromat[9] and lodstats[10] we searched for datasets containing MRM patterns and specific metadata vocabulary in the Linked Open Data (LOD) cloud. For provenance related metadata, the following metadata vocabularies were investigated: PROV-O[11], Dublin Core[12]. Furthermore, we checked for more dataset oriented metadata vocabularies such as DCAT[13], VoID[14], and Data Cube[15]. In order to find out how often metadata vocabularies are already used in the LOD cloud, we used the LODVader service[16], which indexes most active datasets of the LOD cloud. With the help of this index, we were able to perform an in-depth analysis on 43,777 datasets. Out of all searched datasets, 4,843 datasets contain provenance related meta information, which is about 11% of all datasets. This shows that metadata vocabularies already play an important role in existing LOD datasets. Within the metadata-carrying datasets on average 9% of all triples are metadata triples. In the datasets, where PROV-O vocabulary was used, the PROV-O related information covers on average 62% of all the dataset triples. The most popular predicate is *prov:wasDerivedFrom* with more than 128 million occurrences, followed by *dc:language* (94 million), *dc:title* (63 million), *dc:rights* (48 million). The properties describing provenance, rights, and language are the most relevant metadata predicates so far. Furthermore, we were able to find 7,088 datasets, which use more dataset-centric vocabularies and if the Data Cube vocabulary is included, we found about 14,767 matches. When only datasets with DCAT and VoID are evaluated, on average nearly 28% of all triples are metadata triples. This is not a surprise because many of these datasets describe other datasets. The top predicates in this metadata category are: *cube:Dataset* (840 million), *void:vocabulary* (109 million), and *dcat:distribution* (84 million). We also

[7]https://datahub.io/
[8]http://ckan.org/
[9]http://lodlaundromat.org/
[10]http://aksw.org/Projects/LODStats.html
[11]https://www.w3.org/TR/prov-o/
[12]http://dublincore.org/documents/dc-rdf/
[13]https://www.w3.org/TR/vocab-dcat/
[14]https://www.w3.org/TR/void/
[15]https://www.w3.org/TR/vocab-data-cube/
[16]http://lodvader.aksw.org/

analyzed the datasets for URI identifiers, which are specific for MRMs. We were not able to find any dataset using the Singleton Property MRM. When it comes to RDF reification, we did not find any dataset, which uses *rdf:Statement*. Looking for the standard reification predicates, allowed us to identify 15 datasets, where the number of occurrences of *rdf:subject*, *rdf:predicate*, and *rdf:object* differed, which indicates wrong usage of standard reification or inconsistencies in the datasets or indexes.

In addition, we requested help via the Semantic Web[17] and PROV-O[18] mailing lists for known metadata datasets or projects, where management and storage of RDF metadata is an essential part. This search pointed us towards *Nano-publications*, *Bio2RDF*, and *OpenCitations*. Overall, the search strategy and metadata vocabulary analysis helped us to find relevant datasets and it showed us that metadata is already used within the Linked Data community, since more than 11% of all datasets contain metadata information.

3.1.2. Metadata datasets analysis

From the dataset candidates returned by our search strategies, we selected exemplary datasets with more than 5 million triples for a deeper analysis. These datasets are:

Yago 3: is a prominent knowledge base extracted from Wikipedia and other sources [28]. It stores metadata and provenance information per triple using a non-standardized way of assigning triple-ids via turtle comments. The ids are associated with metadata in the same way as in the other MRMs. While there is a source URL and extraction technique recorded for almost every triple, metadata from other dimensions (e.g., geo location, time) is only available for a very small subset of triples.

Artists Knowledge Graph: A knowledge graph, containing fused data about 161,465 artists from 4 sources [16]. For every statement, a provenance object containing detailed traceability metadata is recorded, using a role object[19]. If the same value for one entity attribute occurs in several datasets, multiple provenance objects are attached to it.

Bio2RDF: Within the Bio2RDF [9] project various datasets from the Life Sciences have been converted to RDF. More than 30 datasets contain simple provenance information, which describe each graph. A graph contains all triples from one source. The number of different graphs for the majority of datasets is very small.

LinkLion: is a database for *owl:sameAs* links between entities [33]. In this dataset standard reification is used, to represent which linksets support a specific *sameAs* link. Furthermore, provenance information such as dump time and the used extraction algorithm name are provided.

LinkedGeoData: is a mapping of relational data from OpenStreetMap[20] to RDF. It provides revision metadata information for every node such as version_number, user_id, timestamp, changeset_id [45].

Linked Clinical Trials: translates XML export files of clinical trials to RDF [21]. For some entities (e.g., facility, drug, condition, address, state, and person) the XML document provenance information is kept.

Nano-publications: is a data model, which can be represented in cascaded RDF graphs, one container graph for the nano-publication and three additional graphs for the fact (assertion), provenance and publication information (meta-metadata))[17]. Nano-publications are primarily used in the Life Sciences. One big dataset[21] consists of 204 million associations between gene and disease concepts. For each of these relations the percentile rank of the match score is stored in combination with other provenance information such as *prov:wasDerivedFrom* and *prov:wasGeneratedBy*. In addition, a nano-publication dataset stores metadata about license, right-holder, authors, and creation date. For the checked datasets, the metadata is not diverse, i.e., many/all nano-publications share the same authors.

Open Citations Corpus: The Open Citations Corpus [38] contains information about the author-created bibliographic references present in publications that cite other publications. It consists of around a million cited bibliographic resources with a total number of more than 1.2 million citation links[22]. For bibliographic entities, provenance, and versioning metadata (changes between versions, which agent changed the version and source information) is tracked [39].

Wikidata: Since Wikidata is used as one of the evaluation datasets, a detailed explanation will be given in Section 5. For the sake of completeness of the metadata analysis, we would like to mention that Wiki-

[17] https://lists.w3.org/Archives/Public/semantic-web/2016Sep/0034.html

[18] https://lists.w3.org/Archives/Public/public-prov-comments/2016Sep/0000.html

[19] http://blog.schema.org/2014/06/introducing-role.html

[20] https://www.openstreetmap.org/

[21] http://datadryad.org/resource/doi:10.5061/dryad.gn219

[22] http://rawgit.com/essepuntato/opencitations/master/paper/occ-driftalod2016.html

data is characterized by diverse metadata for individual facts, whereas more than one third of the metadata is dedicated to a temporal dimension.

3.2. *Evaluation dimensions and requirements*

In the following section, we describe the evaluation dimensions and deduce requirements towards our evaluation to study different effects and aspects of these dimensions with regard to MRMs, but also to cover metadata handling in RDF from various angles. With respect to the latter and to close the gap to previous work, an objective of this evaluation is to also examine vendor-specific metadata extensions in large-scale capable, SPARQL 1.1 compliant RDF stores.

During our investigation and based on previous work in RDF store benchmarking, we identified three main dimensions which are influencing the performance of MRMs in RDF stores:

3.2.1. *Dataset characteristics*

Each dataset differs in features such as in/out degrees of entities, property, and value distributions. These dataset characteristics have a direct impact on different evaluation metrics such as the dataset size and loading time. The datasets which are used in this evaluation should contain real-world data, since a realistic combination of these different features is difficult to reproduce in a synthetic dataset. Furthermore, the dataset should contain diverse and not just repetitive data (e.g., different data sources, annotators, dates) and cover different metadata types (e.g., date, provenance) which were seen in the metadata usage analysis. The dataset should be big enough, in order to stress the graph backend. These requirements shall ensure that the amount of (meta)data allows us to create and execute diverse search queries, for which the whole dataset can not be cached by the backend. In nanopublications we observed the concept of meta-metadata. The effect of such nested metadata should be addressed as well.

3.2.2. *Granularity of metadata*

While the dataset characteristics also include features of metadata, from our perspective the granularity level on which metadata is expressed should be considered separately. In the dataset analysis, we observed meta information which describes the whole dataset or huge parts of it, but also more fine-grained metadata for individual triples or entities. In the context of this evaluation, we defined three granularity levels:

Dataset/Graph level: it provides information for all entities and statements within the same dataset/graph. For a dataset like DBpedia, this meta information can be information about the source from which data was extracted, e.g., Wikipedia version or Wikipedia chapter language.

Entity/Resource level: level where all statements about one entity share the same meta information. If facts about an entity from a Wikipedia page are extracted, then the meta information of the Wikipedia page can be used as meta information for all the extracted statements/triples of this entity such as revision information, publication date, number of edits, etc.

Triple/Statement level: level where metadata is expressed for each statement or triple. When dealing with information from different DBpedia language datasets, it is possible that two or more data sources share information about the same real-world entities. In this scenario, it is required to store metadata at a triple/statement level, e.g., to track the origin of every individual fact.

Considering our knowledge graph and fusion use case, we were only interested in datasets which store metadata at the entity or triple level. When fusing data from different datasets into a new dataset, the source information can not be tracked using only dataset level granularity metadata. An MRM evaluation should consider both entity and triple level granularity.

In connection with metadata granularity, the support of an MRM for factorization should be examined in more detail. As factorization, we denote the capability of an MRM to represent the granularity level of metadata in an efficient way on the storage level without metadata redundancy.

3.2.3. *Query characteristics*

Besides the compatibility problem of existing data queries without rewrite, the question of the overhead of MRM usage for data-only queries remains to be answered. Therefore, an MRM evaluation should consider both data-only queries and also mixed (data-metadata) queries. Furthermore, a thorough evaluation should use queries of different complexity classes, ranging from simple to very complex queries. Finally, the complexity and structure of MRM metadata queries should be studied in terms of usability for adopters but also potential effects for RDF store query optimizers.

3.3. *Evaluation Criteria*

Based on the analysis steps and the presented requirements and dimensions, we deduced the following criteria and give metrics for the evaluation, whenever possible.

3.3.1. *Criteria for metadata representations*

Storage cost - evaluates the size of the MRM and its factorization support and will be measured by triple count, as well as serialized file size and overall database size in byte.

Data-only query overhead/impact - Our evaluation measures query time in ms for data-only queries for the MRMs compared to a baseline query/-dataset without MRM specific triples.

Mixed (metadata and data) query execution time - for a set of query templates over data and metadata we compare the execution time (in ms)

Usability - We compare the number of variables, triple patterns, and additional SPARQL elements, which are necessary to query a single triple with a metadata fact, as indicators for the query usability/complexity of an MRM.

3.3.2. *Criteria for metadata extension and SPARQL implementations*

Bulk load - evaluates (meta)data bulk loading capacities for stores and is measured in milliseconds.

SPARQL integration/conformance - evaluates integration of store-specific metadata extensions into SPARQL and validates whether stores are able to handle all SPARQL queries correctly

3.3.3. *Additional criteria*

Backward compatibility of queries - evaluates data-only queries, which should still work after the addition of metadata without the need to rewrite them.

4. Metadata Representation Models

Within the Linked Data community different ways of representing metadata have been developed. The most common MRMs were described in [23] and [22]. Besides two additional MRMs, not evaluated in the literature so far, this work will study the same list of MRMs. In this section, we will present five RDF-compliant MRMs and we will have a brief discussion about native metadata support by graph backends. Figure 1 visualizes the major differences between MRMs and can be used as a visual reference.

4.1. *RDF compliant models*

In order to make it easier for the reader to understand the differences between the MRMs, a running example is used. In the example, two entities are shown with birth year values for *Person p1* and *p2* using the *dbo:birthYear* attribute. For each RDF statement metadata about the last modification date (*dc:modified*) exists. The presented query searches for the most current birth date for each distinct person. The abbreviated names, shown in Figure 1, are used in the upcoming sections, to indicate that the specific realization of an MRM (as displayed in the figure) is meant. This convention allows, e.g., to distinguish between the RDF 1.1 feature "named graphs" and the MRM ngraphs, which utilizes the former.

4.1.1. *Named graphs (ngraphs)*

The *Named Graph* feature, which is supported by every SPARQL 1.1 compliant graph backend, allows for the assignment of one IRI for one or more triples as a graph id. The same IRI can then be used as a subject for a metadata entity, which itself can store the metadata about the associated triple(s) as predicates and objects.

```
# data with birth year values for
# Person p1 and p2 (trig notation)
 <g1> { <p1> dbo:birthYear "1981". }
 <g2> { <p1> dbo:birthYear "1983". }
 <g3> { <p2> dbo:birthYear "1982". }

# metadata:
meta:dbpedia { <g1> dc:modified "2016-11"^^xsd:gYearMonth .
               <g2> dc:modified "2014-12"^^xsd:gYearMonth .
               <g3> dc:modified "2012-01"^^xsd:gYearMonth . }

# query
SELECT ?person ?birth WHERE {
    { GRAPH ?g {?person dbo:birthYear ?birth } .
      GRAPH meta:dbpedia {?g dc:modified ?modified}
    } FILTER NOT EXISTS {
        GRAPH ?g2 {?person2 dbo:birthYear ?birth2 }
        GRAPH meta:dbpedia {?g2 dc:modified ?modified2 }
        FILTER ( ?person = ?person2 && ?modified2 > ?modified )
    }
}
```

The *ngraphs* MRM is easy to understand, since it just builds on top of the existing triple format. Hence, it is possible to reuse existing data queries. In addition, the representation is very compact (for both data and query), which can be seen in the example, the metadata can be reached by only adding the `GRAPH` statement and one additional triple into the search query. Another big advantage of this approach is the fact, that the *ngraphs* MRM supports factorization for all granularity levels. The same metadata IRI can be reused at dataset, entity and triple-level. The one big drawback of the *ngraphs* model is the fact that it uses the *named graph* IRI as a URI for a metadata resource, which itself stores a set of key-value based metadata facts. If the original dataset uses a *named graph* to store the data facts, then a *ngraphs* MRM can not be applied in a backward compatible way.

4.1.2. RDF standard reification (stdreif)

As specified in the RDF standard, it is possible to create a resource which describes a triple and its *subject*, *predicate*, and *object*. The resource IRI can then be used to connect provenance or meta information with the triple.

```
# birth year values for
# Person p1 and p2 (turtle notation)
 <stmt-56e8> rdf:type rdf:Statement ;
             rdf:subject <p1> ;
             rdf:predicate  dbo:birthYear ;
             rdf:object "1981" .
 <stmt-4f83> rdf:type rdf:Statement ;
             rdf:subject <p1> ;
             rdf:predicate  dbo:birthYear ;
             rdf:object "1983" .
 <stmt-4327> rdf:type rdf:Statement ;
             rdf:subject <p2> ;
             rdf:predicate  dbo:birthYear ;
             rdf:object "1982" .

# metadata:
 <stmt-56e8> dc:modified "2016-11"^^xsd:gYearMonth .
 <stmt-4f83> dc:modified "2014-12"^^xsd:gYearMonth .
 <stmt-4327> dc:modified "2012-01"^^xsd:gYearMonth .

# query
SELECT ?person ?birth WHERE {
    { {?g rdf:subject ?person; rdf:predicate dbo:birthYear;
          rdf:object ?birth ; a rdf:Statement . } .
      {?g dc:modified ?modified . }
    } FILTER NOT EXISTS {
        {?g2 rdf:subject ?person2; rdf:predicate dbo:birthYear;
             rdf:object ?birth2 ; a rdf:Statement . } .
        {?g2 dc:modified ?modified2 }
        FILTER ( ?person = ?person2 && ?modified2 > ?modified )
    }
}
```

Compared to the *ngraphs* MRM, it is not possible to reuse existing data queries out of the box. In order for them to work, a custom reasoning mechanism would have to be applied. Furthermore, each original data triple has to be represented by a resource entity, which itself consists of four statements (*rdf:subject*, *rdf:predicate*, *rdf:object*, and *rdf:Statement*). This does not only increase the dataset size, but adds more triple patterns to queries. All four components of a reified resource have to be used as triple patterns in order to find the correct reified triple in the dataset. The resource IRI can be used to access data and metadata. On the positive side it supports datasets, which use named graphs.

4.1.3. N-ary relation (naryrel)

In this MRM, a relationship instance is created as a resource of the *subject-predicate*-pair instead of the *object* of the triple. The *object* is connected to the relationship resource, using a renamed version of the *predicate* (appending a designated suffix). The same relationship resource is utilized, to relate meta information to the statement.[23]

```
# birthyear values for
#Person p1 and p2 (turtle notation)
 <p1> dbo:birthYear <rel-56e8> .
 <rel-56e8> dbo:birthYear-value "1981" .
 <p1> dbo:birthYear <rel-4f83> .
 <rel-4f83> dbo:birthYear-value "1983" .
 <p2> dbo:birthYear <rel-4327> .
 <rel-4327> dbo:birthYear-value "1982" .

# metadata:
 <rel-56e8> dc:modified "2016-11"^^xsd:gYearMonth .
 <rel-4f83> dc:modified "2014-12"^^xsd:gYearMonth .
 <rel-4327> dc:modified "2012-01"^^xsd:gYearMonth .

# query
SELECT ?person ?birth WHERE {
    { { ?person dbo:birthYear ?g. ?g dbo:birthYear-value ?birth . } .
      {?g dc:modified ?modified . }
    } FILTER NOT EXISTS {
        { ?person2 dbo:birthYear ?g2.
          ?g2 dbo:birthYear-value ?birth2 . } .
        {?g2 dc:modified ?modified2 }
        FILTER ( ?person = ?person2 && ?modified2 > ?modified )
    }
}
```

Similar to the *standard reification*, an IRI can be used to access data and metadata. In the case of the *nary-relation* MRM, the IRI is a relation resource IRI. Due to the introduction of the relation resource IRI, one more statement per triple value has to be added. Existing data queries cannot be reused, but compared to the *standard reification* fewer triple patterns are required to access either data values or metadata. Furthermore, it is possible to support datasets with named graphs.

4.1.4. Singleton Property (sgprop)

The singleton property [34] scheme uses a unique property for every triple with associated metadata. This unique property deals as a triple identifier, which can be used to describe the statement with meta information. In order to be able to reconstruct the original property of the statement, every singleton property is linked to its original predicate using a *rdf:singletonPropertyOf* relationship.

```
# birth year values for
# Person p1 and p2 (turtle notation)
 <p1> <56e8-783f-9cd3> "1981" .
 <p1> <4f83-6cd5-88da> "1983" .
 <p2> <4327-367e-439b> "1982" .

# property reconstruction information
 <56e8-783f-9cd3> rdf:singletonPropertyOf dbo:birthYear .
 <4f83-6cd5-88da> rdf:singletonPropertyOf dbo:birthYear .
 <4327-367e-439b> rdf:singletonPropertyOf dbo:birthYear .

# metadata:
 <56e8-783f-9cd3> dc:modified "2016-11"^^xsd:gYearMonth .
 <4f83-6cd5-88da> dc:modified "2014-12"^^xsd:gYearMonth .
 <4327-367e-439b> dc:modified "2012-01"^^xsd:gYearMonth .

# query
 SELECT ?person ?birth WHERE {
```

[23] https://www.w3.org/TR/swbp-n-aryRelations/

```
    { {?person ?g ?birth .
       ?g rdf:singletonPropertyOf dbo:birthYear .} .
      {?g dc:modified ?modified . }
    } FILTER NOT EXISTS {
        {?person2 ?g2 ?birth2.
         ?g2 rdf:singletonPropertyOf dbo:birthYear. }
        {?g2 dc:modified ?modified2 }
        FILTER ( ?person = ?person2 && ?modified2 > ?modified )
    }
}
```

This unique property can be seen as a predicate resource IRI. The predicate resource IRI can be used to access metadata and the original predicate type. Since the predicate resource has the *rdf:singletonPropertyOf* relation, it is possible to use RDFS entailment rules to infer the original statements. When looking at the dataset triples, it is not possible to deduce the direct meaning of a triple predicate. It is always required to use the predicate resource IRI to find the associated *rdf:singletonPropertyOf* property and with it the original predicate. Furthermore, it is possible to support datasets with named graphs.

4.1.5. Companion Properties (cpprop)

As was shown in [22], the *singleton property* representation model suffers from the fact that it creates a new property for every statement in order to create globally unique properties. This results in a very uncommon uniform distribution and large number of properties, and therefore, in increased query times. To limit the creation of new properties and to reduce the influence on the datasets property (frequency) distribution, we propose a novel MRM - *companion properties*.

The *companion properties* representation is inspired by sgprop but uses a fixed naming scheme to create a property, which is unique with respect to the subject of the statement. In order to support the naming scheme, an occurrence counter can be utilized when creating the new dataset. An individual occurrence counter is used per property p for its subject s. The occurrence count is appended as a suffix to every instance of p. Depending on the used profile, every generated property cp has at least one companion property, which is from a graph theoretic view a sibling of p with respect to s. The IRI of this companion property consists of the IRI of cp plus the additional suffix ".SID". For each subject and companion property pair, a statement ID is created which serves as a unique metadata resource identifier for the triple $s\ cp\ o$.

The number of different companion property names is bound by $\sum_{p \in D}(maxOut(p) \cdot 2)$, where $maxOut(p)$ is the maximum number of edges per resource for the property p in the dataset D. When merging two datasets which use this MRM, the naming scheme should include a dataset specific prefix for the counter values, in order to avoid name collisions. Analogous to a singleton property, RDFS entailment rules can be used to infer the original statements.

```
# birth year values for
# Person p1 and p2 (turtle notation)
 <p1> dbo:birthYear.1 "1981" ;
      dbo:birthYear.1.SID <sid-56e8> .
 <p1> dbo:birthYear.2 "1983" ;
      dbo:birthYear.2.SID <sid-4f83> .
 <p2> dbo:birthYear.1 "1982" ;
      dbo:birthYear.1.SID <sid-4327> .

# property reconstruction and SID association vocabulary
 dbo:birthYear.1.SID rdf:idPropertyOf dbo:birthYear.1 .
 dbo:birthYear.1 rdf:companionPropertyOf dbo:birthYear .
 dbo:birthYear.2.SID rdf:idPropertyOf dbo:birthYear.2 .
 dbo:birthYear.2 rdf:companionPropertyOf dbo:birthYear .

# metadata:
 <sid-56e8> dc:modified "2016-11"^^xsd:gYearMonth .
 <sid-4f83> dc:modified "2014-12"^^xsd:gYearMonth .
 <sid-4327> dc:modified "2012-01"^^xsd:gYearMonth .

# inference "rule"
 rdf:companionPropertyOf rdfs:subPropertyOf rdfs:subPropertyOf .

# query
SELECT ?person ?birth WHERE {
    { { ?person ?cp ?birth; ?cpid ?g.
        ?cp  rdf:companionPropertyOf dbo:birthYear.
        ?cpid rdf:idPropertyOf ?cp .  } .
      {?g dc:modified ?modified . }
    } FILTER NOT EXISTS {
       { ?person2 ?cp2 ?birth2; ?cpid2 ?g2.
         ?cp2  rdf:companionPropertyOf dbo:birthYear.
         ?cpid2 rdf:idPropertyOf ?cp2.  } .
        {?g2 dc:modified ?modified2 }
        FILTER ( ?person = ?person2 && ?modified2 > ?modified )
    }
}
```

In contrast to the other triple based MRMs, the identifiers are not required for reconstructing the original triple. Thus it allows, likewise to ngraphs, for sharing of statement identifiers, and additionally multiple identifiers per statement. This enables companion properties to support different granularity levels.

4.2. Vendor Specific models

Some of the database vendors for RDF stores have recognized and discussed the potential of adding a custom support for metadata on statement level or a native way of using statement identifiers in their backends. We investigated the online documentation of several major RDF stores and asked their vendors whether they are planning or already implemented a specific support for metadata. To the best of our knowledge, we provide a short overview of the current state at the time of writing this paper.

4.2.1. *Blazegraph*[24]

The Java-based graph store Blazegraph offers a feature called *Reification Done Right*[25]. It provides an implementation of *SPARQL** and *RDF**. These formalized syntactic extensions [19] of the SPARQL and the Turtle grammar allow to directly associate and query triples with metadata. The existence of a translation to Turtle and SPARQL ensures backward compatibility and provides formal semantics.

Reification syntax - RDF*: To import and bulk load reified RDF data, two additional file formats based on N-Triples and Turtle have been introduced. The extensions allows a statement to occur as subject and/or object of another statement by enclosing it with double angle brackets. «`:Bob foaf:age 23`» `dc:source :Joe`. RDF* even supports nested reified statements to represent meta-metadata. Unfortunately, there is one major limitation caused by the translation into RDF. Multiple reifications of the same triple are translated into one standard W3C reification *rdf:Statement*, and therefore, it is not possible to distinguish grouped annotations (e.g., when a confidence score and the tool which produced the data and the score, are stored as individual values, the confidence values only make sense in the scope of the tool) anymore. One workaround for this issue is explained in Section 6.3.

SPARQL reification extension - SPARQL*: Using SPARQL*, it is possible to bind a whole statement or a statement pattern in RDF* syntax (which can contain variables) to a variable. This variable can be used as subject or object of another SPARQL triple pattern. An example is shown below.

```
SELECT ?age ?src WHERE {
    ?bob foaf:name "Bob" .
    BIND( <<?bob foaf:age ?age>> AS ?t ) .
    ?t dct:source ?src .
}
```

Besides the support for *CONSTRUCT* and *DESCRIBE*, Blazegraph allows data mutation for reified triples using *UPDATE* and *INSERT* queries.

RDR Implementation: The reified statement is embedded directly into the representation of each statement about that reified statement. This is achieved by using indices with variable lengths and recursively embedded encodings of the *subject* and *object* of a statement.

[24] https://www.Blazegraph.com/product/

[25] https://wiki.Blazegraph.com/wiki/index.php/Reification_Done_Right

4.2.2. *Virtuoso*[26]

To the best of our knowledge, the Virtuoso backend does not have extensions for handling metadata use cases. The community and an OpenLink employee have discussed possible extensions [27] [28], but no additional reification feature has been added to Virtuoso yet. In order to use Virtuoso for provenance and metadata scenarios, the RDF-compatible MRMs have to be utilized.

4.2.3. *Others*

Other RDF store providers have created extensions for storing and retrieving metadata more efficiently. AllegroGraph[29] supports the handling of metadata with its *Direct Reification* feature, which uses statement identifiers. Stardog[30] allows for the support of metadata by introducing a statement identifier, which is also used to support property graphs. Both systems provide a proprietary SPARQL extension to query a statement identifier. Unfortunately, at the time of writing this work, none of the systems provide a way to bulk load data making use of statement identifiers. Aside from that, AllegroGraph supports storage and bulk loading of JSON based so-called triple attributes. The fact, that the attributes have to be defined in a schema before they can be loaded, and the missing SPARQL integration limit the usage of this extension.

5. Evaluation Datasets with Metadata

After the review of the available metadata datasets, we decided based on our requirements to use the following datasets for the evaluation:

Wikidata: Wikidata is no native RDF dataset, it instead uses its own Wikidata Statement Model [10]. Claims (which are similar to triples/statements in RDF) can be described with so-called qualifiers consisting of keys and values (analogous statement level metadata in RDF). Qualifiers are used to provide a context or scope for a claim (e.g., how long the *marriedTo*-relation between two persons is valid). In contrast to this factual metadata, there also exists the concept of references, which records provenance for claims.

[26] http://Virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/

[27] https://lists.w3.org/Archives/Public/public-lod/2010Oct/0094.html

[28] http://www.openlinksw.com/weblog/oerling/?id=1572

[29] http://franz.com/agraph/allegrograph/

[30] http://Stardog.com/

In order to reproduce and compare the work from [22], we reused the same Wikidata dump with time, geospatial and source/reference metadata for a part of the claims. The converted RDF dataset based on 2016/01/04 JSON dump contains over 81 million claims, describing around 16 million entities (out of 19 million entities in total) and using 1600 different properties. Since metadata is modelled on statement level, a statement id is kept for every claim (data triple), but 1.5 million claims have qualifiers, only. The 2.1 million qualifiers are based on 953 distinct qualifier keys (denoted as metadata key in Figure 1). In the excerpt below, a shortened example of two different claims with metadata about the presidency of Grover Cleveland is given. Furthermore, the example illustrates the special case, where the same claim occurs more than once, but with different metadata. For a more detailed description of the data, we refer to [22].

```
# 2 Wikidata claims about Grover Cleveland (TriG notation)
<claim1> {
   <claim1> rdf:type wb:Statement .
  # Grover Cleveland held position President of U.S.
   wde:Q35171 wdp:P39 wde:Q11696 .
  # start time qualifier of presidency (claim metadata)
   <claim1> wdp:P580 wdq:abb3c2c8a00 .
  # start time qualifier value object
   wdq:abb3c2c8a00 rdf:type wb:TimeValue ;
     wb:timeValue "+1893-03-04T00:00:00Z"^^xsd:dateTime ;
     wb:timeTimeZone "0"^^xsd:integer ;
     wb:timePrecision "11"^^xsd:integer ;
     wb:timeCalendarModel wde:Q1985727 .
  # as 24th president  (claim metadata)
   <claim1> wdp:P1545 "24" .
}

<claim2> {
   <claim2> rdf:type wb:Statement .
  # Grover Cleveland held position President of U.S.
   wde:Q35171 wdp:P39 wde:Q11696 .
  # as 22nd president
   <claim2> wdp:P1545 "22" .
}
```

Wikipedia history and DBpedia: Since DBpedia data does not come with diverse metadata, we decided to apply the Wikipedia Revision history on top of a company focused dataset. This dataset allows for insights into the data evolution of a DBpedia entity, which is going to be part of future research projects.

In [14], a system has been presented, which can be used to create an RDF dataset with revision information for a Wikipedia chapter (e.g., French, German, or English). We adopted these scripts[31], to transform the revision metadata XML dumps, which are published every month on Wikipedia, into a Turtle representation. Additionally, the script writes metadata such as the corresponding DBpedia instance, the number of revisions per time frames (e.g., months, years), author information, and change dates, to the output turtle file.

Considering that on average more than 277 metadata revision statements exist per DBpedia entity, we decided not to use the complete DBpedia dataset. Therefore, we extracted data from the German and English DBpedia chapters about companies, their associated locations and persons. Focusing the dataset around companies and their related resources, helped to narrow down the dataset. Due to the different types of entity classes, this dataset still ensures a diverse distribution of entity relations within the graph. The reduced dataset contains more than 83 thousand entities (approx. 37,000 companies, 27,000 places, and 19,000 persons). Once extracted, the selected DBpedia resources were enriched with the resource revision meta information for the German and English Wikipedia chapter. The meta information comprises aggregated metadata based on all revisions of an article like the number of revisions (total, last 2 years/months), creation and last modification date, but also links to every Wikipedia revision for the article of the triples entity. In the dataset, a dedicated resource exists for every link, which contains additional information such as editor name and date of the revision. While a link to a revision remains stable, the aggregated metadata is dependent on a specific dump file created at a given point of time. We therefore save meta-metadata for this aggregated metadata in the form of a link to the used dump file. Note that storing provenance for the metadata requires reifying the metadata, too. In total over 23 million revisions are associated with the entities. More than 12,800 properties are used for data statements, but just 21 for metadata keys. The dataset is characterized by a 1:10 data/metadata ratio (10 metadata triples for a data triple) and 1:100 data/revision ratio (100 triples of revision information for one data triple). In contrast to the Wikidata dataset the number of metadata statements (948 million revision information statements and 94 million statements for aggregated metadata plus the links to the revisions) exceeds the number of data statements (9.7 million) by two orders of magnitude. This is motivated by use cases where traceability or provenance information make up a greater portion than the data itself. A fragment of the dataset which shows the full aggregated metadata for one entity, with incomplete data and revision links, is listed below.

```
# DBpedia dataset with revision metadata (trig notation)
# triples of one entity share same metadata
dbr:Ang_Lee-Statements {
  dbr:Ang_Lee
```

[31] https://github.com/dbpedia/Historic

```
      rdf:type  <http://dbpedia.org/ontology/Person> ;
      foaf:givenName  "Ang"@de ;
      foaf:surname  "Lee"@de ; # ...
      owl:sameAs  <http://dbpedia.org/resource/Ang_Lee> .
}

# aggregated metadata for entity
dbr:Ang_Lee-Meta {
  dbr:Ang_Lee-Statements
      dc:created  "2001-08-08T08:21:09"^^xsd:dateTime ;
      dc:modified  "2016-10-14T17:06:34"^^xsd:dateTime ;
      historic:uniqueContributorNb  235 ;
      <http://www.w3.org/2004/03/trix/wp-2/isVersion>  378 ;
      historic:revPerLastMonth1  1 ;
      historic:revPerLastMonth2  1 ;
      historic:revPerYear2016  10 ;
      historic:revPerYear2015  10 .
}

{# meta-metadata for aggregated metadata from above
  dbr:Ang_Lee-Meta
     historic:hasSource  historic:de_xml_16_11_14 .
 # links to revisions metadata resources
  dbr:Ang_Lee-Statements
     historic:hasMainRevision wiki:wiki/Ang_Lee;
     historic:hasOldRevision
        wikiv:index.php?title=Ang_Lee&oldid=10 , # ...
        wikiv:index.php?title=Ang_Lee&oldid=96519084 ,
        wikiv:index.php?title=Ang_Lee&oldid=9713203 .
}
```

6. Evaluation Setup

To allow a comparison of this work with respect to different evaluation hardware and setup, we reproduced the loading and the quin query pattern experiments from [22]. In addition, we measured the execution and loading times as well as the database sizes. For this evaluation, we created a DBpedia based dataset which uses Wikipedia revision information as metadata. Combining the results of all the experiments allows us to take a more detailed look on how different MRMs perform for different datasets and use cases.

The evaluation was executed on an Ubuntu 14.04.3 server system with a 3.19.0-33 kernel, Oracle Java 1.8.0_66, Ruby 2.2.5p319, a 1.8GHz Intel Xeon E5-2630L CPU, 256GB RAM and a 3.6TB hard disk drive. We used Blazegraph in version 2.1.2, Virtuoso 07.20.3215-pthreads and Stardog 4.2.3. The database configuration parameters for Virtuoso and Blazegraph were reused from [22]. According to the settings in [22], we use the recommended 6GB Java heap memory size for Blazegraph. Blazegraph relies on the file system cache to improve disk access and the relatively small memory footprint should keep interruptions by the GC at a minimum. Furthermore, we disabled swap as per recommendation of the Blazegraph performance guide[32]. For Virtuoso we used a buffer according to 32GB available RAM and additional 2GB for the query processor. The recommended settings (4GB heap and 8GB direct memory for 1 billion triples[33]) for Stardog were not sufficient for our dataset and lead to memory exceptions. We therefore set Java parameters to 32GB heap (for better comparison with Virtuoso) and 32GB as maximum direct memory.

According to our requirements for SPARQL integration and bulk loading we did not consider the vendor specific metadata extensions for Stardog and AllegroGraph due to their limitations. Moreover license restrictions prevent the evaluation of AllegroGraph in this work. Therefore we selected Blazegraph, Stardog (without testing the metadata extension) and Virtuoso for this evaluation.

6.1. Dataset Conversion

For the experiment which is described in [22], we reused and extended the existing conversion framework. However, due to its technical design we were not able to generate cpprop representation for the Wikidata dataset.3

To create the DBpedia based dataset, we developed a Java-based framework and command line utility[34], which allows us to apply various metadata representation formats on top of datasets. The framework [12] features a novel JSON representation, which allows for the association of metadata to quad(s) for different levels of granularity. Once the source dataset is converted into the novel JSON representation, this intermediate JSON-based dataset can be used to create test datasets, which use the different metadata representations. All datasets had been converted into several gzipped nquads (.ntx for rdr) files.

6.2. Evaluation Procedures

The evaluation is separated into two parts. First, datasets are loaded, and the following metrics are measured: loading time, database size, statement count. For every MRM and dataset, we created an isolated new database instance to prevent side effects. Once all the data is loaded, the second part, the query execution, is started. For each query template, we restart the backend and clear the cache. Then we run all instances of one query template sequentially.

[32]https://wiki.Blazegraph.com/wiki/index.php/PerformanceOptimization

[33]https://www.stardog.com/docs/#_memory_usage

[34]https://github.com/JJ-Author/meta-rdf

In order to execute the queries, we adapted[35] the framework, which is described in [22]. We measure both, the execution times and the loading times, by retrieving a Ruby time stamp before sending the query, respectively executing the bulk load command, and after the execution is finished. The database sizes were measured with the UNIX `ls` command. We use both a client side and a database internal timeout. The former one is used as a fallback, in case the database timeout did not trigger. For Wikidata, we kept the database timeout of 60 seconds and a client timeout of 120 seconds. As we aim to evaluate also more challenging queries within the DBpedia scenario, we have chosen timeouts of 240 seconds and 400 seconds, respectively.

6.3. Wikidata Scenario

For this work, we extended the scenario, which is described in [22], by evaluating the Blazegraph feature RDR, which had not been studied for the Wikidata use case yet. In order to circumvent the limitation of RDR, that every triple needs to be unique, we do not attach the metadata directly to the data triple. Instead, we use (multiple) statement identifiers as metadata, which are then linked to the actual metadata. This is necessary, to model the Grover Cleveland example shown in Section 5. The technique is illustrated in the following example:

```
<< :s :p :o >> :hasMeta :id1 ;
               :hasMeta :id2 .
```

6.3.1. Wikidata Loading Procedure

When we looked at the original experiment webpage[36], we were surprised about the huge loading times of Blazegraph (e.g., more than 66 hours for ngraphs) in contrast to Virtuoso (4 hours). We repeated loading the data for *ngraphs* and *rdr* with the same database configurations and measured 76 and 80 hours on our machine, respectively. We could identify two minor issues in the original setup, which were causing this huge difference. First the original Wikidata dataset uses a non-RDF compliant encoding of dates. When loading the data into the RDF backend, various warnings are recorded in the log-files, which decreased the insertion throughput. We therefore converted the dates into the appropriate format and repeated measuring the loading time and achieved 47 hours for rdr and 44 hours for ngraphs. Furthermore, we switched the commit process from an incremental commit to a batch commit. This guarantees a fair comparison to Virtuoso, where indexing and commits have been disabled for the bulk loading procedure. This change resulted in an additional speedup and also reduced the database size. The results will be discussed in section 7.

6.3.2. Wikidata Templates (Quins Experiment)

A quin represents a data-metadata look-up query, where for a data triple pattern `s,p,o` the attached metadata key `k` and its corresponding values `v` are queried. For this quin pattern `(s,p,o,k,v)` the authors defined 31 templates based on 31 binary masks of length 5 (from `(0,0,0,0,1)` to `(1,1,1,1,1)`), which define whether the corresponding position of the quin deals as a constant or as a variable in the query. For example the mask `(1,1,1,0,0)` generates queries retrieving all the metadata information for one specific triple (`s,p,o` are constants and `k,v` are variables). Every template has 300 query instantiations, whereas for every template the same pool of 300 randomly sampled quin instances is used for the constants in the query. The query instances are translated into the respective representations of the *ngraphs*, *naryrel*, *sgprop*, *stdreif*, and *rdr* formats. In addition, we replaced the triple pattern (`?p a wikibase:Property`) in the queries with an equivalent FILTER EXISTS statement. We studied the runtime behavior of this optimziation for *ngrahps* and *rdr* which are referred to as *fngraphs* and *frdr*. While this just slightly improves Virtuoso's query execution performance, for Blazegraph various queries do not time out anymore (e.g., *fngraphs*). The performance improvement can be explained with a different query plan, since due to the FILTER statement one join could be omitted. Detailed results are presented on the experiment description website[37].

6.4. DBpedia Scenario

For the DBpedia based dataset, we used a slightly different setup for loading and created a new set of queries, which we will introduce in this section.

6.4.1. DBpedia Loading Procedure

For the DBpedia datasets, we started pre-loading the revisions metadata first. This part of the metadata is independent of the used MRM. We then replicated this database to load the MRM-specific parts of the dataset.

[35]https://github.com/JJ-Author/wikibase-bench

[36]http://users.dcc.uchile.cl/~dhernand/wquery/#results

[37]http://vmdbpedia.informatik.uni-leipzig.de:8088/frey/meta-evaluation/

Table 1

Query templates for DBpedia dataset: DBQ-SIM-01 to DBQ-HAR-02 are data query templates to study the impact of the MRMs for data-only queries. The letter X represents the template variable, which is replaced with an existing constant from the dataset during query instantiation. Every mixed query DBM is an extension of its respective DBQ query by taking additional metadata for selected triple patterns of the query into account.

	Data Query DBQ	**Mixed Query DBM**
SIM-01	Show all properties and objects of the Person X.	. . . as well as the information when the triple (p-o pair) was created.
SIM-02	Show all cities and its population which are located in country X and having more than 20,000 inhabitants.	. . . and additionally the provenance (last wikipedia revision url) information for the city
MED-01	Show properties and its according objects of an entity X, which match exactly with the same entity (owl:sameAs) of another language version of DBpedia.	. . . plus the number of unique contributors for the used sameAs link as confidence indicator.
MED-02	Count (distinct) all companies of a country X.	. . . and the number of revisions in 2016 for the location information (as up-to-dateness indicator) and also the number of contributors for the company type information of the entity
HAR-01	Find potentially matching companies within an industry sector X, having at least one exact match of the OPTIONAL properties label, city and country.	. . . furthermore the modification dates of the sector information of the entities
HAR-02	Find Entities which are associated with a specific geographical region X. The association is defined in that sense, that the entity has an outgoing edge to a Place p, whereas p (or another subject which is the same as p) is part of that region.	. . . and the provenance for the type information.

Unfortunately, we could use this strategy for the triple based MRM, only. Blazegraph and Stardog use a different indexing for rdr and ngraphs, which does not allow to use the triple-based pre-loaded database. For virtuoso, it was possible to reuse the database for ngraphs.

6.4.2. DBpedia Query Templates

In order to allow for a meaningful comparison between the different MRMs, we need to define a set of queries which cover different query types and complexities. As in [22], we also use query templates. But instead of applying systematic pattern based generation, we define a set of individual and heterogeneous templates (based on one template variable), which differ in complexity, the number of triple patterns and used SPARQL features. Defining criteria to measure the complexity of a query independent of the dataset is not trivial. The selectivity of a query is influenced by the query structure itself, but it is also a function of the dataset characteristics. The authors in [37] have studied complexity classes of different SPARQL elements. The conclusions are summarized in three lemmas, whereas Lemma 2 and 3 help to judge complexity independent of dataset characteristics.

Lemma 1: the evaluation of SPARQL queries with AND and FILTERS depends on the size of the dataset (D) and the number of triple patterns (P) in the query and is in $\mathcal{O}(P \cdot D)$.

Lemma 2: establishes that UNIONs between non union-compatible basic graph patterns (BGPs) are *NP-complete*, where two BGPs are union compatible if they share variables.

Lemma 3: states that OPTIONAL values increase the complexity and are *PSPACE-complete*.

Based on this work, we define three complexity classes for the SPARQL templates. The SIMPLE class is characterized by queries with a low number of triple patterns while the MEDIUM class applies for queries containing a large number of triple patterns. Queries consisting of more than one non union-compatible UNION or more than one OPTIONAL are classified as HARD. As outlined in the introduction, knowledge fusion is an important use case if multiple, overlapping graphs are combined in one knowledge graph. Inspired by this use case, we defined two templates per class. We choose randomly a set of 40 instances from our DBpedia evaluation dataset to populate the templates.

In order to measure the overhead of an MRM, when executing data queries, we have created two versions for each template. The first template version executes queries over the data only and the second query template over the data and the metadata. In the results section, these queries are denoted as *data* (DBQ) and *mixed* queries (DBM) respectively. A description of the used data and mixed patterns can be seen in Table 1. We refer to the experiment website for the SPARQL syntax of the used templates and query instances.

7. Evaluation Results

This section presents the results of the qualitative and quantitative MRM comparison as well as the Wikidata and DBpedia experiments, which will be followed by a general discussion about findings.

Table 2

MRM Comparison (factorization, backward compatibility & usability): The level of factorization of each MRM is measured for a set of 100 DBpedia entities and its revision metadata (95,864 meta facts, 950 on avg. per entity). Furthermore, the query complexity between different MRMs is compared. Note: The first number in the statements count column of rdr MRMs corresponds with the number of rdr (nested) statements in the database, whereas the second represents the number of triples, obtained by unnesting the rdr statements. The file format for rdr is `.ntx`, an extension of N-Triples.

	# statements	.nq file size (MB)	backward comp.	#(triple patterns)	#(overhead variables) & #(sparql elements)
raw data	8,444	1.16	/	/	/
cpprop	115,822	19.76	yes - w/ rdfs reasoning	5	3
naryrel	11,202,942	2161.53	no	3	2 & 1 BIND + 3 string-functions
naryrel (shared)	122,812	21.36	no	4	3 & 1 BIND + 3 string-functions
ngraphs	104,308	19.34	quads: no / triples: yes	2	1 & 1 GRAPH
rdr	11,126,845 / 22,169,250	2856.60 (.ntx)	quads: no / triples: yes	2	1 & 1 BIND
rdr (shared)	246,947 / 314,499	44.55 (.ntx)	quads: no / triples: yes	3	2 & 1 BIND
sgprop	11,202,942	2193.75	yes - w/ rdfs reasoning	3	1
sgprop (shared)	122,812	21.52	yes - w/ rdfs reasoning	4	2
stdreif	11,354,934	2188.17	yes - w/ custom reasoning	5	1
stdreif (shared)	141,316	24.63	yes - w/ custom reasoning	6	2

7.1. *Factorization study and qualitative MRM comparison*

Factorization: As already highlighted, the MRMs ngraphs and cpprop are capable of efficiently storing all granularity levels. However, the remaining MRMs do not support factorization, since they are intended for statement level granularity. In order to study this effect with regard to the number of statements, we used a sample of 100 entities from the DBpedia based evaluation dataset for each MRM. Table 2 highlights that the dataset sizes vary significantly. The raw input data size is about 1.2 MB for 100 entities. When different MRMs are applied on the data, the dataset size ranges from about 20MB to more than 2.8 GB. Table 2 clearly shows that *cpprop* and *ngraphs* MRM are using significantly less triples than the other MRMs for the same amount of information, due to its factorization support. Since on average nearly 950 statements describe the revision history of an entity, repeating all these metadata statements per data triple has a knock-on effect on the size of the MRM datasets. Due to the high number of meta information per triple, this study shows very strongly the different characteristics of these MRMs. Based on these numbers, we estimated that for 100.000 entities the dataset size can grow to more than a terabyte for MRMs not supporting factorization, when applied on our DBpedia based evaluation dataset.

Due to the large dataset sizes for some MRMs, we decided to introduce a shared resource, which holds metadata information for one DBpedia entity. For each MRM, which does not support factorization, we link to the shared resource only once per statement, instead of attaching every metadata fact to it. This strategy allows for storing at entity level granularity metadata in a cost effective manner. For rdr, we apply this technique for the revisions, only. However, the aggregated metadata is applied on triple level, using nested rdr statements. The resulting differences are displayed with *shared* in Table 2. Once a shared metadata resource is introduced for each statement, the dataset sizes only vary slightly between all the MRMs. As a result the *ngraphs* MRM is using the least amount of space and stdreif as well as rdr the most.

Query Complexity/Usability: As outlined in Section 4, the data layout and the query structure differ between MRMs, hence the query complexity has to be analyzed for each MRM. First the number of triple patterns and the number of extra SPARQL elements is considered. This gives an indication of how easy it is to read, understand and create a search query. Hence, it might have an impact on how an MRM is going to be adopted by other practitioners. The amount of extra triple patterns and the number of extra SPARQL elements (e.g. *GRAPH*, *BIND*) for each MRM search query is shown in the last two columns of Table 2. The second last column shows the number of triple patterns which are required to find a metadata statement which is associated to a data statement. A query which is based on the *Singleton Property* requires 2 triple patterns to reach the meta information and one more triple pattern to access it. If the meta information is shared for different data statements, then an additional triple pattern is required to access the meta information. This adds up to 3 or 4 triple patterns per query, depending on whether the meta information is shared or not. On top of the extra triple patterns, additional variables or SPARQL elements have to be added to the query. They are displayed in the last column of the table. When looking at the last two columns of the table, the *ngraphs* and *rdr* MRMs show the easiest usability, since the query complexity for these MRMs is the lowest.

Backwards Compatibility: In addition, we checked the backwards compatibility for data queries, meaning the ability to execute existing data queries on a mixed dataset with graph and metadata statements. This is shown in the fourth column. Apart from the *naryrel* MRM, all other MRMs support at least basic backward compatibility, if using a reasoning strategy. The *ngraphs* and *rdr* MRM only support backward compatibility for queries, which do not use the RDF named graph feature. But they do not need to rely on reasoning, in case triples are used, only. If the named graph feature of an RDF dataset is required, the graph IRI has to be treated as metadata, if using *ngraphs* or *rdr* MRM, or one of the other MRMs has to be used.

7.2. *Loading times and sizes*

When looking at the statement counts in Table 3, ngraphs can be identified as the most compact representation. Sgprop and rdr are the most compact triple based MRMs. In the Wikidata scenario, an naryrel serialization scheme similar to sgprop is used, which links the `p-NARY-value` edges shown in Figure 1 to its original predicate names `p`. Thus it has a higher number of statements. When we examined database sizes, we observed it the other way around. While the stdreif MRM transformation results in the highest number of statements, it consumes the least storage in the database files and ngraphs the most, due to the fact that additional index structures for the graph identifiers are maintained. For Blazegraph and Stardog this additional overhead is ranging from 60 to 75 %. Virtuoso uses index structures for graphs per se, which results in an overhead less than 4%. Rdr does consume more storage than the triple based MRMs but less than ngraphs. The difference between the triple based MRM is not significant. Blazegraph increases its journal file with fixed memory blocks (extents), which explains the equal sizes for all triple MRMs.

The rankings in the DBpedia scenario shown in Table 4 are slightly different. Cpprop is the most compact representation after ngraphs, which also is reflected in the smallest database size for Virtuoso and Blazegraph. For these stores the database size difference between sgprop, stdreif and naryrel is very small, too. Except for Virtuoso, ngraphs database files are similar to the Wikidata case the biggest. While Virtuoso and Stardog can benefit from the reduced number of graph identifiers (around 0.8 million for DBpedia vs. 80 million for Wikidata) the graph overhead almost doubles Blazegraph‘s journal size. Due to an optimized scheme for naryrel, its number of statements is equal to sgprop. In contrast, rdr has more statements overhead than stdreif, because we do not use a shared resource for the aggregated metadata.

Examining the loading times of the Wikidata datasets, we observed that naryrel tends to be the slowest and stdreif followed by sgprop the fastest. Although for Blazegraph and Stardog ngraphs is rather slow, Virtuoso processed it most rapidly. Cpprop is the fastest solution for DBpedia. Ngraphs followed by naryrel perform worst. However, the loading of stdreif is slower in relation to the Wikidata experiments. The huge gap between Blazegraph's loading times and its competitors is caused by a limited loading parallelization of the used version. Despite index updates themselves are being executed multi-threaded, the parser is not executed while the index updates are being performed. Furthermore files which could be read in parallel (nquads) are not processed in parallel and if multiple files are provided for bulk-loading these are loaded sequentially, as well. With ongoing progress of the bulkload procedure, we observed a continuously dropping rate of inserted triples per second as well as a decreasing CPU usage but an increasing time of waiting for I/O request completion. Albeit Stardog does load files in parallel and we could observe that it utilized all CPU cores, which explains the short loading times.

Summarising we found that for Stardog and Virtuoso the MRMs are competitive w.r.t loading times. For Blazegraph the variance is much higher, but seems to be dataset dependent. When it comes to database sizes ngraphs files are significantly larger for Stardog and Blazegraph. If Virtuoso in combination with ngraphs or the other MRMs is used, the choice is of no consequence for disk space.

Table 3

Wikidata experiment: The number of statements for the Wikidata dataset, its respective loading times and the final database size for the different MRMs. All loading times are in hours, whereas database sizes are in GiB.

	naryrel	ngraphs	sgprop	stdreif	rdr
#statements	563,678,588	482,371,357	563,676,547	644,981,737	563,676,547
loading time Virtuoso (hours)	3.39	3.04	3.22	3.15	-
db size Virtuoso (GiB)	45.94	46.83	46.25	45.21	-
loading time Blazegraph (hours)	14.57	13.03	7.12	7.09	10.40
db size Blazegraph (GiB)	60.73	98.25	60.73	60.73	66.87
loading time Stardog (hours)	1.12	1.35	1.07	0.85	-
db size Stardog (GiB)	32.34	56.52	32.31	32.19	-

Table 4

DBpedia experiment (db sizes and statement counts): Number of statements for the DBpedia based dataset, its respective loading times and the final database size for the different MRMs. All loading times are in hours, whereas database sizes are in GiB. In the second row the number of statements without counting the revision metadata is shown. The overhead of the MRM specific handling triples is illustrated in contrast to the most compact representation ngraphs (no additional handling triples) in row 3 for a simpler comparison. The last column shows the results for the Wikipedia revisions information (no instance data, no aggregated metadata, or MRM handling triples). Since this is no complete dataset, no values are available for *time complete* rows. The *data* column studies the dataset containing revisions and instance data but without any aggregated metadata and MRM handling triples. Rdr is limited to Blazegraph. Therefore no results are available for the other stores.

	cpprop	naryrel	ngraphs	sgprop	stdreif	rdr	data	revision metadata
total # statements	1,065,086,298	1,078,194,485	1,051,908,211	1,078,194,485	1,104,459,275	1,213,393,958	957,576,433	947,842,639
w/o revisions	117,243,659	130,351,846	104,065,572	130,351,846	156,616,636	265,551,319	9,733,794	0
MRM overhead compared to ngraphs	13,178,087	26,286,274	0	26,286,274	52,551,064	161,485,747	-94,331,778	-
time (MRM) part Virtuoso	0.39	0.47	0.60	0.46	0.48	-	0.29	2.14
time complete dataset Virtuoso	2.53	2.61	2.73	2.60	2.61	-	2.43	-
db size Virtuoso	37.71	39.46	38.26	39.92	38.92	-	33.96	32.65
time (MRM) part Blazegraph	3.46	4.61	-	4.83	5.40	-	0.09	42.51
time complete dataset Blazegraph	45.98	47.13	42.51	47.35	47.91	29.31	42.60	-
db size Blazegraph	77.03	79.72	157.61	80.34	80.07	89.25	68.63	66.60
time (MRM) part Stardog	0.50	0.66	-	0.56	0.56	-	0.17	0.97
time complete dataset Stardog	1.48	1.64	1.75	1.54	1.53	-	1.15	-
db size Stardog	76.94	77.97	86.27	78.72	73.58	-	67.09	56.86

7.3. Wikidata query results

In Figure 2, we can identify stdreif as best solution for Blazegraph for the Wikidata use case; no single timeout occurs. In Virtuoso, stdreif also exhibits a good performance for queries having an execution time longer than 30 milliseconds. For queries faster than that, the additional number of joins caused by the 4 triple patterns has a greater impact on the execution time. While the singleton property is the worst performer for Virtuoso, in Blazegraph there is no huge difference between sgprop, naryrel, and ngraphs. Moreover, sgprop is the best model for Stardog but with no significant difference to stdreif. Though naryrel is faster for simple queries in Stardog, it is not competitive for challenging queries. Surprisingly ngraphs is exceptional slow. Since the Stardog code is not publicly available and we could not find documentation about indexing techniques and other database internals, we can only guess that the database structures for named graphs lead to performance issues for a high number of graphs. The rdr feature which is used to encode the statement identifier and not the metadata directly (caused by the data model of Wikidata as mentioned before), cannot benefit from its indexing strategy. It is the worst performing MRM for the Wikidata use case. Generalising over all quin queries and stores, stdreif performs best.

7.4. DBpedia query template results

Considering the mixed Queries for the DBpedia dataset, ngraphs is the clear winner, as can be seen in Figure 2. For the triple based MRM, naryrel exhibits the best performance in Virtuoso. We can in theory observe the same behavior for Blazegraph. Unfortunately, the naryrel queries for Blazegraph do not return the full number of results. This explains why the queries are executed quickly and why naryrel seems to even outperform ngraphs. Blazegraph showed is-

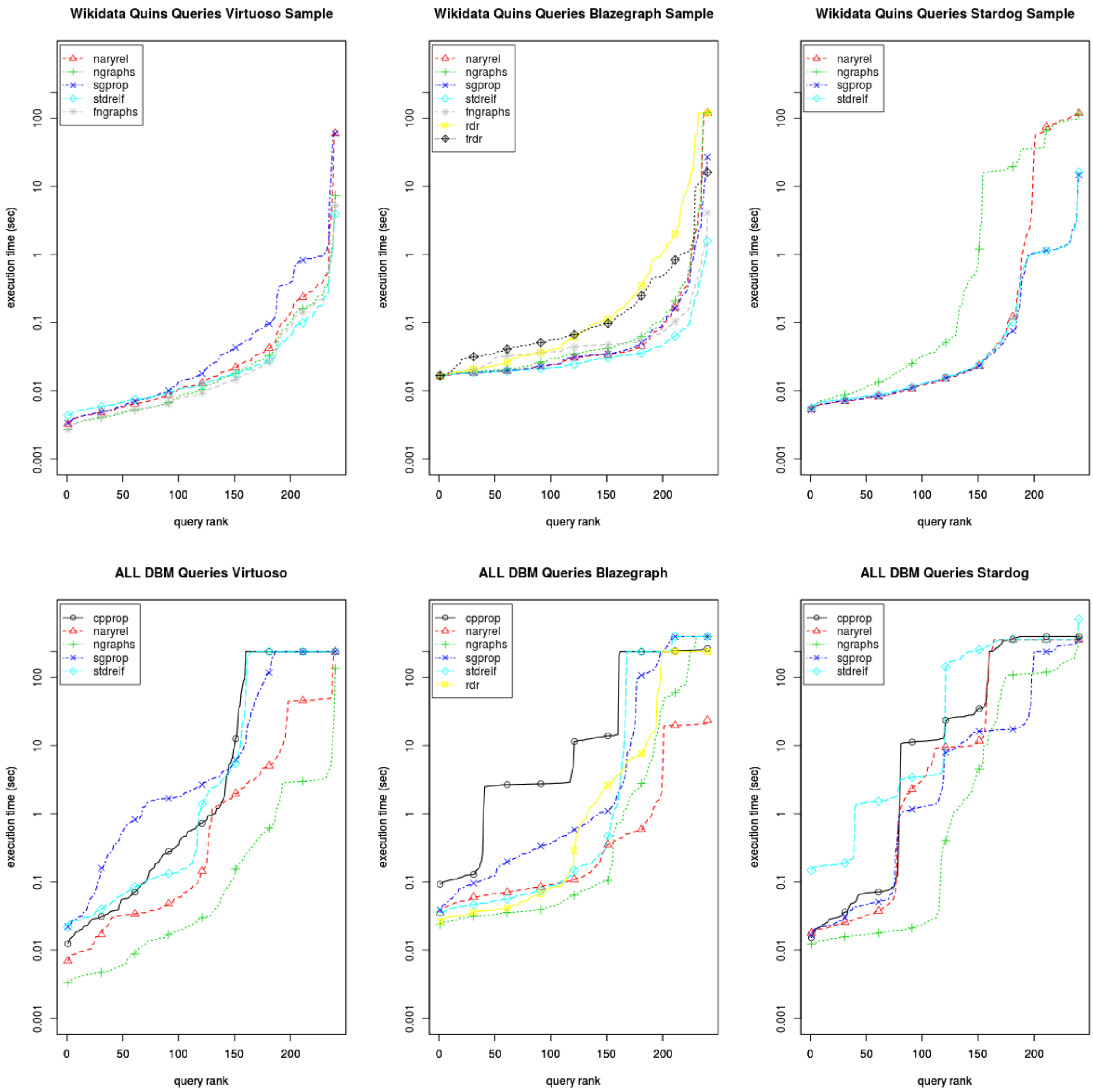

Figure 2. **Comparison of MRM performance for metadata related queries between Wikidata and DBpedia Queries:** Overall results for all queries of DBM patterns (bottom) and a random sample (240 queries) from all Wikidata quin patterns (top). The queries are sorted by its execution time (fastest query has rank 0) for each MRM individually.

sues evaluating queries with multiple `BIND` statements correctly. We tried to circumvent this observation by rewriting the queries for Blazegraph. When executing the rewritten queries, Blazegraph froze when processing the DBM-HAR-01 queries. Therefore, we were not able to test naryrel in a reliable way. As a consequence, the best triple based MRM for Blazegraph is standard reification. However, sgprop outperforms all triple MRMs in Stardog. But it is important to mention naryrel potentially is a better choice for the tested Stardog version based on a trade-off between performance and query stability, since we experienced several non-deterministic HTTP 500 Errors, when we executed sgprop queries (see section 7.6). Stdreif turns out to be the most inefficient approach for Stardog. The rdr feature is almost competitive with ngraphs for the simple and medium queries taking not the revisions into account. While we found performance problems for queries over revision metadata (which are just links to other resources similar to the Wikidata dataset), Blazegraph can leverage its nested statements and benefit from the special structure of the aggregated metadata, which is reified itself. In best case, these nested statements allow Blazegraph to materialize the joins which are necessary in other MRMs. Nonetheless, we cannot observe this advantage for hard queries. For those queries, we encountered several timeouts, which is indicated by the platue in the rdr curve in the plot. De facto there are two plateus, which will be discussed in more detail in section 7.6.

Looking at the overhead of MRMs for regular data queries (Figure 3), ngraphs is the closest to the baseline for the majority of queries against Virtuoso. For queries faster than two seconds, cpprop is competitive with the other triple MRMs. In spite of this, for stdreif and cpprop followed by sgprop occur many timeouts for complex queries. Summing all query execution times, naryrel is the best triple MRM for Virtuoso. In contrast to that naryrel and ngraphs MRMs introduce the most overhead in Stardog, though to the latter is performing well for short queries. Cpprop is the best option. Likewise, sgprop shows similar good results as for mixed queries in Stardog. While there is a significant overhead for queries shorter than one second, rdr is outstanding for challenging data queries. Cpprop deals as second-best option in Blazegraph. Stdreif and ngraphs show almost the same behavior. Sgprop appears as the slowest MRM. Note that naryrel returns incomplete result sets for a fraction of Blazegraph's queries, as already mentioned for mixed queries.

7.5. Dataset characteristics impact

To examine the influence of the dataset and the type of metadata, we compared the DBpedia related mixed simple queries to a query pattern from the quins experiment. The SIMPLE group applies for these kinds of queries. Hence, the query impact is low, so that we potentially can observe an influence of the dataset or the metadata type. We selected the `10010` query pattern, as it projects all properties and values of a specific entity and all its metadata values for a given key. Besides the fact that DBQ-SIM-01 instances are querying for persons only and use a non-varying key constant (creation date) the templates are the same. We can observe for Blazegraph that the trends for the selected quin pattern in Figure 4 are in line with the overall results for Wikidata from Figure 2. Stdreif outperforms the other MRMs, sgprop and naryrel are slower but do not significantly differ. Ngraphs is slower and (f)rdr performs worst. In contrast to that, there is a different order for the DBpedia dataset. While ngraphs undoubtedly is the best, sgprop is much slower. Stdreif deals as best triple MRM candidate. The fact that ngraphs performs better for DBM-SIM-01 can be explained by the characteristics of the metadata: Parts of the metadata facts in the DBpedia dataset (like the creation date) are reified as well in order to store meta-metadata. For ngraphs, the metadata is stored "as is", while for stdreif and the other triple based MRMs the metadata triple is split into several triples. Hence the complexity for query evaluation is higher for these MRMs. For Virtuoso, the results are very similar to the DBM overall results. But the gap between sgprop and the other MRM is drastically bigger. When having a look at the execution times from Wikidata, we see that there is no such gap. Moreover, naryrel is the fastest alternative option for ngraphs for the DBM queries, but the worst MRM for the Wikidata scenario. Taking into account that this is not the case for the overall quins results and additionally that the average execution time for Wikidata and DBpedia are close to each other, we think that this is caused by a general overhead when evaluating the queries for naryrel. Thus, this observation does not seem to reflect a dataset impact. But is noteworthy, that again stdreif is worse in relation to the other MRMs. Following the overall trends for Stardog, sgprop performs well for both datasets. But using the DBpedia based dataset, ngraphs is the fastest approach with respect to query execution times as opposed to Wikidata, where it is the worst solution. This observation seems to support our presumption that ngraphs has performance issues in Stardog, when the number of graph identifiers is large (81 million graphs in Wikidata, around 100 times more compared to DBpedia). Stdreif is notably slower in the DBpedia scenario likewise for Virtuoso. Furthermore, we observed a poor performance executing DBM-SIM-02 queries for naryrel.

7.6. Timeouts, database instabilities & pitfalls

As mentioned before, we implemented an additional client side timeout in the benchmarking framework. For both Blazegraph and Stardog, it is crucial to use this second timeout as fallback to continue benchmarking. For challenging queries these Java based stores had issues terminating the query within the specified

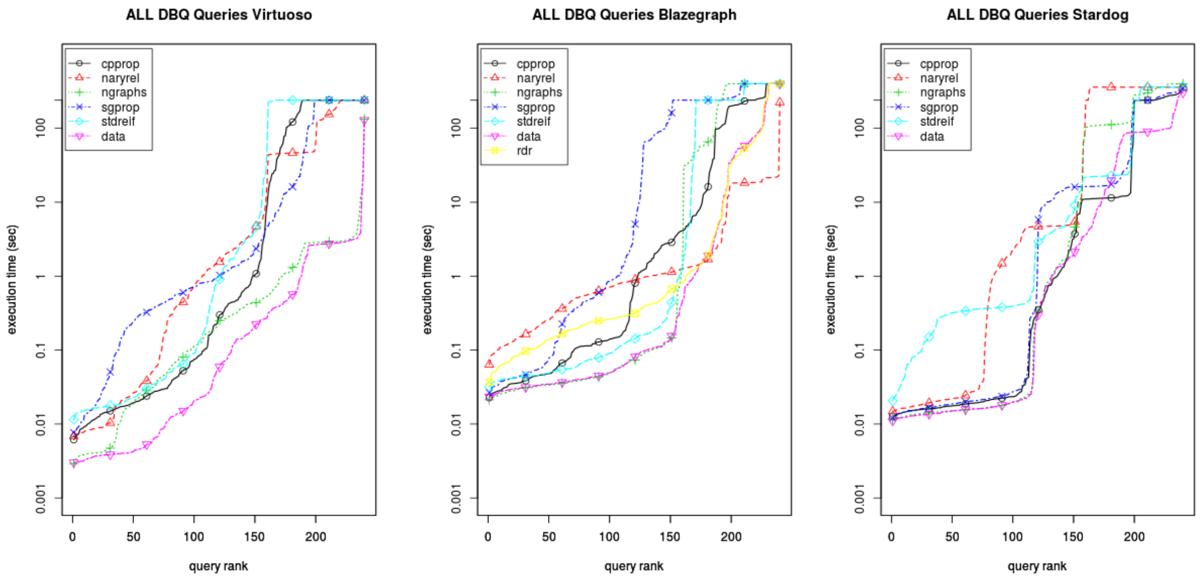

Figure 3. **MRM overhead for data only queries:** Overall results for all DBQ queries in relation to baseline representation containing data triples (purple line)

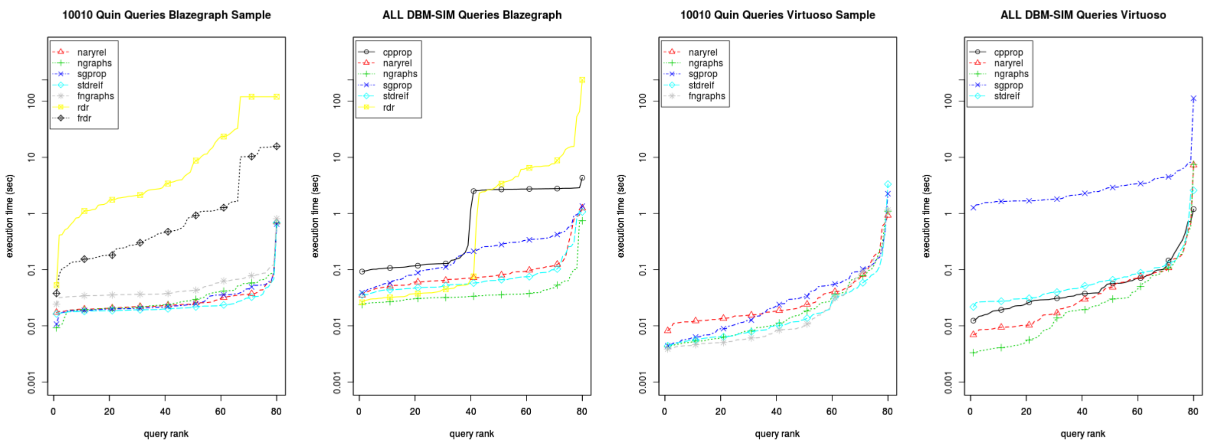

Figure 4. **Studying dataset influence I - Blazegraph & Virtuoso**: A random sample of 80 query instances of the Wikidata quin 10010 query template is compared to all DBM-SIM-01 and DBM-SIM-02 instances. The queries are very similar in structure and/or complexity.

database timeout of 240 seconds. If huge parts of the dataset are being processed, several Java Objects are created. The garbage collection seems to be the reason that both stores struggle and get unresponsive. The framework therefore waits up to 400 seconds to let the store clean up memory and properly terminate the timed-out query before executing the next query. In [22], it had already been reported that subsequent queries did time out non-deterministically. Having a closer look at this issue revealed that challenging queries, which are supposed to be aborted by the database (due to database timeout), continue to run up to several minutes. As a result the original Wikidata setup effectively ran several queries in parallel, which increased the backend pressure even more and caused the timeout of subsequent queries. The client timeout helps to reduce such domino effects by giving the backend a period of 160 seconds to abort the query and enforce the database timeout. However, even this additional timeout is too short for stopping every challenging query. Therefore, a second plateau at 400 seconds can be observed in the plots for various MRMs.

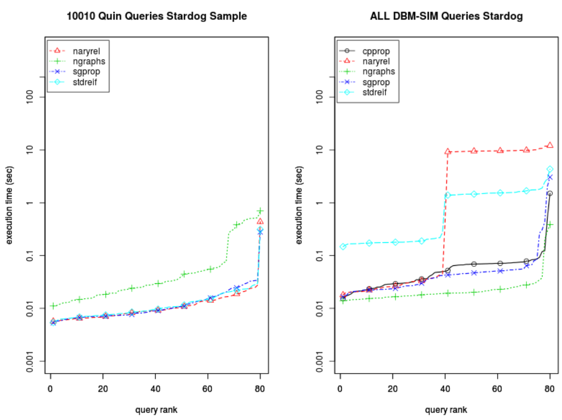

Figure 5. **Studying dataset influence II - Stardog:**

Despite this, we observed both databases transitioning into an undefined state after executing a number of queries. The backends were still running, but did not respond to any command or activity and did not consume CPU time. For several query templates, this was non-deterministic and rerunning the experiment for the query template solved the issue. The garbage collection overhead and problems can be tackled by using off-heap data structures. For Blazegraph a so-called analytic query mode exists, leveraging a custom (off-heap) memory management. Yet, for the rewritten Blazegraph naryrel queries several attempts did not succeed. According to the Blazegraph issue-tracker not all database components and query execution stages utilize this memory management. Likewise, for Stardog, we were not able to run the original HAR-01 templates. Splitting its 3 pairwise OPTIONAL clauses into 6 clauses resolved this issue. Further investigation revealed that due to sub-optimal query optimization and planning, Stardog performs several loop joins and finally exceeds memory. As already stated, internal server errors occurred for subsequent sgprop and cpprop HAR-02 queries, starting at random positions in the queries. After submission of this work a native memory management, trying to address GC issues, has been integrated in the Stardog 5 release (July 2017).

In order to allow for a comparison of Stardog for this template, we decided to use the rewritten template for the other backends as well. Figure 6 illustrates the significant changes in runtime behavior between this template variations. The fix leads to an improvement for sgprop and cpprop for Virtuoso. As a result Blazegraph's performance for sgprop, cpprop and especially ngraphs, declined sharply but increased for rdr. This observation exposes a major problem. The performance of an MRM is not just dependent on its structure, the complexity of the query and database architecture details like data structures and indices. In practice, the way how the template query is expressed and how BGP's are ordered or grouped may have a huge impact on the performance. Depending on the structure, the query optimizer may choose an improper join order. Thus, an MRM based on many joins (e.g., cpprop) has a higher risk that unfavorable join orders are used in the query plan.

To check whether all MRMs are consistent, we compared result sizes. We discovered that a few template queries returned incorrect results. Virtuoso evaluates the FILTER EXISTS expression in HAR-02, which contains an optional variable, as false if the variable is unbound. As opposed to this, Blazegraph and Stardog evaluate this condition as true. We therefore added bound conditions to the queries. A similar issue arise for fngraphs queries in Stardog, which we therefore excluded. Virtuoso shows different results, if strings (containing numbers only) are compared to integers.

7.7. *Comparison with other studies*

The results in [13] show a complete different picture. Regarding sgprop, Virtuoso performed best, Stardog was 3 to 4 times slower than Virtuoso and Blazegraph was much slower than both of them. As the used nary-relation serialization is really different in structure, it is hard to compare with our naryrel results, but instead it makes more sense to compare the trends with stdreif model. Stardog clearly outperformed Blazegraph (5 times slower) and Virtuoso (around 2 to 20 times slower) for queries against this standard reification variant. Several factors may explain the different outcome. The dataset is rather small and from another domain, the evaluation setup differs and the used storage backends have received major updates.

8. Conclusion and Future Work

To summarize, this work defined requirement-based criteria to drive an evaluation of different approaches for metadata handling in three prominent RDF stores. Furthermore, a systematic comparison of several MRMs and its corresponding queries was presented. Based on previous work, additional datasets and use cases which elaborated different aspects about dealing with metadata in RDF datasets were created. Additionally, we introduced a novel metadata representation model

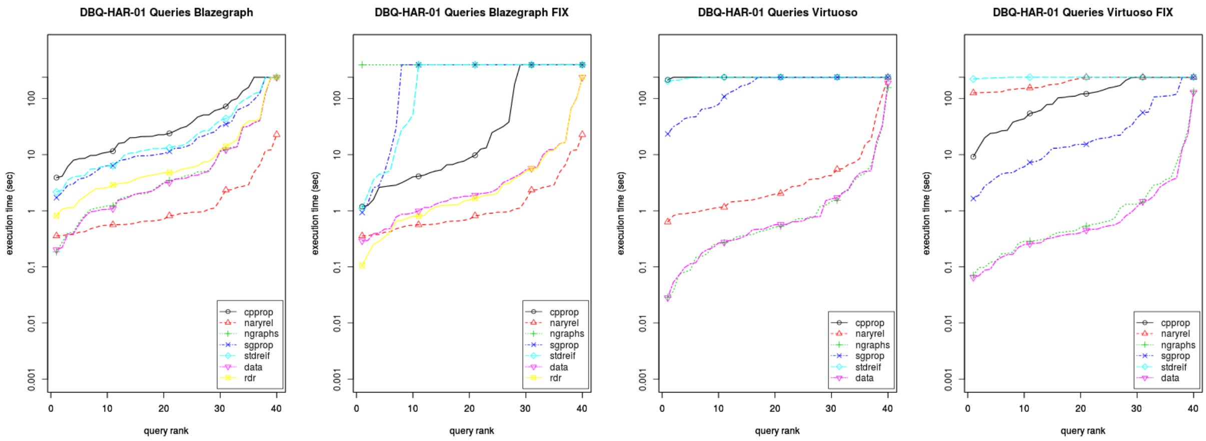

Figure 6. **Query optimization impact:** Comparison of original and rewritten (Stardog fix) DBQ-HAR-01 queries for Blazegraph and Virtuoso

called *Companion Properties*, which has been proven to be a good alternative to existing triple based MRMs for DBQ Queries, even outperforming ngraphs in Stardog. Unfortunately, it did not perform well for challenging DBM queries. Substituting `idPropertyOf` triple patterns, in order to reduce joins and increase performance, will be subject to future research.

The results clearly show, that ngraphs outperforms the other MRMs for challenging mixed queries, which confirms the results presented in [22] for more complex templates than the quin queries. As long as the use case or source dataset does not require the usage of quads, ngraphs is the most suitable solution. In case existing datasets rely on named graphs, naryrel turned out to be a very good alternative for Virtuoso. With the used Blazegraph version, it was not possible to finish the Blazegraph naryrel experiments due to an unresponsive backend. In contrast to [22], we were not able to confirm performance problems with sgprop. It is the best triple MRM for Stardog and it seems to be a more reliable replacement for naryrel in Blazegraph. Stdreif performs good in Blazegraph and Stardog for simple queries, but it has shortcomings in Stardog and for challenging queries in all tested stores. This is not in line with findings in [22], where stdreif had been reported as competitive to ngraphs. The obtained results indicate that metadata characteristics have an impact on the ranking of the MRMs. Ngraphs and rdr support queries against meta-metadata much better than the other MRMs. In general, rdr can compete with ngraphs, if the metadata is on statement level granularity and does not require logical units of metadata facts.

In addition, experimental results show that the MRM ranking differs between data-only and mixed query scenarios. Moreover, ngraphs and rdr offer the best trade-off for both, mixed and data query workloads. When the query templates were created, the query optimizers were strongly impacted by even minimal structural changes in the queries. After investigating incomplete or wrong query results, we encountered the presented SPARQL implementation errors and variations in the used stores. Therefore an adoption of existing SPARQL test suites to check for these errors is advisable. Besides, memory and stability issues for Stardog and Blazegraph were observed, which were caused by garbage collection pressure. Improving query (plan) optimizers and memory management is ongoing work for upcoming major releases of Stardog and Blazegraph. Therefore, it could be interesting to repeat these experiments in the future, in order to evaluate the impact of such changes on the query execution performance.

8.1. Future Work

With regard to benchmarking MRMs, there are many aspects that required to be studied in the future, e.g., parallel query workloads with multiple users. Depending on whether high throughput or low latency are required, it would be interesting to evaluate, which factors influence the performance of a parallel execution. If multiple query operators are evaluated in parallel, this can improve execution performance of an individual query, but it can result in higher costs, which in turn can potentially decrease the overall system performance, if many queries are run in parallel. Hence, a future work could validate, whether the results are also valid for parallel workloads.

In the performance evaluation of this paper (and to the best of our knowledge also in previous MRM studies), the queries are read-only. Having Big Data systems in mind, we can think of scenarios, where data is streamed (added and changed) continuously into the database. Changing metadata of a triple using a shared statement identifier (ngraphs & cpprop) is more complex than for other MRMs. So MRMs optimized for fast reading and factorization, may perform worse in update scenarios. Moreover, the study reported in this paper could be extended to other datasets to gain a better insight into dataset characteristics impact. By using additional datasets and update queries in combination with formalized scores and evaluation procedures, the performance evaluation could be extended to a general-purpose benchmarking framework for metadata handling in RDF stores. Furthermore, the mixed queries used in this evaluation are characterized by selecting and filtering (e.g., for confidence) of metadata. Therefore, metadata has a supportive role for the data parts of a query. We think, it could be interesting to study queries with patterns evaluating metadata statements over different entities or using other more elaborated metadata-centric query patterns, in order to find out whether similar results can be observed. On top of this, different ways in how to query a specific MRM need to be analyzed in more detail. For cpprop and naryrel, other SPARQL expression can be used to reduce the number of joins or BIND statements. Additionally, the injection of query hints during the transformation of a template for an MRM can help to support the query plan selection. In order to improve metadata handling, combinations of MRMs can be considered as well. For the purpose of backwards compatibility and performance, it seems reasonable to use a triple MRM for the first metadata level but ngraphs for meta-metadata. Moreover, rdr could be combined with an adopted version of cpprop to logically tie metadata facts, which belong to the same group. As these queries would require users to have detailed knowledge of the applied MRMs, the usability could be improved by using a SPARQL proxy. Such a service could (similar to our query transformation framework) use special MRM-independent annotations within a query, to translate it into the appropriate format. To go one step further, a more sophisticated metadata-aware system could be developed, which enables unified querying, regardless the used MRMs, granularity levels, and metadata levels.

Acknowledgements

This work was supported by grants from the Federal Ministry for Economic Affairs and Energy of Germany (BMWi) for the SmartDataWeb Project (GA-01MD15010B)[38], as well as from the European Union for the Horizon 2020 project ALIGNED (GA-644055).

References

[1] G. Aluç, O. Hartig, M. T. Özsu, and K. Daudjee. Diversified stress testing of RDF data management systems. In *The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part I*, pages 197–212, 2014.

[2] R. Angles, P. Boncz, J. Larriba-Pey, I. Fundulaki, T. Neumann, O. Erling, P. Neubauer, N. Martinez-Bazan, V. Kotsev, and I. Toma. The linked data benchmark council: a graph and rdf industry benchmarking effort. *ACM SIGMOD Record*, 43(1):27–31, 2014.

[3] C. Baron Neto, K. Müller, M. Brümmer, D. Kontokostas, and S. Hellmann. Lodvader: An interface to lod visualization, analytics and discovery in real-time. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 163–166. International World Wide Web Conferences Steering Committee, 2016.

[4] C. Bizer and A. Schultz. The berlin sparql benchmark. *Int. J. Semantic Web Inf. Syst.*, 5(2):1–24, 2009.

[5] J. Bleiholder and F. Naumann. Conflict handling strategies in an integrated information system. In *IJCAI Workshop on Information on the Web (IIWeb)*, 2006.

[6] J. J. Carroll, C. Bizer, P. J. Hayes, and P. Stickler. Named graphs, provenance and trust. In A. Ellis and T. Hagino, editors, *WWW*, pages 613–622. ACM, 2005.

[7] O. Corby and C. Faron-Zucker. Rdf/sparql design pattern for contextual metadata. In *Web Intelligence*, pages 470–473. IEEE Computer Society, 2007.

[8] X. L. Dong, E. Gabrilovich, G. Heitz, W. Horn, K. Murphy, S. Sun, and W. Zhang. From data fusion to knowledge fusion. *Proc. VLDB Endow.*, 7(10):881–892, June 2014.

[9] M. Dumontier, A. Callahan, J. Cruz-Toledo, P. Ansell, V. Emonet, F. Belleau, and A. Droit. Bio2rdf release 3: a larger connected network of linked data for the life sciences. In *Proceedings of the 2014 International Conference on Posters & Demonstrations Track-Volume 1272*, pages 401–404. CEUR-WS. org, 2014.

[10] F. Erxleben, M. Günther, M. Krötzsch, J. Mendez, and D. Vrandečić. Introducing wikidata to the linked data web. In *International Semantic Web Conference*, pages 50–65. Springer, 2014.

[11] J. D. Fernández, J. Umbrich, A. Polleres, and M. Knuth. Evaluating query and storage strategies for rdf archives. In *Proceedings of the 12th International Conference on Semantic Systems*, SEMANTiCS 2016, pages 41–48, New York, NY, USA, 2016. ACM.

[38] http://smartdataweb.de/

[12] J. Frey and S. Hellmann. Masque: An approach for flexible metadata storage and querying in rdf. In *13th International Conference on Semantic Systems Proceedings (SEMANTiCS 2017) - Posters & Demonstrations Track*, SEMANTiCS '17, Sept. 2017.

[13] G. Fu, E. Bolton, N. Queralt-Rosinach, L. I. Furlong, V. Nguyen, A. P. Sheth, O. Bodenreider, and M. Dumontier. Exposing provenance metadata using different RDF models. In *Proceedings of the 8th Semantic Web Applications and Tools for Life Sciences International Conference, Cambridge UK, December 7-10, 2015.*, pages 167–176, 2015.

[14] F. Gandon, R. Boyer, O. Corby, and A. Monnin. Materializing the editing history of Wikipedia as linked data in DBpedia. In *ISWC 2016-15th International Semantic Web Conference*, Oct. 2016. Poster.

[15] G. Garbis, K. Kyzirakos, and M. Koubarakis. Geographica: A benchmark for geospatial rdf stores (long version). In *International Semantic Web Conference*, pages 343–359. Springer, 2013.

[16] G. Gawriljuk, A. Harth, C. A. Knoblock, and P. Szekely. A scalable approach to incrementally building knowledge graphs. In N. Fuhr, L. Kovács, T. Risse, and W. Nejdl, editors, *International Conference on Theory and Practice of Digital Libraries*, volume 9819 of *Lecture Notes in Computer Science*, pages 188–199. Springer, Springer, 2016.

[17] P. T. Groth, A. Gibson, and J. Velterop. The anatomy of a nanopublication. *Inf. Services and Use*, 30(1-2):51–56, 2010.

[18] Y. Guo, Z. Pan, and J. Heflin. Lubm: A benchmark for owl knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158–182, Oct. 2005.

[19] O. Hartig. Reconciliation of rdf* and property graphs. *CoRR*, abs/1409.3288, 2014.

[20] O. Hartig and B. Thompson. Foundations of an alternative approach to reification in rdf. *CoRR*, abs/1406.3399, 2014.

[21] O. Hassanzadeh, A. Kementsietsidis, L. Lim, R. J. Miller, and M. Wang. Linkedct: A linked data space for clinical trials. *CoRR*, abs/0908.0567, 2009.

[22] D. Hernández, A. Hogan, C. Riveros, C. Rojas, and E. Zerega. 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*, pages 88–103, 2016.

[23] D. Hernández, A. Hogan, and M. Krötzsch. Reifying rdf: What works well with wikidata? In T. Liebig and A. Fokoue, editors, *SSWS@ISWC*, volume 1457 of *CEUR Workshop Proceedings*, pages 32–47. CEUR-WS.org, 2015.

[24] A. Iosup, T. Hegeman, W. L. Ngai, S. Heldens, A. Prat-Pérez, T. Manhardto, H. Chafio, M. Capotă, N. Sundaram, M. Anderson, I. G. Tănase, Y. Xia, L. Nai, and P. Boncz. Ldbc graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. *Proc. VLDB Endow.*, 9(13):1317–1328, Sept. 2016.

[25] M. Koubarakis and K. Kyzirakos. *Modeling and Querying Metadata in the Semantic Sensor Web: The Model stRDF and the Query Language stSPARQL*, pages 425–439. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[26] N. Lopes, A. Polleres, U. Straccia, and A. Zimmermann. Anql: Sparqling up annotated RDFS. In P. F. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Z. Pan, I. Horrocks, and B. Glimm, editors, *The Semantic Web - ISWC 2010 - 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I*, volume 6496 of *Lecture Notes in Computer Science*, pages 518–533. Springer, 2010.

[27] N. Lopes, A. Zimmermann, A. Hogan, G. Lukácsy, A. Polleres, U. Straccia, and S. Decker. Rdf needs annotations. In *Proceedings of W3C Workshop—RDF Next Steps*. W3C, 2010.

[28] F. Mahdisoltani, J. Biega, and F. Suchanek. Yago3: A knowledge base from multilingual wikipedias. In *7th Biennial Conference on Innovative Data Systems Research*. CIDR Conference, 2014.

[29] M. Meimaris and G. Papastefanatos. The evogen benchmark suite for evolving rdf data. In *MEPDaW/LDQ@ ESWC*, pages 20–35, 2016.

[30] M. Meimaris, G. Papastefanatos, S. Viglas, Y. Stavrakas, C. Pateritsas, and I. Anagnostopoulos. A query language for multi-version data web archives. *Expert Systems*, 33(4):383–404, 2016.

[31] G. Montoya, M.-E. Vidal, O. Corcho, E. Ruckhaus, and C. Buil-Aranda. *Benchmarking Federated SPARQL Query Engines: Are Existing Testbeds Enough?*, pages 313–324. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[32] M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. *DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data*, pages 454–469. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[33] M. Nentwig, T. Soru, A.-C. N. Ngomo, and E. Rahm. Linklion: A link repository for the web of data. In *European Semantic Web Conference*, pages 439–443. Springer, 2014.

[34] V. Nguyen, O. Bodenreider, and A. P. Sheth. Don't like rdf reification?: making statements about statements using singleton property. In C.-W. Chung, A. Z. Broder, K. Shim, and T. Suel, editors, *WWW*, pages 759–770. ACM, 2014.

[35] V. Nguyen, O. Bodenreider, K. Thirunarayan, G. Fu, E. Bolton, N. Queralt-Rosinach, L. I. Furlong, M. Dumontier, and A. P. Sheth. On reasoning with RDF statements about statements using singleton property triples. *CoRR*, abs/1509.04513, 2015.

[36] V. Papakonstantinou, G. Flouris, I. Fundulaki, K. Stefanidis, and G. Roussakis. Versioning for linked data: Archiving systems and benchmarks. In *Proceedings of the Workshop on Benchmarking Linked Data (BLINK 2016) co-located with the 15th International Semantic Web Conference (ISWC), Kobe, Japan, October 18, 2016.*, 2016.

[37] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, Sept. 2009.

[38] S. Peroni, A. Dutton, T. Gray, and D. Shotton. Setting our bibliographic references free: towards open citation data. *Journal of Documentation*, 71(2):253–277, 2015.

[39] S. Peroni, D. M. Shotton, and F. Vitali. A document-inspired way for tracking changes of rdf data. In L. Hollink, S. Darányi, A. Meroño-Peñuela, and E. Kontopoulos, editors, *Drift-a-LOD@EKAW*, volume 1799 of *CEUR Workshop Proceedings*, pages 26–33. CEUR-WS.org, 2016.

[40] M. Saleem, Y. Khan, A. Hasnain, I. Ermilov, and A. N. Ngomo. A fine-grained evaluation of SPARQL endpoint federation systems. *Semantic Web*, 7(5):493–518, 2016.

[41] M. Saleem, Q. Mehmood, and A.-C. N. Ngomo. Feasible: A feature-based sparql benchmark generation framework. In M. Arenas, O. Corcho, E. Simperl, M. Strohmaier, M. d'Aquin, K. Srinivas, P. T. Groth, M. Dumontier, J. Heflin, K. Thirunarayan, and S. Staab, editors, *International Semantic*

Web Conference (1), volume 9366 of *Lecture Notes in Computer Science*, pages 52–69. Springer, 2015.

[42] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, and T. Tran. Fedbench: A benchmark suite for federated semantic data query processing. *The Semantic Web–ISWC 2011*, pages 585–600, 2011.

[43] M. Schmidt, T. Hornung, G. Lausen, and C. Pinkel. Sp2bench: A SPARQL performance benchmark. *CoRR*, abs/0806.4627, 2008.

[44] B. Schueler, S. Sizov, S. Staab, and D. T. Tran. Querying for meta knowledge. In *Proceedings of the 17th international conference on World Wide Web*, pages 625–634. ACM, 2008.

[45] C. Stadler, J. Lehmann, K. Höffner, and S. Auer. Linkedgeodata: A core for a web of spatial open data. *Semantic Web*, 3(4):333–354, 2012.

[46] O. Udrea, D. R. Recupero, and V. Subrahmanian. Annotated rdf. *ACM Transactions on Computational Logic (TOCL)*, 11(2):10, 2010.

[47] M. Voigt, A. Mitschick, and J. Schulz. Yet another triple store benchmark? practical experiences with real-world data. In *SDA*, pages 85–94, 2012.

[48] Y. Zhang, P. M. Duc, O. Corcho, and J.-P. Calbimonte. *SRBench: A Streaming RDF/SPARQL Benchmark*, pages 641–657. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.