Maximilian Eibl, Martin Gaedke. (Hrsg.): INFORMATIK 2017,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2017 1891

SPARQL Update queries over R2RML mapped data sources

Jörg Unbehauen[1], Michael Martin[1]

Abstract: In the Linked Data Life Cycle mapping and extracting data from structured sources is an essential step in building a knowledge graph. In existing data life cycles this process is unidirectional, i.e. the data is extracted from the source but changes like cleaning and linking are not fed back into the originating system. SPARQL-to-SQL rewriters create virtual RDF without materializing data by exposing SPARQL endpoints. With the Update extension of our SparqlMap system we provide read/write access to structured data sources to enable a tighter integration of the source systems in knowledge refinement process. in this paper, we discuss three different update methods and further describe in two scenarios how the source system can benefit from feed back from the Linked Data integration.

Keywords: R2RML; SPARQL Update; OBDA; RDB-to-RDF

1 Introduction

The Linked Data Lifecycle is an approach for managing information with the aim of representing knowledge in a structured, machine readable way. The extraction of RDF from structured data sources constitutes an important step in creating such an Linked Data Lifecycle. However, current approaches like ontop [RMHC12], morph [PCS14] or our own SparqlMap [UM16] only provide means for exposing structured data as (virtual) RDF in order to integrate it into a linked data life cycle. As of now, feeding data back into the originating system requires custom transformations and adapters is therefore held back by high upfront costs. SPARQL 1.1 Update [SGP08] offers Linked Data tools a standardized method for updating data. We therefore extended SparqlMap with a SPARQL Update capable endpoint which leverages integration of legacy systems into the Linked Data Lifecycle. This bidirectional integration ultimately allows legacy system to participate from the benefits of Linked Data knowledge management processes. We use mappings expressed in the RDB-to-RDF Mapping Language (R2RML)[DSC12] for the SPARQL Update to SQL Data Manipulation Language (DML) statements.

The rest of our paper is structured as follows. First, we introduce two scenarios to motivate our research. In Sect. 3 we discuss other approaches on enabling SPARQL Update over mapped data sources. Sect. 4 introduces our approach and our proposed solutions to the previously determined requirements. Subsequently, we evaluate our approach in Sect. 5 by

[1] University of Leipzig, AKSW/IIS, Germany [unbehauen|martin]@informatik.uni-leipzig.de

doi:10.18420/in2017_189

simulating SPARQL insert operations. Finally, we draw a conclusion. For the sake of brevity, we abbreviate vocabulary URIs by prefixing them: for URIs of the R2RML namespace we utilize `rr: <http://www.w3.org/ns/r2rml#>`. Further we assume that in our examples, besides the FOAF namespace `foaf: <http://xmlns.com/foaf/0.1/>`, our custom `ex: <http://example.com/>` namespace is also set as base URI.

2 Scenarios

Updates over mapped structured data sources allow interactive modification of the data sources. A straightforward use case is an interactive data manipulation tool like OntoWiki[ADR06] which can persist data modifications using SPARQL update queries. In order to foster a deeper understanding we consider two additional scenarios and determine their requirements on bidirectional RDB-to-RDF mappings.

Scenario 1: Enterprise Search The enterprise search scenario is based on [Fr13] in which Linked Data technologies are used for feeding a full text index. This enterprise search use case is depicted in Fig. 1. Multiple heterogeneous data *source*s are extracted and

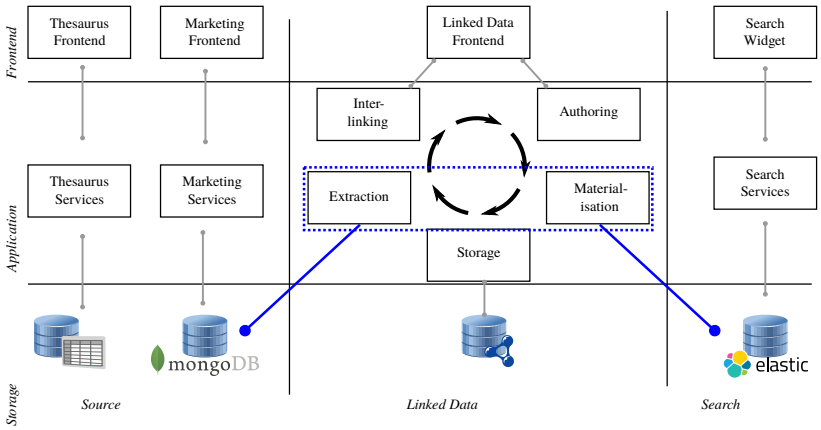

Fig. 1: Enterprise use case of data aggregation and extraction from multiple data sources and feeding an full text search engine.

loaded into a triple store. The extracts are cleaned, interlinked and streamlined using an interactive authoring component, effectively constituting a Linked Data Lifecycle. The resulting knowledge graph is finally materialized into the target *Search* system using a custom program. By using a bidirectional mapping of the search index, we no longer require a custom program for loading the data. Further, as the mapping allows also for read access, the index data is visible from within Linked Data management tools.

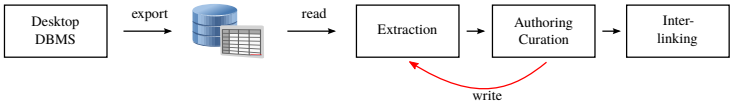

Fig. 2: ETL pipeline, starting with an transient data export and feeding into an Linked Data based enrichment pipeline with the proposed update extension as red arrow.

We can derive the following requirements from this scenario: (**R1**): Non-relational data backends; (**R2**): (Bulk) insertion of large data volumes.

Scenario 2: Extract Transform Load Pipeline In the Extract-Transform-Load (ETL) scenario we discuss how SPARQL update operations can be used to augment a pipelined data operation. Such a pipeline, which is based on the data extraction scenario presented in [BR16], is depicted in Fig. 2.

The date pipeline ingests exports from a desktop Data Base Management System (DBMS) and translates the data via a R2RML mapping into a RDF materialization. This materialization is then modified by an authoring component and further processed. Instead of modifying the materialized RDF extract we suggest that the authoring component issues SPARQL Update queries instead. The challenge in this scenarios is dealing with a data source that either is read-only or is transient in the sense that it is frequently re-created and changes are consequently discarded. An additional challenge poses the schema employed in the desktop DBMS, which is not properly normalized. We can therefore infer the following requirements of this scenario: (**R1**): Non-relational data backends (**R3**): Immutable or transient data sources (**R4**): Denormalized schemata

3 Related Work

Translating the SPARQL to SQL is a well studied technique for accessing relational databases. State-of-the-art SPARQL-to-SQL mappers like ontop [RMHC12], morph [PCS14] do not feature Update capabilities, but SPARQL Update over mapped relational databases was the subject of previous work, with two approaches extending D2R[BC06].

R2RML [DSC12] is a W3C standard for relational to RDF mappings and is supported by most of these rewriters. It utilizes `rr:TermMap` and `rr:TriplesMaps` for a row-wise mapping of relational tables into RDF.

In [GG11] the authors systematically explore the translation of SPARQL into SQL queries, covering both read and write scenarios. It is based on [CLF09], a SPARQL-to-SQL translation approach originally developed triple stores implemented on top of relational database management systems. The paper covers basic mechanisms, but does not give details on how to deal with database constraints or translate more complex, template

based `rr:TermMaps` efficiently. This paper is purely conceptual and does not feature an evaluation.

OntoAccess [HRG10] proposes its own mapping language, R3M, to express bidirectional mappings of relational databases. The approach discusses the insert and delete operations in greater detail and considers schema imposed constraints.

With D2RQ++ [Ra10] the authors propose to integrate a native triple store into their system. This triple store allows insertion of triples that are either not mapped or would violate database schema constraints. Information stored in the triple store can later be conveyed into the relational database, once the constraints are no longer violated.

D2RQ/update [EK12] is a relational schema aware SPARQL Update processor and introduces a series of optimizations in order to prevent schema constraint violations. Further, the authors aim at minimizing the number of SQL statements by grouping related triples together.

4 Approach

In this section we first introduce some basic terminology used in our mapping approach and a small exemplary mapping. Further, we introduce three different methods of updating a mapped triple store.

Our motivation behind creating those methods is our goal to cover as many usage scenarios as possible. Naively, the translation of a SPARQL query for a given update may seem straight forward due to the syntactic similarities between SPARQL and SQL. However, SPARQL updates can easily modify the data that represents its schema. In native triples stores, this does not pose a problem, as those schema triples are stored by the same means as any other triples. In the case of mapped data however, schema information appearing in a virtual graph can originate from the mapping and is tightly coupled to the schema of the data source. The schema of the data source however cannot be modified, as other application in a data landscape will rely upon it.

The data source update (Sect. 4.4) therefore deals with update operations that modify the underlying database. Mapping updates (Sect. 4.5) perform certain large scale data updates and schema modifications. The preprocessor update (Sect. 4.6) allows executing of updates regardless of schema or constraint violations, at the cost of performance.

4.1 Terminology

In this section we briefly revisit the terminology introduced in [USA12]. While this terminology is like R2RML[DSC12] building upon the relation data model, we showed in [UM16] by using views, this terminology can be applied to NoSQL stores. Further, we use the terms *relation*/*table* and *attribute*/*column* interchangeably.

Term map A *term map* is a tuple $tm = (A, ve)$ consisting of a set of relational attributes U from a single relation R and a value expression ve that describes the translation of U into RDF terms (e.g. R2RML templates for generating IRIs). Term maps are the base element of a mapping.

An example for such a *term map* is in Fig. 3 (b) the subject template `employee/{id}`. This term map creates URIs by concatenating the mappings base prefix with "employee/" and the content of the attribute "id" from the "employee" table.

Triple map A triple map *trm* is the triple (tm_S, tm_P, tm_O) of three *term maps* for generating the subject, predicate and object of a triple. All attributes of the three *term maps* must originate from the same relation R. A triple map defines how triples are actually generated from the attributes of a relation (i.e. rows of a table). R2RML allows both SQL tables and views expressed by SQL queries to be used as relation, which is consquently termed *logical table*.

Our definition conflicts with the R2RML `TriplesMap` specification, as a triple map always has one subject, one predicate and one object. R2RML on the other hands allows the definition of multiple predicate/object pairs. As this differentiation is only syntactical we use in our examples the R2RML syntax, as it more commonplace. For the rest of this paper we assume an implicit transformation into single predicate/object map `TriplesMaps` and use triple map for definition purposes.

4.2 Exemplary Mapping

As an exemplary use case we map a two-column *Employee* table Fig. 3 (a) with an R2RML mapping Fig. 3 (b). The R2RML mapping is composed out of two triples maps (i) and (ii) and consequently for each row in the table two triples will be emitted. The triple subject is in both cases build using a template and the *id* column. In case of (i) the *name* is mapped to a `foaf:name`, in case of (ii) *id* is mapped to the custom *id* property.

```
"Employee"
id|   name
__________
1 | 'Mary R.'
2 | 'James T.'
3 | 'Patricia I.'
```

(a) Employee table with three entries

```
<TriplesMapEmployee>
rr:logicalTable [rr:tableName "Employee"];
  rr:subjectMap [rr:template "employee/{id}"];
  rr:predicateObjectMap [  # (i) triple map 1
    rr:predicate foaf:name;
    rr:objectMap [rr:column "name"]];
  rr:predicateObjectMap [  # (ii) triple map 2
    rr:predicate <id> ;
    rr:objectMap [ rr:column "id"]].
```

(b) R2RML mapping of the *Employee* table.

Fig. 3: Relational table mapped by a R2RML mapping.

4.3 Update Pipeline

In the scope of this paper we will discuss only SPARQL 1.1 Update operations that are composed out of INSERTs and DELETEs, therefore not regarding graph manipulations. The concepts presented here however can readily be adapted to meet graph manipulation needs. Our approach for fulfilling the requirements determined in Sect. 2 is an extension of the already existing SparqlMap[UM16] pipeline as presented in Fig. 4. Like in read only scenarios, the first step in write scenario is to parse the request and in case of update queries, skipping over the query analysis step.

The *Mapping Binding* associates each triple of the update query with triple maps that can potentially generate such a triple. In a second step, resources occurring in more than one triple are used to further narrow down candidate triple maps, effectively precomputing the implicit SPARQL join in basic graph patterns. By not grouping triples same-subject wise we can better deal with denormalized data, as for example a table may produce multiple, different subjects based on different templates. This allows us to fulfill with requirement **R4**

In the subsequent *Value Extraction* the values to be modified are determined. Therefore RDF-Terms are compared with the corresponding term maps in order to generate a multiset of column/value pairs. In case of `rr:column` based term maps, a string representation of the RDF node is used, in case of `rr:template` term maps this string is decomposed into column-value pairs based on the template string. RDF-variables are likewise associated with term maps.

After the *Mapping Binding* and *Value Extraction* steps, for each triple we have a set of candidate `rr:TriplesMaps` and a multiset of column/value pairs, that represent potential filter conditions. So far, the insert query used the same processes as described in [UM16], where the *Value Extraction* is part of the query translation process. The *Update Method Select* Fig. 4 (1) is the first step in the SPARQL Update specfic pipeline, which triple-wise

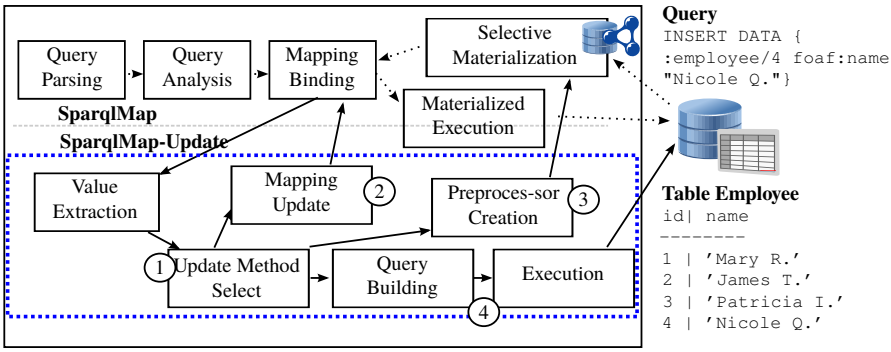

Fig. 4: The SparqlMap processing pipeline with the four proposed extension for bidirectional communication.

determines the subsequent update methods. This process is described in greater details after first describing the three update methods.

4.4 Data Source Update

The data source update Fig. 4 (4) aims at adding, modifying and deleting values or tuples from the underlying database by translating SPARQL updates into SQL DML queries and is comparable to the mechanisms of the other tools described in Sect. 3. The data source update flow can be applied to each triples, for which holds, that a bound triple map at a position wise comparison between the RDF Term and the term map at least one column of the mapped table is involved. Multiple triples that share resources and that map to the same table are grouped together into a single update operation to prevent constraint violations. With those pairs we calculate the actual update request. In case of deletes, where all columns of the mapped table are associated with either a value or a variable, rows can be deleted. If not all columns are matched and they are not used in any other term map, the value of those columns is set to NULL. In case of a constraint violation or the column is still used by another term map, the SPARQL Update is rejected.

To illustrate this using our exemplary mapping consider the SPARQL Update request `DELETE DATA { :employee/3 :id 3 }`. As here not all columns of the table are mapped, we test if setting the *id* column to NULL possible. The *id* column is however used in the subject of Fig. 3 triples map (i), therefore the update request has to be rejected.

Fig. 4 presents an example of a SPARQL Update insert query, which illustrates an other problem: Inserting a single triple results in a mapped virtual graph with two additional triples. Such insert operations can be rejected by an optional evaluation if additional triple maps are affected by the insert.

In case of inserts, constraints violations occur if an insert creates a new row with a previously used primary key or unique column value. We can either use backend specific methods, for example the MySQL `REPLACE INTO` operation, or a decide after a look-up in the database whether an insert is possible, as it targets a NULL value or if the insert has to be rejected or processed by an other update method. One of the problems arising here, is dealing with the lack of schema enforcement in the underlying data stores, caused by either not strictly formulated relational schemata or schema-less data sources. In those cases no schema violations can be detected and potentially inconsistent data may be inserted. We therefore propose to either validate the insert query with a Shapes Constraint Language (SHACL)[2] implementation or a backend specific validation tool, which allows us to fulfill with **R1**.

[2] `https://www.w3.org/TR/shacl/`

4.5 Mapping Update

The alternative to altering the mapped data source is updating the mapping itself. This can be a more efficient way to remove or update all data originating from specific columns/tables or term maps. Consider the following two queries: (a) `DELETE { ?s ?p ?o } WHERE {?s foaf:name ?name}` and (b) `DELETE {?s foaf:name ?o} INSERT {?s rdfs:label ?o}WHERE {?s foaf:name ?o}`. Query (a) deletes everything which has a `foaf:name`, in RDB-to-RDF with Data Source updates on our exemplary mapping, this will delete all rows from the employee table. Query (b) would in the RDB-to-RDF Data Source Update scenario move cell values to another column. As the property `rdfs:label` is not mapped in our exemplary scenario, the query would fail.

Executing a Mapping Update is a modification of the R2RML mapping, by means of SPARQL 1.1 Update queries over the virtual graph. A Mapping Update instead of an Data Source Update can be executed if for a specific triple all associated column/value pairs have at the value position only variables. In other words, a query expressing an equivalent Data Store Update contains no `WHERE` expressions. In case of delete statements, triple maps of this triple pattern can simply be deleted. In case of an DELETE/INSERT queries, constant term maps are replaced with an updated version. In any other case either an exception has to be raised and the SPARQL Update request has to be rejected, or alternatively be processed by one of the other update methods.

In our example a mapping update by query (a) would delete all triple maps of the employee mapping. Query (b) executed as mapping update would replace the constant term maps at the predicate position of Fig. 3 (b) term map (i).

4.6 Preprocessor Update

SparqlMap[UM16] features a lean RDF materialization pipeline for answering queries. Instead of modifying the data source, incoming SPARQL Update queries are stored and integrated into the mapping process. This integration can be performed by rewriting queries, which can dramatically increases query complexity, or by pushing and removing data from materialized graphs, which prohibits join pushing and increases data materialization. As a consequence, we conjecture that this approach is only suitable for small data set sizes, experiencing only a few updates.

This approach is comparable to the idea of using a dedicated overflow triple store in D2RQ++ [Ra10]. The preprocessor Update however allows the execution of updates, that previously were rejected.

The query `DELETE DATA { :employee/3 :id 3 }`, which failed on the data source update can now be executed. In consequence, during the materialization phase of a query this triple is filtered before its RDF terms can be bound to variables.

4.7 Update Method selection

In order to deal with scenario specific requirements, preferences can be configured and methods be excluded. For example in Sect. 2 scenario 2 **R3** defines that the data sources is immutable. Therefore, first the Mapping Update method and as a fall back the Preprocessor Update will be used. In contrast, scenario 1 demands handling of larger data volumes, ruling out the Preprocessor Update method. In this scenario, Data Set Update is the only applicable method, as the mapping is immutable because of it is ties to the logic of the search component. Update queries that cannot be processed by the Data Set Update would be rejected.

5 Implemenation and Evaluation

Our prototypical implementation of the Data Set Update pipeline in SparqlMap is available on github[3]. SparqlMap can be used acting as both command line tool or SPARQL endpoint. As SparqlMap builds on Apache Jena[4] as RDF framework and Apache MetaModel[5] as a pseudo-relational view on both relational and NoSQL DBMS, we therefore can fulfill requirement **R1**.

We evaluated our approach using a subset of the data generated by the Berlin SPARQL Benchmark (BSBM), namely instances of the *review* class on a Intel Xeon E3-1220 server with 8 GB of RAM with all data on a SSD.

The data set of BSBM is synthetic and features three different use cases, all of which represent interactions with a product database. A thorough description of the benchmark can be found in [BS09]. In our test we want to examine the performance characteristics of SparqlMap with varying insert bulk sizes and tests the systems performance in a use case similar to Sect. 2 scenario 1. We therefore created an CSV representation of the SQL output for the *review* instances with 100,000 entries. Mapped via R2RML mapping, this table represents approx. 1.08 million (virtual) triples. During the test we double its size, using varying sizes of bulk inserts. We run the benchmark with inserts of 1, 10, 100 and 1000 Reviews, with each review being represented on average by 10.8 triples.

Times were measured from withing SparqlMap, consequently HTTP overhead is ignored. We measured the average execution time for the four different steps of the process Fig. 4 and depict them in Fig. 5.

In summary, insert operations scale linearly in all four steps of the mapping process. In this scenario, inserting 10 triples takes about 1 millisecond. Notable in Fig. 5 is how the query parsing step dominates query runtime. In all cases approximately this step takes 10 times as

[3] `http://github.com/tomatophantastico/sparqlmap`
[4] `http://jena.apache.org`
[5] `http://metamodel.apache.org`

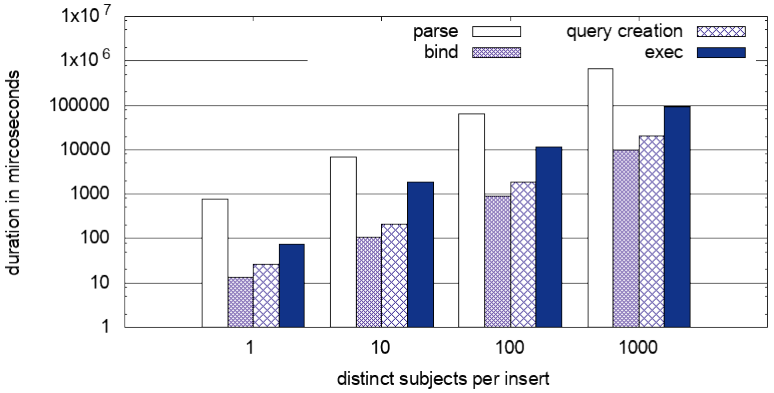

Fig. 5: Query execution time of insert queries into the BSBM `review` table on a log scale.

long as all the other steps combined. Consequently, in these simple insert cases, there is little room for improving the performance without modifying the Apache Jena SPARQL parser. We therefore consider **R2** to be fulfilled.

6 Conclusion

With our evaluation we demonstrate that SparqlMap Update scales well with increasing insert size and further optimizations are not required. We therefore focus on implementing the proposed multi-level validation and update-execution regime. Further, we will in future version support update operations over `rr:sqlViews`, enables us to benchmark the system using the full BSBM read/write scenario. Additionally, we plan on implementing the other update methods and benchmark them as well.

Acknowledgments This work was partly supported by a grant from the German Federal Ministry of Education and Research (BMBF) for the LEDS Project under grant agreement No 03WKCG11C.

References

[ADR06] Auer, Sören; Dietzold, Sebastian; Riechert, Thomas: OntoWiki–a tool for social, semantic collaboration. In: International Semantic Web Conference. Springer, pp. 736–749, 2006.

[BC06] Bizer, Christian; Cyganiak, Richard: D2r server-publishing relational databases on the semantic web. In: Poster at the 5th international semantic web conference. volume 175, 2006.

[BR16] Beretta, Francesco; Riechert, Thomas: Collaborative research on academic history using linked open data: A proposal for the heloise common research model. CIAN-Revista de Historia de las Universidades, 19(1):133–151, 2016.

[BS09] Bizer, Christian; Schultz, Andreas: The Berlin SPARQL Benchmark. IJSWIS, 2009.

[CLF09] Chebotko, Artem; Lu, Shiyong; Fotouhi, Farshad: Semantics preserving SPARQL-to-SQL translation. Data & Knowledge Engineering, 68(10):973–1000, 2009.

[DSC12] Das, Souripriya; Sundara, Seema; Cyganiak, Richard: R2RML: RDB to RDF Mapping Language. Technical report, 2012.

[EK12] Eisenberg, Vadim; Kanza, Yaron: D2RQ/update: updating relational data via virtual RDF. In: Proceedings of the 21st International Conference on World Wide Web. ACM, pp. 497–498, 2012.

[Fr13] Frischmuth, Philipp; Auer, Sören; Tramp, Sebastian; Unbehauen, Jörg; Holzweißig, Kai; Marquardt, Carl-Martin: Towards linked data based enterprise information integration. In: Proceedings of the 2013th International Conference on Semantic Web Enterprise Adoption and Best Practice-Volume 1106. CEUR-WS. org, pp. 26–34, 2013.

[GG11] Garrote, Antonio; García, María N Moreno: RESTful writable APIs for the web of Linked Data using relational storage solutions. In: WWW 2011 Workshop: Linked Data on the Web (LDOW2011). 2011.

[HRG10] Hert, Matthias; Reif, Gerald; Gall, Harald C.: Updating Relational Data via SPARQL/Update. In: Proceedings of the 2010 EDBT/ICDT Workshops. EDBT '10, ACM, New York, NY, USA, pp. 24:1–24:8, 2010.

[PCS14] Priyatna, Freddy; Corcho, Oscar; Sequeda, Juan: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using Morph. In: Proceedings of the 23rd international conference on World wide web. ACM, pp. 479–490, 2014.

[Ra10] Ramanujam, Sunitha; Khadilkar, Vaibhav; Khan, Latifur; Seida, Steven; Kantarcioglu, Murat; Thuraisingham, Bhavani: Bi-directional translation of relational data into virtual RDF stores. In: Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on. IEEE, pp. 268–276, 2010.

[RMHC12] Rodrıguez-Muro, Mariano; Hardi, Josef; Calvanese, Diego: Quest: efficient SPARQL-to-SQL for RDF and OWL. In: ISWC. 2012.

[SGP08] Schenk, Simon; Gearon, Paul; Passant, Alexandre: SPARQL 1.1 Update. Technical report, W3C, 2008. Published online on October 14th, 2010 at `http://www.w3.org/TR/2010/WD-sparql11-update-20101014/`.

[UM16] Unbehauen, Jörg; Martin, Michael: Executing SPARQL queries over Mapped Document Store with SparqlMap-M. In: Proceedings of the 12th International Conference on Semantic Systems. ACM, pp. 137–144, 2016.

[USA12] Unbehauen, Jörg; Stadler, Claus; Auer, Sören: Accessing relational data on the web with sparqlmap. In: Joint International Semantic Technology Conference. Springer Berlin Heidelberg, pp. 65–80, 2012.