# Using Berlin SPARQL Benchmark to Evaluate Relational Database Virtual SPARQL Endpoints

Milos Chaloupka [a,*], and Martin Necasky [a]
[a] *Faculty of Mathematics and Physics, Charles University, Czech Republic*
*E-mails: chaloupka@ksi.mff.cuni.cz, necasky@ksi.mff.cuni.cz*

**Abstract.** The RDF is a well documented and popular format for publishing structured data on the web. It enables consuming data without the knowledge of how the data is internally stored. There are already several native RDF storage solutions that provide SPARQL endpoint. However, they are not widely adopted. It is still more common to store data in relational databases.

There are already several implementations of virtual SPARQL endpoints over a relational database. However, their consistent evaluation is missing. On the other hand, for the native RDF storages there is a state of the art benchmark. In this paper, we show how this well defined benchmark can be used to evaluate virtual SPARQL endpoints.

Keywords: RDB2RDF, R2RML, BSBM, SPARQL

## 1. Introduction

The RDF [1, 2] is a well documented and popular format for publishing structured data on the web. It enables consuming data without the knowledge of how the data is internally stored. Moreover, it supports the evolution of schemas without requiring changes in services which consume the data. One of the features is the ability to provide a web service which can be then used by consumers to query the data using SPARQL language [3], so called SPARQL endpoint.

There are already several native RDF storage solutions that provide SPARQL endpoint. However, they are not widely adopted. It is still more common to store data in relational databases. For example, the top 4 database engines in the ranking of database engines[1] are primarily relational. There is no database engine with native RDF support in top 30 engines.

There is a difference between how data is usually stored and how it should be ideally published. There has been various attempts to close this gap and a W3C Working Group has been established[2] several years ago. The group has produced several standards [4, 5] which specify the transformation of relational data to RDF representation.

However, the transformation of relational data may not be always an ideal solution. Especially, when a SPARQL endpoint is needed then it is needed to load the transformed data into some RDF storage. In that case, it would be ideal to build a virtual SPARQL endpoint directly over a relational database. In that case, the RDF data is not materialized before querying. This is achieved by transforming a given SPARQL query to a corresponding relational query and then transforming the relational result to its RDF equivalent. This allows customers to treat the underlying relational database as if it was a native RDF storage without the need to store the dataset in two systems and synchronise between them.

There are already several implementations of virtual SPARQL endpoints over a relational database. However, their consistent evaluation is missing. There is

*Corresponding author. E-mail: chaloupka@ksi.mff.cuni.cz.
[1] See https://db-engines.com/en/ranking, visited February 2020
[2] See https://www.w3.org/2001/sw/rdb2rdf/, visited March 2020

a state of the art benchmark for the native RDF storages. In this paper, we show how this well defined benchmark can be used to evaluate virtual SPARQL endpoints.

### *1.1. Contribution*

In this paper, we use a state of the art SPARQL benchmark (Berlin SPARQL Benchmark [6]) to evaluate existing approaches to a virtual SPARQL endpoint over a relational database. In particular, the paper has the following contributions:

1. We define aspects that are used to evaluate a virtual SPARQL endpoint solution.
2. We define the mapping from a relational database to an RDF dataset for Berlin SPARQL Benchmark and identify the limitations of such mapping.
3. We identify the gaps of Berlin SPARQL Benchmark framework and provide what is needed to use the framework to evaluate a virtual SPARQL endpoint solution.
4. We evaluate existing virtual SPARQL endpoint solutions.
5. We provide all required information, scripts and tools to repeat the evaluation again.

In section 2, we describe the Berlin SPARQL Benchmark. In section 3, we explain how a mapping from a relational database to an RDF dataset can be defined, also describing the specific mapping. In section 4, we describe the existing virtual SPARQL endpoint solutions and how we have selected them. In section 5 we list related work to virtual SPARQL endpoint benchmarks. In section 6, the performed evaluation is described and the evaluation results are provided.

## 2. Berlin SPARQL Benchmark

The Berlin SPARQL Benchmark (BSBM) [6] is the state of the art benchmark for SPARQL endpoints. The benchmark defines an RDF dataset and SPARQL queries to be executed. Moreover, it defines a semantically identical relational dataset and provides a tool to generate data and a test driver to run the benchmark over a SPARQL endpoint. The latest version of the benchmark is thoroughly described in [7].

BSBM is built around an e-commerce use case - offering a set of products by various vendors and different customers that wrote a review about the products. The products are categorized using their types and features. In the RDF representation, there are 8 classes: Product, Product Type, Product Feature, Producer, Vendor, Offer, Person and a Review. On Figure 1 there is a sample representation of a single product. In the relational representation, there are 8 tables representing the classes and two additional tables. One table contains the relationships between products and product types and the other one between products and product features. All tables are listed with their columns in Figure 2.

```
dataFromProducer1411:Product1435443
 rdf:type bsbm:Product;
 rdf:type bsbm-inst:ProductType1342;
 rdfs:label "Canon Ixus 20010";
 rdfs:comment
  "Mit ihrer hochwertigen Verarbeitung, innovativen
  Technologie und faszinierenden Erscheinung
  verkörpern Digital IXUS Modelle die hohe Kunst
  des Canon Design.";
 bsbm:producer bsbm-inst:Producer001411;
 bsbm:productFeature bsbm-inst:ProductFeature3432;
 bsbm:productFeature
  bsbm-inst:ProductFeature103433;
 bsbm:productFeature
  bsbm-inst:ProductFeature990433;
 bsbm:productPropertyTextual1 "New this year.";
 bsbm:productPropertyTextual2
  "Special Lens with special focus.";
 bsbm:productPropertyNumeric1 "1820"^^xsd:Integer;
 bsbm:productPropertyNumeric2 "140"^^xsd:Integer;
 bsbm:productPropertyNumeric3 "17"^^xsd:Integer;
 dc:publisher dataFromProducer1411:Producer1411;
 dc:date "2008-02-13"^^xsd:date.
```

Fig. 1. Sample RDF representation of a product

```
ProductFeature(nr, label, comment, publisher,
 publishDate)
ProductType(nr, label, comment, parent, publisher,
 publishDate)
Producer(nr, label, comment, homepage, country,
 publisher, publishDate)
Product(nr, label, comment, producer, propertyNum1,
 propertyNum2, propertyNum3, propertyNum4,
 propertyNum5, propertyNum6, propertyTex1,
 propertyTex2, propertyTex3, propertyTex4,
 propertyTex5, propertyTex6, publisher,
 publishDate)
ProductTypeProduct(product, productType)
ProductFeatureProduct(product, productFeature)
Vendor(nr, label, comment, homepage, country,
 publisher, publishDate)
Offer(nr, product, producer, vendor, price,
 validFrom, validTo, deliveryDays, offerWebpage,
 publisher, publishDate)
Person(nr, name, mbox_sha1sum, country, publisher,
 publishDate)
Review(nr, product, producer, person, reviewDate,
 title, text, language, rating1, rating2, rating3,
 rating4, publisher, publishDate)
```

Fig. 2. Relational dataset for BSBM

The dataset can be scaled using one variable: the count of products. According to the number of products, the count of other entities is defined. As the product count increases, the total amount of triples increases almost in a linear fashion. For example, million of triples is in the dataset with 2785 products.

As soon as the RDF dataset is available using a SPARQL endpoint, it is possible to run the benchmark using a provided test driver. The test driver supports specifying custom queries, but it is also possible to use a predefined use case. The Explore use case illustrates queries executed when a consumer is navigating in an e-commerce system and searching for a product. This use case is usually used to benchmark the performance of a SPARQL endpoints. The Explore use case consists of 12 queries (defined in [7]):

1. Find products for a given set of generic features.
2. Retrieve basic information about a specific product for display purposes.
3. Find products for a given more specific set of features.
4. Find products matching two different sets of features.
5. Find products that are similar to a given product.
6. Find products having a label that contains a specific string.
7. Retrieve in-depth information about a specific product including offers and reviews.
8. Give me recent German reviews for a specific product.
9. Get information about a reviewer.
10. Get offers for a given product which fulfill specific requirements.
11. Get all information about an offer.
12. Export information about an offer into another schemata.

The complete Explore use case query mix is the following sequence of 25 queries: 1, 2, 2, 3, 2, 2, 4, 2, 2, 5, 7, 7, 5, 7, 7, 8, 9, 9, 8, 9, 9, 10, 10, 11 and 12. The query 6 is no longer used in the query mix. The queries contain parameters assigned randomly during an execution. So, although query 2 is present in the query mix six times, it retrieves information about different products. The query mix is designed to focus on some queries more than others to match a real e-commerce use case.

## 3. Mapping from relational database to RDF dataset

To present a relational database as an RDF dataset, one needs to define a mapping. The mapping can be either created automatically or defined by a user. The latter usually provides a precise way to declare how the RDF representations should look like while the automatic approach provides a fast way to get some RDF data. Several tools provided the ability to transform a relational query to an RDF dataset even before a standard for mapping existed [8]. For example, the state of the art tool D2RQ [9, 10] used their proprietary mapping language, and the tool provided even the ability to create a virtual SPARQL endpoint. Later, the W3C RDB2RDF Working Group created two standards: Direct Mapping [4] and R2RML [5].

The Direct Mapping is the standard for automatically defined mapping. It works in a way that every table definition is transformed to an RDF class. Every row in the table represents an RDF resource which is an instance of the corresponding class. An RDF triple is generated for each cell. The primary key of the table is used to generate the subject of the triple. The predicate is generated according to the column name, and the object is usually a literal created from the value in the cell. An IRI valued object is created if and only if the column has a foreign key constraint.

Automatically created mapping can be used to generate an RDF dataset if there are no requirements on the representation. That is useful especially when the RDF data are then immediately processed using tools supporting SPARQL. Therefore the transformation to a final RDF form is done with these tools. If we want to publish the RDF data directly, it is more convenient if the mapping can be customized in the mapping. When a user wants to create a virtual SPARQL endpoint over a relational database, it is essential to map the relational data directly to the final RDF form.

### *3.1. R2RML*

The R2RML [5] is a language to express user-defined mappings. The standard defines a set of mapping options and standardizes how a relational dataset should be transformed using an R2RML mapping.

The R2RML mapping declares how the whole RDF dataset can be generated from a relational database. It defines which relational queries should be executed - so called logical tables. The logical table can be either the name of a relational table or any user-defined SQL

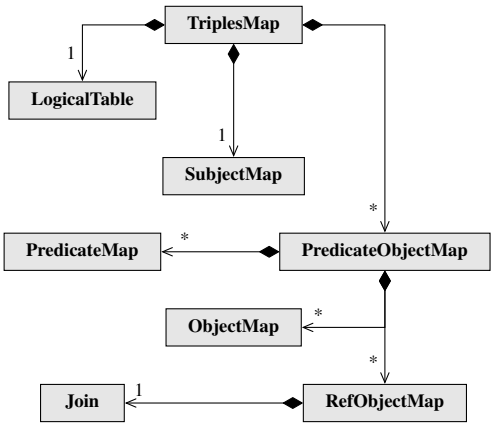


Fig. 3. An overview of R2RML triples map [5]

SELECT statement. After executing a relational query, the corresponding mapping declares a set of rules to generate RDF triples based on every returned row in the result.

Triples generated from a single result row share the same subject defined by subject map and the generated triples are based on individual predicate object maps. An individual generated triple is based on the predicate-object map. It defines how to generate the predicate (using the predicate map) and the object (using the object map). The generated RDF term can be defined either as a constant, a column value or a template. The template prescribes how to merge one or more column values into a text value. That is essential to generate IRIs. For literal RDF terms, it is possible to declare their data type or language. However, both the data type and language cannot be generated using the underlying SQL query. They are constant for all terms generated from the particular mapping. If no data type and language is declared and the value is generated from column values then the data type is automatically chosen according to the type of the relational column.

There is also an alternative way to generate IRIs as objects. It is possible to define the value as a reference to a different triples map. In that case, the object IRI is generated using the referenced triples map subject. However, the referenced triples map subject is based on a different logical table. Therefore, for this particular mapping it is needed to execute relational queries for the logical tables of both triples map and using the defined join (as part of the ref object map) and to pair their rows together and allows to generate all parts of the RDF triple.

The whole triples map schema is shown on Figure 3. To generate the complete RDF dataset from a relational database, it is needed to follow the rules for all triples map. A simplified way of using R2RML mapping to generate an RDF dataset from a relational database is shown on Figure 4. It does not cover all possibilities of the R2RML standards, but it provides the main idea of generating triples using an R2RML mapping.

```
foreach(tm : mapping.triplesmaps)
{
 // Generate triples using PredicateObjectMap
 query = getQuery(tm.logicalTable)

 foreach(row : execute(query))
 foreach(pom : tm.predicateObjectMaps)
 foreach(pm : tm.predicateMaps)
 foreach(om : tm.objectMaps)
 {
  s = generateRDFTerm(row, tm.subjectMap)
  p = generateRDFTerm(row, pm)
  o = generateRDFTerm(row, om)
  yield return RDFTriple(s, p, o)
 }

 // Generate triples using RefObjectMap
 foreach(pom : tm.predicateObjectMaps)
 foreach(rom : pom.refObjectMaps)
 {
  query = getQuery(tm.logicalTable, rom)

  foreach(row : execute(query))
  foreach(pm : tm.predicateMaps)
  {
   s = generateRDFTerm(row, tm.subjectMap)
   p = generateRDFTerm(row, pm)
   o = generateRDFTerm(row,
      rom.parentTriplesMap.subjectMap)
   yield return RDFTriple(s, p, o)
}}}
```

Fig. 4. Generating RDF triples using R2RML

The usage of an R2RML mapping to create a virtual SPARQL endpoint does not have any standardized approach. The only requirement is that the virtual SPARQL endpoint has to be transparent for users. That means that a user should not be able to distinguish between a virtual SPARQL endpoint over a relational database and a SPARQL endpoint over an RDF dataset generated from the same relational database.

### *3.2. Mapping for Berlin SPARQL Benchmark*

Although the BSBM toolset offers the capability to generate both relational and RDF dataset, there is not defined any mapping between these two datasets. Therefore, before any evaluation it is needed to define the corresponding R2RML mapping. We have restricted us to create an R2RML mapping without using user-defined SQL statements as logical tables. It allows the tools to perform any available optimizations as it has complete information from the relational schema

about keys and indexes. This restriction is specific to virtual SPARQL endpoints. There is no need for a similar restriction when a whole RDF dataset is generated from a relational database.

The first issue is that using the R2RML standard it is not possible to declare the RDF type and language of a literal other than a constant. However, as shown on Figure 2 there is a table `Review` with columns `text` and `language`. These two columns should be combined to create a literal. This is not supported in the R2RML standard as the language for any literal has to be specified as a constant in the mapping file. There is a possible workaround - one can create a triples map for every language possible and as a logical table there can be a SQL query filtering the rows by language. That would break the restriction we set on us. Moreover, that would be hard to maintain as it would require a specific mapping for every possible language. Therefore, we have decided to dismiss this functionality and treat all languages as "en".

The other issue is related to another limitation of R2RML. Using the ref object map, it allows to reference a TriplesMap for object values using a defined join condition. However, the join condition is expressed as a set of columns that should be used for joining the corresponding logical tables. This corresponds to the concept of foreign join conditions in relational databases. It usually represents relationships with cardinality 1:1 or 1:N. To represent cardinality M:N, an extra table can be used - a table that contains keys to both corresponding tables. For this relationship, there is no specific support in R2RML. It can be modeled by a specific triples map for the relationships table.

However, this approach works under a condition, that the relationships table contains all information needed to construct subject values. Only object values can be defined as a reference to another triples map. Subject values are constructed only using a logical table. Based on our restriction, that means that subject values can be constructed using only the columns in the relationship tables.

This condition is not met in the BSBM dataset. For example, a product instance (see Figure 1) is generated using three tables. The table `Product` is used to generate most of the properties. The subject is generated using the columns `publisher` and `nr`. However, the column `publisher` is only in this table and not in the other ones containing data for product instances: table `ProductTypeProduct` which contains the relationship with product types and a table `ProductFeatureProduct` which contains the relationship with product features. To solve this issue we decided to modify the BSBM tooling, so the `publisher` column is not a part of IRIs. As an example, a triple from Figure 1 is modified to have a subject `bsbm-inst:Product1435443` and the object for predicate `dc:publisher` is modified to `bsbm-inst:Producer1411`. The only difference is that the namespaces are not prefixed by publishers and the global `bsbm-inst` is used.

One of our key aimed contributions is to make the evaluation repeatable. Therefore, all changes made to the BSBM benchmark are published in a github repository.[3] There is also the final mapping file in the same repository.[4]

## 4. Evaluated virtual endpoints

To identify the tools for the evaluation, we have used Google Scholar[5] to find mentions of virtual SPARQL endpoints in last three years.

There are various approaches to the SPARQL virtual endpoint implementations. One of the first tools is the D2RQ Platform [9, 10]. However, this tool has not been updated for several years.[6] It was initially created before the W3C standards were introduced and therefore the tool does not support the R2RML mapping. The tool transforms a given SPARQL query into multiple relational queries and then processes the results in memory. This causes memory and performance issues. Because the tool has not been updated for several years and it does not have a support of R2RML standard, we have decided not to include it in our evaluation.

There was an unrelated research focused on SPARQL to SQL translation by Chebotko [11, 12]. This approach was focused to use only a single relational query but it required to have a specific relational schema - a single table with three columns: one for subject, one for predicate and one for object. The Chebotko's approach was later used to create Ultrawrap [13]. It prepares an "R2RML view", a single subquery which returns all triples according according to the R2RML file. That is a union of multiple select statements, one for every predicate object mapping.

[3] It can be found in our fork of the BSBM repository: https://github.com/mchaloupka/bsbm-r2rml, visited February 2020

[4] The file can be accessed on https://github.com/mchaloupka/bsbm-r2rml/blob/develop/src/main/dist/rdb2rdf/mapping.ttl, visited February 2020

[5] See https://scholar.google.com/, visited November 2019

[6] See http://d2rq.org/, visited March 2020

The Ultrawrap solution has been commercialized and the author did not wanted to make it available for research purposes. Therefore, we were not able to evaluate this tool.

### 4.1. Morph

The Morph translation algorithm [14] is also based on Chebotko's approach. They redefine the approach so it does not require to have the data prepared in an "R2RML view" as it is done in Ultrawrap. The main idea is that for every basic graph pattern it adds a subquery. The subquery is similar to the mentioned "R2RML view" but it is optimized. The view is based only on the mappings that may not be immediately filtered out. For example, if the basic graph pattern defines the predicate of matching triples the generated subquery will not use the mappings that produce a different predicate. After that, various optimization of the relational are applied.

### 4.2. Ontop

Lately, the Ontop framework has added support to provide virtual SPARQL endpoint [15–17]. The query translation algorithm uses the data ontology to generate relational queries. The ontology is based on the mapping and the schema of the underlying relational database. Moreover, the tool uses several SQL optimization methods to further optimize the queries. The tool has been adopted in various academic and industrial use cases.[7]

### 4.3. SparqlMap

Previously mentioned solutions Ultrawrap, Morph and Ontop use a simple SPARQL variable representation in SPARQL queries (described in [18]). That means that a SPARQL variable value is represented in the result of a relational query using a single column. There may be additional columns to provide type or language but the actual value is stored in a single column. The authors of SparqlMap [19] have chosen a different approach. For the variable representation, they represent a variable using multiple columns - one for string, one for boolean, one for numeric and one for date time values. Moreover, they use one or more additional columns to represent IRI values. As evaluated in [18], this approach gives the ability to correctly handle various corner cases in the SPARQL to SQL transformation but for a performance cost.

[7] See https://ontop-vkg.org/research/#projects, visited March 2020

### 4.4. EVI

The EVI solution [20] has a goal to produce SPARQL queries that are as close to manually written relational queries as possible. The main concept is based on value binders - another output of the transformation of a given SPARQL query. The value binders represent how the SPARQL result for a given SPARQL query should be reconstructed from the relational result of the corresponding relational query. A simple variant of that approach was already used in the D2RQ solution but it was processed in memory. The EVI solution processes in memory only single rows from relational results as every SPARQL query is transformed to a single relational query. However, the value binders are used even during the transformation algorithm - as a SPARQL variable can be potentially represented by any columns, the transformation algorithm is using them to understand the variables at any point of the transformation.

## 5. Related work

As mentioned in section 4, one of the pioneering transformation solutions is the D2RQ Platform. The tool was later compared against native RDF storages using the Berlin SPARQL Benchmark [21] and using a real world scenario [22].

There are various surveys about approaches for transformation of relational database into RDF [23–25]. However, these surveys are focused mainly on the supported features and how the tools internally work.

We have not found any benchmark of the available solutions, except the ones that were published together with them. The Ultrawrap solution was compared to native RDF storages [13]. The Morph and SparqlMap solution was compared to the D2RQ solution [14, 19]. The Ontop solution was compared to native RDF storages [16]. The EVI solution was compared to the Ontop solution and to a native RDF storage [20].

However, we have not identified any work that compares these solutions against each other. Moreover, the solutions does not even use a unified approach to evaluation. Although, some of them use the BSBM benchmark, the details of how the benchmark was used is not described. As mentioned in section 3.2, the BSBM

tooling cannot be used as it is so they had to modify the benchmark somehow.

## 6. Evaluation

In this paper, the evaluation is focused only on the aspects which can be evaluated using the described BSBM tooling. We do not cover support, soundness or performance for SPARQL queries and parts of R2RML standard that are not part of the BSBM query mix.

The evaluation consists of the following aspects:

- Usability - What does it take to deploy, configure and run the virtual SPARQL endpoint.
- Completeness - Support for the selected use case.
- Soundness - For the supported queries, verify that they provide correct results.
- Performance - How long does it take to execute the selected use case.

The following subsections describe the details of evaluating the individual aspects and the evaluation results.

### *6.1. Usability*

The usability evaluation evaluates how hard it is to get the tool running.

For every tool, we needed to get a virtual SPARQL endpoint running using an R2RML file. Ideally, it should be clear from the provided documentation how to run the tool and it should not involve multiple steps. Moreover, it should be correctly documented. It is an advantage, if there is a living community around the tool, so if an issue is raised, there is someone able to help.

We assign a point for every satisfied aspect. The aspects were evaluated as follows:

- Quick start guide - an available quick start guide - at least a short documentation describing how to start the tool should be available.
- Documentation - a complete documentation as it would be expected from a mature product, describing various use cases, referencing other required tools and standards.
- Active community - as all tools are available on GitHub,[8] we give this point if at least 100 users are watching the repository.
- Easy to try - following the provided user guide, it should not require any technical skills to execute a sample query.
- Easy to use - an extra point for the tool, if it is not only easy to evaluate a sample query but it also quickly provides the SPARQL endpoint given the connection string and the R2RML mapping. Ideally, there should be a single command to start the endpoint.

The **Morph** solution is available as open source.[9] There is a guide which shows how to run a single SPARQL query. For each query execution, it is needed to prepare a file containing the following information:

- Database connection
- Path of the R2RML file
- Output path for the results file
- Path of a file containing the SPARQL query

With such a file it is possible to execute a Morph tool to execute the SPARQL query over a relational database.

We have not found out how to create a running SPARQL endpoint. There is no comprehensive documentation, but according to the code it seems to be an unsupported feature. There is almost no community, the tool seems to be maintained by a single person (F. Priyatna) but according to the issue tracking, it seems that there are few active users.

The **SparqlMap** solution is also available as open source.[10] There is a documentation explaining how to run a SPARQL endpoint. The SPARQL endpoint can be started by a single command with several parameters specifying path to the R2RML mapping and the access to the database. It worked without any significant issues except the fact that the SPARQL endpoint runs on a slightly different address than what is documented.[11]

The community situation is similar to the Morph community. The tool seems to be maintained by a single person (J. Unbehauen). According to the issue tracking, it seems that there are few active users.

[8] See https://github.com/, visited March 2020

[9] See https://github.com/oeg-upm/morph-rdb, visited February 2020

[10] See https://github.com/tomatophantastico/sparqlmap, visited February 2020

[11] The documentation declares that the endpoint to be accessible on `localhost:8080/sparql` but the endpoint is accessible on `http://localhost:8090/api/ROOT/sparql`.

The **Ontop** solution is available as open source.[12] There is also a portal[13] where the tool is documented.

The tool originally required a proprietary mapping format. They provided a Protege[14] plugin to do the conversion. The endpoint was actually an extension to Sesame Workbench.[15] A user had to define a new Sesame repository as a virtual RDF store and provide a path to the files which contains the mapping.

However, starting from version 3[16] they support R2RML directly and also provide CLI tool to perform the mapping format conversions. It also provides the ability to start an endpoint with any additional tools. Moreover, a docker container is provided.[17]

The Ontop solution has the most living community from the evaluated tools. There are several people (mainly from a research group at the Free University of Bozen Bolzano) contributing to the project. According to the issue tracking, it seems that there are several active users. Moreover, the repository has a bigger traffic than other solutions so it seems that there is the biggest development.

Even the **EVI** solution is available as open source.[18] It is available as a library, but there is an additional repository which provides the virtual SPARQL endpoint. The SPARQL endpoint can be started by running a single command with several parameters specifying for example the path to the R2RML mapping and the access to the database. It is also possible to use a configuration file to provide these values.

There is no community around this tool. The tool is developed and maintained by a single person (M. Chaloupka) and there are almost no issues from other users.

The **summary** of the points given to individual tools is shown in the table 1. The Ontop solution has received more points than any other solution.

### 6.2. Completeness

The completeness evaluation measures whether the tool is able to execute the Berlin SPARQL Benchmark queries as described in section 2. For the executed queries, we do not consider whether the returned results are correct, only whether the tool is able to process the mapping, receive a query using an HTTP-based SPARQL endpoint, connect to a database and return some results. Moreover, in terms of completeness we care also which ones of the main relational database engines are supported by the tool. For the evaluation, we consider Oracle,[19] MS SQL[20] and MySQL[21] database engines. They were selected based on their ranking.[22]

Table 1
The usability evaluation results

| | Morph | SparqlMap | Ontop | EVI |
|---|---|---|---|---|
| Quick start guide | ✓ | ✓ | ✓ | ✓ |
| Documentation | ✗ | ✓ | ✓ | ✗ |
| Active community | ✗ | ✗ | ✓ | ✗ |
| Easy to try | ✓ | ✓ | ✓ | ✓ |
| Easy to use | ✗ | ✓ | ✓ | ✓ |
| Total | 2 | 4 | 5 | 3 |

For completeness, we decided to give points in a way that reflects the ability to execute the BSBM queries using an HTTP-based SPARQL endpoint on a selected relational database. For every supported query from the selected BSBM use case we give 1 point for every suppported relational database engine and 1 more point if the solution provides an HTTP-based SPARQL endpoint out of the box. So, for example, if a solution provides an HTTP-based SPARQL endpoint and supports 4 queries on 2 database engines we will give the solution 12 points.

As already mentioned, we have not found a way of how to start an HTTP-based SPARQL endpoint using the **Morph** solution. Therefore the BSBM tooling was not able to access it. However, we were able to try the queries manually. The solution supports various database engines (including MS SQL, Oracle and MySQL). We were able to execute a simple SELECT query with WHERE clause containing only one basic graph pattern. However, all the queries in the selected BSBM use case failed to execute. We reported this issue.[23]

[12] See https://github.com/ontop/ontop, visited February 2020
[13] See https://ontop-vkg.org/, visited February 2020
[14] See https://protege.stanford.edu/, visited February 2020
[15] See https://rdf4j.org/, visited February 2020
[16] See https://github.com/ontop/ontop/releases/tag/ontop-3.0.0, visited February 2020
[17] See https://hub.docker.com/r/ontop/ontop-endpoint, visited February 2020
[18] See https://mchaloupka.github.io/EVI/, visited February 2020
[19] See https://www.oracle.com/database/, visited February 2020
[20] See https://www.microsoft.com/en-us/sql-server/sql-server-2019, visited February 2020
[21] See https://www.mysql.com/, visited February 2020
[22] See https://db-engines.com/en/ranking_trend/relational+dbms, visited February 2020
[23] See https://github.com/oeg-upm/morph-rdb/issues/34, reported June 2018, in February 2020 the issue is still opened

Table 2
Completeness evaluation results

| | Morph | SparqlMap | Ontop | EVI |
|---|---|---|---|---|
| HTTP endpoint | ✗ | ✓ | ✓ | ✓ |
| RDBS support | 3 | 3 | 3 | 1 |
| Supported queries | 0 | 0 | 12 | 12 |
| Points | 0 | 0 | 48 | 24 |

The **SparqlMap** solution supports various database engines (including MS SQL, Oracle and MySQL). However, we encountered a similar issue as with the Morph solution. We were able to execute a simple query but we were not able to execute any complex query that is used in the BSBM use case. We reported this issue.[24]

The **Ontop** solution supports various database engines (including MS SQL, Oracle and MySQL). We were able to execute all twelve queries from the selected use case.

The **EVI** solution supports only one database engine right now - MS SQL. However, using this set up we were able to execute all queries from the selected use case.

The **summary** of the completeness evaluation is shown in Table 2. According to this table, the only two solutions usable for the BSBM benchmark use case are Ontop and EVI. The Ontop solution provides a wider range of database engines supported, so from the completeness perspective Ontop gets the best results.

### *6.3. Soundness*

The soundness evaluation measures correctness of the results of particular queries from the BSBM use case. As described in the completeness evaluation, the Morph and SparqlMap solutions are not able to execute any of the SPARQL queries from the selected BSBM use case. Therefore, the soundness is evaluated only for Ontop and EVI.

To evaluate the queries we not only checked the results but we also checked the relational queries used. To avoid the situation when the result would be correct only by coincidence. Moreover, in case that the results are not correct we wanted to understand what exactly is the issue. and to understand why the results are not correct. We have done that using a profiling tool which is a part of MS SQL. We have found that both tools generate very different queries.

[24] See https://github.com/tomatophantastico/sparqlmap/issues/35, reported May 2018, in February 2020 the issue is still opened

We have identified one issue using the Ontop solution. The Query 11 is processed by Ontop almost correctly. Simply said, the query takes an object `X` and retrieves all triples in the form `{X,_,_}` or `{_,_,X}`. The Ontop solution transforms the input SPARQL query to an union which should cover all possibilities. In the used mapping there are some triples where the object is an IRI directly generated from a column value (representing web pages). The problem is, that Ontop seems to lose the information that the object should be an IRI and not a string. Therefore, the Ontop solution does not consider these values when evaluating the Query 11.

The Ontop solution represents a SPARQL variable in a relational query using three columns to identify the type, language and the actual value of the variable. We have described it in the paper [18]. The main issue of this approach is that a relational column is typed (for all result rows the type is the same) while a SPARQL variable is not typed (every result can have the variable mapped to a value with a different type). Ontop used to solve this by using always a string column. As a consequence, it means that the native relational sorting does not work correctly - for example, 10 is treated as smaller than 5 as even numbers are treated as strings. This issue would happen in the query 10, but we have found that the Ontop has fixed this issue. They use `DECIMAL` to represent the numeric value. It is possible that in some corner cases, the approach will not be correct but using the BSBM queries we have not found any issue.

We have also observed that the Ontop solution does not always use a single relational query which corresponds completely to the SPARQL query. The `DESCRIBE` clause is performed as several relational queries - first to retrieve a list of objects to be described and the following ones to retrieve the object descriptions. Moreover, it seems that the Ontop preloads the list of RDF classes that are present in the dataset. As some of the classes are defined in the database, it has to load some data into memory. This approach have its advantages and disadvantages. On one hand, it is able to optimize SPARQL queries as it is able to produce optimized queries as it knows which classes exist and it is not needed to query it every time. On the other hand, if the database data changes often the preloaded set of classes will not be up to date and therefore the queries may not be handled correctly. However, this is not included in our evaluation results as in our evaluation we do not focus on a scenario when the database changes over time.

Table 3
Soundness evaluation results

| | Ontop | EVI |
|---|---|---|
| Query 1 | ✓ | ✓ |
| Query 2 | ✓ | ✓ |
| Query 3 | ✓ | ✓ |
| Query 4 | ✓ | ✓ |
| Query 5 | ✓ | ✓ |
| Query 6 | ✓ | ✓ |
| Query 7 | ✓ | ✓ |
| Query 8 | ✓ | ✓ |
| Query 9 | ✓ | ✓ |
| Query 10 | ✓ | ✓ |
| Query 11 | ✗ | ✓ |
| Query 12 | ✓ | ✓ |
| Correct queries | 11 | 12 |

To **summarize** the soundness evaluation, we decided to give 1 point per every correct query. As mentioned above, using the BSBM queries, we have found that one case is not handled correctly by the Ontop solution. We list our findings in Table 3.

### *6.4. Performance*

The BSBM test driver measures the performance of a provided SPARQL endpoint. The performance is measured in the terms of how many query mixes per hour are executed. However, the benchmark can be adjusted using two main arguments: how big the dataset is and how many clients runs the queries at the same time. The usable scale depends on the used machine. We have used an Ubuntu Linux virtual machine hosted in Azure. The used machine size is D4sv3 which provides 4 virtual CPUs and 16GB of RAM with storage of 120GB. Based on the performance of that machine, we have selected the arguments as all combinations of the following:

- The generated BSBM dataset with the product count of 10, 100, 1000, 10000, 100000, 200000, 500000 or 1000000
- The BSBM test driver client count of 1, 2, 4, 8, 16 or 32

To simplify the execution and to make the measurement repeatable, we provided the whole benchmark as a script[25] that can be easily executed again.

[25] See https://github.com/mchaloupka/r2rml-benchmark, visited February 2020

As we already mentioned in section 3.2, we have slightly updated the BSBM benchmark. Originally, the BSBM benchmark is not able to generate other than MySQL dataset for a relational database. We have added a support for a MS SQL dataset in the exactly same way as MySQL is done. Moreover, we have added the ability to generate multiple datasets at once, so it is possible to generate MySQL and MS SQL datasets at the same time with the same data.

The query execution speeds for 1 client used are shown on Figure 5. For smaller datasets, the EVI solutions offers significantly better performance. As the dataset grows the difference is smaller and for the largest dataset the performance is almost the same. There is an interesting difference between the Ontop on MS SQL and MySQL. For smaller datasets the MySQL variant is faster, which changes for larger datasets where the MySQL variant is slower.

The execution speed with 8 clients seems to be almost twice as fast as the speed when only 1 client is used. The biggest difference seems to be for a dataset with 10000 products where the count of total query mixes per hour is more than three times larger than with only 1 client. If the client count is further increased, there is no significant change as shown on Figure 7. So, it seems that having as many processor cores as clients provides the biggest benefits. At the same time, it looks like the virtual endpoint implementations do not use multiple cores anyhow to serve a single client. This is especially obvious for small datasets where the majority of execution time is used by the transformation.

As mentioned, we have executed the tests also using more than 1 client. Having more clients provide bigger speed especially on smaller datasets. As the dataset grows, the difference is smaller as the ability of the used relational database to execute queries quickly in parallel is lower. The query execution speeds when using 8 clients are shown on Figure 6.

For larger datasets, the execution time depends on the speed of the underlying layer. On Figure 8 there is included a comparison with the speed of the manually created relational queries retrieving the same results. The used relational queries are a part of the BSBM test driver. They are semantically the same as the corresponding SPARQL queries. However, we do not consider them written in the most optimal way possible. We did not want to modify the queries, to stick to the BSBM framework as much as possible. Interestingly, the MySQL relational database tends to be significantly (almost four times) faster for smaller datasets

| Dataset scale | 10 | 100 | 1000 | 10000 | 100000 | 200000 | 500000 | 1000000 |
|---|---|---|---|---|---|---|---|---|
| EVI - MS SQL | 9653 | 8866 | 7164 | 3363 | 1412 | 928 | 652 | 481 |
| Ontop - MS SQL | 3109 | 3073 | 2233 | 1654 | 701 | 713 | 507 | 461 |
| Ontop - MySQL | 3162 | 3476 | 2666 | 2406 | 1028 | 1039 | 466 | 334 |

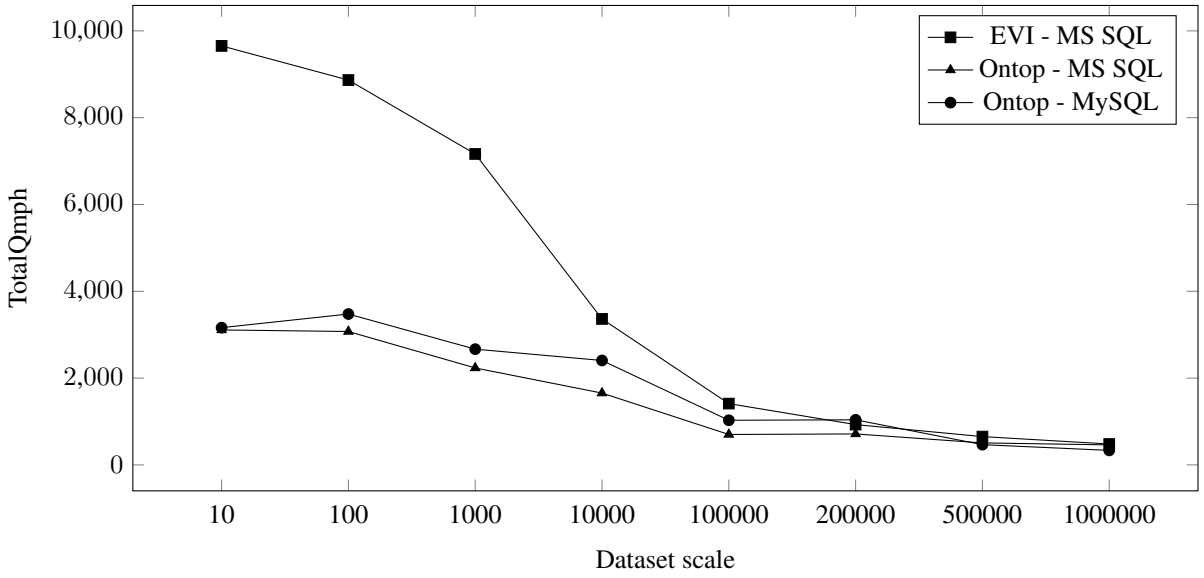


Fig. 5. Query mixes per hour using 1 client

| Dataset scale | 10 | 100 | 1000 | 10000 | 100000 | 200000 | 500000 | 1000000 |
|---|---|---|---|---|---|---|---|---|
| EVI - MS SQL | 21466 | 22347 | 18776 | 10048 | 2467 | 1495 | 851 | 634 |
| Ontop - MS SQL | 7230 | 7268 | 5449 | 3635 | 1305 | 1242 | 773 | 656 |
| Ontop - MySQL | 7432 | 7900 | 6412 | 5322 | 2090 | 1952 | 795 | 618 |

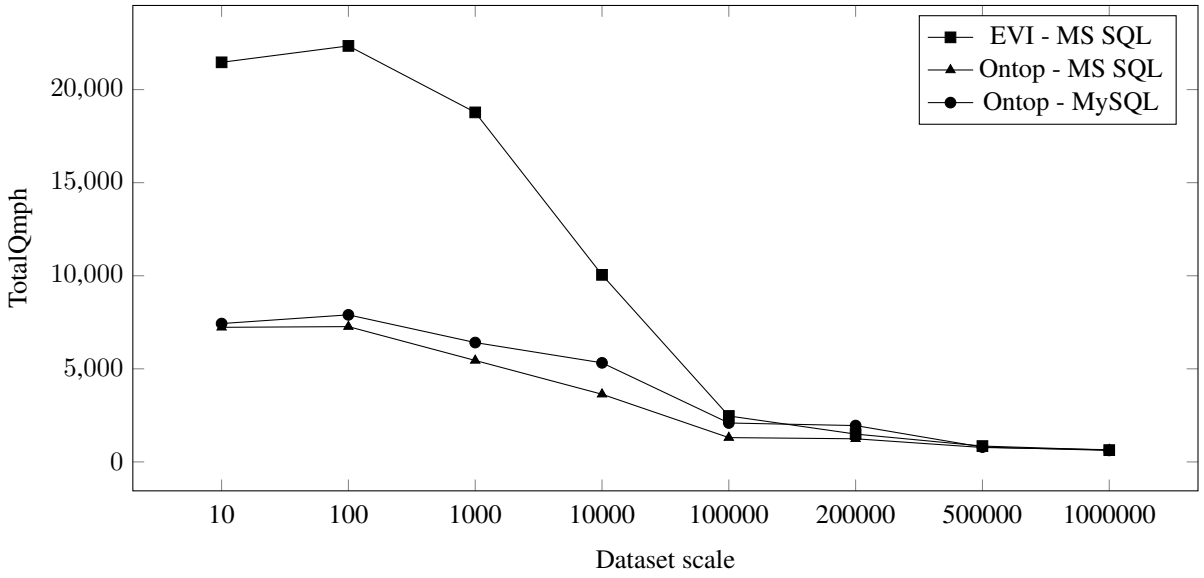


Fig. 6. Query mixes per hour using 8 clients

| Dataset scale | 1 | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| EVI - MS SQL | 3363 | 6723 | 9321 | 10048 | 10388 | 10377 |
| Ontop - MS SQL | 1654 | 2658 | 3778 | 3635 | 3634 | 3701 |
| Ontop - MySQL | 2406 | 4027 | 5354 | 5322 | 5240 | 4903 |

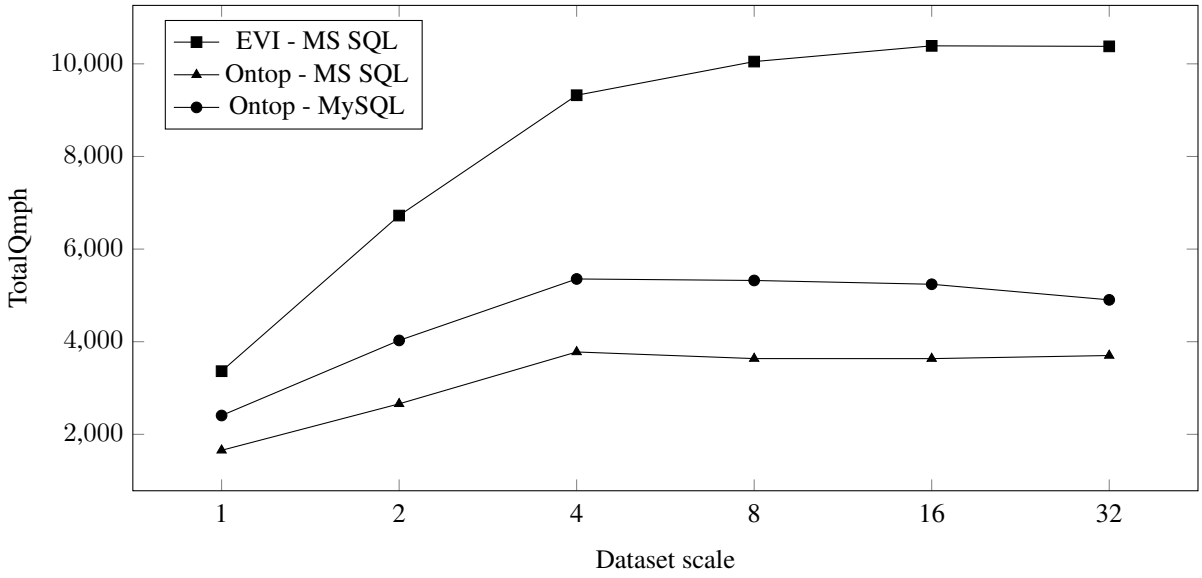


Fig. 7. Query mixes per hour using multiple clients with dataset scale of 10000

| Dataset scale | 10 | 100 | 1000 | 10000 | 100000 | 200000 | 500000 | 1000000 |
|---|---|---|---|---|---|---|---|---|
| EVI - MS SQL | 9653 | 8866 | 7164 | 3363 | 1412 | 928 | 652 | 481 |
| Ontop - MS SQL | 3109 | 3073 | 2233 | 1654 | 701 | 713 | 507 | 461 |
| Ontop - MySQL | 3162 | 3476 | 2666 | 2406 | 1028 | 1039 | 466 | 334 |
| MS SQL | 73358 | 51803 | 29312 | 7030 | 1803 | 1084 | 626 | 529 |
| MySQL | 217927 | 150838 | 30872 | 4749 | 401 | 259 | 102 | 51 |
| Virtuoso | 15714 | 14878 | 13302 | 12582 | 2522 | 595 | ✗ | ✗ |

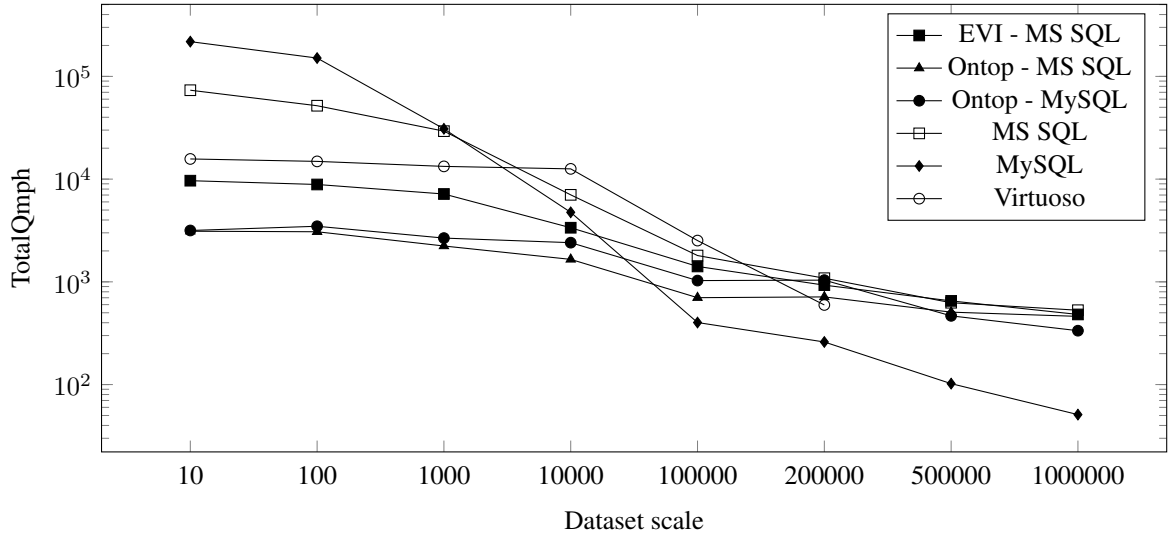


Fig. 8. Query mixes per hour using 1 client with other solutions

Table 4
Performance evaluation results

| Scale | EVI | Ontop - MS SQL | Ontop - MySQL |
|---|---|---|---|
| 10 | 100% | 32% | 32% |
| 100 | 100% | 35% | 39% |
| 1000 | 100% | 31% | 37% |
| 10000 | 100% | 49% | 71% |
| 100000 | 100% | 50% | 73% |
| 200000 | 89% | 69% | 100% |
| 500000 | 100% | 78% | 71% |
| 1000000 | 100% | 96% | 69% |
| Average | 99% | 55% | 62% |

and MS SQL seems to be significantly (more then ten times) faster for larger datasets.

The comparison with the speed of the manually created relational queries also shows that the cost of the transformation from SPARQL to SQL query and then back from relational result to SPARQL result is significant especially for small queries. As the dataset grows, the performance depends more on the underlying relational database so the transformation cost is minimal. Interestingly, for larger dataset the virtual endpoint may be actually faster than the manually created relational queries in BSBM because the virtual endpoint solutions generate more optimized relational queries. This was visible when using MySQL database.

For reference, on Figure 8 there is also a comparison with a native RDF storage - the Virtuoso Universal Server.[26] We have selected it as it is a well performing native RDF storage when benchmarked using BSBM [26]. For smaller datasets, the Virtuoso solution is significantly faster than any virtual SPARQL endpoint. As the dataset grows, loading the dataset into Virtuoso and querying took more and more time. For the dataset size of 200 000 products, the Virtuoso solution was actually slower than the virtual SPARQL endpoints. Moreover, the Virtuoso solution was not able to handle bigger datasets at all using the selected hardware.

To **summarize** the performance evaluation, we decided to use the execution speed using 1 client. For every tested dataset scale, we give 100% score to the fastest solution. The other solutions gets their score accordingly to the ratio between the execution speed and the speed of the fastest solution. The result is shown in Table 4.

[26] See https://virtuoso.openlinksw.com/, visited March 2020

Table 5
Individual evaluation results overview

| Aspect | Morph | SparqlMap | Ontop | EVI |
|---|---|---|---|---|
| Usability | 2 | 4 | 5 | 3 |
| Completeness | 0 | 0 | 48 | 24 |
| Soundness | 0 | 0 | 11 | 12 |
| Performance | 0% | 0% | 62% | 99% |

Table 6
Overall evaluation results

| Aspect | Morph | SparqlMap | Ontop | EVI |
|---|---|---|---|---|
| Usability | 40% | 80% | 100% | 60% |
| Completeness | 0% | 0% | 100% | 50% |
| Soundness | 0% | 0% | 92% | 100% |
| Performance | 0% | 0% | 62% | 99% |
| Average | 10% | 20% | 88% | 77% |

### 6.5. Summary

From the usability and completeness perspective, the Ontop solution is a clear winner. It underlines the fact that the Ontop solution seems to be the most mature and used solution from all selected solutions. The Morph and SparqlMap solutions were identified as not suitable for the BSBM use case.

From the soundness perspective, the EVI solution was slightly better than Ontop. We have identified an issue in handling one of the used SPARQL queries. However, the difference is small.

The performance evaluation has identified that the query transformation overhead is bigger in the Ontop solution compared to the EVI solution. As the dataset grows, this overhead is less significant because the performance is mainly limited by the performance of the underlying relational database.

For every aspect, we have assigned a score to individual solutions. An overview of achieved scores in individual evaluations is shown in Table 5. In Table 6, the achieved score is shown as relative to the maximum score that could have been achieved by a tool.

We believe that the importance of individual aspects depends on the exact use case. Therefore, we have decided to put the same weight on all aspects - the overall score is calculated as the average of individual relative results. The Ontop solution has achieved the best overall score.

## 7. Conclusion

In this paper, we have described the evaluation of virtual SPARQL endpoints using Berlin SPARQL Benchmark tools. We have shown, how the tools can be used to evaluate virtual SPARQL endpoint solutions and what are the gaps of the benchmark from that perspective. We have provided patches for these gaps. Moreover, we have provided scripting to orchestrate the benchmark.

According to the evaluation, it seems that there is still a place for improvements for the virtual endpoint solutions. For the BSBM use case, there are two usable solutions right now: Ontop and EVI. The EVI solution provides a better performance, although for very large datasets the Ontop solution performs similarly. On the other hand, the EVI solution is limited to the usage of MS SQL database only. Therefore, if another database engine is used then Ontop is the only usable choice.

The Ontop solution has been identified as a mature and used solution. There is a complete documentation provided. Moreover, if that is not enough it will be possible to ask for help as there is a living community around the tool. The EVI solution provides a better performance but from all other perspectives the Ontop solution is better.

Regarding the performance, there is still a room for improvements. When using a native RDF storage, the performance will be significantly better if the dataset will be small enough for the selected hardware. However, it requires the data to be migrated from the relational database to the RDF storage. Moreover, it has to be done whenever data is changed. Therefore, this approach cannot be used if the dataset changes often.

Based on this paper, it should be possible to evaluate virtual SPARQL endpoints consistently in the future. We have published not only the approach to run the evaluation but also all additional files and tools required. The evaluation should be performed when any of the solutions will be improved or when a new solution will be created.

Moreover, we have identified limitations of the R2RML mapping language using BSBM use case. It simulates usage in an e-commerce solution. Based on this, we believe that these limitations should be addressed in the future as they may affect real world scenarios.

**Acknowledgments.** This work was supported by the Czech Science Foundation (GAR), grant number 19-01641S.

## References

[1] RDF Working Group, Resource Description Framework, W3C Recommendation, W3C, 2014, https://www.w3.org/RDF/.

[2] M. Lanthaler, D. Wood and R. Cyganiak, RDF 1.1 Concepts and Abstract Syntax, W3C Recommendation, W3C, 2014, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[3] S. Harris and A. Seaborne, SPARQL 1.1 Query Language, W3C Recommendation, W3C, 2013, http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[4] E. Prud'hommeaux, M. Arenas, A. Bertails and J. Sequeda, A Direct Mapping of Relational Data to RDF, W3C Recommendation, W3C, 2012, http://www.w3.org/TR/2012/REC-rdb-direct-mapping-20120927/.

[5] S. Das, R. Cyganiak and S. Sundara, R2RML: RDB to RDF Mapping Language, W3C Recommendation, W3C, 2012, http://www.w3.org/TR/2012/REC-r2rml-20120927/.

[6] C. Bizer and A. Schultz, The Berlin SPARQL Benchmark., *Int. J. Semantic Web Inf. Syst.* **5**(2) (2009), 1–24. http://dblp.uni-trier.de/db/journals/ijswis/ijswis5.html#BizerS09.

[7] C. Bizer and A. Schultz, Berlin SPARQL Benchmark (BSBM), 2011, Accessed: January 2020. http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/.

[8] S.S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T.T. Jr, S. Auer, J. Sequeda and A. Ezzat, A Survey of Current Approaches for Mapping of Relational Databases to RDF., Technical Report, W3C RDB2RDF Incubator Group, 2009, http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf. http://www.w3.org/2005/Incubator/rdb2rdf/RDB2RDF_SurveyReport.pdf.

[9] C. Bizer and A. Seaborne, D2RQTreating non-RDF databases as virtual RDF graphs, *World Wide Web Internet and Web Information Systems* (2005).

[10] R. Cyganiak, D2RQ. Accessing Relational Databases as Virtual RDF Graphs., Accessed: January 2020.

[11] A. Chebotko, S. Lu and F. Fotouhi, Semantics Preserving SPARQL-to-SQL Translation, *Data Knowl. Eng.* **68**(10) (2009), 973–1000. doi:10.1016/j.datak.2009.04.001.

[12] A. Chebotko, S. Lu, H.M. Jamil and F. Foutouhi, Semantics Preserving SPARQL-to-SQL Query Translation for Optional Graph Patterns, Technical Report, Wayne State University, 2006.

[13] J. Sequeda and D.P. Miranker, Ultrawrap: SPARQL Execution on Relational Data, *Web Semantics: Science, Services and Agents on the World Wide Web* **22**(0) (2013).

[14] F. Priyatna, O. Corcho and J. Sequeda, Formalisation and Experiences of R2RML-based SPARQL to SQL Query Translation Using Morph, in: *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, ACM, New York, NY, USA, 2014, pp. 479–490. ISBN 978-1-4503-2744-2. doi:10.1145/2566486.2567981.

[15] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro and G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web* **8** (2016). doi:10.3233/SW-160217.

[16] M. Rodriguez-Muro, M. Rezk, J. Hardi, M. Slusnys, T. Bagosi and D. Calvanese, Evaluating SPARQL-to-SQL Translation in ontop, in: *Proc. of the 2nd Int. Workshop on OWL Rea-*

*soner Evaluation (ORE 2013)*, CEUR Workshop Proceedings, http://ceur-ws.org/, Vol. 1015, 2013, pp. 94–100.

[17] M. Rodríguez-Muro and M. Rezk, Efficient SPARQL-to-SQL with R2RML mappings, *Web Semantics: Science, Services and Agents on the World Wide Web* **33**(1) (2015).

[18] M. Chaloupka and M. Nečaský, A Survey of Approaches to Representing SPARQL Variables in SQL Queries, in: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, H. Panetto, C. Debruyne, W. Gaaloul, M. Papazoglou, A. Paschke, C.A. Ardagna and R. Meersman, eds, Springer International Publishing, Cham, 2017, pp. 300–317. ISBN 978-3-319-69459-7.

[19] J. Unbehauen, C. Stadler and S. Auer, Accessing Relational Data on the Web with SparqlMap, in: *JIST*, 2012. http://svn.aksw.org/papers/2012/SPARQLComponent/JIST_SparqlMap/public.pdf.

[20] M. Chaloupka and M. Neaský, Efficient SPARQL to SQL Translation with User Defined Mapping, in: *Knowledge Engineering and Semantic Web: 7th International Conference, KESW 2016, Prague, Czech Republic, September 21-23, 2016, Proceedings*, Springer International Publishing, Cham, 2016, pp. 215–229. ISBN 978-3-319-45880-9.

[21] C. Bizer and A. Schultz, Benchmarking the performance of storage systems that expose SPARQL endpoints, in: *In Proceedings of the ISWC Workshop on Scalable Semantic Web Knowledgebase*, 2008.

[22] A.J.G. Gray, N. Gray and I. Ounis, Can RDB2RDF Tools Feasibily Expose Large Science Archives for Data Integration?, in: *The Semantic Web: Research and Applications*, L. Aroyo, P. Traverso, F. Ciravegna, P. Cimiano, T. Heath, E. Hyvönen, R. Mizoguchi, E. Oren, M. Sabou and E. Simperl, eds, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 491–505. ISBN 978-3-642-02121-3.

[23] D.-E. Spanos, P. Stavrou and N. Mitrou, Bringing Relational Databases into the Semantic Web: A Survey, *Semant. Web* **3**(2) (2012), 169209–.

[24] M.A.G. Hazber, R. Li, B. Li, Y. Zhao and K.M.A. Alalayah, A Survey: Transformation for Integrating Relational Database with Semantic Web, in: *Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences*, ICMSS 2019, Association for Computing Machinery, New York, NY, USA, 2019, pp. 6673–. ISBN 9781450361897. doi:10.1145/3312662.3312692.

[25] F. Michel, J. Montagnat and C. Faron Zucker, A survey of RDB to RDF translation approaches and tools, Research Report, I3S, 2014, ISRN I3S/RR 2013-04-FR 24 pages. https://hal.archives-ouvertes.fr/hal-00903568.

[26] P. Boncz, O. Erling and M.-D. Pham, *Advances in Large-Scale RDF Data Management*, in: *Linked Open Data – Creating Knowledge Out of Interlinked Data: Results of the LOD2 Project*, S. Auer, V. Bryl and S. Tramp, eds, Springer International Publishing, Cham, 2014, pp. 21–44. ISBN 978-3-319-09846-3. https://doi.org/10.1007/978-3-319-09846-3_2.