Enabling OBDA query translation over CSVs

No Author Given

No Institute Given

Abstract. Ontology-Based Data Access (OBDA) has traditionally focused on providing ontology-based access to relational databases (RDB) data, either by RDF materialisation or SPARQL-to-SQL query translation techniques. With the advent of mapping languages or annotations such as RML or CSVW, recently OBDA has been also used to generate materialised RDF instances of data available in semi-structured data formats such as CSV. So far, query translation techniques have been applied over such data format by considering a CSV file as a single table that can be loaded in an RDB. However, such techniques do not take into account those characteristics that are normally present in real-world CSV files: data are not necessarily normalised, data may not adhere to strict consistency rules (e.g., datatypes, multiplicity of values), and explicit relationships across different files (e.g., joins) are often missing. In this paper, we present a framework for enabling query translation over a set of CSV files by using a combination of CSVW annotations and RML mappings with FnO transformation functions. Exploiting these inputs, the framework creates an enriched RDB representation of the CSV files together with the corresponding R2RML mappings, enabling the use of any existing query translation (SPARQL-to-SQL) techniques proposed in literature. We validate our proposal against a set of real-world CSV files in the domain of smart cities, comparing query result completeness and performance with state-of-the-art engines.

Keywords: OBDA · Query Translation · CSVW · RML · R2RML.

1 Introduction

Semi-structured data formats, and particularly spreadsheets in the form of CSV or Excel files, are one of the most widely used formats to publish data on the Web. This is especially true in the case of open government data portals, as described in the annual report of the European Data Portal (EDP) [4]. For example, Table 1 shows the formats that are most commonly used to publish data in a set of mature EU open data portals, together with the percentage of spreadsheet-based datasets according to the total number of published datasets in each of them.

There are several reasons why spreadsheets are so popular for data publication on the Web. First, they are easy to generate by data providers. In some cases, they are even used as one of the main ways to manage data inside organisations [11]. Second, they are easy to consume with usual office tools (e.g., Excel,

Data Portal	1st Format	2nd Format	3rd Format
Spain	**CSV (50%)**	**XLS (35%)**	JSON (33%)
Norway	**CSV (77%)**	GEOJSON (17%)	JSON (14%)
Italy	**CSV (76%)**	JSON (35%)	XML (25%)
Croatia	**XLS (63%)**	**CSV (40%)**	HTML (33%)

Table 1: Most commonly used formats (and percentage over the total number of datasets) to expose data in mature EU open data portals

LibreOffice) and there are advanced tools that can be used to process them (e.g., OpenRefine, Tableau). However, more advanced consumers (e.g. application developers, knowledge workers that need to integrate data from different sources) often have to face some relevant challenges when consuming spreadsheet-based data: there is no standard way to query data in them as it can be done with other types of data formats, such as RDB, JSON or XML, among others; data are difficult to integrate since data constraints and relationships across different files are not explicit; data are often difficult to understand since column names are generally heterogeneous.

Some of the aforementioned challenges may be dealt with by considering a Semantic Web approach. Data publishers or third parties may be willing to provide their data using an RDF-based representation, so as to comply with the 5-star open data scheme [1]. This may be done either using materialised or virtual RDF views of the data that is currently being exposed as spreadsheets. In this context, Ontology-Based Data Access (OBDA) [19] may be appropriate. OBDA aims at providing a unified view and common access to a set of data sources based on ontologies and mappings, either by generating materialised views (RDF files) or by translating SPARQL queries into queries that are supported by the underlying source.

Existing OBDA engines [3][20][9] have been already used in order to deal with spreadsheets[2]. Most of these engines allow generating RDF materialisations of the CSV data and those that provide virtual RDF views usually treat a CSV file as if it was a single not-normalised relational database table with no keys or integrity constraints, important elements that are used by RDB engines for efficient querying. However, there are significant differences between RDBs and CSVs, what makes this approach not to be completely appropriate. Data stored in RDBs follow an RDB schema that must be declared in advance, while CSV files are more flexible since they do not necessarily have a predefined schema (at most the names of columns are defined in the header of the file). As data constraints are not necessarily imposed in CSVs (e.g., the range of values of a numerical column cannot be defined, the maximum number of characters of a string cannot be specified), multiple values may appear in a cell, independent concepts may appear in the same table, etc. Indeed, the problem gets even harder when we want to provide a unified view over multiple CSVs, since relationships (e.g.,

[1] `https://5stardata.info/en/`

[2] Without loss of generality, from now on we will refer to CSVs as it is an open format.

Primary Key - Foreign Key) between columns in those files are not explicitly defined and hence cannot be easily identified and exploited.

Several languages have been proposed to specify the mappings needed to provide RDF views over CSV files: CSVW [26] and RML+FnO (RML [9] combined with the Function Ontology [7]). However, none of them in isolation can deal with all the aforementioned challenges, as discussed in Section 3. Therefore, we propose using the information contained in both of them simultaneously in order to **enable query translation over heterogeneous and real CSVs**. Our approach takes as input a set of CSV files together with a set of CSVW annotations and RML+FnO mappings. It analyses the annotations and mappings so as to obtain details on the underlying schema, required transformation functions, missing information, etc., and generates and populates an enriched and normalised RDB schema from the CSV files and an R2RML mapping document, so that existing SPARQL-to-SQL techniques and optimisations can be used to query them. In more detail, our research hypotheses in this paper are:

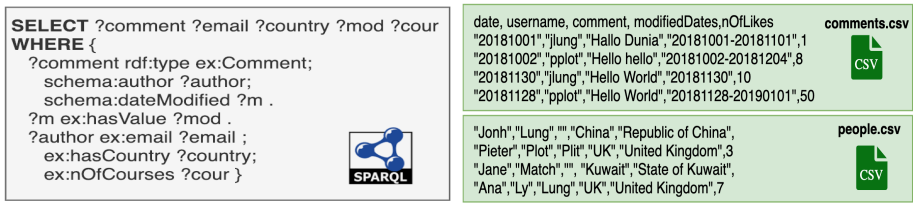
1. It is possible to make use of existing SPARQL-to-SQL translation techniques to perform query translation over CSV files, via automatic generation of an enriched RDB model that represents the original CSV data, plus an R2RML mapping that exploits RML+FnO mappings and CSVW annotations, hence being able to benefit from the optimisations for query translation already proposed in the literature.
2. The completeness of the results of evaluating a SPARQL query over a set of CSV files is improved when RML+FnO mappings and CSVW annotations are taken into account, in comparison with current proposals.
3. Using such an approach, the performance of query evaluation improves over current proposals.

One requirement that we imposed to ourselves when we started this work was: **our approach should use as much as possible existing resources and not introduce yet another mapping language nor require a new SPARQL-to-SQL implementation**. There are two main reasons behind this requirement. Proposing a new language will bear a consequence of having a new learning curve for the users, in addition to numerous existing mapping languages. SPARQL-to-SQL techniques and optimisation have been studied and implemented for a decade and rather than investing efforts of creating a new implementation, we decide to use existing engines.

The paper is organised as follows: Section 2 provides a motivating example that especially focuses on the main characteristics of CSV files that cannot be handled in current naïve OBDA-based query translation over CSVs. Section 3 discusses the main properties of RML+FnO and CSVW, which our proposal makes use of. Section 4 describes our approach for the automatic generation and population of an enriched RDB schema and R2RML mapping document. Section 5 evaluates our approach, using several experiments with real-world CSV files from the smart city domain. Section 6 presents related work and Section 7 presents conclusions and areas for future work.

2 Motivating Example

In this section, we show an example that motivates the need for an improved solution to deal with the challenges associated to OBDA-based query translation over a collection of CSV files. Figure 1a shows a SPARQL query that lists comments (message text and modified date) and their authors (email, country and number of courses taught). The data for answering this query is available in two CSV files (*people.csv* and *comments.csv*, Figure 1b). The CSV file *comments.csv* contains data about comments, such as the author's username and the dates when the message was created and modified. The CSV file *people.csv* contains information about the authors, such as name, surnames, data about the country of origin and information about the number of courses that they teach. A priori, the files do not have any relationship among them, hence it would not be possible to retrieve the information requested by the SPARQL query. Finally, the example shows the expected SPARQL query result (Figure 1c). All the files used in this example are available at the supplemental material, to facilitate reproducibility.



(a) SPARQL query **(b)** CSV files

?comment, ?email, ?country, ?mod, ?cour
"Hallo Dunia", "jlung@domain.com", dbr:China, 2018-10-01, 0
"Hallo Dunia", "jlung@domian.com", 2018-10-01, dbr:China, 2018-11-01, 0
"Hello hello", "pplot@domain.com", 2018-10-02, dbr:United_Kindom, 2018-12-04, 3
...

(c) SPARQL results

Fig. 1: Motivating example: evaluating a SPARQL query over two CSV files. Figure 1a shows a SPARQL query that lists comments made by an author, together with its modified dates, and the author's email, country, number of courses taught. Figure 1b shows the two CSV files used to evaluate the SPARQL query and Figure 1c shows the expected result.

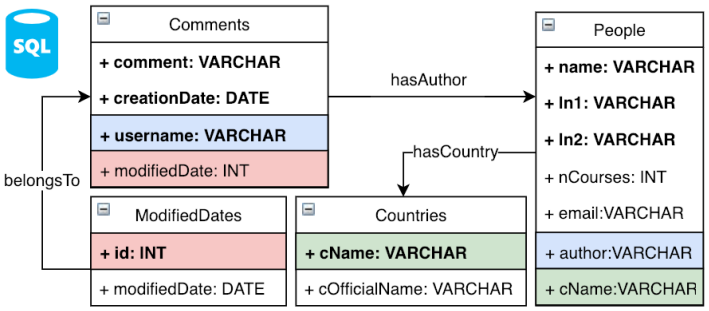
In Section 1 we already identified some challenges that are common when dealing with data available in CSVs, and which are not sufficiently considered in current OBDA approaches. Now we discuss them in more detail, together with a description of how they appear in our motivating example:

– **NS: CSVs do not provide a schema.** CSV files provide minimal information about their underlying schema, in the form of column names in the header, if at all present. Existing OBDA-based query translation approaches

for CSVs rely on loading the CSVs as a single RDB table, without specifying any additional constraints. However, mappings and annotations can be used to enrich the generated RDB schema with additional information, such as column datatypes, primary key (PK) identification, etc.

- **NN: CSVs are not normalised.** CSV files are not usually structured to follow the normal forms proposed by Codd [5]. The use of CSV allows organising multiple concepts in a single file (2NF) and even multiple values in a cell (1NF). Mappings and annotations can be used to help generate the corresponding 3NF database representation. For example:
  - CSVs in 1NF: The fourth column (*modifiedDates*) of the file *comments.csv* contains multiple values. This issue can be addressed thanks to annotations, because they can give information about the separator between the values.
  - CSVs in 2NF: Columns 3 and 4 of the file *people.csv* are non-prime attributes that do not depend on the PK of the table (column 1 together with column 2). Mapping information can be used so as to divide the CSV file into two different tables (*People* and *Countries*), with the corresponding relationships between them.
- **NU: Data may be in a non-uniform format.** The first column of *comments.csv* represents the dates of each comment. The format of this column is YYYYMMDD (first 4 digits represent the year followed by 2 digits for the month and finally the last 2 digits for the day). No information about the format of the date is provided. The same may happen with the format of number, Boolean or specific string. Mappings and annotations can be used to define the format of those types.
- **MD: Missing data.** There are some cases where our CSV files are missing some relevant data:
  - The first one can be identified in the SPARQL query. It asks for the email, which is not available in the CSV files. This can be solved by taking into account transformation functions in mappings that allow generating derived information. For example, the email may be generated using a transformation function applied to columns 1 and 2 from *people.csv*, together with domain information (i.e. "@domain.com").
  - The second one can be seen in column 3 of *people.csv*. Here we see that the first and third rows do not have any data in their third column (which is the second family name, something that is common in some European countries). The empty string in this column indicates that such information does not exist and will be represented as NULL in the RDB.
  - The third one can be seen in the last column of *people.csv*. The column contains the number of courses that they teach. The empty string in this column means that the corresponding value is 0.
- **MM: Missing metadata.** The headers of the file *people.csv* are missing. Column names are essential for the generation of the RDB schema and mappings. Annotations may provide sufficient metadata to generate them.

– **IJ: Joins are not explicit.** A priori, the CSV files of the example do not have any explicit relationship. However, joins among CSVs may be defined if transformation functions are included in the mapping. In our motivating example, the join is defined by applying a set of transformation functions over the *name* (column 1) and *ln1* (column 2) of the *people.csv* and the *username* column of *comments.csv*.
– **IC: Irrelevant Columns.** There is no need to transform all the columns of a CSV file into database columns because not all of them are relevant. For example, the last column (*nOfLikes*) of *comments.csv* is not mapped. Hence it will not appear in the generated RDB.



(a) Enriched RDB schema from the CSV files from Figure 1b

```
<TriplesMapPerson>
 rr:logicalTable [ rr:tableName "People"; ] ;

 rr:subjectMap [
  rr:template "http://ex.com/Person/{name}{ln1}{ln2}";
  rr:class schema:Person; ];

 rr:predicateObjectMap [
  rr:predicate schema:hasCountry;
  rr:refObjectMap [
   rr:parentTriplesMap <TriplesMapCountry>;
   rr:joinCondition [
    rr:child "cName";  rr:parent "cName";
   ]; ]; ];
...
```

```
<TriplesMapModifiedDates>
 rr:logicalTable [ rr:tableName "ModifiedDates"; ] ;

 rr:subjectMap [
   rr:template "http://ex.com/ModifiedDate/{id}";
   rr:class ex:ModifiedDate;  ];

 rr:predicateObjectMap [
   rr:predicate ex:belongsTo;
   rr:refObjectMap [
    rr:parentTriplesMap <TriplesMapPublication>;
    rr:joinCondition [
     rr:child "id"; rr:parent "modifiedDate";
    ]; ]; ];
....
```

(b) R2RML mappings that allow exploiting the data in the enriched RDB

Fig. 2: Enriched RDB schema and R2RML mappings based on the *comments.csv* and *people.csv* CSV files so as to enable SPARQL-to-SQL evaluation over them

Figure 2 shows the RDB schema and part of the R2RML mapping documents necessary for enabling OBDA query translation over the *comments.csv* and *people.csv* CSV files. In the next sections we show how RML+FnO mappings and CSVW annotations can be used to generate these outputs.

3 Mappings and Annotations for CSV

R2RML [6] is a W3C Recommendation for describing transformation rules from RDB to RDF. This mapping language has been extended by RML so as to provide support to other data formats, such as XML, CSV, JSON, etc. Both languages provide basic transformation functions to concatenate strings, which are especially useful for generating URIs from columns/fields of the dataset. Recently, RML has been integrated with the Function Ontology (FnO) [7] to support other types of transformations. CSVW annotation [26] is a W3C Recommendation to describe CSV files.

We now describe relevant properties of RML+FnO and CSVW, summarised in Table 2, which are useful to deal with the challenges identified in Section 2:

- **Transformation functions.** String concatenation functions are supported by both CSVW (`csvw:aboutUrl`, `csvw:valueUrl`) and RML (`rr:template`). In addition, more complex functions can be specified using the RML+FnO `fnml:functionValue` property. Two special cases of transformation functions are related to how to generate default values and NULL representations in the enriched RDB. These two cases can be handled by RML (`fnml:functionValue`) and CSVW (`csvw:defaultValue`, `cvwv:null`).
- **Datatypes and data formats.** RML allows defining the datatype of RDF literals using the `rr:datatype` property and CSVW allows specifying the datatype (`csvw:datatype` property) and format (`csvw:format` property) of the CSV' columns.
- **Keys and Constraints.** In CSVW the property `csvw:primaryKey` can be used to declare explicitly the primary key of a table. This is not available with RML, but we can assume that the primary key is the column(s) used in the Subject Map (`rr:subjectMap`), as it should generate a unique URI for every resource. As for the foreign key, the use of RML's `rr:joinCondition` can be seen as an indication that the column used is a foreign key. CSVW provides an explicit way to declare whether a column is a foreign key, using the `csvw:foreignKeys` property. CSVW provides a couple of properties (e.g., `csvw:mininum` or `csvw:maximum`) to specify the range of numerical columns.
- **Normalization.** The CSVW property `csvw:separator` indicates the character used to separate multiple values in the cells of a CSV column, what is relevant when a CSV file is in 1NF. The use of multiple RML Triples Maps with joins between the same CSV file can be used as an indication that it contains multiple concepts (2NF).
- **Metadata.** The property `csvw:rowTitles` can be used to specify column names in the case the first row is not used to specify the column names.

As we have seen, RML+FnO and CSVW have different purposes (mapping vs annotating, respectively) and the use of only one of them is not enough to generate an enriched database representation. In the next section we will see how the two proposals can be used together in our approach.

Challenges	Relevant Properties
Describe the corresponding concept (MM)	rr:class, csvw:propertyUrl
Describe the corresponding property (MM)	rr:predicateMap, csvw:propertyUrl
Add header to a CSV file (MM)	csvw:rowTitles
Column datatype (NS)	csvw:datatype
Constraining values (NU)	csvw:minimum, csvw:maximum
Specify the format of a column (NU)	csvw:format
Specify a join (IJ)	rr:refObjectMap, csvw:foreignKeys
Specify unique values columns (NS)	csvw:primaryKey
Transform value (MD)	rr:templateMap, fnml:functionValue, csvw:valueUrl
Support for multiple values in one cell (NN)	csvw:separator
Primary key (NS)	rr:subjectMap, csvw:primaryKey, csvw:aboutUrl
Default for missing values (MD)	csvw:default
Specify NULL values (MD)	csvw:null
Specify NOT NULL constraint (NS)	csvw:required
Specify columns to be tranformed (IC)	rr:reference, rr:template, csvw:columns, csvw:aboutUrl, csvw:valueUrl

Table 2: Properties of CSVW and RML+FnO that can be used to address the challenges of dealing with CSVs in OBDA-based query translation

4 Approach

In this section, we explain how we use RML+FnO mappings and CSVW annotations to enable query translation over heterogeneous CSVs, by generating an enriched RDB and a set of R2RML mappings. In Figure 3 we show the workflow of our approach that consists of: (i) the definition of a set of basic SQL transformation functions using FnO, (ii) the data translation process to generate the enriched RDB that represents the original CSV files, (iii) the mapping translation process to generate R2RML mappings from the input RML+FnO and CSVW and (iv) the application of SPARQL-to-SQL optimisation techniques to evaluate a query.

4.1 Definition of FnO_{SQL}

We assume that the majority of the transformations that need to be applied to CSV data can be done using a combination of basic SQL functions, which are supported by the majority of RDBMS (e.g., `concat`, `lower`, `substring`). This allows: (i) improving the performance of the transformation process, and (ii) avoiding the need of ad-hoc programming of functions in a programming language. These functions are declared in FnO and realised as FnO_{SQL} for their usage in RML mappings.

4.2 Data translation: from CSV to RDB

We analyse the information provided by RML mappings and CSVW notations to generate the database model, define datatypes and restrictions for each table

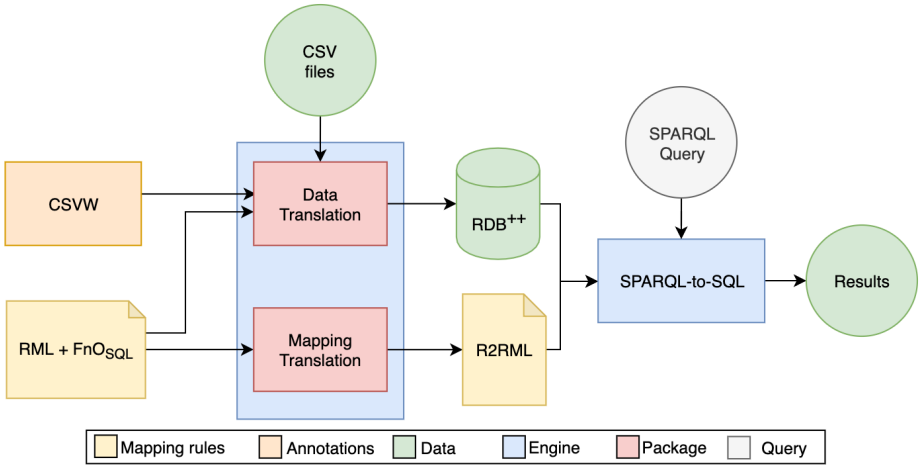


Fig. 3: Proposed workflow to enable query translation over heterogeneous CSVs

column, and relations between tables (PK/FK). In the following sections we describe each step in more detail, using the sample CSV files provided in Section 2, together with their corresponding CSVW annotations and RML+FnO_{SQL} mappings. The mappings and annotations, together with the final generated RDB schema for our motivation example, are available at the supplemental material.

Generation of a basic database schema. In the generation of the SQL database model (tables and their columns), the naïve approach used by existing SPARQL-to-SQL proposals is to generate a table for each CSV file. Normally, the name of the CSV file serves as the name of the database table, possibly with some encoding transformation in order to generate a valid table name. This approach assumes that CSVs are normalised and all files have column names that are used as table columns.

RML+FnO mappings and CSVW annotations provide additional information, as described in Section 3. We analyse this information to enrich the initially generated RDB. Based on the sample CSVs (*people.csv* and *comments.csv*), we perform the following steps: CSVW annotations are used to:

- CSVW is used to define row titles for the *people.csv*, using the `csvw:rowTitles` property that allows the definition of the Triples Map in RML for that file.
- CSVW is used to split *comments.csv* into two different tables (*Comments* and *ModifiedDates*), using the `csvw:separator` property on the column *modifiedDates* and generate the corresponding Triples Map for the *ModifiedDates* table.
- RML+FnO is used to generate new columns from `rr:objectMaps` that contain functions. The column name will be the name of the property from the predicate, without its prefix.
- RML+FnO is used to generate a new table based on the information provided by the country Triples Map.
- RML+FnO is used to generate the rest of the tables based on the name of the CSVs defined in the source property and the names of the columns based on the `rr:objectMap` properties.

Generation of Restrictions. The table definition has to include the datatype of each column. In the naïve approach, all the columns are loaded into the RDB instance as strings, so any special feature based on the datatype (e.g., get the year of a date) is not possible. Our approach uses the information in CSVW annotations (`csvw:datatype` or `csvw:base`) to generate the correct datatype for the columns. CSVW can also be used to assist the definition of columns with certain datatypes such as:

- Numerical column format (`csvw:format`, `csvw:pattern`, `csvw:decimalChar`).
- Maximum and minimum of numerical columns (`csvw:minimum`, `csvw:maximum`).
- Datetime column or boolean format (`csvw:format`).
- NULL (`csvw:null`) or default (`csvw:default`) values.

The database schema is improved by setting the datatype of each column and modifying the CSV data using the `csvw:format` property, so that they can be loaded correctly in the RDB instance. This step is very relevant in the case of the datetime columns, as there are multiple formats for defining a date[3], but generally the RDBMS only allows a specific one. Using the information provided by CSVW annotations, the format of a date column is known and data can be modified to be loaded correctly into the RDB instance.

Generation of Constraints. The last step is the generation of the RDB constraints. The naïve approach does not define any constraint over the generated database schema. This may affect the performance of query evaluation, since relevant columns (e.g., PK, FK) are usually indexed by most RDBMS.

Our approach defines the PK of tables as the column(s) involved in the generation of the subject. For example, the PK of the *people.csv* is composed by the columns *name*, *ln1*, and the PK of *comments.csv* is composed by the columns *username*, *date*, and *comment*. We also define the FK using the information provided in the `rr:parent` in a `rr:joinCondition` definition. This constraint can only be defined in case that the parent's reference is the PK of that table (e.g., join condition between countries and people). However, in the case of CSVs, joins may involve columns that are not part of the PK in their tables and some basic transformation functions. In this specific case, we create a new column in the table following the same approach as the `rr:objectMap` that contains transformation functions, the name of the predicate without the prefix as the name of the column. The generated column is indexed for improving the join condition over this column during the query evaluation. For example, in our motivating example, the join between people and comments involves several transformation functions over the `rr:joinCondition`, so a new indexed column, called *author*, is added to the schema of the *People* table. Then we populate the *author* column translating the basic transformation functions in the RML mapping to a SQL expression, as shown in Listing 1.1.

[3] `https://www.w3.org/TR/tabular-data-primer/#date-format`

Listing 1.1: Translated SQL expressions for populating join columns

$author_{people}$ = LOWER(CONCAT(SUBSTRING(name,1,1),ln1))
$username_{comments}$ = LOWER (username)

4.3 Mapping Translation: From RML+FnO to R2RML mappings

We generate a R2RML mappings using the provided RML+FnO mappings. In Figure 4 we show the original RML+FnO mapping with transformation functions to generate an implicit join and the translated R2RML mapping where the columns generated during the data translation process are taking into account to perform the implicit joins efficiently. We apply the following rules to translate RML+FnO to R2RML:

- The properties of RML are transformed to their corresponding R2RML properties[4]: `rml:logicalSource` to `rr:logicalTable`, `rml:source` to `rr:tableName` and `rml:reference` to `rr:column`.
- The `rr:objectMap` properties that contain `fnml:functionValue` are transformed to a `rr:column` property together with the reference to the new column generated in the aforementioned steps.
- The `rr:joinCondition` properties that contain `fnml:functionValue` are transformed to `rr:child` and `rr:parent` properties together with the references to the columns generated in the aforementioned steps.

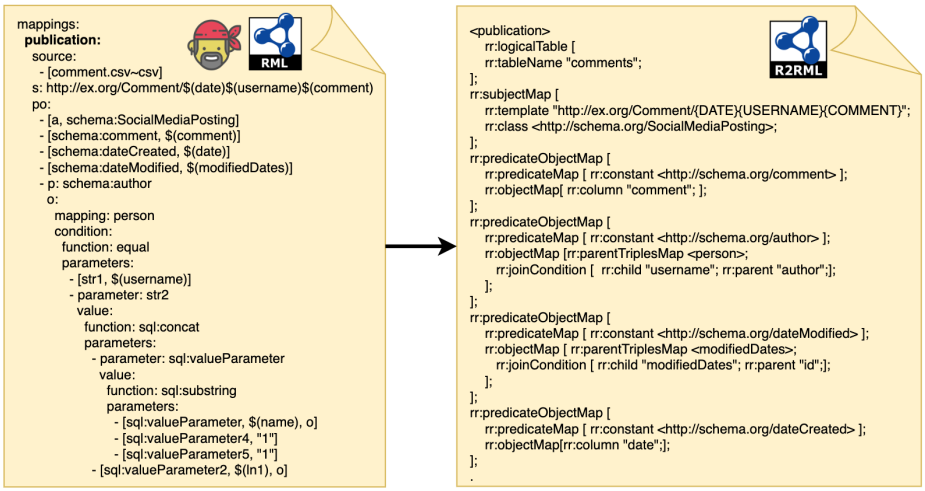


Fig. 4: Mapping Translation from RML+FnO (using YARRRML spec) to R2RML of publication TriplesMap

[4] `http://rml.io/RML_R2RML.html`

5 Evaluation

In this section we evaluate our approach and compare it with morph-RDB, an existing SPARQL-to-SQL OBDA engine that provides support for querying CSV files[5]. As far as we are aware, there are no general-purpose testbeds in the state of the art that allow validating our hypotheses. Therefore, we have created a testbed as follows: 1) we selected and downloaded a set of CSV files having challenges (e.g., NS, NU, MD, MM) that we identified in Section 2, available as open data, from the domain of public transport and general city data; 2) we created the corresponding annotations and mappings for these CSV files; 3) we proposed several SPARQL queries to exploit these data, with an increasing degree of complexity (Table 3). The transport data used in these experiments is defined using the GTFS specification[6]. This model provides information such as schedules, stops or routes using 13 different inter-related CSV files. The specification defines restrictions over the CSV columns, such as maximum, minimum or formats. One GTFS feed specifies the information of one type transportation mode (e.g., metro, train or tram). Linking these feeds makes it easier for route planners to consider several types of transportation modes to provide information to citizens about multimodal routes. The general city data used in our testbed is related to museums and sport-centres that contain a column with information of the public transport stops that are close to them.

We evaluate our proposal in two different ways: completeness and performance. In the first evaluation (Q1-Q5), we use real datasets (GTFS, museums and sport-centers) from a medium-sized city. As the size of these datasets are rather small, the time difference between our approach and an existing approach is neglectable, so we focus here on the completeness of the result. In the second evaluation (Q6-10), we take the GTFS data from the first evaluation and generate a set of synthetic datasets and we focus on the performance of the query evaluation.

Query	Description	# sources	Selectivity
Q1	Stops with the same name from different transport means	2	High
Q2	Stops nearby city museums	2	High
Q3	Stops nearby city sport centres	2	High
Q4	Stops from different transport means with same name and nearby a museum	3	High
Q5	Stops nearby museums and sport centres	3	High
Q6	List all the shapes	1	Low
Q7	List all possible routes from each stop	4	Low
Q8	List all wheelchair accessible stops along a particular route	4	High
Q9	List all non-wheelchair accessible stops along a particular route	4	High
Q10	List all the corresponding shapes from each trip	2	Low

Table 3: Queries used in our evaluation

[5] We tried other R2RML engines such as Ontop, but the executions failed during the parsing process of the R2RML mappings, which we have checked that are valid

[6] `https://developers.google.com/transit/gtfs/reference/`

5.1 Query result completeness on real datasets

We evaluate the queries (Q1-Q5) over a set of GTFS, shops and museum data. We use the GTFS stop file from two different transport means (train and tram) with 104 and 1263 rows respectively, together with 64 instances of museums and 74 instances of sport-centres. These datasets are published in an open data portal and we measure the number of responses (Table 4). In these cases, the column with information about the transport in original data from sport-centres and museums has the following structure: "TransportType1: StopName1,StopName2. TransportType2: StopName1,StopName2... TransportTypeN: StopName1.". Using the corresponding SQL function for regular expressions together with CSVW annotations about the separator character, the approach can generate the columns with the stopNames and the corresponding transport type. Additionally, transformation functions are applied to normalise stop names (`lower,trim`). We checked and compared the results from morph-RDB and our approach and verified that our approach gives a better result in terms of completeness, due to its ability to deal with heterogeneous challenges that we mentioned previously.

Query	Plain R2RML+database (using Morph-RDB)	Our proposal
Q1	45	**124**
Q2	0	**21**
Q3	0	**28**
Q4	0	**18**
Q5	0	**17**

Table 4: Number of results for each query and engine

5.2 Query evaluation performance on synthetic datasets

In the second evaluation we want to check the query evaluation performance and its relationship with dataset size. For this reason, we create the naïve RDB schema that represents GTFS and we use VIG [17][7] to generate multiple datasets with various scale factors and query them with queries Q6-Q10.

We measure the indexing time together with the query-execution time of each query. The hardware used to perform the experiments was an Intel(R) Xeon(R) CPU E5-2603 v3 @ 1.60GHz 20 cores, 100G memory with Ubuntu 16.04LTS. Each query has been evaluated three times with a timeout of 2 hours. The results are shown in Table 5 with the average time in seconds.

From the obtained results, we can see that our proposal fits better in the cases where query selectivity is low (Q7, Q10), when there are multiple joins among the tables (Q8) or when the original CSV data need transformations (Q9). For example, morph-RDB fails at Q9 because according to the GTFS spec, the false value of wheelchair_boarding property at stops table is 2 so it needs to be modified to load correctly in the RDB. Our proposal resolves this problem

[7] We configure VIG with multiple fixed columns to be consistent with the GTFS specification

Query	GTFS-1		GTFS-5		GTFS-10		GTFS-50		GTFS-100	
	E1	E2	E1	E2	E1	E2	E1	E2	E1	E2
Q6	**4.91**	5.84	**11.88**	14.69	**21.31**	26.52	**101.05**	183.04	**205.05**	932.06
Q7	**2.29**	3.97	30.58	**6.71**	115.39	**11.03**	2696.25	**106.20**	>2h	**773.99**
Q8	1.897	**1.792**	5476.05	**5.32**	>2h	**8.73**	>2h	**94.42**	>2h	**752.23**
Q9	error	**2.47**	error	**5.32**	error	**8.63**	error	**93.84**	error	**751.65**
Q10	>2h	**1222.7**	>2h	>2h	>2h	>2h	>2h	>2h	>2h	>2h

Table 5: Results of the query evaluation performance (time in seconds) over multiple sizes of a GTFS dataset (the number indicates the scale factor used in VIG). E1 corresponds to the use of morph-RDB on a plain database generated from the CSV files and E2 to our proposal. Error means that the number of results is 0.

applying transformation functions over the original data. The exploitation of the mappings and annotations of CSV to generate the enriched database schema that represents the original CSV files improves the query performance because the SPARQL-to-SQL optimisations (that normally exploit database index) can be applied. In cases where the queries are simple (e.g., Q6) the performance of the proposals is similar, as the optimisations do not have a major effect. In some cases, (Q6-GTFS100) the proposal fits worst because it is affected by the indexing time.

6 Related Work

morph-xR2RML [18] has been proposed as an extension of RML and provides query translation for JSON and MongoDB. The authors in [10] proposed the exploitation of widely used vocabularies (e.g., Hydra, DCAT) and their alignment with RML with the purpose of generating materialized RDF instances.

A concept that predates OBDA mappings is the one of adapter [27], proposed by Wiederhold. An adapter (aka wrapper) is used to transform heterogeneous data sources into a common data model, which can then be processed and integrated by a mediator. Classical examples of systems that implemented this concept were SIMS [2], Information Manifold [21], GARLIC [23]. A more specific context related to our work is that of approaches that allow querying directly information stored in files (e.g. [12], Drill[8], NoDB [1], Bash-Datalog [22]).

A lot of work has been made on supporting the transformations of spreadsheets into RDF, such as XLWrap [16], SML [25] or CSVW [26], and tools such as Tarql[9], Vertere-RDF[10], OpenRefine[11] or Karma[15].

Another area related to our work that is worth mentioning is the definition and application of data transformation functions into mappings, something that is especially common when transforming JSON or CSV data into RDF.

[8] `https://drill.apache.org/`
[9] `https://github.com/tarql/tarql`
[10] `https://github.com/mmmmmrob/Vertere-RDF`
[11] `http://openrefine.org/`

Works related to this topic are focused on developing ad-hoc and programmed functions. For example, R2RML-F [8] allows using functions in the value of the `rr:objectMap` property, so as to modify the value of the table columns from a relational database. Another example that follows the same approach, using RML and CSV files, is described in [14][13]. KR2RML [24], used in Karma, also extends R2RML by adding transformation functions in order to deal with nested values. OpenRefine allows such transformations with the usage of GREL functions, which can be also used in its RDF extension.

None of the aforementioned approaches take advantage of additional information in mappings and annotations provided by users in order to generate an enriched database representation together with its corresponding R2RML mappings with the aim to use existing off-the-shelf OBDA engines.

7 Conclusions and Future Work

In this paper, we have presented an approach for allowing OBDA-based query translation over heterogeneous CSV files. We have developed a framework that takes as input a set of CSV files, CSVW annotations, and an RML+FnO mapping, and generates as output an enriched RDB instance with data from the CSV files together with R2RML mappings, so that they can be used by any state-of-the-art R2RML-compliant OBDA engine. This is the first contribution that enables OBDA-based query translation over real-world CSVs, without the need to specify yet another mapping language and implement yet another OBDA engine or modify existing ones. As part of our future work, we will focus on improving the performance a of our approach proposing new optimisations in the query-translation process. In addition, we will study and extend our approach to be used with other types of data (e.g., XML, JSON).

References

1. I. Alagiannis, R. Borovica, M. Branco, S. Idreos, and A. Ailamaki. NoDB: efficient query execution on raw data files. In *Proc. ACM SIGMOD*, pages 241–252. ACM, 2012.
2. Y. Arens, C. Y. Chee, C.-N. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Intelligent and Cooperative Information Systems*, 2(02):127–158, 1993.
3. 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(3):471–487, 2017.
4. W. Carrara, C. Radu, and H. Vollers. *Open Data Maturity in Europe*. 2017.
5. E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)*, 4(4):397–434, 1979.
6. S. Das, S. Sundara, and R. Cyganiak. R2RML: RDB to RDF Mapping Language, W3C Recommendation 27 September 2012. *www.w3.org/TR/r2rml*, 2012.
7. B. De Meester, A. Dimou, R. Verborgh, and E. Mannens. An ontology to semantically declare and describe functions. In *ISWC*, pages 46–49. Springer, 2016.

8. C. Debruyne and D. O'Sullivan. R2RML-F: Towards Sharing and Executing Domain Logic in R2RML Mappings. In *LDOW@ WWW*, 2016.
9. A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van de Walle. RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data. In *LDOW*, 2014.
10. A. Dimou, R. Verborgh, M. V. Sande, E. Mannens, and R. Van de Walle. Machine-interpretable dataset and service descriptions for heterogeneous data access and retrieval. In *Proc. Semantics*, pages 145–152. ACM, 2015.
11. S. Household. Zero-based budgeting then and now: Technology remakes the ZBB rules. Accessed: 2018-12-07.
12. S. Idreos, I. Alagiannis, R. Johnson, and A. Ailamaki. Here are my data files. here are my queries. where are my results? In *Proceedings of 5th Biennial Conference on Innovative Data Systems Research*, number CONF, 2011.
13. A. C. Junior, C. Debruyne, R. Brennan, and D. O'Sullivan. FunUL: a method to incorporate functions into uplift mapping languages. In *Proceedings of the 18th International Conference on Information Integration and Web-based Applications and Services*, pages 267–275. ACM, 2016.
14. A. C. Junior, C. Debruyne, and D. OSullivan. Incorporating functions in mappings to facilitate the uplift of CSV files into RDF. In *ISWC*, pages 55–59. Springer, 2016.
15. C. A. Knoblock, P. Szekely, J. L. Ambite, A. Goel, S. Gupta, K. Lerman, M. Muslea, M. Taheriyan, and P. Mallick. Semi-automatically mapping structured sources into the semantic web. In *ESWC*, pages 375–390. Springer, 2012.
16. A. Langegger and W. Wöß. XLWrap–querying and integrating arbitrary spreadsheets with SPARQL. In *ISWC*, pages 359–374. Springer, 2009.
17. D. Lanti, G. Xiao, and D. Calvanese. VIG: Data scaling for OBDA benchmarks. *Semantic Web*, (Preprint):1–21.
18. F. Michel, L. Djimenou, C. Faron-Zucker, and J. Montagnat. Translation of relational and non-relational databases into RDF with xR2RML. In *11th International Conference on Web Information Systems and Technologies (WEBIST'15)*, pages 443–454, 2015.
19. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. In *Journal on data semantics X*, pages 133–173. Springer, 2008.
20. F. Priyatna, O. Corcho, and J. Sequeda. Formalisation and experiences of R2RML-based SPARQL to SQL query translation using morph. In *Proc. of WWW*, pages 479–490. ACM, 2014.
21. A. L. A. Rajaraman, J. Ordille, et al. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, 1996.
22. T. Rebele, T. P. Tanon, and F. Suchanek. Bash datalog: Answering datalog queries with unix shell commands. In *ISWC*, pages 566–582. Springer, 2018.
23. M. T. Roth and P. M. Schwarz. Don't scrap it, wrap it! a wrapper architecture for legacy data sources. In *VLDB*, volume 97, pages 25–29, 1997.
24. J. Slepicka, C. Yin, P. A. Szekely, and C. A. Knoblock. KR2RML: An Alternative Interpretation of R2RML for Heterogenous Sources. In *COLD*, 2015.
25. C. Stadler, J. Unbehauen, P. Westphal, M. A. Sherif, and J. Lehmann. Simplified RDB2RDF Mapping. In *LDOW@ WWW*, 2015.
26. J. Tennison, G. Kellogg, and I. Herman. Model for tabular data and metadata on the web. W3C recommendation. *World Wide Web Consortium (W3C)*, 2015.
27. G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.