Materialization of OWL Ontologies from Relational Databases: A Practical Approach

Sergio Alejandro Gómez[1,2(✉)] and Pablo Rubén Fillottrani[1,2]

[1] Laboratorio de I+D en Ingeniería de Software, Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, San Andrés 800, Bahía Blanca, Argentina
{sag,prf}@cs.uns.edu.ar

[2] Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC-PBA), La Plata, Argentina
https://lissi.cs.uns.edu.ar/

Abstract. Providing both end-users and applications with a uniform way to query legacy databases through a high-level ontology that models both the business logic and the underlying data sources is the main concern in Ontology-based Data Access (OBDA). Our goal in this research is providing tools for performing OBDA with relational and non-relational data sources. Within the OBDA framework, in this work, we present a prototype tool that can access an H2 database, allowing the user to explicitly express mappings, and populating an ontology that can be saved for later querying. We report on the current functionality of our tool, which includes creating, loading, saving a global ontology populated with a database or a CSV file. For the latter, we devised a language for specifying the underlying schema of the CSV file. We argue that this language is better suited than current alternatives such as JSON. Also, the system allows the user to visually express mappings from the database to the ontology and the ability to create databases for testing the behavior of the system in the presence of increasing workloads. Our tests indicate that the system can handle a moderate workload of tables of tens of thousands of records but fails to handle tables of millions of records.

Keywords: Ontology-based data access · Ontologies · Description Logics · Web Ontology Language · Relational databases

1 Introduction

Ontology-based Data Access (OBDA) [1,2] is concerned with providing end-users and applications with a way to query legacy databases through a high-level ontology that models both the business logic and the underlying data sources. Modern knowledge-based applications have replaced the representation of business logic by using a high-level representation of the business intelligence which is decoupled from the application code. This allows for improved flexibility. In Semantic Web applications [3], the business intelligence is represented by ontologies expressed in the Web Ontology Language 2 (OWL 2) [4]. Briefly, an ontology

P. Pesado and M. Arroyo (Eds.): CACIC 2019, CCIS 1184, pp. 285–301, 2020.
https://doi.org/10.1007/978-3-030-48325-8_19

is a logical theory formed by a collection of concepts and roles and also a set of concept and role assertions [5]. The relationship holding among the concepts and roles in the ontology are described in terms of inclusion and equality axioms. Ontologies used to represent business logic are then used by ontology reasoners to conclude implicit knowledge (i.e. not present in the database). The conclusions that can be got include making explicit the implicit terminology of concepts defined by the ontology, determining if a certain individual is a member of a concept, or determining if two individuals are related through a role, determining if a concept is subsumed by another concept, or if a role is subsumed by another role.

Thus, the classic OBDA architecture [1] is composed of a global database, a legacy database and a bridge between the ontology and the database. The bridge between the ontology and the data sources is addressed by mappings that define how to express records of the database as ontological assertions. Relational databases are comprised of relations (tables), that in term are defined by data schemas, which define the names and domains of table attributes as well as any integrity constraints that might apply to them, and are composed of records. Ontologies, on the other hand, are composed of axioms, and concept and roles assertions. The mappings define how to populate the ontology in terms of the elements of the database. Basically, the concept and role fillers are defined by SQL queries that indicate how to populate them. Notice that in the case of having several databases, a federation system can be used that allows to see the set of databases as a unified database. In this work, however, we will not take this possibility into account.

In this research, we are concerned with providing tools for performing OBDA with relational and non-relational data sources. Several tools have been developed by other research groups (see for instance [6–9] that we reviewed in [10]). Some of those tools are closed-source while others are open-source, some are downloadable and can be used as stand-alone applications or as programming libraries. While many times they are a good starting point for building applications, many times they are not flexible enough. In that regard, we are developing a tool, which nowadays is in a prototypical state, that can access an H2 database, allowing the user to explicitly formulate mappings, and populating an ontology that can be saved for later querying and visualization. See [10,11] for previous reports on the functionality of the application and its prospective application areas.

We present the advances we have made on the development of such a tool. In particular, we have added a form that allows end-users to fully specify in a high-level manner the nature of mappings and by writing SQL queries as well. We also added a module that allows testing on how our application behaves in the presence of increasing demands. We introduce a language that allows the user to precisely define the contents of a CSV file, we use that information to interpret the contents of a CSV file and then translating into OWL. We also discuss how this materialization tool could be used in the context of an e-government application. We provide with a downloadable prototype and a user manual at

http://cs.uns.edu.ar/~sag/obda-v4. We assume that the reader has a basic knowledge of Description Logics (DL) [12], relational databases [13] and the Web Ontology Language [4].

This work consolidates and extends results presented in [14]. We have included a new language for specifying the underlying schema of a CSV file, its implementation and an analysis of its performance. Also, we have also included an analysis of how this prototypical application could be integrated with an electronic government setting where public open data has to be machine processed.

The rest of the paper is structured as follows. In Sect. 2, we briefly recapitulate the concepts associated with materializing ontologies from tables. In Sect. 3, we present a novel development in the system that allows a naïve user to define a mapping from tables to ontologies in a visual manner. In Sect. 4, we present an alternative language for describing CSV meta information. In Sect. 5, we show an empirical evaluation of the performance of the prototype creating tables and ontologies. In Sect. 6, we present a case study where we show how the proposed application could be used for supporting data handling in an e-government application in a municipality. In Sect. 7, we review related work. In Sect. 8, we conclude and foresee future work.

2 Materialization of OWL Ontologies from Relational Databases

An ontology is a logical theory formed by set of axioms and assertions describing the business logic. The mappings describe how to map relational views into ontological vocabulary. Given a data access instance formed by a relational database $\mathcal{D}$, an ontological vocabulary $\mathcal{V}$, a set of ontological axioms $\mathcal{O}$ over $\mathcal{V}$, and a set of mappings $\mathcal{M}$ between $\mathcal{V}$ and $\mathcal{D}$, there are two approaches to answer a query Q over $\mathcal{V}$: (i) *materialization*: ontological facts are materialized (i.e. classes and properties participating in mappings are populated with individuals by evaluating SQL queries participating in mappings) and this gives a set of ontological facts $\mathcal{A}$ and then Q is evaluated against $\mathcal{O}$ and $\mathcal{A}$ with standard query-answering engines for ontologies, or (ii) *virtualization*: Q should be first rewritten into SQL using $\mathcal{O}$ and $\mathcal{M}$ and then SQL should be executed over $\mathcal{D}$.

In this work, we will only use the materialization approach. Materializing an OWL ontology from a relational database requires exporting the database contents as a text file in OWL format. For doing this, we need to export the schema information of each table as Tbox axioms and the instance data of the tables as Abox assertions. Here, we review the formalization for exporting database relations as ontologies as we presented it in [10] according to the directions given by [1,15]. Building an ontology from a database requires creating at least a class C_T for every table T, and for every attribute a of domain d in T we need two inclusion DL axioms $C_T \sqsubseteq \exists a$ and $\exists a^- \sqsubseteq d$. Primary key values k_i serve the purpose of establishing the membership of individuals to classes as DL Abox assertions of the form $C_T(C_T\#k^j)$. For indicating that a^j is the value of attribute a, we will use a role expression of the form $C_T\#a(C_T\#k^j, C_T\#a^j)$.

When it is clear from context, we might drop the prefix $C_T\#$ for simplifying our notation. A foreign key fk in table T_1 referencing a primary key field in table T_2 will also require to add two Tbox axioms $C_{T_1} \sqsubseteq \exists \mathsf{ref\_}fk$ and $\exists \mathsf{ref\_}fk^- \sqsubseteq C_{T_2}$ and an Abox assertion $\mathsf{ref\_}fk(k^j, fk^t)$ for expressing that the individual named k^j in C_{T_1} is related to the individual named fk^t in C_{T_2}. Besides, in any case, if we want to consider a subset of a table for its mapping into an ontology, we might define an SQL query that will act as an SQL filter. In this work, we will only deal with the translation into OWL of single tables and one-to-many relations (see [10] for details):

Definition 1 (Mapping of a table with a single primary key). *Let T be a table with schema $T(\underline{k}, a_1, \ldots, a_n)$ and instance $\{(k^1, a_1^1, \ldots, a_n^1), \ldots, (k^m, a_1^m, \ldots, a_n^m)\}$. To map T into a DL terminology $\mathcal{T}$, we have to create a class T and for each attribute a_i of domain D_i we have to add two axioms: $T \sqsubseteq \exists a_i$, indicating that every T has an attribute a_i, and $\exists a_i^- \sqsubseteq D_i$, meaning that the domain of a_i is D_i. The assertional box $\mathcal{A}$ for T will contain $\{T(k^1), \ldots, T(k^m)\}$. Given a key value k_j, $j = 1, \ldots, m$, for every attribute a_i, $i = 1, \ldots, n$, of the schema and instance value a_i^j (i.e. the value of i-th attribute of the j-th individual), produce a property $a_i(k^j, a_i^j)$.*

Example 1. Consider a table for representing people with schema Person (personID, name, sex, birthDate, weight) and instance as on the left side of Fig. (1). This table is created by the SQL script presented in the right side of Fig. (1).

personID	*name*	*sex*	*birthDate*	*weight*
1	John	true	2010-01-01	100.0
2	Mary	false	2009-01-01	60.0

```
create table "Person" (
        "personID" int unsigned not null
        auto_increment primary key,
        "name" varchar(20) not null,
        "sex" boolean, "birthDate" date,
        "weight" real );
insert into "Person"("name", "sex", "birthDate",
    "weight") values ('John', true, '2010-01-01', 100.0);
insert into "Person"("name", "sex", "birthDate",
    "weight") values ('Mary', false, '2009-01-01', 60.0);
```

Fig. 1. On the left, relational instance of the table Person and, on the right, SQL script for creating the table Person

The table Person is interpreted in Description Logics according to Definition 1, as $\Sigma = (\mathcal{T}, \mathcal{A})$ in Fig. 2. Description Logic ontologies are implemented in the OWL language, which includes an XML serialization which we partially present in Fig. 3 by showing the representation for John.

We now recall how to map two tables participating in a one-to-many relationship.

Definition 2 (Mapping of a one-to-many relationship). *Let $A(\underline{k_1}, a_1, \ldots, a_n)$ and $B(\underline{k_2}, b_1, \ldots, b_m, k_1)$ be two tables participating in a one-to-many relationship where k_1 is both the primary key in A and a foreign key in B.*

$$\mathcal{T} = \left\{ \begin{array}{ll} \mathsf{Person} \sqsubseteq \exists \mathsf{personID}, & \exists \mathsf{personID}^- \sqsubseteq \mathsf{Integer}, \\ \mathsf{Person} \sqsubseteq \exists \mathsf{name}, & \exists \mathsf{name}^- \sqsubseteq \mathsf{String}, \\ \mathsf{Person} \sqsubseteq \exists \mathsf{sex}, & \exists \mathsf{sex}^- \sqsubseteq \mathsf{Boolean}, \\ \mathsf{Person} \sqsubseteq \exists \mathsf{birthDate}, & \exists \mathsf{birthDate}^- \sqsubseteq \mathsf{Date}, \\ \mathsf{Person} \sqsubseteq \exists \mathsf{weight}, & \exists \mathsf{weight}^- \sqsubseteq \mathsf{Real} \end{array} \right\}$$

$$\mathcal{A} = \left\{ \begin{array}{ll} \mathsf{Person}(\mathsf{Person}\#1), & \mathsf{personID}(\mathsf{Person}\#1, 1), \\ \mathsf{name}(\mathsf{Person}\#1, \mathsf{John}), & \mathsf{sex}(\mathsf{Person}\#1, \mathsf{true}), \\ \mathsf{birthDate}(\mathsf{Person}\#1, 2001\text{-}01\text{-}01), & \mathsf{weight}(\mathsf{Person}\#1, 100.0), \\ \mathsf{Person}(\mathsf{Person}\#2), & \mathsf{personID}(\mathsf{Person}\#2, 2), \\ \mathsf{name}(\mathsf{Person}\#2, \mathsf{Mary}), & \mathsf{sex}(\mathsf{Person}\#2, \mathsf{false}), \\ \mathsf{birthDate}(\mathsf{Person}\#2, 2009\text{-}01\text{-}01), & \mathsf{weight}(\mathsf{Person}\#2, 60.0) \end{array} \right\}.$$

Fig. 2. Ontology $\Sigma = (\mathcal{T}, \mathcal{A})$ representing the table *Person* from Example 1

```
<owl:Class rdf:about="http://cs.uns.edu.ar/~sag#Person"/>
<!-- http://cs.uns.edu.ar/~sag/Person/personid=1 -->

<owl:NamedIndividual rdf:about="http://cs.uns.edu.ar/~sag/Person/personid=1">
<rdf:type rdf:resource="http://cs.uns.edu.ar/~sag#Person"/>
<Person:birthDate rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime">
2010-01-01T00:00:00</Person:birthDate>
<Person:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">John</Person:name>
<Person:personID rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">1</Person:personID>
<Person:sex rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</Person:sex>
<Person:weight rdf:datatype="http://www.w3.org/2001/XMLSchema#double">100.0</Person:weight>
</owl:NamedIndividual>
```

Fig. 3. Part of the OWL code for the definition of the class Person from Example 1

Tables A and B are translated in DL according to Definition 1. Besides, the two axioms are added: $B \sqsubseteq \exists \mathsf{ref}\_k_1.A$ *and* $\exists \mathsf{ref}\_k_1^-.B \sqsubseteq A$. *And for every tuple* $(k_1^i, a_1^i, \ldots, a_n^i)$ *of A related to a tuple* $(k_2^j, b_1^j, \ldots, b_m^j, k_1^i)$ *in B, an assertion* $\mathsf{ref}\_k_1(k_2^j, k_1^i)$ *is added.*

Example 2 (Continues Example 1). Consider a one-to-many relation of table Person from Example 1 with a table Phone(phoneNumber, personID), populated as shown in Fig. 4. Notice that personID is a foreign key referencing table Person.

phoneNumber (pk)	*personID (fk)*
555-0000	1
555-0001	1

Fig. 4. Relational instance of table Phone from Example 2

Notice that phoneNumber is the primary key while personID is a foreign key referencing key-values of the table Person. Concerning the one-to-many relation and according to Definition 2, two axioms are added to the ontology: $\mathsf{Phone} \sqsubseteq \exists \mathsf{ref}\_\mathsf{personID}.\mathsf{Person}$ and $\exists \mathsf{ref}\_\mathsf{personID}^-.\mathsf{Phone} \sqsubseteq \mathsf{Person}$. Let $p_1 = \mathsf{Phone}\#555\text{-}0000$ be an IRI for the first phone and $p_2 = \mathsf{Phone}\#555\text{-}0001$ for the second one. The assertions $\mathsf{Phone}(p_1)$, $\mathsf{phoneNumber}(p_1, 555\text{-}0000)$, $\mathsf{personID}(p_1, 1)$,

ref_personID(p_1, Person#1), Phone(p_2), phoneNumber(p_2, 555-0001), personID (p_2, 1), ref_personID (p_2, Person#1), are then added to the ontology indicating that 555-0001 and 555-0002 are phone numbers and that the person with id 1 owns these phone numbers. Notice how the IRIs for the phones are built concatenating both the name of the class and the value of the respective key values. Assertions prefixing the name of the field with ref_ that relate the person and his/her phone are added too.

3 Visual Mapping Specification

The specification of the mappings for obtaining the fillers of concept from a table is usually a complex matter for naïve end-users. Remember that a mapping is basically a SQL query that defines how the fillers of concept, property or role are computed in terms of the contents of a database. When there is no support for composing mappings, the user has to write such SQL from scratch. We believe that adding support for building the mappings will improve the user experience of a prospective user of OBDA technology.

With the idea of providing support to end-users in their quest of creating concepts for populating ontologies from database contents, we created a module that allows to visually specify a mapping from a table. The module retrieves the tables from the database, and allows to select a table. Once the table is selected, its fields can be selected too. The user can then introduce what conditions each field of the table has to satisfy. Besides, one field (usually the key field of the table) has to be selected to fill the concept. The module then will automatically generate the SQL filter for filling the concept by extracting the records from the table, and will also add a subclass axiom to the ontology.

Example 3. Consider again the table Person from Example 1 and suppose that some user of the system wants to define the concept "heavy, young, male individual". Suppose also that the user models a heavy individual as somebody who weighs at least a hundred kilograms, a young individual as someone who was born after 2001, and a male individual as someone of male sex. People of male sex are codified as having the column named sex as true while females are codified as false. Although this is a trivial example, it shows the complexities that run into database modeling that produce a degradation of the representation of the world and that are unretrievable afterwards. The user will then visually specify the conditions for an individual to be a member of the concept YoungHeavyMalePerson in a form like the one presented in Fig. 6. Notice how the user specifies which database field corresponds to the key (i.e. the name of the individuals), in this case personID. In turn, the system will generate a SQL query as shown in Fig. 5.

After the execution of the query that will compute the individuals that fill the concept, the system will add to the current ontology the triples expressing that those individuals are the fillers of the concept YoungHeavyMalePerson. Besides, in order to relate this concept to its superconcept, the axiom YoungHeavyMalePerson $\sqsubseteq$ Person will be added to the current ontology as well.

```
SELECT "Person"."personID" FROM "Person"
WHERE "Person"."birthDate" >= '2001-01-01'
AND "Person"."weight" >= 100 AND "Person"."sex" = true
```

Fig. 5. SQL query for the specification of the concept YoungHeavyMalePerson of Example 3

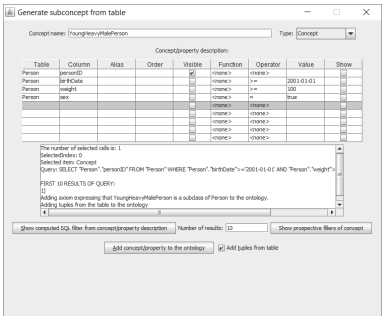

Fig. 6. Visual concept specification of the concept YoungHeavyMalePerson

This will lead to the situation presented in Fig. 7. The new class YoungHeavyMalePerson is defined as a subclass of Person and John, whose personID role is "1" becomes a member of YoungHeavyMalePerson. Notice also that no new individuals are defined as John is already present in the ontology because he is a Person. In this sense, we adhere to the unique name assumption as much as we can although this is not required by the formalism. Also notice how the intensional definition of the concept is lost in the ontology (other than being a subclass of Person) and only its extension is maintained in the ontology (as the set of its fillers).

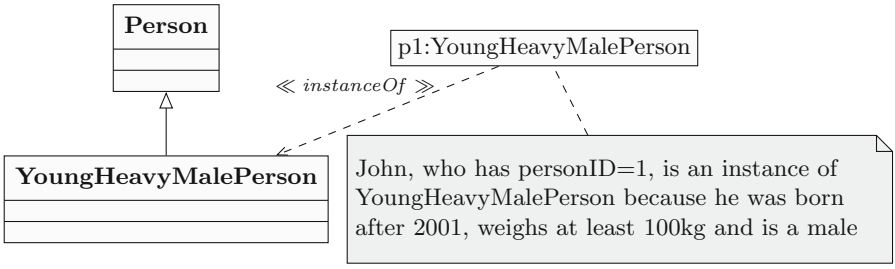

Fig. 7. Situation arisen by specifying a subclass of Person named YoungHeavyMale Person

Another feature that the current version of the system includes is the possibility of specifying a subclass by means of an explicit SQL query.

Example 4. Continuing Example 3, the concept FemalePerson (which defines a subset of the table Person formed by women) is specified by means of the SQL query:

```
SELECT "personID" FROM "Person" WHERE "sex" = false
```

This can be done by using the form presented in Fig. 8. Notice the additional OWL code in the ontology generated by out tool which is presented in Fig. 9 expressing that a female person is a person (i.e. FemalePerson $\sqsubseteq$ Person is an axiom in the ontology) and that Mary is both a female person and a person (i.e. FemalePerson(Mary) and Person(Mary) are assertions in the ontology).

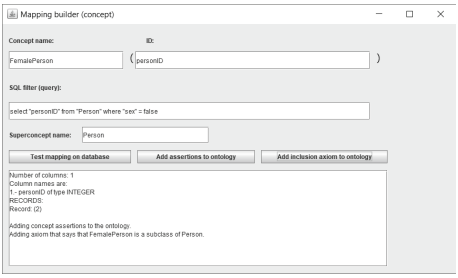

Fig. 8. Specification of the subclass FemalePerson of Person by a SQL query

```
<owl:Class rdf:about="http://cs.uns.edu.ar/~sag#FemalePerson">
<rdfs:subClassOf rdf:resource="http://cs.uns.edu.ar/~sag#Person"/>
</owl:Class>
...
<owl:NamedIndividual rdf:about="http://cs.uns.edu.ar/~sag/FemalePerson=2">
<rdf:type rdf:resource="http://cs.uns.edu.ar/~sag#FemalePerson"/>
...
</owl:NamedIndividual>
```

Fig. 9. Portion of OWL code for introducing subconcept FemalePerson

4 Specification of Schemas for CSV Files

A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields enclosed in delimiters and separated by commas. A CSV file stores tabular data (numbers and text) in plain text, in which case each line will have the same number of fields. Comma separated files are used for the interchange of database information between machines of two different architectures. The plain-text character of CSV files largely avoids incompatibilities such as byte-order and word size. The files are human-readable, so it is easier to deal with them in the absence of perfect documentation or communication.

Example 5. In Fig. 10, we show the CSV table for the table Person of Example 1.

```
"personID","name","sex","birthDate","weight"
"1","John","true","2010-01-01","100.00"
"2","Mary","false","2009-01-01","60.00"
```

Fig. 10. CSV file for the table Person of Example 1

Despite its simplicity, the lack of both standardization and schema information in CSV files poses a disadvantage, forcing application programs to guess or ask the user for delimiter and field-separators characters. For solving this problem, the W3C Working Group has proposed a format for specifying CSV metadata [16] based mostly in JSON (JavaScript Object Notation)[1]. Although this solution works in practice, we think that JSON files, although their human readability, are not simple enough for naive users. We then propose a simpler language for specifying the schema (or meta information) of a CSV file as defined by the BNF grammar presented in the left side of Fig. 11. We believe that our language is simple enough to be human-readable and complex enough for its purpose. The declarations have to be *sound* (i.e each declared field in CSV Meta information file must be present in the CSV file), *complete* (i.e. each field in the CSV file must be declared in the CSV meta information file) and *ordered* (i.e. the order in which fields appear in both the CSV file and the CSV meta information file must be the same).

In the right side of Fig. 11, we provide an example of a file for defining the schema of a CSV file that would represent the data provided in Example 5. Although the declarations for *number-of-key-fields* and *number-of-fields* seem redundant, we think that they offer a way for validating that the user is doing things correctly. Valid identifiers begin with a letter and continue with letters and numbers. For now we consider only the types: integer, real, string, boolean and date, but this can be easily extended. If the number of rows to be translated is not specified then *all* is assumed by default. If no field separator is specified, the comma is assumed by default. If no quotation character is specified, the double quotation mark is assumed by default. The parser validates that both the number of key-fields and fields declared matches those that were defined.

The contents of the CSV file are validated and processed according to the definitions given in the CSV schema file. Then the contents of the CSV file are loaded into an H2 database which is translated into OWL as explained in Sect. 2. Our implementation approach parses first the meta-information schema file and then the CSV file; it then generates a SQL script that is used to create an H2 table, that is translated into OWL.

[1] See https://www.w3schools.com/js/js_json_intro.asp (checked on 2020-02-20).

```
⟨list-commands⟩ ::= ⟨command⟩
    | ⟨command⟩⟨list-commands⟩

⟨command⟩ ::= ⟨field-separator-def⟩
    | ⟨quote-separator-def⟩
    | ⟨class-def⟩
    | ⟨number-of-key-fields-def⟩
    | ⟨number-of-fields-def⟩
    | ⟨key-field-def⟩
    | ⟨field-def⟩
    | ⟨number-of-rows-def⟩

⟨field-separator-def⟩ ::= field-separator ⟨character⟩

⟨quotation-separator-def⟩ ::= quotation-character-for-fields ⟨character⟩

⟨class-def⟩ ::= class-name ⟨identifier⟩

⟨number-of-key-fields-def⟩ ::= number-of-key-fields ⟨positive-integer⟩

⟨number-of-fields-def⟩ ::= number-of-fields ⟨positive-integer⟩

⟨key-field-def⟩ ::= keyfield ⟨identifier⟩ type ⟨type-id⟩

⟨field-def⟩ ::= field ⟨identifier⟩ type ⟨type-id⟩

⟨number-of-rows-def⟩ ::= number-of-rows-to-translate ⟨quantity-def⟩

⟨type-id⟩ ::= integer
    | real
    | string
    | boolean
    | date

⟨quantity-def⟩ ::= all
    | ⟨positive-integer⟩
```

```
field-separator ,
quotation-character-for-fields "
class-name Person
number-of-key-fields 1
key-field personID type integer
number-of-fields 4
field name type string
field sex type boolean
field birthDate type date
field weight type real
number-of-rows-to-translate all
```

Fig. 11. On the left, BNF grammar for the meta information language for defining CSV schemas, and, on the right, example of providing schema information for the CSV file in Fig. 10

5 Experimental Evaluation

We now discuss some of the tests we have performed in order to test how our application handles increasing demands in database size. The performance of our system is affected mainly by the fact that we chose to materialize tables as triples (i.e. class membership, property and roles assertions) and also by three factors: (i) the system is implemented in the JAVA programming language; (ii) the database management system that we use is H2[2], and, (iii) the handling of the global ontology is done via the OWL API [17,18].

Our tests were conducted on an ASUS notebook having an Intel Core i7, 3.5 GHz CPU, 8 GB RAM, 1 TB HDD, Windows 10. They involved the creation of simple databases composed by a single table containing 100 fields of text type filled with an increasing number of records. Table 1 summarizes our results. As it can be seen, our implementation starts having problems at tables with 100,000 records; although an ontology can be generated and saved to the disk, when we try to load the ontology we saved previously, we get an error inside the code of the OWL API, indicating that library cannot handle such a data load. When running a test for creating a database of a million records, the H2 database produces an error (which is understandable as it is maintained in RAM). Likewise, in

[2] See http://www.h2database.com.

Table 1. Running times for ontology generation from H2 database

Number of records	Database file size [Megabytes]	Time for creating the ontology [seconds]	Ontology file size [Megabytes]	Time for loading the ontology [seconds]
1,000	0.80	1.029	8.65	4.014
10,000	8.82	5.345	87.26	15.106
100,000	98.11	66.48	19,053.36	Out of memory error
1,000,000	1,080.60	Out of memory error	–	–

Table 2. Running times for ontology generation from CSV file

Number of records	CSV file size [Megabytes]	Time for loading CSV file [seconds]	Time for creating ontology [seconds]	Size of ontology file [Megabytes]	Time for loading ontology from disk [seconds]
100	0.048	1.313	0.437	0.871	1.579
1,000	0.568	1.764	1.111	8.560	4.784
10,000	6.636	9.161	3.743	8.215	12.577
100,000	75.987	85.855	37.137	872.883	Out of memory error
1,000,000	856.188	936.473	–	–	–

Table 2, we can see the times for loading CSV files of 100 fields containing integer values and also an increasing number of records. Therefore, we conclude that our application can only handle tables with a size tens of thousands records and is not able of handling tables of a hundred thousand records. Because of this limitation, we think that we will be forced to use query-rewriting techniques [1] for delegating the evaluation of queries to the database management system instead of the ontology management library.

6 Case Study: Support for e-Gov in Municipalities

The importance of social policies has grown in recent years at all levels of government (whether municipal, provincial, national and international), since they represent one of the main tools to combat economic inequalities that occur globally [19] and serve to meet the needs of many vulnerable groups. The provision of public social action services to citizens become an obligation for governments, just as such services are a human right, such as are access to water, energy, health, education and other services.

Despite the global relevance, universality in the provision of public services is a challenge for each government due to the variety of contexts in which such services are provided, including the needs of specific social groups, the capacities of each government, and the context-specific conditions (such as territory, political, cultural, economic, etc.) [19,20]. In some municipalities of the province of Buenos Aires in Argentina, the following challenges are observed for the provision of public social action services: (i) the services are provided by several municipal government agencies and there is no consolidated information on how the services are being delivered; (ii) currently, there are ad hoc applications that support the process for the delivery of each service, these applications work in isolation, without sharing data; (iii) there is no strategy for the delivery of these services using multiple channels; (iv) the digital channels that could be used are not being exploited properly; (v) there is no software infrastructure that allows the rapid development of applications for the delivery of social action services [21].

Based on these challenges, it is necessary to find a way to publish data contained in legacy and current applications and used in various state institutions in a way that can be integrated, accessed, modified and consulted in a format that is uniform, distributed and scalable. In this sense, the technologies of the Semantic Web have matured enough to be considered as a viable solution for the publication and integration of institutional data. In particular, using semantics implies conceiving systems where the meaning of the data is explicitly specified and is taken into account to design their functionalities. This idea has become crucial for a wide variety of information processing applications and has received much attention in the artificial intelligence, database, web and data mining communities.

As a case study of the operation of the application developed, we present an example loosely based on the public data available in the Municipality of Bahía Blanca. A preliminary version was presented in [11][3]. Let us take as an example three tables, presented in Fig. 12 with the details of the beneficiaries of all social assistance in the period selected in the Municipality of Bahía Blanca[4], where we have a table called *Program* representing social assistance programs, another with the beneficiaries called *Beneficiary* and a third called *Person* with the data of the people enrolled in the aid programs.

The program table schema contains the identifier (which is the primary key) and program name. The table of beneficiaries of aid programs contains the identification (id) of the benefit (which is the primary key of the table), the document number of the beneficiary (which is a foreign key), the amount received, the date of reception and the help program for which the help was received (which is a foreign key). The people table contains the person's social ID (personal document) number (which is the primary key) and his/her last and first name.

[3] The authors would like to thank funding from Comisión de Investigaciones Científicas (Project *Herramientas para el desarrollo y la entrega de servicios públicos digitales de acción social para municipios bonaerenses – PIT-AP-BA 2016).*

[4] See http://datos.bahiablanca.gob.ar/dataviews/74266/social-aids.

Program table

id(pk)	*name*
0001	Municipal Occupational Training Program

Beneficiary table

id(pk)	*dni(fk)*	*amount*	*date receiving help*	*program(fk)*
1	22,000,000	12000	2019-03-10	0001
2	26,000,000	16000	2019-05-04	0001
3	22,000,000	13000	2019-05-04	0001

Person table

dni(pk)	*name*
22,000,000	Smith, John
26,000,000	Doe, Jane

Fig. 12. Relational tables from Bahia Blanca municipality public information site

The Tbox axioms in Fig. 13 represent the schema information of the tables in Fig. 12 and the Abox assertions in Fig. 14 represent the relational instance.

$\mathsf{Program} \sqsubseteq \exists \mathsf{id},$
$\exists \mathsf{name}^{-} \sqsubseteq \mathsf{String},$
$\mathsf{Beneficiary} \sqsubseteq \exists \mathsf{dni},$
$\exists \mathsf{ref\_dni}^{-} \sqsubseteq \mathsf{Beneficiary},$
$\mathsf{Beneficiary} \sqsubseteq \exists \mathsf{dateReceivingHelp},$
$\exists \mathsf{program}^{-} \sqsubseteq \mathsf{Integer},$
$\mathsf{Person} \sqsubseteq \exists \mathsf{dni},$
$\exists \mathsf{name}^{-} \sqsubseteq \mathsf{String}.$

$\exists \mathsf{id}^{-} \sqsubseteq \mathsf{String},$
$\mathsf{Beneficiary} \sqsubseteq \exists \mathsf{id},$
$\exists \mathsf{dni}^{-} \sqsubseteq \mathsf{Integer},$
$\mathsf{Beneficiary} \sqsubseteq \exists \mathsf{amount},$
$\exists \mathsf{dateReceivingHelp}^{-} \sqsubseteq \mathsf{Date},$
$\mathsf{Beneficiary} \sqsubseteq \exists \mathsf{ref\_program.Program},$
$\exists \mathsf{dni}^{-} \sqsubseteq \mathsf{Person},$

$\mathsf{Program} \sqsubseteq \exists \mathsf{name},$
$\exists \mathsf{id}^{-} \sqsubseteq \mathsf{Integer},$
$\mathsf{Beneficiary} \sqsubseteq \exists \mathsf{ref\_dni.Person},$
$\exists \mathsf{amount}^{-} \sqsubseteq \mathsf{Real},$
$\mathsf{Beneficiary} \sqsubseteq \exists \mathsf{program},$
$\exists \mathsf{ref\_}\mathit{program}^{-} \sqsubseteq \mathsf{Program}.$
$\mathsf{Person} \sqsubseteq \exists \mathsf{name},$

Fig. 13. Terminology for the schema of the data from the municipality

Next, we present the technology for querying the OWL ontologies materialized from this database. SPARQL (SPARQL Protocol and RDF Query Language) [22] is a declarative query language for RDF that allows to retrieve and manipulate data stored in the Semantic Web represented as RDF statements. A SPARQL endpoint accepts queries for any web-accessible RDF data and returns results via HTTP. The results of SPARQL queries can be returned in a variety of formats such as XML, JSON, RDF and HTML. Then, with this solution, data from the municipality's administration can be published in an uniform, public, modern, open format that can be queried both by people and by applications. For instance, finding who were the ten people who, during the year 2019, collected the most of a plan of at least $\$15,000$ could be done by means of the SPARQL query as shown in Fig. 15. This would allow both applications to use it as a web service or end-users users to query the data in a web page.

7 Related Work

With ViziQuer, Cerans et al. [23] provide an open source tool for web-based creation and execution of visual diagrammatic queries over RDF/SPARQL data. The tool supports the data instance level and statistics queries, providing visual counterparts for most of SPARQL 1.1 select query constructs, including aggregation and subqueries. A query environment can be created over a user-supplied SPARQL endpoint with known data schema. ViziQuer provides a visual interface

for expressing user queries in SPARQL posed against an ontology. In contrast, we provide the user with an interface for describing subclass expressions and inclusion axioms by means of restrictions imposed on records of a relational table with the aim of populating an ontology that could later be exposed and queried as an SPARQL endpoint.

$$
\begin{array}{lll}
\mathsf{Program}(0001), & \mathsf{name}(0001, 'MunicipalOccupationalTrainingProgram'), & \\
\mathsf{Beneficiary}(1), & \mathsf{id}(1,1), & \mathsf{dni}(1,22000000), \\
\mathsf{amount}(1,12000), & \mathsf{dateReceivingHelp}(1,\text{'2019-03-10'}), & \mathsf{program}(1,0001), \\
\mathsf{Beneficiary}(2), & \mathsf{id}(2,2), & \mathsf{dni}(2,26000000), \\
\mathsf{amount}(2,16000), & \mathsf{dateReceivingHelp}(2,\text{'2019-05-04'}), & \mathsf{program}(2,0001), \\
\mathsf{Beneficiary}(3), & \mathsf{id}(3,3), & \mathsf{dni}(3,22000000), \\
\mathsf{amount}(3,13000), & \mathsf{dateReceivingHelp}(2,\text{'2019-05-04'}), & \mathsf{program}(2,0001), \\
\mathsf{Person}(22000000), & \mathsf{name}(22000000, 'Smith, John'), & \mathsf{Person}(26000000), \\
\mathsf{name}(26000000, 'Doe, Jane'). & &
\end{array}
$$

Fig. 14. Relational instance from the data of the municipality

```
PREFIX mbb <http://www.mbb.gov.ar/>
SELECT ?name ?date ?amount
FROM <http://www.mbb.gov.ar>
WHERE
{
    ?beneficiary mbb:ref_dni ?person .
    ?beneficiary mbb:amount ?amount .
    ?beneficiary mbb:date_receiving_help ?date .
    ?person mbb:name ?name .
    BIND (YEAR(?date) as ?year)
    FILTER (?year = 2019 && ?amount >= 15000)
}
ORDER BY DESC(?amount)
LIMIT 10
```

```
-----------------------------------------
| name          | date       | amount |
=========================================
| "Doe, Jane"   | 2019-05-04 | 16,000 |
-----------------------------------------
```

Fig. 15. On the left, the SPARQL query for finding who were the ten people who, during the year 2019, collected the most of a plan of at least $15,000; and, on the right, the result of the execution of the query by a SPARQL engine against the ontology data in Fig. 14

Christodoulou et al. [24] make the case that structural summaries over linked-data sources can inform query formulation and provide support for data integration and query processing over multiple linked-data sources. To fulfil this aim, they propose an approach that builds on a hierarchical clustering algorithm for inferring structural summaries over linked-data sources. Thus, their approach takes as input an RDF repository and then reverse engineers an ontology using clustering techniques to detect prospective classes. In contrast, we take a database and the user proposes SQL queries to express subconcepts intensionally; when the SQL queries are executed, the fillers of the concept populate the ontology building an extensional de facto definition of the concept. In that regard, the work of Christodoulou et al. can be considered complementary to ours.

Barrasa et al. [25] propose R2O, an extensible and declarative language to describe mappings between relational DB schemas and ontologies implemented in RDF(S) or OWL. R2O provides an extensible set of primitives with well defined semantics. This language has been conceived expressive enough to cope

with complex mapping cases arisen from situations of low similarity between the ontology and the DB models. R2O allows the user to express complex queries in terms of ontologies in a language that is similar to the relational algebra but aimed at ontologies. Therefore, this approach is complementary to ours because it allows to query the ontology once it is published in OWL format.

8 Conclusions and Future Work

We presented a framework for performing ontology-based data access by means of performing a materialization approach. Our implementation is JAVA-based and relies on a H2 database management system and a JAVA library called OWL-API for accessing and querying databases and maintaining an OWL database in main memory, respectively. We presented several enhancements that we have made to the previous iteration of our prototype implementation, which now includes a visual mapping specification functionality and allows to maintain a global database that can be either created from scratch or loaded from disk, modified and later saved again to disk. From the experimental evaluation to which we subjected our system, we conclude that our application is able to handle a moderate workload of a table of tens of thousands of records but fails to handle table of the order of millions of records. In this regard, we think that we will be forced to use query-rewriting techniques for delegating the evaluation of queries to the database management system instead of the ontology management library. Part of our current research efforts are aimed in this direction. Other form of improvement lies in the possibility of addressing the federation of databases for performing integration of multiple heterogeneous data sources. We introduced a language for defining the schema information of a CSV file, and then how to interpret the contents of a CSV file for performing its translation into OWL. We discussed how this materialization tool could be used in the context of an e-government application showing how relational data can be publish as open data in OWL.

Acknowledgments. This research is funded by Secretaría General de Ciencia y Técnica, Universidad Nacional del Sur, Argentina and by Comisión de Investigaciones Científicas de la Provincia de Buenos Aires (CIC-PBA).

References

1. Kontchakov, R., Rodríguez-Muro, M., Zakharyaschev, M.: Ontology-based data access with databases: a short course. In: Rudolph, S., Gottlob, G., Horrocks, I., van Harmelen, F. (eds.) Reasoning Web 2013. LNCS, vol. 8067, pp. 194–229. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39784-4_5
2. Xiao, G., et al.: Ontology-based data access - a survey. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI 2018), pp. 5511–5519 (2018)
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Sci. Am. **284**(5), 34–43 (2001)

4. Bao, J., Kendall, E.F., McGuinness, D.L., Patel-Schneider, P.F.: OWL 2 Web Ontology Language Quick Reference Guide, 2nd edn. W3C Recommendation, 11 December 2012
5. Gruber, T.R.: A translation approach to portable ontologies. Knowl. Acquisition **5**(2), 199–220 (1993)
6. Calvanese, D., Giacomo, G.D., Lembo, D., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web **2**(1), 43–53 (2011)
7. Jimenez-Ruiz, E., et al.: BootOX: practical mapping of RDBs to OWL 2. In: The 14th International Semantic Web Conference, pp. 113–132 (2015)
8. de Medeiros, L.F., Priyatna, F., Corcho, O.: MIRROR: automatic R2RML mapping generation from relational databases (2015)
9. Pinkel, C., et al.: RODI: benchmarking relational-to-ontology mapping generation quality. Semantic Web, 1–26 (2016)
10. Gómez, S.A., Fillottrani, P.R.: Towards a framework for ontology-based data access: materialization of OWL ontologies from relational databases. In: Pesado, P., Aciti, C. (eds.) X Workshop en Innovación en Sistemas de Software (WISS 2018), XXIV Congreso Argentino de Ciencias de la Computación, CACIC 2018, pp. 857–866 (2018)
11. Gómez, S.A., et al.: Desarrollo de herramientas para acceso a bases de datos heterogéneas basado en ontologías en el contexto de la entrega de servicios públicos digitales. Primer Encuentro de Centros Propios y Asociados de la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, pp. 235–238 (2018)
12. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: An Introduction to Description Logic. Cambridge University Press, Cambridge (2017)
13. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, 6th edn. McGraw-Hill Education, New York (1983)
14. Gómez, S.A., Fillottrani, P.: A framework for OBDA - current state and perspectives. In: Actas del XXV Congreso Argentino de Ciencias de la Computación (CACIC 2019), pp. 920–929 (2019)
15. Arenas, M., Bertails, A., Prud'hommeaux, E., Sequeda, J.: A Direct Mapping of Relational Data to RDF. W3C Recommendation, 27 September 2012
16. Tennison, J.: CSV on the Web - A Primer, W3C working group note, 25 February 2016
17. Horridge, M., Bechhofer, S.: The OWL API: a Java API for OWL ontologies. Semantic Web **2**(1), 11–21 (2011)
18. Matentzoglu, N., Palmisano, I.: An Introduction to the OWL API. Technical report, The University of Manchester (2016)
19. Bertot, J., Estevez, E., Janowski, T.: Universal and contextualized public services - digital public service innovation framework. Gov. Inf. Q. **33**, 211–222 (2016)
20. Estévez, E., Fillottrani, P., Janowski, T., Ojo, A.: Government information sharing - a framework for policy formulation. In: Pin-Yu, C., Yu-Che, C. (eds.) E-Governance and Cross-boundary Collaboration - Innovations and Advancing Tools, pp. 23–55. IGI Global (2011)
21. Fillottrani, P., Estévez, E., Cenci, K., Pesado, P., Pasini, A., Thomas, P.: Herramientas para el desarrollo y la entrega de servicios públicos digitales de acción social para municipios bonaerenses. In: IV Congreso Internacional Científico y Tecnológico-CONCYT 2017 (2017)
22. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language for RDF W3C recommendation, 21 March 2013. https://www.w3.org/TR/rdf-sparql-query/

23. Čerāns, K., et al.: ViziQuer: a web-based tool for visual diagrammatic queries over RDF data. In: Gangemi, A., et al. (eds.) ESWC 2018. LNCS, vol. 11155, pp. 158–163. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98192-5_30
24. Christodoulou, K., Paton, N.W., Fernandes, A.A.: Structure inference for linked data sources using clustering. In: Proceedings of the Joint EDBT/ICDT 2013 Workshops, EDBT 2013, pp. 60–67 (2013)
25. Barrasa, J., Corcho, Ó., Gómez-Pérez, A.: R2O, an extensible and semantically based database-to-ontology mapping language. In: Proceedings of the Second Workshop on Semantic Web and Databases, SWDB 2004, vol. 3372 (2004)