Automatic Mapping Rules and OWL Ontology Extraction for the OBDA *Ontop*

Fayez Khazalah
Wayne State University

Zaki Malik
Wayne State University

Abdelmounaam Rezgui
New Mexico Tech

***Abstract*—Extracting *Ontop* mapping rules and OWL ontology manually from a relational schema is a tedious task. We present an automatic approach for extracting *Ontop* mappings and OWL ontology from an existing database schema. The end users can access the underlying data source through SPARQL queries. A SPARQL query is written according to the extracted ontology and the end user does not need to know about the underlying data source and its schema. The proposed approach takes into consideration the different relationships between entities of the database schema. Instead of extracting a flat ontology that is an exact copy of the database schema, it extracts a rich ontology. The extracted ontology can also be used as an intermediate between a domain ontology and the underlying database schema. The experiment results indicate that the extracted mappings and ontology are accurate. i.e., end users can query all data (using SPARQL) from the underlying database source in the same way as if they have written SQL queries.**

I. Introduction

The *Ontop* framework [1] is fast becoming one of the best approaches for Ontology-based data access (based on the success of the well-known D2RQ framework). In [2], it is shown that the *Ontop* framework is efficient, and achieves good performance when compared with other well-known systems (e.g., *Jena*, *Sesame*, etc.). *Ontop* deploys query rewriting techniques with Semantic Query Optimization in an efficient manner. Thus, the queries execute faster. Moreover, redundant data is eliminated in the optimization process, which is beneficial when SQL queries are written by inexpert users [2]. In addition, another evaluation study [3] shows that the performance of the SQL queries that are generated by *Ontop* are superior compared to both other systems that translate SPARQL queries into SQL, for example, D2RQ and Virtuoso RDF Views, and other well known triple stores, for example, OWLLIM, Stardog, and Virtuoso.

The continuous development of the *Ontop* framework, in addition to providing a tool to translate R2RML mappings [4] into *Ontop* mappings and vise versa, will increase the popularity of the *Ontop* framework and make it the de facto for ontology-based data access. Henceforth, our proposed approach focuses on the automated extraction of *Ontop* mappings from a relational schema beyond the simple extraction approach that only extracts simple mapping rules and an ontology that is an exact copy of the underlying relational schema, without considering the conceptual differences between the two worlds.

Although mappings extraction is not the core focus of the *Ontop* research group, (initially assuming that the *Ontop* mappings can be written manually), the recent release added an automatic extraction option for *Ontop* mappings and ontology from an existing data source in the *OntoPro* plugin for Protegé [5]. However, this tool follows a basic approach for automating the process of extraction *Ontop* mappings. In other words, it simply extracts an ontology that is an exact copy of the relational schema and does not consider the relationships between relational entities. This complicates the process of mapping the extracted ontology with domain ontologies that have a rich structure than that of a relational schema. Some shortcomings and incompatibilities in the extracted mappings, defined in the following:

- It does not recognize a binary relation. Instead, it extracts incorrect *Ontop* mappings for representing binary relations.
- It represents the n-ary relation as n separate relations between the n relations that are composing the n-ary relation. Thus, it needs n separate SQL queries to retrieve the data the represents the n-ary join relation, which is inefficient.
- It does not recognize a recursive reference. Thus, it fails to extract *Ontop* mapping rules for representing recursive references.

The remainder of the paper is organized as follows. Section II gives a short background of the ontology-based data access and *Ontop* framework, while Section III presents the proposed approach for *Ontop* mappings and OWL ontology extractions from a relational schema. A brief overview of related works is presented in Section IV, while Section V concludes the paper.

II. Ontology Based Data Access (OBDA)

OBDA is an area of research in which the focus is to provide tools for end-users to access data sources through a high-level of conceptual view, that is presented using ontologies [6]. It assumes the availability of an ontology that acts as an intermediate layer between the end-users, and the underlying data source [7]. However, the end-users are assumed not to be aware about the underlying database schema, structure of entities, and storage details [8].

Several pproaches for direct access to relational data using ontology-based data access and SPARQL queries have been introduced in the literature (D2RQ server [9], Virtuoso RDF [10], to name a few). However, these systems have some, but serious drawbacks (e.g., lack of the semantics support and poor query performance [1]).

The *Ontop* framework [1] has been proposed to tackle the shortcomings of the existing approaches. It is an OBDA framework that supports on-the-fly SPARQL queries over RDBs through OWL and RDFS ontologies. *Quest* [1] is a reasoner and SPARQL engine that represents the core of *Ontop*. In contrast with conventional RDF triple store that transform relational data into RDF triple before querying it, the *Quest* engine accesses the relational data and reasoning over

COLLABORATECOM 2014, October 22-25, Miami, United States

DOI 10.4108/icst.collaboratecom.2014.257493

it directly and on the fly, without transforming it into OWL assertions or RDF triples. Thus, it eliminates the performance issues related to memory limitations over large data. This mode of access is called *virtual ABox* mode. The *Quest* supports access to RDBs by using mapping rules (written in a mapping language) to translate SPARQL queries into SQL queries. It also deploys the query rewriting techniques efficiently and utilizes the high performance, scalability, and the maturity of the underlying RDBMS for executing and answering SPARQL queries. The architecture of Quest is shown in Figure 1. It gets as inputs, an ontology (aka TBox) and an OBDA Model (that consists of data source definitions that contain information on how to access the defined sources, and a set of mappings that are defined using *Ontop*'s mapping language). When *Quest*

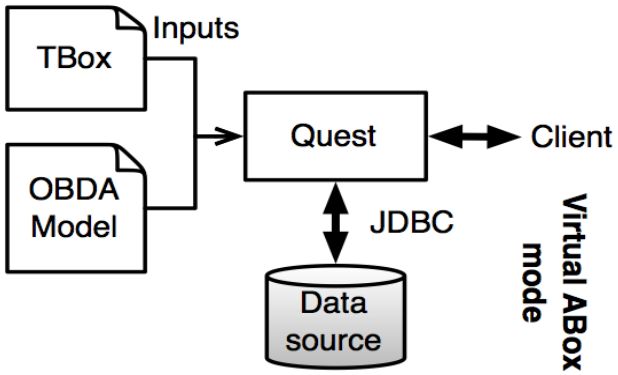

Fig. 1: Quest reasoner architecture [11].

gets a SPARQL query over the ontology, it utilizes the set of mappings over the relational database to translate it from a query over the OWL ontology into an SQL query over the relational data source. When a client sends a SPARQL query request to the *Quest* engine, the following steps are taken by the *Quest* engine. First, it rewrites the query using the ontology terms (i.e., TBox). Second, it unfolds the query into a single SQL query (using the *Ontop* mappings rules). Finally, it executes the query and sends back the result to the client. The *Quest* uses a powerful mapping language introduced in [8] for writing the *Ontop* mapping rules. A mapping rule is composed of two parts: a source part that is simply an SQL query, and a target part represents an ABox assertion template that is mapped with the source query. The template is simply a set of RDF triples written in Turtle format. The columns of the SQL query are mapped to the subject and object of the target's template triples, and the values of columns in the retrieved result are used to generate the virtual ABox assertions. When a client executes a SPARQL query over the OWL ontology, Quest uses the mapping rules to rewrite the SPARQL query over the OWL ontology into relational SQL queries over the database. After an SQL query is executed, Quest returns the results to the client as ABox assertions which are simply RDF triples that represent the virtual data instances of ontology classes and their relations.

A. Motivation Example

In this section, we illustrate *Ontop* framework through an example. Assume we have a relational database schema (*BookStore*) that is shown in Figure 2. It stores and manages data about books and the authors of books. Assume we also have an OWL ontology (*OntoBookStore*) as shown in Figure

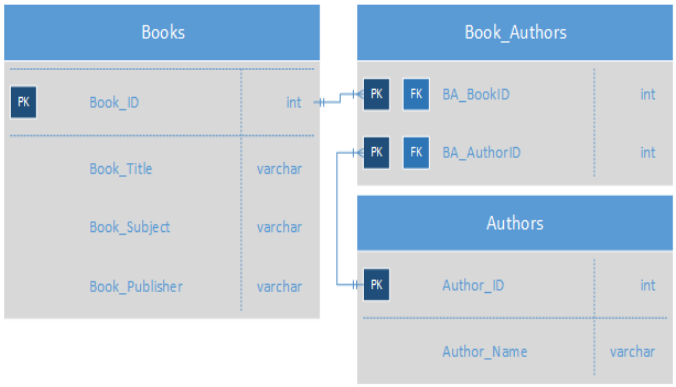

Fig. 2: BookStore Schema.

3 that is similar or equivalent to (*BookStore*), in such that both are representing the same domain of knowledge. Through using *Ontop* framework, we can query the relational schema. *Ontop*'s mapping language provides the support for defining and managing mapping rules between the ontology and its equivalent or similar relational schema for the purpose of accessing the relational data. An *Ontop*'s mapping rule is simply an axiom that relates relational entities and attributes to the correspondent concepts and properties of a similar or equivalent ontology.

We want, in this example, to declare the minimal number of *Ontop* mapping rules that are efficient and comprehensive for the purpose of querying the underlying relational data through SPARQL queries. We take a relational schema and an ontology as inputs and manually find a set of mapping rules that relates the relational schema's tables and fields to the classes and attributes of the ontology. The mechanism that we follow to declare the set of mappings is as follows:

- Find the mapping correspondences between entities/fields of the relational schema and concepts/properties of the equivalent ontology. For example, *Books* table in Figure 2 is paired with *Book* class in Figure 3.
- For each table/class pair found in above (i.e., *Books*/*Book* and *Authors*/*Author*), define a mapping rule that connects the table to its correspondent class from the ontology. The body of the mapping rule is composed of two parts: target and source. The target part is an ABox assertion template that maps the elements of the OWL ontology with their correspondent elements in the relational schema. It can also have one or more triples. The source part is simply an SQL query that represents all the database entities that are part of the mappings performed in the target part.
- Construct the subject template that represents the unique id of the virtual instance of the class that is derived from the related table. This subject template will plays as the subject for all the triples that belong to the mapping rule. The subject template is defined using the following format:
  $:< Class\_Name > /\{< Table\_Primary\_Key >\}$
  where $< Table\_Primary\_Key >$ is the primary key of the relational table that is equivalent to the ontology class.
- Add the following triples to the target part of each rule:
  - a class triple that maps the given table with the equivalent class from the ontology.
  - a data property triple for each field in the table that maps the field in the table with the correspondent data attribute

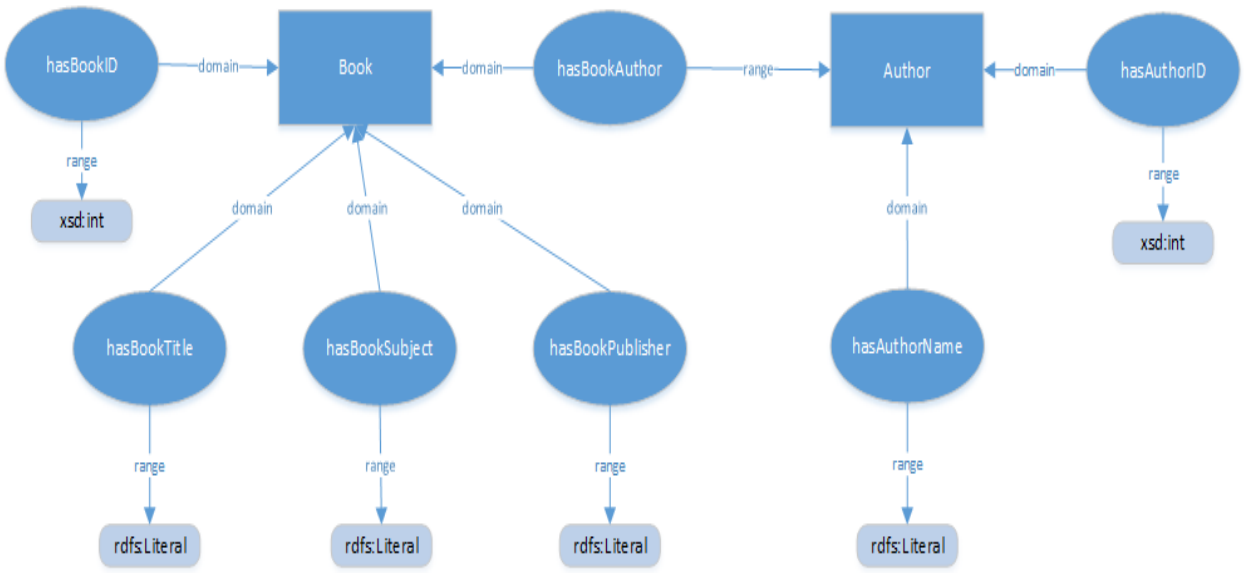

Fig. 3: $OntoBookStore$, an equivalent ontology for $BookStore$ schema.

in the ontology.

- an object property triple for each foreign key field in the table. It maps the foreign key and its referenced primary key in the other table with the correspondent relation in the ontology. The relation is simply an object property that has the class that is a correspondent to the foreign key's table as its domain and the class that is a correspondent to the primary key's table as its range.

- Write the appropriate SQL query and add it to the source part of the mapping rule.
- The remaining table ($Book\_Authors$) does not have an equivalent class. However, this table is simply a binary-join relation on $Books$ and $Authors$ tables. If we look at the ontology, we will find that the object property $hasBookAuthor$ is equivalent to the table $Book\_Authors$ as it represents the relation between $Book$ and $Author$ classes. Thus, to represent this binary relation we add a mapping rule with only one triple in the target part. The subject of the triple is a subject template for class $Book$ with the part of the composite primary key (for table $Book\_Authors$) that references table $Books$ and the object is a subject triple for class $Author$ with the second part of the composite primary key that references table $Authors$.

The extracted mapping rules are shown in Table I. Figure 4 also shows a pictorial representation for the mapping rules and their associations with the relational schema and the OWL ontology, where the black dotted arrows represent the mappings between the elements of (the relational schema and ontology) and the mapping rules, and the red dotted arrows represent the relations among mapping rules. The motivating example shows that extraction of ontology and $Ontop$ mapping rules from a relational schema is tedious and needs much time in addition to expert people. Therefore, there is a need to automate the extraction process.

III. Ontop Mapping Rules and OWL Ontology Extraction from a Relational Schema

The extraction process is composed of three modules: *Schema Metadata Extractor* module that uses a *Connection Wrapper* (implemented through JDBC API [12]) to extract the definition of the relational schema (i.e., the SQL DDL details), *Ontop Mappings Extractor* ($OMsE$) module that uses the schema metadata to extract the required $Ontop$ mapping rules to enable end users from accessing the given relational schema through $SPARQL$ queries, and *OWL Ontology Extractor* (OOE) module that depends on the metadata and $Ontop$ mapping rules extracted from the first module to generate the equivalent OWL ontology for the given relational schema. $OMsE$ module takes the description of the schema (as SQL DDL) and extracts $Ontop$ mappings rules. The proposed approach is built on top of the $Quest$ inference system. Before we show our approach for extracting $Ontop$ mappings and OWL ontology from a relational database schema, we define terms used hereafter:

A. The Metadata of a Relational Schema

- Σ: A metadata of a relational schema that represents the entities and their relationships in a particular domain. It is defined as Σ: {$\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_t$}, where $\mathcal{T}_i$ is a particular table/entity in the schema and t is the number of tables.
- $\mathcal{T}_i$: {$\mathcal{N}_i$, $\mathcal{A}_i$, $\mathcal{PK}_i$, $\mathcal{FK}_i$, $\mathcal{NK}_i$}, where $\mathcal{N}_i$ is the name of the table, $\mathcal{A}_i$ is the set of all $\mathcal{T}_i$'s attributes, $\mathcal{PK}_i$ is the set of $\mathcal{T}_i$'s primary key attributes, $\mathcal{FK}_i$ is the set of $\mathcal{T}_i$'s foreign key attributes, and $\mathcal{NK}_i$ is the set of T_i's non-key attributes.
- $\mathcal{A}_i$: {a_1:d_1, a_2:d_2, ..., a_n:d_n}, where n is the number of attributes in $\mathcal{T}_i$ and a_j:d_j is the pair of attribute j's name (a_j) and the SQL data type (d_j).

B. Extracting Ontop mapping rules from a relational schema

Assuming we have a relational schema metadata Σ (that is extracted from the $SQL\ DDL$) as an input to the $OMsE$, we apply the following rules in the $OMsE$ to extract the mapping rules $\mathcal{M}$ from Σ:

1) Independent Table (IT) rule: Let $\mathcal{T}_i$ be a table that has a primary key $\mathcal{PK}_i$. Let $\mathcal{PK}_i$ be either a single attribute key or a composite key, such that $\mathcal{PK}_i = \{\mathcal{K}_{P_i^1}, \mathcal{K}_{P_i^2}, \ldots, \mathcal{K}_{P_i^n}\}$, where $\mathcal{K}_{P_i^t}$ is the attribute t in $\mathcal{PK}_i$ and n is the total number of attributes that are composing $\mathcal{PK}_i$. If the foreign keys set

TABLE I: Mapping rules between $OntoBookStore$ ontology and $BookStore$ relational schema.

1) map-books:

TARGET: $:book/\{BOOK\_ID\}\ a\ :Book;$ **:hasBookID** $\{BOOK\_ID\};$ **:hasBookTitle** $\{BOOK\_TITLE\};$

SOURCE: **SELECT** $*$ **FROM** $books$

2) map-authors:

TARGET: $:author/\{AUTHOR\_ID\}\ a\ :Author;$ **:hasAuthorID** $\{AUTHOR\_ID\};$ **:hasAuthorName** $\{AUTHOR\_NAME\};$

SOURCE: **SELECT** $*$ **FROM** $authors$

3) map-authors-to-books:

TARGET: $:book/\{BA\_BOOKID\}\ :hasBookAuthor\ :author/\{BA\_AUTHORID\};$

SOURCE: **SELECT** $ba\_bookid,\ ba\_authorid$ **FROM** $book\_authors$

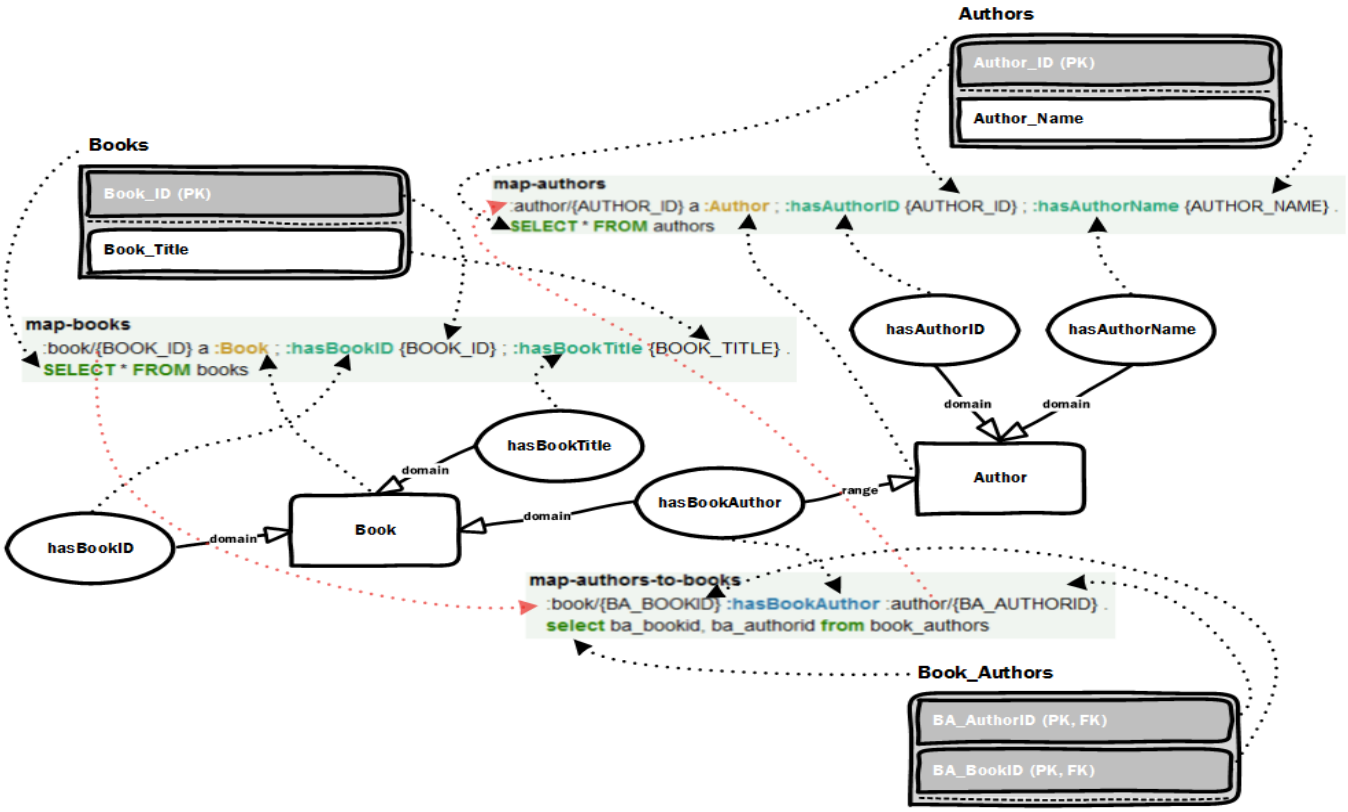

Fig. 4: A pictorial mapping between $BookStore$ schema, $OntoBookStore$ ontology, and the mapping rules.

$\mathcal{FK}_i$ of $\mathcal{T}_i$ is $null$, we say that $\mathcal{T}_i$ is an IT. We can represent the target of an IT rule by a class triple and a data property triple for each attribute that belongs to $\mathcal{T}_i$. The source query part of $Ontop$ mapping rule for IT is simply an SQL select statement on the table.

Example 1. *IT rule. Consider a table dept with a primary key deptno of type integer and the non-key attributes dname and loc of type string. Because dept has a primary key and has no any foreign key constraints, IT rule applies on dept. The $Ontop$ mapping rule for dept is shown in Table II.*

2) Dependent Table (DT) rule: Let $\mathcal{T}_i$ be a table that has a primary key $\mathcal{PK}_i$. Let $\mathcal{PK}_i$ be either a single attribute key or a composite key, such that $\mathcal{PK}_i = \{\mathcal{K}_{P_i^1}, \mathcal{K}_{P_i^2}, \ldots, \mathcal{K}_{P_i^n}\}$, where $\mathcal{K}_{P_i^t}$ is the attribute t in $\mathcal{PK}_i$ and n is the total number of attributes that are composing $\mathcal{PK}_i$. Let also the foreign keys set $\mathcal{FK}_i$ of this table to be $\{\mathcal{K}_{F_i^1}, \mathcal{K}_{F_i^2}, \ldots, \mathcal{K}_{F_i^m}\}$, where $\mathcal{K}_{F_i^t}$ is the foreign key t in $\mathcal{FK}_i$ and m is the number of foreign keys in $\mathcal{T}_i$. Let $\mathcal{K}_{F_i^x} = \{\mathcal{K}_{f^1}, \mathcal{K}_{f^2}, \ldots, \mathcal{K}_{f^v}\}$, where $\mathcal{K}_{f^t}$ is the attribute t in $\mathcal{K}_{F_i^x}$ and v is the number of attributes in foreign key $\mathcal{K}_{F_i^x}$. If $\mathcal{FK}_i$ is not null, we say that $\mathcal{T}_i$ is an DT. There are two steps for extracting $Ontop$ mapping rules from $\mathcal{T}_i$ that satisfies the DT rule. The first step is to extract an $Ontop$ mapping rule as we do in IT rule. The second step is to extract an $Ontop$ mapping rule for each foreign key in $\mathcal{T}_i$. We skip the first step because it is the same as in the previous rule. The details of the second step follow. For each foreign key $\mathcal{K}_{F_i^t}$ in $\mathcal{T}_i$ that references a table $\mathcal{T}_j$, we extract

TABLE II: Mapping rule for table *dept*.

TARGET:	**:dept**/$deptno = \{deptno\}$ a **:dept**; **:dept#deptno** $\{deptno\}$; **:dept#dname** $\{dname\}$; **:dept#loc** $\{loc\}$;
SOURCE:	**SELECT** $*$ **FROM** $dept$

an *Ontop* mapping rule with $target$ and $source$ parts. The $target$ part has two object property triple templates. The first triple represents the many-to-one relation from $\mathcal{T}_i$ to $\mathcal{T}_j$. The other triple is simply the inverse of first one. In other words, it represents the one-to-many relation from $\mathcal{T}_j$ to $\mathcal{T}_i$. Thus, the predicate of the second triple is the inverse of the predicate of the first one.

Example 2. *DT rule. Consider tables dept (from the previous example) and emp. emp has the primary key empno and the non-key attributes empname, job, and sal. It also has the department number field (deptno) that represents a foreign key that references deptno (which is the primary key of dept). In addition, it has the manager attribute (mgr) that represents a foreign key reference to itself. Because emp has a primary key and some foreign key constraints, DT rule applies on emp. The first step is to extract an Ontop mapping rule for emp the same way as in the case of IT. The outcome of this step is the Ontop mapping rule shown in Table III. Next, an Ontop mapping rule is extracted for each foreign key in emp. emp has two foreign keys fields: mgr and deptno. The first one is a recursive reference to emp. Thus, RT rule is applied in this case to extract the Ontop mapping rule that represents the self-reference. We leave the details of extracting the RT rule to the next example. The Ontop mapping rule for the foreign key deptno is then extracted, as shown in Table IV.*

3) Recursive Table (RT) rule: RT. Let $\mathcal{T}_i$ be a table that has a primary key $\mathcal{PK}_i$. Let $\mathcal{PK}_i$ be either a single attribute key or a composite key, such that $\mathcal{PK}_i = \{\mathcal{K}_{P_i^1}, \mathcal{K}_{P_i^2}, \ldots, \mathcal{K}_{P_i^n}\}$, where $\mathcal{K}_{P_i^t}$ is the attribute t in $\mathcal{PK}_i$ and n is the total number of attributes that are composing $\mathcal{PK}_i$. Let also $\mathcal{K}_F$ to be a foreign key in table $\mathcal{T}_i$, such that it references the same table (i.e., recursive). Let $\mathcal{K}_F = \{\mathcal{K}_{f^1}, \mathcal{K}_{f^2}, \ldots, \mathcal{K}_{f^v}\}$, where $\mathcal{K}_{f^t}$ is the attribute t in $\mathcal{K}_F$ and v is the number of attributes in $\mathcal{K}_F$. If there is at least one $\mathcal{K}_F$ that is a self reference on the table $\mathcal{T}_i$, we say that this table is an RT. For representing a foreign key $\mathcal{K}_F$ that is a self reference in $\mathcal{T}_i$, we extract an *Ontop* mapping rule with a $target$ part that has one object property triple that represents the recursive reference $\mathcal{K}_F$ on $\mathcal{T}_i$.

Example 3. *RT rule. We go back to the previous example and consider table emp. The field mgr in emp is a foreign key that references emp itself. Thus, we apply the RT rule. The outcome is the Ontop mapping rule shown in Table V.*

4) Binary Join Table without non-key Attributes (BJ) rule: BJ. Let $\mathcal{T}_i$, $\mathcal{T}_j$, and $\mathcal{T}_k$ are tables with primary keys $\mathcal{PK}_i$, $\mathcal{PK}_j$, and $\mathcal{PK}_k$, respectively. If (1) the primary key of $\mathcal{T}_i$ is composed of two parts ($\mathcal{PF}_j$ and $\mathcal{PF}_k$), where the former is both the first part of $\mathcal{PK}_i$ and the foreign key that references $\mathcal{PK}_j$ of $\mathcal{T}_j$, and the latter is both the second part of $\mathcal{PK}_i$ and the foreign key that references $\mathcal{PK}_k$ of $\mathcal{T}_k$, and (2) all $\mathcal{T}_i$'s attributes are in the primary key, we say that $\mathcal{T}_i$ is a binary join table. We can represent this kind of binary relation in ontology without adding an equivalent class entity for table $\mathcal{T}_i$. Instead, we add one object property for each one-to-many relationship. The first object property has the extracted concept of $\mathcal{T}_j$ as its domain and the extracted concept of $\mathcal{T}_k$ as its range. The second object property has the extracted concept of $\mathcal{T}_k$ as its domain and the extracted concept of $\mathcal{T}_j$ as its range. In other words, each object property is simply the inverse of the other. Thus, we represent the target of BJ rule by using two object property triples. The first triple represents the one-to-many sub-relation from $\mathcal{T}_j$ to $\mathcal{T}_k$, and the second one (which is simply the reverse of the first one) represents the one-to-many sub-relation from $\mathcal{T}_k$ to $\mathcal{T}_j$, as we mentioned above.

Example 4. *BJ rule. Assume we have the tables employee, project and assignments. employee has the primary key empid and the non-key attributes fname and lname, project has the primary key projid and the non-key attribute projname, and assignments has a composite primary key of emp_id that references empid in employee and proj_id that references projid in project and it has no non-key fields. Table assignments represents a binary join table that connects both employee and project. Both employee and project are independent tables. Thus, their Ontop mapping rules are extracted according to the IT rule as shown in Table VI. Table assignments has a composed primary key of emp_id and proj_id, such that the former foreign key references table employee and the latter foreign key references table project. In other words, the primary key is only composed of these two foreign keys, such that each foreign key references another table. In addition, table assignments does not have any fields other than the ones that are composing its primary key and both foreign keys that are representing the binary join. Thus, the BJ rule applies here on assignments. The resulted Ontop mapping rule is shown in Table VII.*

5) n-ary Join Table (NJ) rule: NJ. Let $\mathcal{T}_i$ be a table that has a primary key $\mathcal{PK}_i$. Let $\mathcal{PK}_i$ be either a single attribute key or a composite key, such that $\mathcal{PK}_i = \{\mathcal{K}_{P_i^1}, \mathcal{K}_{P_i^2}, \ldots, \mathcal{K}_{P_i^n}\}$, where $\mathcal{K}_{P_i^t}$ is the attribute t in $\mathcal{PK}_i$ and n is the total number of attributes that are composing $\mathcal{PK}_i$. Let also the foreign keys set $\mathcal{FK}_i$ of this table to be $\{\mathcal{K}_{F_i^1}, \mathcal{K}_{F_i^2}, \ldots, \mathcal{K}_{F_i^m}\}$, where $\mathcal{K}_{F_i^t}$ is the foreign key t in $\mathcal{FK}_i$ and m is the number of foreign keys in $\mathcal{T}_i$. Let $\mathcal{K}_{F_i^x} = \{\mathcal{K}_{f^1}, \mathcal{K}_{f^2}, \ldots, \mathcal{K}_{f^v}\}$, where $\mathcal{K}_{f^t}$ is the attribute t in $\mathcal{K}_{F_i^x}$ and v is the number of attributes in foreign key $\mathcal{K}_{F_i^x}$. Let $\mathcal{NJK}_i$ be $\{\mathcal{K}_{J_i^1}, \mathcal{K}_{J_i^2}, \ldots, \mathcal{K}_{J_i^m}\}$, where $\mathcal{K}_{J_i^t}$ is the foreign key t in $\mathcal{NJK}_i$ and m is the number of foreign keys in $\mathcal{T}_i$ that are forming the m-ary join of tables $\mathcal{T}_1$, $\mathcal{T}_2$, ..., $\mathcal{T}_m$ in $\mathcal{T}_i$, such that $\mathcal{K}_{J_i^1}$ is referring to table $\mathcal{T}_1$, $\mathcal{K}_{J_i^2}$ is referring to table $\mathcal{T}_2$, ..., and $\mathcal{K}_{J_i^m}$ is referring to table $\mathcal{T}_m$. Let $\mathcal{K}_{J_i^x} = \{\mathcal{K}_{f_x^1}, \ldots, \mathcal{K}_{f_x^{n_x}}\}$, where $\mathcal{K}_{f_x^t}$ is the attribute t in $\mathcal{K}_{J_i^x}$ that is part of the m-ary join and n_x is the number of attributes in foreign key $\mathcal{K}_{J_i^x}$. Let $\mathcal{K}_{J_i^x}$ is referring to the primary key $\mathcal{PK}_x$ of table $\mathcal{T}_x$, such that the attributes $\mathcal{K}_{f_x^1}$, ..., $\mathcal{K}_{f_x^{n_x}}$ are referring to the attributes $\mathcal{K}_{P_x^1}$, ..., $\mathcal{K}_{P_x^{n_x}}$, respectively. If n is greater than 2, we say that $\mathcal{T}_i$ is an NJ table. There are three steps for extracting *Ontop* mapping rules from $\mathcal{T}_i$ that satisfies the NJ rule. The first step is to extract an *Ontop* mapping rule as we do in the IT rule. The second step is to extract an *Ontop* mapping rule for each foreign key in $\mathcal{T}_i$ that does not belong to $\mathcal{NJK}_i$. The last step is to extract an *Ontop* mapping rule

TABLE III: Mapping rule for representing table emp.

TARGET:	$:emp/empno = \{empno\}$ a $:emp$; **emp#empno** $\{empno\}$; **:emp#empname** $\{empname\}$; **:emp#sal** $\{sal\}$;
SOURCE:	**SELECT** $*$ **FROM** emp

TABLE IV: Mapping rule for representing the reference $deptno$ in table emp.

TARGET:	$:emp/empno = \{emp\_empno\}$ **emp#hasDept** $:dept/deptno = \{dept\_deptno\}$. $:dept/deptno = \{dept\_deptno\}$ **emp#hasEmp** $:emp/empno = \{emp\_empno\}$.
SOURCE:	**SELECT** $emp.empno$ **AS** emp_empno, $dept.deptno$ **AS** $dept\_deptno$ **FROM** emp, $dept$ **WHERE** $emp.deptno = dept.deptno$

TABLE V: Mapping rule for representing the self reference mgr in table emp.

TARGET:	$:emp/empno = \{emp\_child\_empno\}$ **emp#mgr** $:emp/empno = \{emp\_parent\_empno\}$.
SOURCE:	**SELECT** $emp\_child.empno$ **AS** emp_child_empno, $emp\_parent.empno$ **AS** emp_parent_empno **FROM** $emp\ emp\_child$, $emp\ emp\_parent$ **WHERE** $emp\_child.mgr = emp\_parent.empno$

TABLE VI: Mapping rules for tables $employee$ and $project$.

TARGET:	**:employee**$/empid = \{empid\}$ a **:employee**; **:employee#empid** $\{empid\}$; **:employee#fname** $\{fname\}$; **:employee#lname** $\{lname\}$;
SOURCE:	**SELECT** $*$ **FROM** $employee$
TARGET:	**:project**$/projid = \{projid\}$ a **:employee**; **:project#projid** $\{projid\}$; **:project#projname** $\{projname\}$;
SOURCE:	**SELECT** $*$ **FROM** $project$

from the foreign keys in $\mathcal{NJK}_i$ which are representing the n-ary join relation in $\mathcal{T}_i$. We skip the first two steps because they are the same as in IT and DT rules. The details of the last step follow. We extract an $Ontop$ mapping rule to represent the n-ary join relation in $\mathcal{T}_i$. The $target$ part of this rule has a number of object property triples that is equal to the number of joined tables (n_m). However, because both the subject and predicate for all triples are the same, we include the subject and predicate in the first triple and omit them from the rest of triples. We do this by separating objects by a comma as we mentioned before using $Turtle$ format. The predicate $hasNaryJoin$ has the domain : $\mathcal{T}_i$ and the ranges : $\mathcal{T}_1$, : $\mathcal{T}_2$, ..., and : $\mathcal{T}_m$. OWL and $SPARQL$ cannot represent n-ary relations. To overcome this issue, we represent the n-ary relation by only one predicate that is named $hasNaryJoin$. It maintains the n-ary relation tightly-coupled by having all joined tables as its ranges.

Example 5. *NJ rule. Assume we have the tables employee, component, product and assembly. employee has the primary key empid and the non-key attribute empname, component has the primary key compid and the non-key attributes comptype and compname, and product has the primary key prodid and the non-key attributes prodtype and prodname. assembly has a composite primary key (empid, compid, prodid) that references employee, component, and product tables, respectively, and the non-key attribute description. employee, component, and product are independent tables. Thus, their Ontop mapping rules are extracted according to the IT rule as shown in Table VI. assembly represents an NJ that connects the three tables employee, component, and prdouct. Thus, the NJ rule applies. First, we extract a mapping rule (according to the IT rule format) to represent assembly. Second, an Ontop mapping rule that represents this 3-ary relation is extracted. These two rules are shown in Table IX.*

C. Extracting OWL Ontology from Ontop Mapping Rules

The process of extracting an OWL ontology $\mathcal{W}$ (that is equivalent to the relational schema) from existing $Ontop$ mappings becomes straight forward. It scans the $Ontop$ mappings in $\mathcal{M}$. For each $Ontop$ mapping rule, it checks the target templates; if the type of the template is a $class\ triple$, a new class is added to $\mathcal{W}$; if its type is an $object\ property\ triple$, a new object property is added to $\mathcal{W}$; and if it is a $data\ property\ triple$, a new data property is added to $\mathcal{W}$. In addition, the domain and range of each extracted object or data property are also extracted and added to $\mathcal{W}$. Thus, at the end of this process we will have a complete OWL ontology with classes, properties, and relations.

D. Implementation and Experiments

We have implemented a prototype for the proposed approach in Java. The end-users can access any remote relational data through a JDBC connection. After establishing the connection, end-users can extract the $Ontop$ mapping rules and OWL ontology from the underlying data source, pose SPARQL queries over the extracted ontology (to access the relational data) and get the results back. In addition, end-users can alter both the extracted rules and the ontology according to their needs.

We have evaluated the proposed approach using different freely available relational databases. The evaluation process is composed of two steps: In the first step, the ontology and mapping rules are extracted. In the second step, SPARQL queries (over the extracted ontologies) are used to access the relational data sources. Evaluation results have showed that all the data instances in the underlying data sources can be accessed indirectly (only using SPARQL queries) in the same efficient way as in the SQL queries.

For the lack of space, we give only one example on the

TABLE VII: Mapping rule for table *assignments*.

TARGET:	**:project**/$proj\_id = \{proj\_id\}$ **:project#hasEmployee** **:employee**/$emp\_id = \{emp\_id\}$. **:employee**/$emp\_id = \{emp\_id\}$ **:employee#hasProject** **:project**/$proj\_id = \{proj\_id\}$.
SOURCE:	**SELECT** $*$ **FROM** $assignments$

TABLE VIII: Mapping rules for tables *employee*, *component* and *product*.

TARGET:	**:employee**/$empid = \{empid\}$ a **:employee**; **:employee#empid** $\{empid\}$; **:employee#empname** $\{empname\}$;
SOURCE:	**SELECT** $*$ **FROM** $employee$
TARGET:	**:component**/$compid = \{compid\}$ a **:component**; **:component#compid** $\{compid\}$; **:component#comptype** $\{comptype\}$; **:component#compname** $\{compname\}$;
SOURCE:	**SELECT** $*$ **FROM** $component$
TARGET:	**:product**/$prodid = \{prodid\}$ a **:product**; **:product#prodid** $\{prodid\}$; **:product#prodtype** $\{prodtype\}$; **:product#prodname** $\{prodname\}$;
SOURCE:	**SELECT** $*$ **FROM** $product$

TABLE IX: Mapping rule for table *assembly*.

TARGET:	**:assembly**/$empid = \{empid\}; compid = \{compid\}; prodid = \{prodid\}$ a **:assembly**; **:assembly#empid** $\{empid\}$; **:assembly#compid** $\{compid\}$; **:assembly#prodid** $\{prodid\}$; **:assembly#description** $\{description\}$;
SOURCE:	**SELECT** $*$ **FROM** $assembly$
TARGET:	**:assembly**/$empid = \{assembly\_empid\}; compid = \{assembly\_compid\}; prodid = \{assembly\_prodid\}$ **:assembly#hasNaryJoin** **:employee#empid** $\{employee\_empid\}$, **:component#compid** $\{component\_compid\}$, **:product#prodid** $\{product\_prodid\}$.
SOURCE:	**SELECT** $assembly.empid$ **AS** $assembly\_empid$, $assembly.compid$ **AS** $assembly\_compid$, $assembly.prodid$ **AS** $assembly\_prodid$, $employee.empid$ **AS** $employee\_empid$, $component.compid$ **AS** $component\_compid$, $product.prodid$ **AS** $product\_prodid$ **FROM** $assembly$, $employee$, $component$, $product$ **WHERE** $assembly.empid = employee.empid\ AND\ assembly.compid = component.compid\ AND\ assembly.prodid = product.prodid$

evaluation process. Consider the tables *dept* and *emp* that have been used in the first three examples. An instance of these two tables is shown in Table X.

TABLE X: Data instance for *dept* and *emp* tables.

deptno	dname	loc
10	Accounting	New York
20	Research	Dallas
30	Sales	Chicago
40	Operations	Boston

(a) dept

empno	empname	job	mgr	sal	deptno
7839	King	President	Null	5000	10
7566	Jones	Manager	7839	2975	20
7902	Ford	Analyst	7566	3000	20
7369	Smith	Clerk	7902	800	20
7782	Clark	Manager	7839	2450	10
7698	Blake	Manager	7839	2850	30
7521	Ward	Salesman	7698	1250	30

(b) emp

```
PREFIX : <http://experiments.org/

:dept rdf:type owl:Class .
:emp rdf:type owl:Class .

:dept#deptno rdf:type owl:DatatypeProperty .
:dept#dname rdf:type owl:DatatypeProperty .
:dept#loc rdf:type owl:DatatypeProperty .
:emp#deptno rdf:type owl:DatatypeProperty .
:emp#empno rdf:type owl:DatatypeProperty .
:emp#empname rdf:type owl:DatatypeProperty .
:emp#job rdf:type owl:DatatypeProperty .
:emp#mgr rdf:type owl:DatatypeProperty .
:emp#sal rdf:type owl:DatatypeProperty .

:emp#hasDEPT rdf:type owl:ObjectProperty .
:emp#hasEMP rdf:type owl:ObjectProperty .
```

Listing 1: Ontology extracted from *dept* and *emp* tables.

```
PREFIX emp: <http://experiments.org/emp#>
PREFIX dept: <http://experiments.org/dept#>

SELECT ?e ?eName ?eJob ?eMgrName ?dDeptNo ?dDName
WHERE {
  ?e a :emp .
  ?e emp:empno ?eEmpNo .
  ?e emp:empname ?eName .
  ?e emp:job ?eJob .
  OPTIONAL {
            ?e emp:mgr ?eMgr .
            ?eMgr a :emp .
            ?eMgr emp:empname ?eMgrName .
  }
  ?e emp:deptno ?dDeptNo .
  ?eDeptNo a :dept .
  ?eDeptNo dept:dname ?dDName .
}
```

Listing 2: SPARQL query to access the database instance shown in Table X.

TABLE XI: The SPARQL query's result in Listing 2.

e	eName	eJob	eMgrName	dDeptNo	dDName
:emp/empno=7782	Clark	Manager	King	:dept/deptno=10	Accounting
:emp/empno=7839	King	President	null	:dept/deptno=10	Accounting
:emp/empno=7369	Smith	Clerk	Ford	:dept/deptno=20	Research
:emp/empno=7566	Jones	Manager	King	:dept/deptno=20	Research
:emp/empno=7902	Ford	Analyst	Jones	:dept/deptno=20	Research
:emp/empno=7521	Ward	Salesman	Blake	:dept/deptno=30	Sales
:emp/empno=7698	Blake	Manager	King	:dept/deptno=30	Sales

IV. Related Work

There is a plenty of works in the literature about extracting an ontology from an existing relational schema. Most

approaches share common rules for the extracting process. The most common rules that are repeated in the different approaches (for examples, in [13], [14], [15], [16], [17], [18], [19], and [20]) are as follows: default (or basic approach), binary relationship, n-ary relationship, hierarchy, and fragmentation rules (See [7] for a comprehensive survey). A default rule is simply a basic and naive approach for extracting equivalent OWL concepts and properties from relational tables and their attributes. The basic approach is to map a relation to an OWL class, a non-foreign key attribute to an OWL data property, a foreign key attribute to an OWL object property, and a relation row to an individual of an OWL class. The first two rules of our proposed approach (i.e., independent and dependent rules) are similar to the basic approach. In addition, all the previously mentioned approaches have some kind similar rules for the default and binary rules. [15], [17], and [18] do not have a rule for representing the n-ary relationship. The hierarchy (or sub-class) rule is not considered by [14] ,[16], and [18], whereas the fragmentation rule is not considered in all mentioned approaches, except in [19] . Hierarchy and fragmentation rules are very similar and one cannot distinguish between them unless he/she knows the intend of the schema designers or by mining the relational instance. The latter usually requires mining the data instance if enough data are available or using heuristic approaches, etc. Thus, most existing approaches misrepresent the hierarchy and fragmentation rules. In other words, they apply the hierarchy rule when in fact they should apply the fragmentation rule, and vice versa. In our approach, we decide to leave the discussion of the hierarchy and fragmentation rules for a future research as they primary depend on the availably of a relational instance with sufficient data. In addition, we leave the discussion of extracting equivalent OWL axioms for the relational constraints for future research.

V. Conclusion and Future Work

Manually extracting $Ontop$ mapping rules and OWL ontology from a relational schema is a very complex and time consuming process. In this paper, we propose an approach for extracting $Ontop$ mappings and OWL ontology from a relational schema. The proposed approach defines an $Ontop$ mapping rule's template for each type of relational entities. It covers the extraction rules for independent, dependent, recursive, binary join, and n-ary join tables. It also defines algorithms for automatically extracting $Ontop$ mappings for a relational entity according the defined templates. OWL ontology is then easily extracted from those mappings rules This work can be extended to cover other type of relations, such as fragment entities, sub-entities (inheritance), enumerated attributes, and others. In addition, we will develop approaches for extracting equivalent ontological elements for database constraints, for example, check, enum, etc.

References

[1] M. Rodriguez-Muro, J. Hardi, and D. Calvanese, "Quest: Efficient sparql-to-sql for rdf and owl," in *Demos of the 12th Int. Semantic Web Conf. (ISWC 2012)*, 2012.

[2] M. Rodrıguez-Muro, R. Kontchakov, and M. Zakharyaschev, "Ontop at work," in *Proc. of the 10th OWL: Experiences and Directions Workshop (OWLED 2013).*, 2013.

[3] M. Rodriguez-Muro, M. Rezk, J. Hardi, M. Slusnys, T. Bagosi, and D. Calvanese, "Evaluating sparql-to-sql translation in ontop," 2013.

[4] R. Cyganiak, S. Sundara, and S. Das, "R2RML: RDB to RDF mapping language," Tech. Rep., Sep.

[5] M. Rodriguez-Muro, L. Lubyte, and D. Calvanese, "Realizing ontology based data access: A plug-in for protégé," in *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*. IEEE, 2008, pp. 286–289.

[6] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Ontology-based database access." in *SEBD*, 2007, pp. 324–331.

[7] D.-E. Spanos, P. Stavrou, and N. Mitrou, "Bringing relational databases into the semantic web: A survey," *Semantic Web*, vol. 3, no. 2, pp. 169–209, 2012.

[8] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, "Linking data to ontologies," in *Journal on data semantics X*. Springer, 2008, pp. 133–173.

[9] C. Bizer and A. Seaborne, "D2rq-treating non-rdf databases as virtual rdf graphs," in *Proceedings of the 3rd international semantic web conference (ISWC2004)*, vol. 2004, 2004.

[10] O. Erling and I. Mikhailov, "Virtuoso: Rdf support in a native rdbms," in *Semantic Web Information Management*. Springer, 2010, pp. 501–519.

[11] M. Rodriguez-Muro and D. Calvanese, "Quest, an owl 2 ql reasoner for ontology-based data access," in *Proc. of the 9th Int. Workshop on OWL: Experiences and Directions (OWLED 2012)*, ser. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, vol. 849, 2012.

[12] M. Fisher, J. Ellis, and J. C. Bruce, *JDBC API Tutorial and Reference*, 3rd ed. Pearson Education, 2003.

[13] I. Astrova, "Rules for mapping sql relational databases to owl ontologies," in *Metadata and Semantics*. Springer, 2009, pp. 415–424.

[14] A. Buccella, M. R. Penabad, F. J. Rodriguez, A. Farina, and A. Cechich, "From relational databases to owl ontologies," in *Proceedings of the 6th National Russian Research Conference*, 2004.

[15] K. M. Albarrak and E. H. Sibley, "Translating relational & object-relational database models into owl models," in *Information Reuse & Integration, 2009. IRI'09. IEEE International Conference on*. IEEE, 2009, pp. 336–341.

[16] L. Lubyte and S. Tessaris, "Automatic extraction of ontologies wrapping relational data sources," in *Database and Expert Systems Applications*. Springer, 2009, pp. 128–142.

[17] F. Cerbah, "Mining the content of relational databases to learn ontologies with deeper taxonomies," in *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, vol. 1. IEEE, 2008, pp. 553–557.

[18] C. Curino, G. Orsi, E. Panigati, and L. Tanca, "Accessing and documenting relational databases through owl ontologies," in *Flexible Query Answering Systems*. Springer, 2009, pp. 431–442.

[19] N. Alalwan, H. Zedan, and F. Siewe, "Generating owl ontology for database integration," in *Advances in Semantic Processing, 2009. SEMAPRO'09. 3rd Int. Conf. on*. IEEE, 2009, pp. 22–31.

[20] S. H. Tirmizi, J. Sequeda, and D. Miranker, "Translating sql applications to the semantic web," in *Database and Expert Systems Applications*. Springer, 2008, pp. 450–464.