Integrating Relational Databases with Semantic Web Ontologies: Reasoning and Query Answering using Views

Linhua Zhou  
School of Computer Science, Zhejiang University, CN  
uajunsir@zju.edu.cn

Yuxin Mao  
School of Computer Science, Zhejiang University, CN  
zlhyjy@zju.edu.cn

Huajun Chen  
School of Computer Science, Zhejiang University, CN  
maoyx@zju.edu.cn

Abstract

Recently, there has been a growing need for integrating legacy relational databases with semantic web ontologies, however, experience in building such applications has revealed a gap between semantic web languages and relational data model. We present a formal mapping system to bridge the gap and study the problem of reasoning and query answering using view underlying the mapping system. Particularly, we consider a special mapping called “constraint mapping” between database integrity constraints and OWL axioms. We then propose an approach to incorporating these OWL axioms and description logic reasoning into query rewriting using views to enable the algorithm to answer more types of queries. The approach has been used to develop a pilot data integration application for neuroscience community.

Motivation

Semantic Web provides enhanced capabilities for data integration by making data semantics explicit through machine-understandable ontologies. It is based upon Resource Description Framework (RDF) and Web Ontology Language(OWL), the semantic web languages proposed by W3C Consortium. Relational databases are commonly used to build the backbone of the modern information systems. With the popularity of semantic web, there is an ongoing increase of interest and endeavor in transforming and integrating legacy relational databases into semantic web. For example, in life science community, researchers are trying to transform legacy neuroscience relational databases into RDF/OWL format to enable integrative query answering and reasoning (Hugo Y.K. 2007). However, experience in building such applications has revealed a gap between semantic web languages and relational data model; this gap becomes more noticeable if we consider expressive languages such as OWL-DL or OWL-Full.

To understand the problems in specific, consider the following example. For a hospital application, one might require each patient to have at least one record of disease. In a relational database, this could be captured by a foreign key constraint stating that, for each tuple in patient table, an associated tuple in disease table must exist. Such a constraint would be used to ensure data integrity during database updates. In an OWL knowledge base, the same requirement could be captured by an owl:someValuesFrom axiom asserting that for all patient, they have at least one kind of disease. However, the key difference is that the OWL axiom would not be used to check data integrity. One can add a new patient into the knowledge base without specifying a kind of disease for the patient and the knowledge base system would not consider it as an error, rather, the patient would be simply inferred to have some unknown disease. As a matter of fact, OWL knowledge base consider the axiom as a kind of TBox knowledge for reasoning.

More issues arise while mapping legacy relational databases to OWL ontologies. Most available approaches merely consider the mappings between tables/columns and classes/properties, with no consideration of database integrity constraints which are actually important information and knowledge. Problem comes up while one tries to do query answering and reasoning. For the same example, if new patients with no diseases specified are added into the OWL knowledge base (OWL-KB) and one issues a query of all patients having at least one kind of disease, the OWL-KB would discard all of those newly added patients. As can be seen in our approach, although the constraints, which are mapped to different OWL axioms, are not used by OWL-KB to check data integrity, they can absolutely be used to enhance query answering and rewriting.

This paper presents a view-based mapping mechanism to bridge the gap between OWL ontologies and schemata of relational data model in a seamless manner, and study the problem of answering queries using views underlying this mapping system. In particular, we consider a special mapping called “constraint mapping” between database integrity constraints and OWL axioms. We then propose an approach to incorporating these OWL axioms and description logic reasoning into query rewriting using views to enable the algorithm to answer more types of queries. The approach has been used to develop a pilot data integration application for neuroscience community. For formal discussion, we focus attention on the SHIQ (Ian Horrocks 2003) description logic language, which is the underlying formalism for semantic web ontology languages.

Preliminaries

**SHIQ Description Logic**

**Syntax.** Given $R$ as a finite set of transitive and inclusion axioms with normal role names $N_R$, a $\text{SHIQ-role}$ is
views aims to compute a rewriting of the query in terms of ping specification and source data. Query rewriting using the views and then evaluates the rewriting directly against the source data. The queries in actual applications are usually conjunctive queries. A view is a named query. In this paper, we consider conjunctive query for \(SHIQ\).

**Definition 1 (SHIQ Conjunctive Queries)** Let \(N_V\) be a countably infinite set of variables disjoint from \(N_C, N_R, N_1\) and \(N_J\). An atom is an expression \(A(v)\) (concept atom) or \(R(v, v')\) (role atom), where \(A\) is a concept name, \(R\) is a role, and \(v, v'\) \(\in V\). A conjunctive query \(q\) is a non-empty set of atoms in the form:

\[
q(X) : e_1(X_1, Y_1), \ldots, e_n(X_n, Y_n),
\]

\(q\) belongs to a new alphabet \(Q\) of queries that is disjoint from \(N_C, N_R, N_1\) and \(N_V\).

\(e_1(X_1, Y_1), \ldots, e_n(X_n, Y_n)\) are called the subgoals in the query body, and are either a concept atom in the form of \(A(v)\) or a role atom in the form of \(R(v, v')\).

\(X_1, \ldots, X_n\) are either variables from \(N_V\) or constants from \(N_1\), and \(X \subseteq X_1 \cup \ldots \cup X_n\) are called distinguished variables. \(Y_1, \ldots, Y_n\) are called existential variables.

As like in conventional literatures (Halevy 2001), the distinguished variables must have a value binding while doing query evaluation, instead, existential variables do not have to bind to any instance or data values.

**SHIQ-RDM Mapping System**

**SHIQ-RDM View**

The basic mapping system consists of a mapping formalism between a target \(SHIQ\) ontology and a set of source relational schemata that employs the form of LaV (i.e., each relational predicate is defined as a view over the ontology).

**Definition 2 (SHIQ-RDM View)** A \(SHIQ-RDM\) view is in this form: \(\mathcal{R}(\vec{X}) : \neg \exists \vec{Y} \cdot \mathcal{Q}(\vec{X}, \vec{Y})\) where:

- \(\mathcal{R}(\vec{X})\) is called the head of the view, and \(\mathcal{R}\) is a relational predicate.
- \(\exists \vec{Y} \cdot \mathcal{Q}(\vec{X}, \vec{Y})\) is called the body of the view, and is a \(SHIQ\) conjunctive query over the ontology.
- The \(\vec{X} \subseteq \vec{X}'\) is the set of distinguished variables, \(\vec{Y}'\) is the set of existential variables.

Figure 1 is a \(SHIQ-RDM\) mapping scenario consisting of the target FOAF ontology and two source relational data sources “\(\text{zju}\)” and “\(\text{ycmi}\)”. Example 1 illustrates the corresponding \(SHIQ-RDM\) views. Note we use RDF triple syntax to describe concept atom and role atom.

**Example 1 (SHIQ-RDM View Examples)**

\begin{verbatim}
<table>
<thead>
<tr>
<th>zu Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1: zjusem{?en, ?em, ?eh} :-</td>
</tr>
<tr>
<td>(?y1, rdf:type, foaf:Person),</td>
</tr>
<tr>
<td>(?y1, foaf:name, ?en),</td>
</tr>
<tr>
<td>(?y1, foaf:muco, ?em),</td>
</tr>
<tr>
<td>(?y1, foaf:shoolHomepage, ?eh)</td>
</tr>
<tr>
<td>v2: zjusem_account(?en, ?em) :-</td>
</tr>
<tr>
<td>(?y1, rdf:type, foaf:Person),</td>
</tr>
<tr>
<td>(?y1, foaf:name, ?en),</td>
</tr>
<tr>
<td>(?y1, foaf:holdsAccount, ?y2),</td>
</tr>
<tr>
<td>(?y2, rdf:type, foaf:OnlineChatAccount),</td>
</tr>
</tbody>
</table>
\end{verbatim}

\(^{1}\)The FOAF project: http://www.foaf-project.org/.
Figure 1: $SHIQ$-$RDM$ Mapping Scenario.

Table 2: Constraint Mapping. $T_1$, $T_2$ are relational tables, and are mapped to class $A, C$ respectively.

<table>
<thead>
<tr>
<th>Primary Key Constraint: $T_1$</th>
<th>Axiom: $A \leq 1$hasPK.$T_1 \cap 1$hasPK.$T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal Semantics: $\forall x : [A(x) \rightarrow 3y : hasPK(x, y)]$ AND $\forall x, y_1, y_2 : [A(x) \land hasPK(y_1) \land hasPK(y_2) \rightarrow y_1 \equiv y_2]$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Foreign Key Constraint: $T_1, T_2$</th>
<th>Axiom: $A \leq 2$PK.$C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal Semantics: $\forall x : [A(x) \rightarrow 3y : R(x, y) \land C(y)]$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\leq n$-Cardinality Constraint: $T_1, T_2$</th>
<th>Axiom: $A \leq n$R.C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal Semantics: $\forall x : [A(x) \rightarrow \exists R^2(x, C) \geq n]$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\geq n$-Cardinality Constraint: $T_1, T_2$</th>
<th>Axiom: $A \geq n$R.C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Formal Semantics: $\forall x : [A(x) \rightarrow \exists R^2(x, C) \geq n]$</td>
<td></td>
</tr>
</tbody>
</table>

Rewriting $SHIQ$ Queries using $SHIQ$-$RDM$ Views

In this section, we present an algorithm to rewrite conjunctive $SHIQ$ queries into SQL queries on the basis of the mapping system presented in previous section.

The Problem

Given a $SHIQ$ TBox $T$, a set of source relational instances $I$, a set of $SHIQ$-$RDM$ views, plus a set of constraint mappings $CM$, the central problem we are interested in is how to compute answers to $SHIQ$ queries by rewriting the ontology queries into relational queries, on the basis of the mapping system.

Example 3 ($SHIQ$ Conjunctive Query Example) Note again, we use RDF triple syntax to describe the concept atoms and role atoms in the query body.


Herein below, we present a query rewriting approach which can be generally divided into three stages: 1) Reasoning over views; 2) Splitting views into smaller mapping rules; 3) Rewriting queries using mapping rules.

Reasoning over Views

In our approach, we try to enable description logic reasoning in the query rewriting process. This goal is achieved by including an extra reasoning process over the $SHIQ$-$RDM$ views. In this process, a set of rules based on $SHIQ$ TBox axioms, including the axioms generated by constraint mappings, are applied on the original view definitions. First of all, we formally specify these rules as follows.

Definition 3 (View Reasoning Rules) Let $x, x_1, x_2, y$ denote either variable names from $N_v$ or constant names from $N_I$, let $A(x)$ and $R(x_1, x_2)$ denote concept and role atom, and let $B$ denote the set of atoms in the view body.

1. $\subseteq$-rule: IF $a)A \subseteq A', b)A(x) \in B \lor C.A'(x) \notin B$ THEN ADD $A'(x)$ into $B$.
2. $\subseteq$-rule: IF $a)R \subseteq R', b)R(x_1, x_2) \in B, c)R'(x_1, x_2) \notin B$ THEN ADD $R'(x_1, x_2)$ into $B$.

Constraint Mapping

Database integrity constraints such as Primary Key, Foreign Key, and Cardinality Constraint are actually important information. However, most of available mapping approaches merely consider the mappings between tables/columns and classes/properties disregarding the knowledge captured in constraints. We propose a special mapping called “constraint mapping” between database integrity constraints and $SHIQ$ axioms. We then show in the next section an approach to incorporating these $SHIQ$ axioms and description logic reasoning into query rewriting using views to enable the algorithm to answer more types of queries. Table 2 illustrates the set of constraint mapping rules for generating $SHIQ$ axioms.

As examples, suppose $\text{?en}$ is the primary key of table $\text{yuci:emp(?en,?em,?eh)}$, $\text{zju:account(?an,?ah)}$ has a foreign key relationship with $\text{zju:account(?an,?ah)}$ on $\text{?an}$, and the database requires that all person has exactly 2 accounts. We could generate the following $SHIQ$ axioms.

Example 2 (Generated $SHIQ$ Axioms)

1. foaf:Person $\subseteq \exists$foaf:name $\sqcap \leq 1$foaf:name
2. foaf:Person $\subseteq \exists$foaf:holdsAccount,foaf:Account
3. foaf:Person $\subseteq \leq 2$foaf:holdsAccount,foaf:Account
4. foaf:Person $\subseteq \geq 2$foaf:holdsAccount,foaf:Account

Rewriting $SHIQ$ Queries using $SHIQ$-$RDM$ Views

In this section, we present an algorithm to rewrite conjunctive $SHIQ$ queries into SQL queries on the basis of the mapping system presented in previous section.

The Problem

Given a $SHIQ$ TBox $T$, a set of source relational instances $I$, a set of $SHIQ$-$RDM$ views, plus a set of constraint mappings $CM$, the central problem we are interested in is how to compute answers to $SHIQ$ queries by rewriting the ontology queries into relational queries, on the basis of the mapping system.

Example 3 ($SHIQ$ Conjunctive Query Example) Note again, we use RDF triple syntax to describe the concept atoms and role atoms in the query body.


Herein below, we present a query rewriting approach which can be generally divided into three stages: 1) Reasoning over views; 2) Splitting views into smaller mapping rules; 3) Rewriting queries using mapping rules.

Reasoning over Views

In our approach, we try to enable description logic reasoning in the query rewriting process. This goal is achieved by including an extra reasoning process over the $SHIQ$-$RDM$ views. In this process, a set of rules based on $SHIQ$ TBox axioms, including the axioms generated by constraint mappings, are applied on the original view definitions. First of all, we formally specify these rules as follows.

Definition 3 (View Reasoning Rules) Let $x, x_1, x_2, y$ denote either variable names from $N_v$ or constant names from $N_I$, let $A(x)$ and $R(x_1, x_2)$ denote concept and role atom, and let $B$ denote the set of atoms in the view body.

1. $\subseteq$-rule: IF $a)A \subseteq A', b)A(x) \in B \lor C.A'(x) \notin B$ THEN ADD $A'(x)$ into $B$.
2. $\subseteq$-rule: IF $a)R \subseteq R', b)R(x_1, x_2) \in B, c)R'(x_1, x_2) \notin B$ THEN ADD $R'(x_1, x_2)$ into $B$. 
3. \( \exists \)-rule: \( \text{IF } a(A) \subseteq B, b(A(x)) \in (B), \text{c)} \text{There is no y such that } S(x, y) \in B \text{ and } C(y) \in B \text{ THEN ADD both } S(x, y) \text{ and } C(y) \text{ to } (B) \text{ where y is a new existential variable.} \)

4. \( \forall \)-rule: \( \text{IF } a(A) \subseteq \forall S.C, b(A(x)) \in B, \text{c)} \text{There is a } x \text{ such that } S(x, y) \in B \text{ and } C(x, y) \notin B \text{ THEN ADD } C(x, y) \text{ to } B. \)

5. \( \geq n \)-rule: \( \text{IF } a(A) \subseteq \geq n S.C, b(A(x)) \in B, \text{c)} \text{h} \text{x has no n } S\text{-related } y \text{ such that } C(y) \in B \text{ and } y \neq y_j \text{ for } 1 \leq i \leq j \leq n \text{ THEN ADD into } B \text{ with n new role atom } S(x, y_i) \text{ and n new concept atom } C'(y_j) \text{ for all } 1 \leq i \leq n. \)

6. \( \leq n \)-rule: \( \text{IF } a(A) \subseteq \leq n S.C, b(A(x)) \in B, \text{c)} \text{x has more than n } S\text{-related } y \text{ such that } C(y) \in B, d) \text{There are two } S\text{-related } y, y_2 \text{ such that } C(y_1) \in B \text{ and } C(y_2) \in B \text{ with } y_1 \neq y_2 \text{ THEN Replace all concept atoms } A'(y_2) \in B \text{ with } A'(y_1) \text{ and all role atoms } R'(x, y_2) \in B \text{ with } R'(x, y_1). \)

7. Inv-rule: \( \text{IF } a(x) \in B \text{ THEN ADD } R(x, y) \text{ where } y \text{ is the inverse role of } R, b) R(x, y) \in B \).

For example, suppose the TBox has the following axioms, applying these axioms together with the axioms generated by constraint mappings (c.f. Example 2) to v4 in Example 1 results in the extended view as below.

**Example 4 (TBox Axioms)**

1. foaf:OnlineChatAccount \( \subseteq \) foaf:Account
2. foaf:accountServiceHomepage \( \subseteq \) foaf:homepage

**Example 5 (Extended View Example)**


\[
\begin{align*}
& (\text{?y1, rdf:type, foaf:Person}), \\
& (\text{?y1, foaf:name, ?en}), (\text{?y1, foaf:mbox, ?em}), \\
& (\text{?y1, foaf:holdsAccount, ?yh}), \\
& (\text{?y2, rdf:type, foaf:OnlineEcommerceAccount}), \\
& (\text{?y2, foaf:accountName, ?an}), \\
& (\text{?y2, foaf:accountServiceHomepage, ?ah}), \\
& (\text{?y2, rdf:type, foaf:Account}), \\
& (\text{?y2, foaf:homepage, ?ah}), \\
& (\text{?y1, foaf:holdsAccount, ?y3}), \\
& (\text{?y3, rdf:type, foaf:Account}).
\end{align*}
\]

This extra reasoning process is necessary and essential because it directly incorporates the constraints information and TBox knowledge into the view definitions, and enables the rewriting algorithm to answer more types of query. For example, obviously \( Q_1 \) can not be answered by using the original views because the query makes a reference to foaf:OnlineAccount and foaf:homepage which do not appear in any original view definitions at all. For another example, by incorporating axiom 3,4 in Example 2 into the view, the system can answer the query of all person who has exact two accounts. Most importantly, because all TBox knowledge and constraints information has been encoded into the views as new concept atoms or role atoms, it greatly facilitates the implementation and improves the performance of the query rewriting algorithm since the algorithm does not need to consider the TBox knowledge and the constraints information anymore while doing query translation.

### Splitting Views

In the second step, the extended views are split into smaller rules called **Class Mapping Rules**. The purpose is to make it easier to match and replace the bodies of queries and views in the rewriting process.

**Definition 4 (Class Mapping Rule CMR)** A **CMR** rule is in the form

\[
t_1(x_1), ..., t_n(x_n) : \rightarrow R(x_1, y_1), ..., R_m(x, y_m)
\]

- \( t_1(x_1), ..., t_n(x_n) \) are relational predicates.
- \( Ax \) is a concept atom, \( R_1(x_1, y_1), ..., R_m(x, y_m) \) are role atoms.

Intuitively, a **CMR** rule defines a mapping from the relational predicate to a subset (triple group) of view body.

**Definition 5 (Triple Group) **A triple group of a view body is a set of triples that have the same subject. For example, the first four triples of v4 can be considered as a triple group, the next three triples can be viewed as another triple group. In Example 6, \( R_1 \) defines the mapping from ycmi:emp(?en,?em,?an) to the first triple group, and \( R_2 \) defines the mapping to the second triple group.

Triple group serves similar purpose as the MCD proposed in the MiniCon algorithm (Rachel Pottinger 2001). A triple group represents a fragment of the view body which can later be matched and combined more easily.

**Example 6 (Class Mapping Rule Examples)**

- **R1**

  \[
  \text{ycmi:emp(?en,?em,?an) :-}
  \]

  \[
  \begin{align*}
  & (\text{y1, rdf:type, foaf:Person}), \\
  & (\text{y1, foaf:name, ?en}), (\text{y1, foaf:mbox, ?em}).
  \end{align*}
  \]

- **R2**

  \[
  \text{ycmi:emp(?an,?ah) :-}
  \]

  \[
  \begin{align*}
  & (\text{y2, rdf:type, foaf:OnlineEcommerceAccount}), \\
  & (\text{y2, foaf:accountName, ?an}), (\text{y2, foaf:homepage, ?ah}).
  \end{align*}
  \]

- **R3**

  \[
  \text{zju:emp(?en,?em,?eh) :-}
  \]

  \[
  \begin{align*}
  & (\text{y3, rdf:type, foaf:Person}), \\
  & (\text{y3, foaf:name, ?en}), (\text{y3, foaf:mbox, ?em}), \\
  & (\text{y3, foaf:schoolHomepage, ?eh}).
  \end{align*}
  \]

- **R4**

  \[
  \text{zju:emp__account(?en,?ah) :-}
  \]

  \[
  \begin{align*}
  & (\text{y4, rdf:type, foaf:Person}), \\
  & (\text{y4, foaf:name, ?en}), (\text{y4, foaf:holdsAccount, ?ah}).
  \end{align*}
  \]

Algorithm 1 illustrates the rule generation process. In general, for each view, the algorithm firstly partitions the view body into triple groups. Next, the algorithm generates one class mapping rule for each **triple group**. Sometimes, we need to merge the rules, for example, R3 and R4 are merged into the R3-4 because they are both descriptions for foaf:Person class.

### Query Rewriting

In this phase, the Algorithm 2 transforms the input \( SHIQ \) query using the **CMR** mapping rules, and outputs a set of valid rewritings that only refer to a set of view heads. The key to the query rewriting is to find out the **applicable class mapping rule**, namely, those mapping rules that are
applicable class mapping rule

body of the query and partitions the body into triple groups

query body with the rule heads to yield candidate rewritings.

capable of providing (partial) answers, and then replace the query body with the rule heads to yield candidate rewritings.

In details, the rewriting algorithm starts by looking at the body of the query and partitions the body into triple groups as Algorithm 1 does. Next, it begins to look for rewritings for each triple group by trying to find an applicable mapping rules.

Definition 6 (Applicable Class Mapping Rule) Given a triple group \( t \) of a query \( Q \), a mapping rule \( m \) is an applicable class mapping rule \( \mathcal{AM} \) with respect to \( t \), if (a) there is a triple mapping \( \varphi \) that maps every triple in \( t \) to a triple in \( m \), and (b) variables in \( t \) appearing in the query head also appear in the head of \( m \).

Definition 7 (Triple Mapping) Let \( t_1 \), \( t_2 \) denotes RDF(s) triples and let \( \text{Vars}(t_1) \) denotes the set of variables in \( t_1 \). \( t_1 \) is said to be mapped with \( t_2 \) iff there is a functional variable mapping \( \varphi \) from \( \text{Vars}(t_1) \) to \( \text{Vars}(t_2) \) such that \( t_2 = \varphi(t_1) \).

Algorithm 1: Generating Class Mapping Rules

**Require:** Set of SHIQ-RDM view \( V \)
1: Initialize mapping rules list \( M \), sparql query \( Q \)
2: for all \( v \) in \( V \) do
3: Partition the triples in \( v \) into triple groups.
4: Let \( T \) be the set of triple groups generated for \( v \)
5: for all triple group \( t \) in \( T \) do
6: create new mapping rule \( m \)
7: \( \mathcal{R}(m) = \mathcal{R}(v) \)
8: \( \mathcal{E}(m) = t \)
9: add \( m \) to \( M \)
10: end for
11: end for
12: Merge rules if they describe the same class.
13: Output mapping rule list \( M \)

Because of space limitation, only two candidate rewritings are illustrated here.

Given the algorithms, we give an analysis on the complexity of the algorithm. Let \( n \) be the number of triple groups in \( Q_1 \), let \( m \) be the number of mapping rules, it is not difficult to see that in the worst cases the rewriting can be done in time \( O(m^n) \). We sketch the analysis as follows. For each triple group, the possible number of applicable mapping rules is \( m \). Therefore, in the worst case, there might exist \( m^n \) combinations of possible rewritings for \( Q_1 \). The worst case experiment in the the evaluation section reflects the correctness of this statement. We note that all rewriting algorithms in LaV settings are limited because a complete algorithm must produce an exponential number of rewritings (Halevy 2001).

**Implementation and Evaluation**

The approach has been used to develop a data integration application for neuroscience community². The system uses a neuroscience ontology to mediate queries and search across a set of neuroscience and biological databases such as NeuronDB, BrainPharm, Uniprot, etc.

Based on the system, an experiment has been done to evaluate the performance and scalability of the algorithm. We considered two general types of relational schema: chain schema and star schema. We also considered the worst case in which two parameters were looked upon: (1) The number of triple groups of query and (2) the number of sources.

**Chain scenario.** A chain schema has a line of relational tables that are joined one by one. The chain scenario simulates the case where a line of inter-linked relational tables are mapped to a line of inter-related classes. The A of Fig. 3 shows the performance in the chain scenario with the increasing length of the chain and also the number of views. The algorithm can scale up to 300 views under 10 seconds.

**Star scenario.** A star schema has a centralized relational tables which is joined with a set of other tables. The star scenario simulates the case where source relational tables are mapped to a target ontology with large branching factor²

²The link will be put here after the blind review process.
like a star. The B of Fig. 3 depicts the performance with increasing branching factor of the star and also the number of views. The algorithm can scale up to 300 views under 1 seconds.

**Worst case analysis.** In this experiment, we set up 10 sources, and for each source, 8 chained tables are mapped to 8 classes, respectively. The C of Fig. 3 indicates the cost of rewriting increases fast as the number of triple groups and number of sources increases: in the case of 8 groups, the cost reaches 25 seconds with only 4 sources. The experiment result is in accordance with the complexity analysis in previous section. Intuitively, the worst case happens when the query has large numbers of triple groups or spans over large numbers of classes, and for each class, the numbers of class mapping rule are also large. In this case, for each triple group of the query, there is a lot of applicable mapping rules, and thus there would be large number of possible rewritings, since virtually all combinations produce valid rewritings, and complete algorithm is forced to form an exponential number rewritings.

### Related Work

Integrating relational databases with semantic web ontologies has always been a hot topic in semantic web communities. Typical works include D2R3[^1], Virtuoso[^2], RDF Gateway, (An Yuan. 2005), etc. In comparison, our mapping system is featured in consideration of the constraint mappings, and the query rewriting algorithm has been incorporated with the description logic reasoning capability which has never been reported in those approaches.

Our work is also a complement to conventional approaches to integrating relational databases by using description logic such as Information Manifold(Ygal Arens 1996), SIMS (T. Kirk & Srivastava 1995), DLR(Diego Calvanese 1998), ALCQI (Diego Calvanese 2005), etc. Besides, the rewriting algorithm absorbs basic ideas from conventional rewriting techniques reported in relational database literatures such as the bucket algorithm and the inver-rule algorithm (Halevy 2001). Our contribution lies in integrating the constraint mapping information and description logic reasoning capability into query rewriting using views and implementing a practical OWL-to-Relational query rewriting algorithm for semantic web communities.

[^1]: http://sites.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/
[^2]: http://virtuoso.openlinksw.com

### Conclusion

We report a mapping formalism to bridge the gap between the OWL ontologies and schemata of relational data model (RDB), and present an approach to rewriting $\mathcal{SHIQ}$ conjunctive queries into SQL queries on the basis of the mapping system. In particular, we consider a special mapping called Constraint Mapping between RDB integrity constraints and $\mathcal{SHIQ}$ axioms. By including an extra mapping process over the view, the TBox knowledge and the constraints information are incorporated into the view definitions, thus enabling the query rewriting algorithm to answer more types of queries. Open issues remained include the consideration of the mapping of multi-values dependency of relational data model, and consideration of more expressive description logic languages such as $\mathcal{SHOIN}^+(D)$.

### References


Ian Horrocks, Peter F. Patel-Schneider, F. v. H. 2003. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. J. of Web Semantics 1(1):"7–26".

