Rewriting of SPARQL/Update Queries for Securing Data Access

Said Oulmakhzoune[1,2], Nora Cuppens-Boulahia[1], Frederic Cuppens[1], and Stephane Morucci[2]

[1] IT/Telecom-Bretagne, 2 Rue de la Chataigneraie, 35576 Cesson Sevigne - France
{said.oulmakhzoune,nora.cuppens,frederic.cuppens}@telecom-bretagne.eu
[2] Swid, 80 Avenue des Buttes de Cosmes, 35700 Rennes - France
{said.oulmakhzoune,stephane.morucci}@swid.fr

Abstract. Several access control models for database management systems (DBMS) only consider how to manage select queries and then assume that similar mechanism would apply to update queries. However they do not take into account that updating data may possibly disclose some other sensitive data whose access would be forbidden through select queries. This is typically the case of current relational DBMS managed through SQL which are wrongly specified and lead to inconsistency between select and update queries. In this paper, we show how to solve this problem in the case of SPARQL queries. We present an approach based on rewriting SPARQL/Update queries. It involves two steps. The first one satisfies the update constraints. The second one handles consistency between select and update operators. Query rewriting is done by adding positive and negative filters (corresponding respectively to permissions and prohibitions) to the initial query.

1 Introduction

RDF [1](Resource Definition Framework) is a data model based upon the idea of making statements about resources (in particular Web resources) in the form of triple (subject, predicate, object) expressions. SPARQL [2] has been defined to easily locate and extract data from an RDF graph. There are also some recent proposals to extend SPARQL to specify queries for updating RDF documents. SPARUL, also called SPARQL/Update [3], is an extension to SPARQL. It provides the ability to insert, update, and delete RDF triples. The update principle presented by this extension is to delete concerned triples and then insert new ones. For example the following query illustrates an update of the graph 'http://swid.fr/employees' to rename all employees with the name 'Safa' to 'Nora'.

```
PREFIX emp:  <http://swid.fr/emp/0.1/>
WITH <http://swid.fr/employees>
DELETE { ?emp emp:name 'Safa' }
INSERT { ?emp emp:name 'Nora' }
WHERE
{  ?emp rdf:type emp:Employee.
   ?emp emp:name 'Safa' }
```

The 'WITH' clause defines the graph that will be modified. 'DELETE' defines triples to be deleted. 'INSERT' defines triples to be inserted. Finally, 'WHERE' defines the quantification portion.

In the literature, several access control models for database management systems have been defined to implement a security mechanism that controls access to confidential data. For instance, view model for relational database, Stonebraker's model [4] for Ingres database, query transformation model, etc. These proposals assume that similar mechanism would apply for select and update operators, which is not generally true. They do not enforce consistency between the consultation and modification of data.

For example in the case of a SPARQL query, we assume that for two given predicates p and q, we are allowed to select and modify the value of p, but we are not allowed to select the value of q. If we update the value of p using a condition on the predicate q, we can deduce the value of q (see the examples 2 and 3 in section 4.2). Here we use permission to update in order to disclose confidential information which we are not allowed to see. Unfortunately, this problem is not taken into account by several access control models and still exists in many implementations including current relational DBMS compliant with SQL.

Our approach is to rewrite the user SPARQL update query by adding some filters to that query. It involves two steps: (1) Satisfy the security constraints associated with 'update' and (2) handle the consistency between select and update operators.

This paper is organized as follows. Section 2 presents our motivating example. Section 3 presents some notations and definitions. Section 4 formally defines our approach to manage SPARQL update queries. Section 5 presents some related works. Finally, section 6 concludes this paper.

2 Motivating Example

The model of view is an interesting access control model for relational databases. It works pretty well for select operator, but when we move to update, there may be some illegal disclosure of confidential data. Let's take an example to illustrate this problem.

We suppose that we have an 'Employee' table with fields 'name', 'city' and 'salary' and with the following data (see Table 1). We create a user named Bob. We suppose then that Bob is not allowed to see the salary of employees.

Table 1. Result of select query on Employee table

name	city	salary
Said	Rennes	45 000
Toutou	Madrid	60 000
Aymane	London	55 000
Alice	Paris	90 000
Safa	Paris	45 000

According to the view model, we create a view with fields 'name' and 'city'. Then, we give to Bob the permission to 'select' on this view. Let Employee_view be that view, it is defined as follows:

```
Query: CREATE VIEW 'Employee_view' AS (select name, city from Employee)
```

We suppose now that Bob is allowed to select fields of 'Employee_view'. So, he can execute the following query:

```
Query: SELECT * FROM 'Employee_view'
```

The result of that query is presented on the table 2. We note that Bob cannot see the employees' salary. Now let us assume that he is allowed to update data of the 'Employee' table. He updates, for example, the city of employees who earn 45 000 using the following SQL query:

Table 2. Result of select query on Employee_view

name	city
Said	Rennes
Toutou	Madrid
Aymane	London
Alice	Paris
Safa	Paris

```
Query: UPDATE Employee SET city="Brest" WHERE salary=45 000
```

Now, he takes a look at Employee_view in order to see if its content has been changed or not. So, he executes the following query:

```
Query: SELECT * FROM 'Employee_view'
```

As we can see if we compare with the content of Employee_view (Table 3), the city of employees Said and Safa is changed to "Brest". So Bob deduces that their salary is 45000, which he is not allowed to. Although Bob is not permitted to see the salary of the employee table, he is able to learn, through an update command, that there are two employees, Said and Safa, with a salary equal to 45000.

Table 3. Result of select query on Employee_view

name	city
Said	**Brest**
Toutou	Madrid
Aymane	London
Alice	Paris
Safa	**Brest**

This kind of problem exists in all relational databases such as Oracle. It lies in the SQL specification. It does not come from the security policy (Bob could have permission to update the salaries without necessarily being allowed to consult them). It comes from inadequate control on the update query. Let us show how to handle this in the case of SPARQL queries.

3 Notations and Definitions

An RDF database is represented by a set of triples. So, we denote E the set of all RDF triples of our database. We denote $E_{subject}$ (respectively $E_{predicate}$, E_{object}) the projection of E on subject (resp. predicate and object).

We define a "condition of RDF triples" as the application $\omega : E \rightarrow Boolean$ which associates each RDF triple $x = (s, p, o)$ of E with an element of the set $Boolean = \{True, False\}$.

$$\omega : E \rightarrow Boolean,\ x \rightarrow \omega(x)$$

For each element x of E, we say that $\omega(x)$ is satisfied if $\omega(x) = True$. Otherwise we say that $\omega(x)$ is not satisfied.

We define also the "simple condition of RDF triples" as the condition of RDF triples that uses the same operators and functions as the SPARQL filter ($regex, bound, =, <, >$ ...) and constants (see [2] for a complete list of possible operators).

Example 1. The following condition means that if the predicate of x is salary then its value should be less than or equal to 60K.
$(\forall x = (s, p, o) \in E), \omega(x) = (p = \text{emp:salary}) \wedge (o \leq 60\ 000)$

Definition of involved condition. Let $n \in \mathbf{N}^*$, $\{p_i\}_{1 \leq i \leq n}$ be a set of predicates of $E_{predicat}$ and $\{\omega_i\}_{1 \leq i \leq n}$ be a set of simple conditions. Let x be an element of E where $x = (s, p, o)$. The condition expressing that s (subject of x) must have the properties $\{p_i\}_{1 \leq i \leq n}$ such that the value of each property p_i satisfies the condition ω_i, is called an involved condition associated with $\{(p_i, \omega_i)\}_{1 \leq i \leq n}$. This condition, denoted ω, could be expressed as follows: $(\forall x = (s, p, o) \in E)$

$$\omega(x) = \begin{cases} True & \text{if } (\exists (x_1, ..., x_n) \in E^n)/(\forall 1 \leq i \leq n) x_i = (s, p_i, o_i) \\ & \text{where } o_i \in E_{object} \text{ and } \omega_i(x_i) = True \\ False & \text{Otherwise} \end{cases}$$

Let $x = (s, p, o)$ be an element of E. In the case of a simple condition ω_{simple}, we only need the s, p and o to evaluate $\omega_{simple}(x)$. But in the case of an involved condition $\omega_{involved}$ associated with $\{(p_i, \omega_i)\}_{1 \leq i \leq n}$, the value of x is not sufficient to evaluate $\omega_{involved}(x)$. It requires knowledge about other elements of E (sharing the same subject with x).

We denote the constant condition of RDF triple Ω_{True} the application defined as follows:

$$\begin{aligned} \Omega_{True} : E &\rightarrow Boolean \\ x &\rightarrow True \end{aligned}$$

Definition. Let ω be a condition on RDF triples. We define the subset of E that satisfies the condition ω, denoted $I(\omega)$, as follows:

$$I(\omega) = \{x \in E | \quad \omega(x) = True\}$$

We define the complement of the set $I(\omega)$ in E, denoted $\overline{I(\omega)}$, as follows:

$$\overline{I(\omega)} = \{x \in E | \quad x \notin I(\omega)\} = E \backslash I(\omega)$$

Theorem 1: Let ω be a condition on RDF triples, $I(\bar{\omega}) = \overline{I(\omega)} = E \backslash I(\omega)$

Proof of theorem 1
$x \in I(\bar{\omega}) \iff \{x \in E | \quad \bar{\omega}(x) = True\} \iff \{x \in E | \quad \overline{\omega(x)} = \underline{True}\}$
$\iff \{x \in E | \quad \omega(x) = False\} \iff \{x \in E | \quad x \notin I(\omega)\} \iff x \in \overline{I(\omega)}$. $\square$

4 Principle of Our Approach

Let ω be a condition of RDF triples and $\{p_i\}_{1 \leq i \leq n}$ a set of predicates of $E_{predicate}$. We define our security rules as the permission or prohibition to select (or update) the value of predicates $\{p_i\}_{1 \leq i \leq n}$ if the condition ω is satisfied.

Our approach involves two steps:

– (1) Satisfy the security constraints associated with ‘update’.
– (2) Handle the inconsistency between ‘select’ and ‘update’.

4.1 Update Access Control

In this case we similarly treat the ‘DELETE’ and ‘INSERT’ clause of the update query.

Let D_{Query}, I_{Query} and W_{Query} be respectively the DELETE, INSERT and WHERE clause of a user’s update query. There are two cases, the case of prohibition and the case of permission.

Prohibition case: We assume that a user u is not allowed to update the value of predicates $\{p_i\}_{1 \leq i \leq n}$ if the condition ω is satisfied. Since the security policy is closed then this user is allowed to update the value of predicates that are not in $\{p_i\}_{1 \leq i \leq n}$. Now, if this user tries to update at least one predicate p of $\{p_i\}_{1 \leq i \leq n}$, then we must check if the condition ω is not satisfied (prohibition case). Which means adding the negative filter of ω to W_{Query}.

We deduce then the following expression in the case of prohibition:

$$[(\exists p \in D_{Query} \cup I_{Query}) | p \in \{p_i\}_{1 \leq i \leq n}] \rightarrow [\text{Filter}(W_{Query}, \overline{\omega})] \quad (1)$$

Where Filter(GP,C) means adding the SPARQL filter of the RDF condition C to the group of patterns GP.

Let I^u_{Integ} be the set of elements that the user u is permitted to update. For a given predicate q we denote S_q a set of triple of E that has the predicate q. Let $S^u_{pred} = \bigcup_{i=1}^{n} S_{p_i}$.

The user u is not allowed to update the value of predicates $\{p_i\}_{1 \le i \le n}$ if the condition ω is satisfied. This is equivalent to: $\overline{I^u_{Integ}} = S^u_{pred} \cap I(\omega)$. According to the result of theorem 1, we deduce that:

$$I^u_{Integ} = \overline{S^u_{pred}} \cup I(\overline{\omega})$$

Proof of integrity

Let x be an element of E that has been updated by the user's transformed query. There are two cases: (i)$x \notin S^u_{pred}$ (ii) $x \in S^u_{pred}$. In the case (i) we have $x \in \overline{S^u_{pred}} \subseteq I^u_{Integ}$. So, $x \in I^u_{Integ}$. In the case (ii), we have $x \in S^u_{pred}$. According to the expression 1 above, x satisfies the negation of the condition ω, which means that $x \in I(\overline{\omega}) \subseteq I^u_{Integ}$. So, $x \in I^u_{Integ}$. In both cases, $x \in I^u_{Integ}$, which proves that the expression 1 above preserves the integrity.

Permission case: We assume that a user u is allowed to update the value of predicates $\{p_i\}_{1 \le i \le n}$ if the condition ω is satisfied. So:

- (i) The user u is not allowed to update the value of predicates $\{p_i\}_{1 \le i \le n}$ if the condition $\overline{\omega}$ is satisfied.
- (ii) He is not allowed also to update the value of predicates that are not in $\{p_i\}_{1 \le i \le n}$. Which means that he is not allowed to update the value of predicates $E_{predicate} \backslash \{p_i\}_{1 \le i \le n}$ if the condition Ω_{True} is satisfied.

According to the expression (1), the prohibition (i) is equivalent to:

$$[(\exists p \in D_{Query} \cup I_{Query}) | p \in \{p_i\}_{1 \le i \le n}] \rightarrow [\text{Filter}(W_{Query}, \overline{\overline{\omega}})]$$

We have $\overline{\overline{\omega}} = \omega$. So,

$$[(\exists p \in D_{Query} \cup I_{Query}) | p \in \{p_i\}_{1 \le i \le n}] \rightarrow [\text{Filter}(W_{Query}, \omega)] \tag{2}$$

The prohibition (ii) is equivalent to:

$$[(\exists p \in D_{Query} \cup I_{Query}) | p \in E_{predicate} \backslash \{p_i\}_{1 \le i \le n}] \rightarrow [\text{Filter}(W_{Query}, \overline{\Omega_{True}})]$$

We have $\overline{\Omega_{True}} = False$, so (ii) is equivalent to:

$$[(\exists p \in D_{Query} \cup I_{Query}) | p \notin \{p_i\}_{1 \le i \le n}] \rightarrow [\text{Filter}(W_{Query}, False)] \tag{3}$$

Adding false filter to W_{Query} means ignoring the execution of the query. So, we treat the case (ii) first. If we need to add a false filter then we do not have to treat the case (i), since the query is ignored. Otherwise, we treat the case (i).

4.2 Consistency between Consultation and Modification

This section treats the second step of our approach. It handles the consistency between the 'select' and 'update' operators. This treatment is done only by analysing the quantification portion of the update query.

Let u be a user and I^u_{Conf} (resp. I^u_{Integ}) be the set of elements that the user u is permitted to select (resp. to update).

In general, in the case of the select operator, the result of the query must be a subset of I^u_{Conf} (Figure 1 (A)) in order to preserve confidentiality. Similarly for the update operator, the modified data must be a subset of I^u_{Integ} in order to preserve the integrity (Figure 1 (B)). As shown in the motivating example, these two rules are not sufficient to preserve confidentiality when the user has both rights to select and rights to update. In other words, the user can update an element x of $I^u_{Conf} \cap I^u_{Integ}$ using reference to an element y of $\overline{I^u_{Conf}} = E \backslash I^u_{Conf}$ in order to deduce the value of y associated with x, which he is not allowed to see. There are two cases: (1) $I^u_{Conf} \cap I^u_{Integ} = \emptyset$, (2)$I^u_{Conf} \cap I^u_{Integ} \neq \emptyset$. It is obvious that in the first case (1), there is no such problem. However, in the current SQL implementation, we still have another kind of problem of confidentiality. For exemple, in our motivating example, if we suppose that the user Alice is allowed only to select names and cities of employees and she is allowed to update only their salaries. We are in the case (1). Alice updates the salary of employees who earn exactly 45000 using the folowing SQL query:

```
SQL> UPDATE Employee SET salary=salary+100 WHERE salary=45 000;
2 rows updated
```

Although, Alice is not permitted to see the salary of employees, she has been able to learn, through this update command, that there are two employees with a

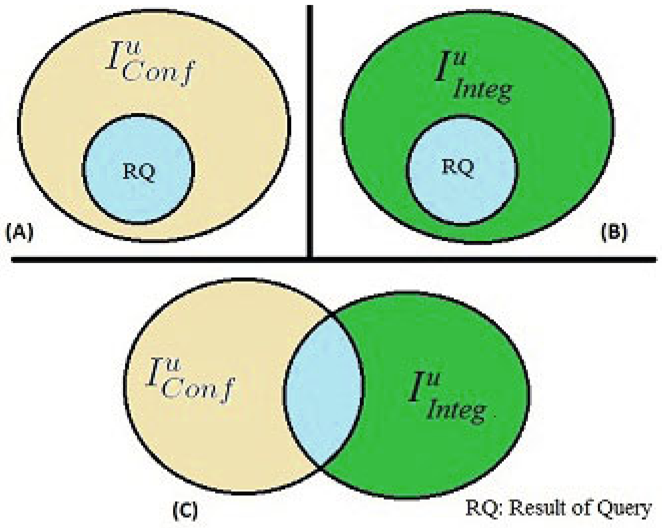

Fig. 1. Consistency between select and update operators

salary 45000. This corresponds to a write down prohibited by the Bell-LaPadula model. To solve it, we have just to delete the information message (number of rows that has been updated).

Now let us assume that $I^u_{Conf} \cap I^u_{Integ} \neq \emptyset$. Let $x \in I^u_{Conf} \cap I^u_{Integ}$, we denote $I_{Ref}(x)$ a subset of elements of $\overline{I^u_{Conf}}$ that are in relation with x, i.e $I_{Ref}(x) = \{y \in \overline{I^u_{Conf}} | \exists R, \textit{ a binary relation over E such that } xRy\}$. If $I_{Ref}(x)$ is not empty, then we can deduce the value of each element of $I_{Ref}(x)$. We simply update the value of x using elements of $I_{Ref}(x)$ in the where clause of our query. So, our problem is equivalent to the following proposition:

$$[(\exists x \in I^u_{Conf} \cap I^u_{Integ}) | I_{Ref}(x) \neq \emptyset] \rightarrow [Interference(select, update)] \tag{4}$$

$Interference(select, update)$ means that there exists an interference between select and update operators. To solve this problem, we must control W_{Query} (the where clause of the query) to avoid referencing a value that is not in I^u_{Conf}.

There are two cases, the case of prohibition and the case of permission.

Prohibition case. Let ω be a condition of RDF triples. We assume that a user u is not allowed to see (select) values of predicates $\{p_i\}_{1 \leq i \leq n}$ where $n \in N^*$, if the condition ω is satisfied.

Let S^u_{pred} be a set of all possible triples of E where predicates are elements of $\{p_i\}_{1 \leq i \leq n}$. Since we are in the case of prohibition, so, $\overline{I^u_{Conf}} = S^u_{pred} \cap I(\omega)$. If the W_{Query} of the update query uses at least one predicate of the set $\{p_i\}_{1 \leq i \leq n}$, then the condition ω should not be satisfied in order to enforce the security policy. In other words we have to introduce the negative filter of ω in W_{Query}. This is equivalent to:

$$[(\exists p \in W_{Query}) | p \in \{p_i\}_{1 \leq i \leq n}] \rightarrow \text{Filter}(W_{Query}, \overline{\omega(x_p)}) \tag{5}$$

where x_p is the triple pattern of W_{Query} using the predicate p.

Proof of Confidentiality: We assume that the user u tries to update an element x of $I^u_{Conf} \cap I^u_{Integ}$ by referencing an element $y \in E$. It is obvious that if $y \in I^u_{Conf}$ there is no such problem. We assume that $y \notin I^u_{Conf}$ i.e $y \in \overline{I^u_{Conf}}$. Since $\overline{I^u_{Conf}} = S^u_{pred} \cap I(\omega)$, so, $y \in I(\omega)$ and $y \in S^u_{pred}$.

Let us proceed by contradiction to proof that the update of x has not occurred. We have $y \in S^u_{pred}$, so according to the expression 5 above, the negative filter of ω has been added to W_{Query}. If we assume that the user update takes place, then y satisfies the negation of ω i.e $y \in I(\overline{\omega}) = \overline{I(\omega)}$. This means that $y \notin I(\omega)$, so this is *Contradiction*. So, the update of x has not occurred.

Involved condition: Let P be the graph pattern of W_{Query}. Let ω be an involved condition associated with $\{(q_i, \omega_i)\}_{1 \leq i \leq m}$ where $\{q_i\}_{1 \leq i \leq m}$ is a set of predicates and $\{\omega_i\}_{1 \leq i \leq m}$ is a set of simple conditions. In the case of an involved condition, negative filter is equivalent to: (i) at least one simple condition of

$\{\omega_i\}_{1\leq i\leq m}$ is not satisfied or (ii) at least one predicate of $\{q_i\}_{1\leq i\leq m}$ does not appear. That is W_{Query} corresponds to the following transformed query:

$$W_{Query} = P_1 \ \mathit{UNION}\ P_2$$

where P_1 and P_2 are the following graph patterns:

$$P_1 = (\text{P AND } (\mathit{UNION}_{i=1}^{m}(tp_i \text{ FILTER } \overline{\omega_i(tp_i)})))$$

$$P_2 = (\text{P } \mathit{OPT}_{i=1}^{m}(tp_i) \text{ FILTER } (\vee_{i=0}^{m} !\mathit{bound}(?obj_i)))$$

such that $\{tp_i = (?emp, q_i, ?obj_i)\}_{1\leq i\leq m}$. AND, $UNION$, OPT and $FILTER$ are respectively the SPARQL binary operators (.), UNION, OPTIONAL and FILTER. P_1 guarantees that at least one simple condition of $\{\omega_i\}_{1\leq i\leq m}$ is not satisfied. P_2 represents the set of elements that does not have at least one predicate of $\{q_i\}_{1\leq i\leq m}$.

In SPARQL 1.1 version [3], *NOT EXIST* and *EXISTS* are respectively two filters using graph pattern in order to test for the absence or presence of a pattern. In the case of prohibition for an involved condition, it comes to test the non existence of triples pattern $\{tp_i \text{ FILTER } \omega_i(tp_i)\}_{1\leq i\leq m}$. In this case W_{Query} corresponds, for example, to the following transformed query: $W_{Query} = P\ \mathit{FILTER\ NOT\ EXIST}\ (\mathit{AND}_{i=1}^{m}(tp_i \text{ FILTER } \omega_i(tp_i)))$

Permission case. Let ω be a condition of RDF triples. We assume that a user is allowed to see (select) the value of predicates $\{p_i\}_{1\leq i\leq n}$ where $n \in \mathbb{N}^*$, if the condition ω is satisfied. Since our security policy is closed, this permission could be expressed as the following prohibitions:

– (a) The user is not allowed to see the value of predicates $E_{Predicate}\backslash\{p_i\}_{1\leq i\leq n}$
– (b) The user is not allowed to see the value of predicates $\{p_i\}_{1\leq i\leq n}$, if the condition $\overline{\omega}$ is satisfied.

Let us apply the expression 5 to the case (a) and then to the case (b). The case (a) could be expressed as prohibition to see the value of predicates $E_{Predicate}\backslash\{p_i\}_{1\leq i\leq n}$ if the condition Ω_{True} is satisfied. Ω_{True} is a simple condition, so according to the prohibition algorithm, if W_{Query} uses at least one predicate of the set $E_{Predicate}\backslash\{p_i\}_{1\leq i\leq n}$ then we add the corresponding negative filter of Ω_{True} to W_{Query}, i.e.:

$$[(\exists p \in W_{Query})|p \in E_{Predicate}\backslash\{p_i\}_{1\leq i\leq n}] \rightarrow [\text{Filter}(W_{Query},\overline{\Omega_{True}(x_p)})]$$

where x_p is the triple pattern of W_{Query} using the predicate p. This is equivalent to:

$$[(\exists p \in W_{Query})|p \notin \{p_i\}_{1\leq i\leq n}] \rightarrow [\text{Filter}(W_{Query},\text{False})] \quad (6)$$

Adding the False filter to W_{Query}, means ignoring the execution of the query. So, we do not have to treat the case (b) since the query is ignored.

Now if all predicates of W_{Query} belong to $\{p_i\}_{1\leq i\leq n}$. We treat the case (b). According to the prohibition algorithm, (b) is equivalent to:

$$[(\forall p \in W_{Query})|p \in \{p_i\}_{1\leq i\leq n}] \rightarrow [\text{Filter}(W_{Query},\overline{\omega(x_p)})]$$

The negative filter of $\overline{\omega}$ is the positive filter of ω. So, the case (b) is equivalent to the following proposition:

$$[(\forall p \in W_{Query})|p \in \{p_i\}_{1\leq i\leq n}] \rightarrow [\text{Filter}(W_{Query},\omega(x_p))] \tag{7}$$

Example 2. (case of involved condition)

We assume that Bob is not allowed to see the name, age and salary of network department employees where their age is greater than 30. This prohibition could be expressed as “Bob is not allowed to see values of predicates {emp:name, emp:age, emp:salary} if the involved condition ω, defined below, is satisfied”.

$(\forall x = (s, p, o) \in E)$

$$\omega(x) = \begin{cases} True & \text{if } (\exists(value_1, value_2) \in E^2{}_{Object})|\omega_1(x_1) = \text{True and } \omega_1(x_2) = \text{True} \\ & \text{where } x_1 = (\text{s,emp:dept},value_1) \text{ and } x_2 = (\text{s,emp:age},value_2) \\ False & \text{Otherwise} \end{cases}$$

such that $(\forall y = (s', p', o') \in E)$

$$\omega_1(y) = ((p'\text{=emp:dept}) \wedge (o'\text{="Network"})) \vee ((p'\text{=emp:age}) \wedge (o' \geq 30))$$

We assume also that Bob is allowed to update the city of all employees. Bob tries to update the city of employees whose name is ”Alice”. So the corresponding Bob’s update query will be as follows:

```
WITH <http://swid.fr/employees>
DELETE { ?emp emp:city ?city }
INSERT { ?emp emp:city 'Rennes' }
WHERE{ ?emp  rdf:type   emp:Employee;
             emp:name   "Alice". }
```

We note that Bob uses the predicate name on W_{Query}. We know also that this predicate is prohibited to be selected under the involved condition ω. We assume that the employee Alice works in the network department and she is 34 years old. So the execution of this query allows Bob to deduce that there is an employee named ”Alice” on the network department and her age is greater than 30.

According to the result above, the transformed query will be as follows:

```
WITH <http://swid.fr/employees>
DELETE { ?emp emp:city ?city }
INSERT { ?emp emp:city 'Rennes' }
WHERE
{  {?emp rdf:type emp:Employee; emp:name "Alice".
     {  {?emp emp:dept ?dept. FILTER(?dept !="Network")}
        UNION
        {?emp emp:age ?age. FILTER(?age<30)}
     }
   }UNION{
     ?emp rdf:type emp:Employee; emp:name "Alice".
     OPTIONAL{?emp emp:dept ?dept} OPTIONAL{?emp emp:age ?age}
     FILTER(!bound(?dept) || !bound(?age))
   }
}
```

In the case of SPARQL 1.1, the transformed query will be as follows:

```
WITH <http://swid.fr/employees>
DELETE { ?emp emp:city ?city }
INSERT { ?emp emp:city 'Rennes' }
WHERE
{  ?emp rdf:type emp:Employee;
        emp:name "Alice".
   FILTER NOT EXIST{
      ?emp emp:dept ?dept. FILTER(?dept ="Network")
      ?emp emp:age ?age. FILTER(?age>=30) }
}
```

i.e. the update will be done only on employees who are not in the network department (or do not have the department property) or are less than 30 years old (or do not have the age property).

5 Related Works

SPARQL is a recent query language. Even if there is a clear need to protect SPARQL queries, there is only one proposal to define an approach to evaluate SPARQL with respect to an access control policy. This approach called "fQuery" [5] handles only the case of the select operator. But there is still no proposal to securely manage SPARQL/Update.

In the case of SQL, security is based on view definitions. Using GRANT and REVOKE operators, one can specify which views a given user (or user role) is permitted to access. This approach works pretty well for select operator, but when we move to update there may be some illegal disclosure of confidential data (see our motivating example). This may be considered a security bug of current relational DBMS implementation.

An interesting approach for relational DBMS based on query transformation was suggested by Stonebraker [4]. In this case, the query transformation is specified by adding conditions to the WHERE clause of the original query. The author assumes that a similar mechanism would apply to both select and update operators, which is not generally true. He does not handle the problem of consistency between data selection and update. Our approach and algorithms could be applied to Stonebraker's approach by using domains instead of predicates.

Query transformation is also the approach suggested by Oracle in its Virtual Private Database (VPD) mechanism [6]. However, VPD does not include means to specify security policy requirements so that query transformation must be hard coded in PL-SQL by the database administrator. Thus, VPD does not define a general approach to automatically transform queries with respect to an access control policy.

Regarding XML database security, an interesting work to control updates was suggested by Gabillon [7]. This approach takes into account interactions between the read and write privileges. The suggested solution is to evaluate the write operation (update, insert, delete) on the view that the user is permitted to see. In other words, users cannot perform write operations on nodes they cannot see. However, our approach is less restrictive since it allows a user to update predicates that this user is not allowed to see while maintaining consistency between the select and update operators.

6 Conclusion and Future Works

In this paper, we have defined an approach to protect SPARQL/Update queries using query transformation. It involves two steps. The first one is to satisfy the update constraints. The second one is to handle consistency between 'select' and 'update' operators by analyzing the WHERE clause of the update query. Query rewriting is done by adding filters to the initial query: Positive filters corresponding to permission and negative filters corresponding to prohibition.

A possible extension would be to define a user friendly specification language to express such an access control policy. For this purpose, a possible direction for future work would be to derive the filter definition from the specification of an access control policy based on RBAC [8] or OrBAC [9]. Another extension will be to integrate our approach into service composition management.

References

1. Klyne, G., Carroll, J.: Resource description framework (rdf): Concepts and abstract syntax, `http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/`
2. Prud'Hommeaux, E., Seaborne, A.: Sparql query language for rdf (January 2008), `http://www.w3.org/TR/rdf-sparql-query/`
3. Schenk, S., Gearon, P., Passant, A.: Sparql 1.1 update (June 2010), `http://www.w3.org/TR/2010/WD-sparql11-update-20100601/`
4. Stonebraker, M., Wong, E.: Access control in a relational data base management system by query modification. In: Proceedings of the 1974 Annual Conference, pp. 180–186 (1974)
5. Oulmakhzoune, S., Cuppens-Boulahia, N., Cuppens, F., Morucci, S.: fQuery: SPARQL Query Rewriting to Enforce Data Confidentiality. In: Foresti, S., Jajodia, S. (eds.) Data and Applications Security and Privacy (DBSec). LNCS, vol. 6166, pp. 146–161. Springer, Heidelberg (2010)
6. Huey, P.: Oracle database security guide : using oracle virtual private database to control data access, ch. 7, `http://download.oracle.com/docs/cd/E11882_01/network.112/e10574.pdf`
7. Gabillon, A.: A formal access control model for xml databases. In: Jonker, W., Petković, M. (eds.) SDM 2005. LNCS, vol. 3674, pp. 86–103. Springer, Heidelberg (2005)
8. Ferraiolo, D.F., Sandhu, R., Gavrila, S., Kuhn, D.R., Chandramouli, R.: Proposed NIST Standard for Role-Based Access Control. ACM Transactions on Information and Systems Security (TISSEC) 4(3) (2001)
9. Abou El Kalam, A., El Baida, R., Balbiani, P., Benferhat, S., Cuppens, F., Deswarte, Y., Miège, A., Saurel, C., Trouessin, G.: Organization Based Access Control. In: 8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003), Lake Como, Italy (June 2003)