Pragmatic XML Access Control using Off-the-shelf RDBMS

Bo Luo, Dongwon Lee, and Peng Liu

The Pennsylvania State University

Abstract

As the XML model gets more popular, new needs arise to specify *access control* within XML model. To meet these needs, various XML access control models and enforcement methods have been proposed recently. However, by and large, these approaches either assume the support of security features from XML databases or use proprietary tools outside of databases. Since there are currently few commercial XML databases with such capabilities, the proposed approaches are not yet practical. Toward this problem, therefore, we explore the question of "Is is possible to fully support XML access controls using RDBMS?" By formalizing XML and relational access control models using the *deep set* operators, we show that the problem of XML access control atop RDBMS is amount to the problem of converting XML deep set operators into equivalent relational deep set operators. We show the conversion algebra and identify the properties to ensure the correct conversion. Finally, we present three practical implementations of XML access controls using off-the-shelf RDBMS and their performance results.

1 Introduction

The XML model [9] has emerged as the *de facto* standard for storing and exchanging information in the Internet Age. As more information is exchanged and processed over the Web, the issues of security become increasingly important. Such issues are diverse, spanning from data level security to network level security to high-level access controls. In this paper, in particular, our focus is on how to support *access controls* for XML data.

Current access control research can be categorized into two groups: access control modeling and access control enforcement mechanisms. Table 1 illustrates the current development of access control model research. First row refers to (research-oriented) access control models developed for XML and relational models, respectively, while second row refers to state-of-art open-source or commercial products for each model. In general, not all the features proposed by modeling research community (first row) are implemented in existing access control enforcement approaches. For instance, to our best knowledge, most industrial or open source XML database products do not have supports for fine-grained XML access controls yet (to be elaborated in Section 3).

Recently, many access control methods extending the XML model to incorporate security aspects have been proposed (e.g., XACML [23], [12], [3], [55]). To the lesser or greater extent, however, XML access control enforcement mechanisms proposed in the research community neglect the fact that the most XML data still resides in RDBMS behind the scenes. In the scenario of RDBMS-backed XML database systems (hereafter *XRDB*), XML data is shredded into relations and stored in RDBMS; query-answering is conducted through a conversion layer so that users interact with the system as if it is native XML. In the scenario of XML publishing, relational data is compiled into XML format for distribution and exchange; users receive documents as if they were originated from XML model. For both scenarios, we enjoy the benefit of XML model while taking advantage of the maturity of the off-the-shelf RDBMS. In both scenarios, it is desirable to natively specify access controls on the XML side (upper-left quadrant of Table 1), but they need to be enforced on the RDBMS side (lower-right quadrant). We believe that current XML access control

Table 1: The overview of XML and Relational access control model supports.

XML	**Relational**	
XML Access Control Models (e.g., [12], [3])	Relational Access Control Models (e.g., [25])	**Models**
XML Databases (e.g., Galax [53], Tamino [51])	Relational Databases (e.g., Oracle, DB2, SQL Server)	**Products**

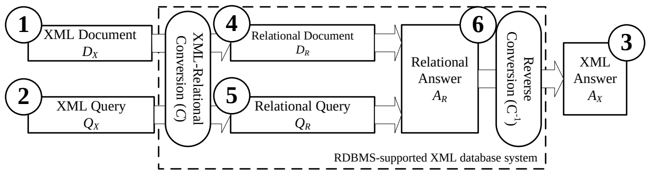
enforcement mechanism research is in a sense re-inventing wheels without utilizing existing relational access control models (i.e., upper-right quadrant) or leveraging on security features that are readily available in relational products (i.e., lower-right quadrant). In short, therefore, our goal in this paper is to answer the following question in Table 1:

> **When is it (not) possible to support the upper-left quadrant (i.e., XML access controls) using the lower-right quadrant (i.e., RDBMS)? Why? How?**

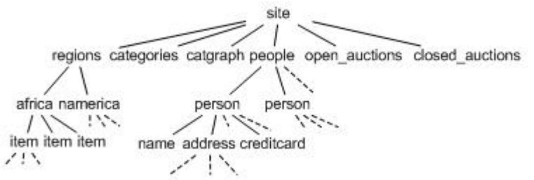
As illustrated in Figure 1(a), in an XRDB system: XML data D_X are first converted into D_R and stored in RDBMS; user issues XML query Q_X (XPath or XQuery) using published XML schema; Q_X is then converted into Q_R (in SQL) and evaluated against D_R; relational answer A_R is finally converted back to XML answer A_X and returned to user.

Challenges. First, the major challenges of supporting XML access controls in XRDB systems stem from the inherent discrepancy of XML and relational data models. Relational data model features a structure of two-dimensional table, while XML features a hierarchical data model. When XML data are shredded into relational data model by some transformation algorithms, not all transformation algorithms can fully preserve structural properties of XML model [1]. Therefore, the inherent incompatibility of two data models leads to the fundamental discrepancy between two access control models. Second, relational access control policies define authorized actions of "cells," where each cell is an impartible element and whose accessibility is explicitly expressed. However, XML nodes are hierarchically nested, and XML data model inherently takes "answer by subtree model" (e.g., querying for `//foo` yields the whole subtree rooting at node `<foo/>`). Therefore, for any XML node, an action could be: authorized (or unauthorized) to the whole subtree, or partially authorized. The later case does not occur in relational access control model. Finally, in XML model, we can control the access right of each individual node. In traditional relational model, the smallest granularity that one may control is a column via GRANT/REVOKE. Therefore, one needs to employ more recent developments of RDBMS access controls (e.g., Oracle VPD) to enable row/cell level access control.

Key contributions. (1) To our best knowledge, this work is the first one to algebraically formalize XML access control in both native XML (XDB) and XRDB environment. (2) This work takes the first steps to define the *equivalent objects* and *equivalent operations* between native XML and XRDB systems. With this concept, we can migrate all the exciting features of native XML systems into XRDB by converting the atomic operations into equivalent relational counterparts. In this paper, we take the feature of fine-grained XML access control for a pilot study, and the results are encouraging. (3) This work shows for the first time that the "security" of XRDB can be achieved by finding the "equivalent" relational operators for three specific deep-set operators. This finding provides a viable way to build secure XRDB systems. (4) Finally, this work proposes several practical approaches to implement the viable way "discovered" by our theory.


1
XML Document D_X
2
XML Query Q_X
XML-Relational Conversion (C)
4
Relational Document D_R
5
Relational Query Q_R
Relational Answer A_R
6
Reverse Conversion (C^{-1})
3
XML Answer A_X
RDBMS-supported XML database system

(a) Overview



(b) XMark DTD

Figure 1: Overview architecture and an example XML schema.

Table 2: Notations used throughout the paper.

Symbol	Definition
Q_X/Q_R	An XPath (resp. SQL) query
ACR_X/ACR_R	Access control rule set in XML (resp. relational) security model
R_X/R_R	An individual access control rule in XML (resp. relational) model
V_X/V_R	An XML (resp. relational) view
R^+/R^-	Positive/negative access control rule
$Q\langle D\rangle$ or $\langle Q\rangle$	The answer of the query Q against data D (D can be omitted if clear)
SA	Safe answer where no part of them violates access control policy
X2R	XML-2-Relational
$C()/C^{-1}()$	X2R conversion algorithm such as XRel, XParent, or inlining (resp. reverse algorithm)
XRDB(C)	RDBMS-supported XML database (i.e., XRDB) using a conversion algorithm C

2 Related Work

2.1 XML and Relational Access Control

First, on the model side, several authorization-based XML access control models are proposed. Starting with [48] for HTML documents, [13, 12] provides access controls and their enforcement by associating an authorization sheet to each XML document or DTD. Among comparable proposals, in [3], access control environment for XML documents and techniques to deal with authorization priorities and conflict resolution issues are proposed. [29] introduced provisional authorization model and XACL. [21] formalizes the way of specifying *objects* in XML access control using XPath. Finally, recall that the use of authorization priorities with propagation and overriding is an important issue in XML access controls, similar to that in OODB [18].

Most of the proposals adopt either *role-based access control* (e.g. [56]) or *credential-based access control* (e.g. [4]). The major difference between them is the way they identify users. Credential-based access control is more flexible and powerful in this aspect. However, in the research of access control enforcement mechanisms, people tend to choose a relatively simple access control model (e.g. [40, 8, 35, 46, 44, 46, 41]) to avoid distraction.

XML access control enforcement mechanisms in native XML environment have been intensively studied in recent years. Generally speaking, they are categorized into four classes: (1) engine level mechanisms implement node-level security check inside XML database engine; they tag each XML node with a label [14, 11, 58] or an authorization list [60, 28], and enforce security check at query processing. (2) view based approaches build security views that only contain access-granted data [54, 17, 30]. [1](3) pre-processing approaches check user queries and enforce access control rules before queries are evaluated, such as the static analysis approach [40, 41], QFilter approach [35], function-based approach [46], access condition table approach [44] policy matching tree[45], secure query rewrite (SQR) approach [38], etc. (4) [8] considers access control of streaming XML data and apply security check at client side, using a filtering mechanism. [19, 37]. Moreover, [5, 10] focus on XML access control policies and enforcement as well as encryption issues in information pushing or brokerage systems. More recently, protecting the privacy and security associated with XML tree structure (instead of content) becomes an emergent topic [19, 39].

Relational access control models can be classified into two categories: *multilevel security models* [27, 57, 49] and *discretionary security models (DAC)*. Most real world database systems implement a table/column level DAC similar to the one implemented in System R [24]. View-based approaches is the traditional method to enable row-level access control, while Oracle's VPD is the most recent development. Finally, some advanced access control models (e.g., [25, 26]) are proposed in a more theoretical manner.

[1]When a view-based approach implements virtual views without materializing them, it is inherently a pre-processing approach.

2.2 XML and Relational Conversion

Toward conversion between XML and relational models, an array of research has addressed the particular issues lately. On the industry side, database vendors are busily extending their databases to adopt XML types. Shredding and non-shredding are two major pathes that followed by commercial products. Oracle provides both un-shredded (CLOB) and shredded storage options [42]. Microsoft supports XML shredding and publishing through mid-tier approach in SQL Server 2000, and adds CLOB storage in SQL Server 2005 [47]. IBM proposes the first native XML storage in DB2 9, but shredded XML storage (through schema decomposition) is still kept as an important feature [43, 6].

On the research side, various proposals have been made recently, mainly either schema-based (e.g., [16, 52, 31]) or schema-oblivious (e.g., [20, 59]) approaches. In terms of access control, some commercial products apply existing column level access control of RDBMS on XML data stored in CLOB columns. None of these approaches supports or discusses fine-grained access control. Finally, to our best knowledge, the only work that is directly relevant to our proposal is [55]. [55] proposes an idea of using RDBMS to handle XML access controls, in a rather limited setting. In our vision paper [32], we addressed some issues and challenges of enforcing XML access control atop RDBMS. We provide the algebraic analysis and explore practical solutions in this paper.

Our framework is not tied to a particular conversion method. Throughout this paper, we use shared-inlining [52] and XRel [59] as the examples of schema-based and schema-oblivious conversion methods, respectively. Briefly, XRel decomposes XML documents into document, element, attribute, text, and path tables. The structure of the element and path table are: `element(docID, elementID, parentID, depth, pathID, st, ed, idx, reidx)` and `pth(pathID,pathexp)`, respectively. In this approach, each node is stored as one record in the `element` table, and each distinct path is stored as one record in the `pth` table. As a simple example, we decompose an XMark document using XRel and show part of `element` table in Figure 2 (b). As we can see, element 252 is a node of path 164 ("`/site/people`", as stored in the `path` table); which starts from offset 33996 (byte) and ends at 36229 in the original XML document.

It uses `pth.pathexp` to keep the path expressions (similar to XPath), and `element.st` and `element.ed` to mark the start and end offset of each node in the document.

Throughout the rest of this paper, we use the online auction DTD of XMark [50] as the exemplar schema, shown in Figure 1(b). Table 2 summaries symbols used in this paper.

3 Preliminaries

3.1 XML Access Control Policy

Access control models define the semantics and syntax of access control policies. Although they could be very complicated, the essential of access control models is to describe *subjects*, *objects*, *actions* and all the variations around it. Fortunately, there is no discrepancy in identifying *subjects* and defining *actions* in XML and relational environment, e.g., they both could adopt role-based access control to identify user's access rights; or both adopt credential-based access control that uses attributes to denote rights. As we described in Section 1, shredding XML access control models into relational ones is a challenging task, because of the fundamental discrepancies of XML and Relational data models. Therefore, challenges reside in *object*-related components of access control models, while issues that only relate to *subjects* and *actions* are trivial. Thus, our subsequent discussion focuses more on *object* part. In this paper, we adopt the model proposed in [12] as the basis; other models like [40, 8, 35, 46] can be used as well with a reasonable change.

Definition 1 (XML Access Control Rule) *An XML access control policy is specified by a set of 4-tuple* **access control rules***: $R_X = \{\textbf{subject}, \textbf{object}, \textbf{action}, \textbf{sign}\}$, where* subject *is to whom an authorization is granted (i.e., role),* object *is a set of XML nodes (in XPath) to which the policy is applied,* action *can be either read, write, or update, and* sign $\in \{+, -\}$ *refers to either access granted or denied, respectively.* □

In this model, access is prohibited by default. If conflict occurs between positive and negative rules, negative rule takes precedence. Compared to the 5-tuple model of [12], we simplified it by eliminating the *type* $\in \{LC, RC\}$ that refers to either local check (LC) – authorization is only applied to nodes in context, i.e., text child, or recursive check (RC) – authorization is propagated to all descendants, respectively. In our model, all access controls are by default RC, complying with the XML semantics (i.e., projecting out a node `<foo/>` yields the entire subtree rooting at `<foo/>`). LC rules are converted to an RC rule by adding "`/text()`" to its *object* field.

3.2 XML to Relational Conversion

We model the X2R conversion algorithm (surveyed in Section 1) as follows:

Remark 1 *A relational to XML conversion method contains: (1) $C_D()$ to convert XML to relational data, (2) $C_Q()$ to convert XML query (XQuery or XPath) to SQL, and (3) C^{-1} to convert relational answer back to XML.* □

That is, $Q_R = C_Q(Q_X)$, $D_R = C_D(D_X)$, and $A_X = C^{-1}(A_R)$. From this, the process of "evaluating XML query on XRDB" can be modeled as the following Equation:

$$A_X = C^{-1}(A_R) = C^{-1}(Q_R\langle D_R\rangle) = C^{-1}(C_Q(Q_X)\langle C_D(D_X)\rangle) \tag{1}$$

Remark 2 *An X2R conversion algorithm is* lossless *iff: (1) (lossless node conversion) $\forall$ XML node $x_i, C_D^{-1}(C_D(x_i)) = x_i$; (2) (lossless node set decomposition) $\forall$ XML node set $\{x_1, ..., x_n\}, C_D^{-1}(C_D(\{x_1, ...x_n\})) = C_D^{-1}(\{C_D(x_1), ...C_D(x_n)\}) = \{C_D^{-1}(C_D(x_1)), ...C_D^{-1}(C_D(x_n))\}$; and (3) (exclusive conversion) $C_D(x_1) = C_D(x_2)$ only when $x_1 = x_2$, and $C_D^{-1}(r_1) = C_D^{-1}(r_2)$ only when $r_1 = r_2$.* □

Remark 3 *An X2R conversion algorithm is* correct *iff: $\forall$ query Q and $\forall$ document X, $Q\langle X\rangle = C^{-1}(Q_R\langle D_R\rangle) = C^{-1}(C_Q(Q_X)\langle C_D(X)\rangle)$.* □

Definition 2 (Soundness) *An X2R conversion algorithm A is **sound** iff it is* lossless *and* correct. □

In the remainder of the paper, we assume that the conversion algorithm being used is *sound*. Finally, we ignore the order of XML nodes when we compare the correctness, since this feature is not supported in most X2R conversion algorithms.

In the research community, most X2R conversion algorithms only support a subset of XQuery/XPath. For instance, many of them support parent-child (/), ancestor-descendant (//), wildcard (*) and predicates. Later, we will show that our approach does not alter the query or data conversion algorithm. Therefore, the query conversion totally depend on the X2R conversion algorithm; i.e. for a particular X2R conversion method X, our algorithm supports everything that X supports. For ease of understanding, we do not use predicates in the examples, however, we test queries with predicates in our experiments.

3.3 Deep set operators

In [36], we propose deep set operators for XML, as extensions of conventional set operators defined in XPath [2] and XQuery [7]. Here, we briefly revisit them, and later demonstrate how they are used to formalize XML access control.

Definition 3 (deep set operators) *The* `deep-union` *operator ($\overset{D}{\cup}$) takes two node sequences $\langle P\rangle$ and $\langle Q\rangle$ as operands, and returns a sequence of nodes (1) who exist as a node or as a descendant of the nodes in "either" operand sequences, and (2) whose parent does not satisfy (1). Formally, $\langle P\rangle \overset{D}{\cup} \langle Q\rangle = \{n|(n \in \langle P_d\rangle \vee n \in \langle Q_d\rangle)\ \wedge (n :: parent() \notin \langle P_d\rangle \wedge n :: parent() \notin \langle Q_d\rangle)\}$ where $P_d = P/descendant - or - self()$. The* `deep-intersect` *operator ($\overset{D}{\cap}$) takes two node sequences $\langle P\rangle$ and $\langle Q\rangle$ as operands, returns a sequence of nodes (1) who exist as a node or as a descendant of the nodes in "both" operand sequences, and (2) whose parent does not satisfy (1). Formally, $\langle P\rangle \overset{D}{\cap} \langle Q\rangle = \{n|(n \in \langle P_d\rangle \wedge n \in \langle Q_d\rangle)\ \wedge (n :: parent() \notin$*

$\langle P_d\rangle \vee n :: parent() \notin \langle Q_d\rangle)\}$. Finally, the `deep-except` *operator ($\overset{D}{-}$) takes two node sequences $\langle P\rangle$ and $\langle Q\rangle$ as operands, for each node $\langle p_i\rangle$ in $\langle P\rangle$, it remove $\langle p_i\rangle \overset{D}{\cap}_X \langle Q\rangle$ from the subtree of $\langle p_i\rangle$ and return the remaining.* □

For instance, (1) a query "`<b/>` ∪ `<a><b/></a>`" yields {`<b/>`, `<a><b/></a>`}, but "`<b/>` $\overset{D}{\cup}$ `<a><b/></a>`" yields only `<a><b/></a>`; (2) a query "`<b/>` ∩ `<a><b/></a>`" yields *Null*, but "`<b/>` $\overset{D}{\cap}$ `<a><b/></a>`" yields `<b/>`; and (3) a query "`<a><b/><c/></</>` − `<b/>`" yields `<a><b/><c/></</>`, but "`<a><b/><c/></</>` $\overset{D}{-}$ `<b/>`" yields only `<a/><c/></</>`.

3.4 XML access control in XDB and XRDB

The goal of XML access control is in a sense to ensure that only *safe answer* (SA) is returned to users. As shown in [34, 36], safe answer of a query Q includes all the XML nodes n such that: (1) n is part of $\langle Q\rangle$, (2) the access of n is granted by positive rules, and (3) the access of n is *not* denied by negative rules. That is, the precise semantics of "safe XML answer," SA_X, can be modeled as:

$$SA_X = \langle Q_X\rangle \overset{D}{\cap}_X (\langle ACR^+\rangle \overset{D}{-}_X \langle ACR^-\rangle) \tag{2}$$

$$= \langle Q_X\rangle \overset{D}{\cap}_X [(\langle R^+_{X_1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^+_{X_n}\rangle) \overset{D}{-}_X (\langle R^-_{X_1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^-_{X_m}\rangle)] \tag{3}$$

Equation 1 models how XML query is evaluated in XRDB to return XML answer, A_X. Similarly, Equation 2 models how only safe XML answers, SA_X, are returned. Therefore, we have:

Definition 4 (Secure XRDB) *An XRDB is called secure iff ∀ access control rule set ACR_X and ∀ query Q_X, it always returns the safe answer A_X:*

$$A_X \equiv SA_X \tag{4}$$

$$\iff \quad C^{-1}(\{C_Q(Q_X)\langle C_D(D_X)\rangle\}') \equiv \langle Q_X\rangle \overset{D}{\cap}_X [(\langle R^+_{X1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^+_{Xn}\rangle) \overset{D}{-}_X (\langle R^-_{X1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^-_{Xm}\rangle)] \tag{5}$$

□

Note that $\{C_Q(Q_X)\langle C_D(D_X)\rangle\}'$ indicates that access control mechanism intervenes in relational query processing. Our goal in this paper is to enforce XML access controls on RDBMS so that Equation 4 holds in *XRDB* setting. In this way, we need to develop *relational* access control rules and *relational* deep set operators that are equivalent to their corresponding *XML* access control rules and *XML* deep set operators.

4 XML Access Control in XRDB: The Theory

All entities of the 4-tuple XML access control model, except the *object* entity, can be directly adopted to relational access control model. Since the object entity is specified in XPath, we may apply an X2R algorithm $C(R_X.object)$ to get $R_R.object$. As a result, we can convert XML access control rules to "equivalent" relational access control rules:

$$\mathbf{R_R} = \{\mathbf{R_X.subject}, \mathbf{C(R_X.object)}, \mathbf{R_X.action}, \mathbf{R_X.sign}\}$$

However, the converted relational access control rules cannot be directly enforced in XRDB – naive enforcement of R_R may not generate correct answer, or even leads to security leakage, as demonstrated in the following example:

Example 1. Consider two rules of Figure 2(a) with XRDB(XRel) – that is, XML data are stored in RDBMS using XRel [59] conversion algorithm. The "element" table is partly shown in Figure 2(b). Rule 1 indicates that a user is allowed to access `<person>` nodes, i.e., nodes 293 and 299 (second and third record in Figure

1. {user, /site/people/person, read, +}
2. {user, /site/people/person/credicard, read, -}

(a)

DOCID	ELEMENTID	PATHID	ST	ED	
0	252	164	33996	36229	<people>
0	293	165	35592	35826	<person>
0	299	165	35832	36217	<person>
0	303	188	35989	36032	<creditcard>

(b)

```
SELECT  e0.DOCID, e0.ELEMENTID, e0.PATHID, e0.ST, e0.ED
FROM document d, element e0, pth p0
WHERE p0.pathexp LIKE '#%/people'
AND e0.pathid = p0.pathid AND d.docid = e0.docid
```

(c)

```
SELECT  e0.DOCID, e0.ELEMENTID, e0.PATHID, e0.ST, e0.ED
FROM document d, element e0, pth p0
WHERE p0.pathexp LIKE '#%/people#/person'
AND e0.pathid = p0.pathid AND d.docid = e0.docid
```

(d)

Figure 2: Examples of naive enforcement of "equivalent" relational rules leading to incorrect answer or security leakage.

2 (b)), and rule 2 indicates that a user cannot access `<credicard>` nodes, i.e., node 303. Naive enforcement of the rules will grant access to the record of element 293 and 299, and revoke the access to the element 303.

Now, a query "`//people`" is desired to yield an answer containing two `<person>` nodes, since they are the descendants (that are accessible) of the requested node. However, the converted SQL query (Figure 2(c)) yields no answer since access to the record of element 252 is prohibited by default. Moreover, for a query "`//person`", the converted SQL (Figure 2(d)) returns both `<person>` nodes to the user (with the unauthorized `<creditcard>` node). This is so because both records of element 293 and 299 are accessible, while revoking access to element 303 does not affect its ancestor. □

4.1 Object and Operation Equivalency

To solve the problem illustrated in Example 1, we propose our framework of supporting access control in XRDB systems. First, we define object and operation equivalency between XML and relational.

Definition 5 (Object Equivalency) *When both $R = C(X)$ and $X = C^{-1}(R)$ hold for XML node set X and relation R, we consider X and R equivalent w.r.t. C/C^{-1}, and denote as $X \equiv R$.* □

Note that, when we talk about equivalency of X and R, we have to predefine the context, i.e., select the X2R conversion algorithm C/C^{-1}. For a XML node set X, $C(X)$ may be different under different X2R conversion algorithms.

Definition 6 (Operation Equivalency) *Suppose $X_1 \equiv R_1$ and $X_2 \equiv R_2$ w.r.t. C/C^{-1}. Then, an XML operation OP_X is equivalent to a relational operation OP_R (denoted as $OP_X \equiv OP_R$) w.r.t. C and C^{-1} if:*

$$C(X_1\ OP_X\ X_2) = C(X_1)\ OP_R\ C(X_2) = R_1\ OP_R\ R_2$$ □

It is worth to note that XML operator takes two node sets as operands while its equivalent relational counterpart may not take two generic relations as operands. Rather, each operand is the equivalent objects of the corresponding XML node set, which may be tables, columns, records, or cells. Many relational operations require operands to be domain compatible (e.g., intersect, union etc.). We loosen this requirement for OP_R.

With the concept of operation equivalency, we can migrate all the exciting features of XML into XRDB by converting the atomic operations into equivalent relational counterparts. Our problem of secure XRDB is then articulated as follows:

Lemma 1. *In XRDB(C), if we can find relational operators, $\overset{D}{\cup}_R$, $\overset{D}{\cap}_R$, and $\overset{D}{-}_R$, which are equivalent to XML deep set operators, $\overset{D}{\cup}_X$, $\overset{D}{\cap}_X$, and $\overset{D}{-}_X$, w.r.t. the X2R conversion algorithm C, we are able to enforce XML access control in XRDB(C) such that Equation (4) always holds.* ■

Proof. First, according to the definition of object and operation equivalency, we are looking for $A_X = SA_X \equiv SA_R$, which means: $SA_R = C(SA_X)$. Since $\overset{D}{\cup}_R \equiv \overset{D}{\cup}_X$, $\overset{D}{\cap}_R \equiv \overset{D}{\cap}_X$ and $\overset{D}{-}_R \equiv \overset{D}{-}_X$ w.r.t. $C()$ and $C^{-1}()$, according to the definition of equivalent operation, we have:

$$\begin{aligned}
SA_R &= \mathbf{C}(\langle Q_X\rangle \overset{D}{\cap}_X [(\langle R^+_{X1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^+_{Xn}\rangle) \overset{D}{-}_X (\langle R^-_{X1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^-_{Xm}\rangle)]) \\
&= \mathbf{C}(\langle Q_X\rangle) \overset{D}{\cap}_R \mathbf{C}([(\langle R^+_{X1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^+_{Xn}\rangle) \overset{D}{-}_X (\langle R^-_{X1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^-_{Xm}\rangle)]) \\
&= \mathbf{C}(\langle Q_X\rangle) \overset{D}{\cap}_R [\mathbf{C}(\langle R^+_{X1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^+_{Xn}\rangle) \overset{D}{-}_R \mathbf{C}(\langle R^-_{X1}\rangle \overset{D}{\cup}_X ... \overset{D}{\cup}_X \langle R^-_{Xm}\rangle)] \\
&= \mathbf{C}(\langle Q_X\rangle) \overset{D}{\cap}_R [\mathbf{C}(\langle R^+_{X1}\rangle) \overset{D}{\cup}_R ... \overset{D}{\cup}_R \mathbf{C}(\langle R^+_{Xn}\rangle) \overset{D}{-}_R \mathbf{C}(\langle R^-_{X1}\rangle) \overset{D}{\cup}_R ... \overset{D}{\cup}_R \mathbf{C}(\langle R^-_{Xm}\rangle)]
\end{aligned}$$

Since we have $Q_X \equiv Q_R$, $R_{Xi} \equiv R_{Ri}$, therefore:

$$\begin{aligned}
&\mathbf{C}(\langle Q_X\rangle) \overset{D}{\cap}_R [\mathbf{C}(\langle R^+_{X1}\rangle) \overset{D}{\cup}_R ... \overset{D}{\cup}_R \mathbf{C}(\langle R^+_{Xn}\rangle) \overset{D}{-}_R \mathbf{C}(\langle R^-_{X1}\rangle) \overset{D}{\cup}_R ... \overset{D}{\cup}_R \mathbf{C}(\langle R^-_{Xm}\rangle)] \\
= \quad &\langle Q_R\rangle \overset{D}{\cap}_R [\langle R^+_{R1}\rangle \overset{D}{\cup}_R ... \overset{D}{\cup}_R \langle R^+_{Rn}\rangle \overset{D}{-}_R \langle R^-_{R1}\rangle \overset{D}{\cup}_R ... \overset{D}{\cup}_R \langle R^-_{Rm}\rangle]
\end{aligned}$$

As a conclusion, we are able to compose a SA_R within XRDB(C) such that $SA_R = C(SA_X)$. Since all steps above are reversible, we also have $SA_X = C^{-1}(SA_R)$. (q.e.d)

According to Lemma 1, in order to support access control in XRDB, we need to find equivalent operations such that $\overset{D}{\cup}_R \equiv \overset{D}{\cup}_X$, $\overset{D}{\cap}_R \equiv \overset{D}{\cap}_X$ and $\overset{D}{-}_R \equiv \overset{D}{-}_X$. Object and operation equivalency is based on specific X2R conversion method, therefore, the existence and representation of relational deep set operators also heavily depends on the particular conversion method C. Hereafter, we analyze the role of each deep set operator in Equation 2 and the existence of its equivalent relational counterpart under different X2R conversion algorithms.

4.2 On Equivalent Conversion of Deep Set Operators

Deep-union operator is used to integrate all the nodes that are defined accessible by individual positive rules (also, all the nodes that are defined inaccessible by individual negative rules), as shown in Equation 2. With the property $P \overset{D}{\cup} Q \subseteq P \cup Q$ [36], Remark 1 is rewritten into:

$$\langle P\rangle \overset{D}{\cup}_X \langle Q\rangle = \{n | (n \in \langle P\rangle \vee \ n \in \langle Q\rangle) \ \wedge (n \notin \langle P//*\rangle \wedge \ n \notin \langle Q//*\rangle)\} \tag{6}$$

Since n is an XML object and $\langle P\rangle$, $\langle Q\rangle$, $\langle P//*\rangle$, $\langle Q//*\rangle$ are all sets of XML objects, when C/C^{-1} is sound according to Definition 3, we have:

$$\begin{aligned}
C(\langle P\rangle \overset{D}{\cup}_X \langle Q\rangle) &= \{C(n) | [C(n) \in C(\langle P\rangle) \vee \ C(n) \in C(\langle Q\rangle)] \ \wedge [C(n) \notin C(\langle P//*\rangle) \wedge \ C(n) \notin C(\langle Q//*\rangle)]\} \\
&= \{r | [r \in C(\langle P\rangle) \vee \ r \in C(\langle Q\rangle)] \ \wedge [r \notin C(\langle P//*\rangle) \wedge \ r \notin C(\langle Q//*\rangle)]\}
\end{aligned}$$

Here, since we are to find $\overset{D}{\cup}_R$ such that $C(\langle P\rangle) \overset{D}{\cup}_R C(\langle Q\rangle) = C(\langle P\rangle \overset{D}{\cup}_X \langle Q\rangle)$:

$$C(\langle P\rangle) \overset{D}{\cup}_R C(\langle Q\rangle) = \{r | [r \in C(\langle P\rangle) \vee \ r \in C(\langle Q\rangle)] \ \wedge [r \notin C(\langle P//*\rangle) \wedge \ r \notin C(\langle Q//*\rangle)]\}$$

The condition of $[r \in C(\langle P\rangle) \vee r \in C(\langle Q\rangle)]$ is essentially the regular union. It is composed by set containment and Boolean operations. In XRDB, set containment check is supported when the soundness requirement in Definition 2 is fulfilled, and Boolean operation is generally supported in RDBMS. $[r \notin C(\langle P//*\rangle) \wedge \ r \notin C(\langle Q//*\rangle)]$ tends to support deep semantics. It requires XRDB to be able to identify $r \in C(\langle P//*\rangle)$ for any given relational object r and set $C(\langle P\rangle)$. This could be achieved in two ways: (1) directly calculate the containment relationship between r and all elements of $C(\langle P\rangle)$; or (2) enumerate all descendants of each element of $C(\langle P\rangle)$, and check if n is identical to any of them.

Lemma 2. *To implement deep-union operator in XRDB(C), the X2R conversion algorithm C should: (1) fulfil the soundness requirement stated in Definition 2; and (2) for given node n and node set $\langle P\rangle$, it should be able to check the containment condition of: $C(n) \in C(\langle P//*\rangle)$, e.g., it should recognize if $C(n)$ is a descendant of any node $C(p_i)$;* ∎

At present, all X2R conversion algorithms (we are aware of) fulfill the above conditions.

If a naive implementation of XRDB access control fails to support deep-union, instead, it implements regular union operator to arbitrarily collect "accessible nodes" and "forbidden nodes" into two sets; and this will not cause any security leak. This is because the positive set does not contain extra node, and the negative set does not miss any necessary node. However, this will cause duplicate nodes in the sets of accessible nodes, and then possibly in the answers to queries.

Deep-intersect operator is used to calculate the exact overlapping of user requested data and accessible data (i.e. $\langle Q\rangle$ and $\langle ACR\rangle$). Like deep-union, deep-intersect operator is defined as:

$$C(\langle P\rangle \overset{D}{\cap}_X \langle Q\rangle) = \{r | [r \in C(\langle P\rangle) \wedge \; r \in C(\langle Q\rangle)] \; \wedge [r \notin C(\langle P//*\rangle) \vee \; r \notin C(\langle Q//*\rangle)]\} \qquad (7)$$

Therefore, any X2R conversion algorithm that supports deep-union is able to support deep-intersect. That is, Lemma 2 could be directly extended to deep-intersect. On the other hand, if an XRDB fails to implement deep-intersect operator, instead, it uses regular intersection, as a result: (1) if a query asks for a descendant of an access-granted node, the whole node should be returned, but may be missed (i.e., mistakenly "jailed" by XRDB); (2) if a query asks for a node, where only part of its subtree is granted access, the access-granted descendants should be returned, but might be missed (such as shown in Example 1).

Example 2. In Example 1, A query "`//people`" yields `<people>` nodes, i.e. element 252, as shown in Figure 2 (b) record 1. Meanwhile, *object* field of access control rule 1, "`/site/people/person`", yields `<person>` nodes, i.e. element 293 and 299, as shown in Figure 2 (b) record 2 and 3. In XRel, each XML node is marked with a "*start*" and an "*end*" offset. Node containment is checked through a comparison of these offsets: for two node p_1 and p_2, if ($p_1.start < p_2.start$) and ($p_1.end > p_2.end$), then p_2 is an descendant of p_1. In our example, we can tell that node 293 and 299 are descendants of node 292. Therefore, "`//people` $\overset{D}{\cap}_X$ `//person`" will yield node 293 and node 299. Comparing with Example 1, "`//people` $\cap$ `//person`" yields *Null*. □

The operands of XML deep-union/intersect operators may contain different nodes. In RDBMS, where domain compatibility is strictly enforced, their relational equivalent counterpart might be domain incompatible (e.g. a row "intersect" a cell). This happens when schema-based X2R conversion methods (e.g. [16, 52]) are employed, where different XML nodes could be converted to tables, rows, or cells. To tackle this problem, we can employ new RDBMS techniques such as Oracle VPD (Virtual Private Database) to enable us to fine-control relational tables to create relational views with any group of cells from a table.

Deep-except is used to remove inaccessible nodes from the answer. Recall that, in our XML access control model, all nodes are inaccessible by default. When a user is prohibited to access a node, there is no need to write a negative rule (R^-) to revoke its accessibility unless the node is covered by positive rules (ACR^+). In this way, negative rules are only used to specify exceptions to global permissions, i.e. "revoke" access proposed by ACR^+. Deep except operator is used to enforce negative rules. Regarding whether deep except could be implemented in XRDB with X2R conversion algorithm C, it depends upon the characteristics of the negative rules contained in the access control policy. In particular, we distinguish two types of negative rules, as shown below.

Definition 7 (Node elimination vs. Descendant elimination negative rules) *A negative rule in ACR restricts user from access a set of nodes* $\{r_1^-, ...r_n^-\}$. *If* ***none*** *of the nodes is a descendant of the context node of a positive rule, i.e.:*

$$r_i^- \notin \langle R^+//*\rangle, \quad \forall r_i^- \in \{r_1^-, ...r_n^-\}; \forall \langle R^+\rangle \in \langle ACR^+\rangle$$

then it is called a ***node elimination (NE)*** *negative rule. Else, if one of the nodes is a descendant of the context node of a positive rule, i.e.:*

$$r_i^- \in \langle R^+//*\rangle, \quad \exists r_i^- \in \{r_1^-, ...r_n^-\}; \exists \langle R^+\rangle \in \langle ACR^+\rangle$$

it is called a ***descendant elimination (DE)*** *negative rule.* □

Intuitively, "*Node elimination*" negative rule removes context node from $\langle ACR^+\rangle$, while "*descendant elimination*" negative rule removes descendants from context node of $\langle ACR^+\rangle$.

For XML nodes covered by node elimination negative rules $\langle ACR_1^-\rangle$, deep-except operator directly removes them from $\langle ACR^+\rangle$, without breaking any XML nodes in the original document or creating any new nodes:

$$\langle ACR^+\rangle \overset{D}{-}_X \langle ACR_1^-\rangle = \{n | n \in \langle ACR^+\rangle \ \wedge \ n \notin \langle ACR_1^-\rangle\}$$

Essentially, this is the regular except semantics. In this way, in XRDB, we have,

$$C(\langle ACR^+\rangle) \overset{D}{-}_R C(\langle ACR_1^-\rangle) = C(\langle ACR^+\rangle \overset{D}{-}_X \langle ACR_1^-\rangle) = \{r | r \in C(\langle ACR^+\rangle) \ \wedge \ r \notin C(\langle ACR_1^-\rangle)\}$$

To support deep except operator for node elimination negative rules only, the conditions described in Lemma 2 still apply. However, it takes more burden to process descendant elimination negative rules, where real "deep" semantics is required. That is,

$$\langle ACR^+\rangle \overset{D}{-}_X \langle ACR_2^-\rangle = \{deepRemove(n, n \overset{D}{\cap}_X \langle ACR_2^-\rangle) | n \in \langle ACR^+\rangle\}$$

where $deepRemove(p, \langle Q\rangle)$ takes a node and a set of its descendants as operands, removes the descendants from the subtree of the node and return the remaining. This function may not be directly converted to relational.

Lemma 3. *When deep-except operator takes node specified by descendant elimination negative rules as the second operand, it is implemented through* $deepRemove()$ *operation. To implement deep-except operator that supports descendant elimination negative rules in XRDB(C), the X2R conversion algorithm X should: (1) fully satisfy Lemma 2; and (2) for any node* n_1 *and its descendant* n_2*,* $C(n_2)$ *should be part of* $C(n_1)$*; and in the reverse conversion of* $n_1 = C^{-1}(C(n_1))$*, node* n_2 *in the subtree is entirely converted from* $C(n_2)$*.*■

Example 3. For instance, in Example 1, Rule 2 is a descendant elimination negative rule since it revoke access towards descendants of Rule 1's context node (`<person>`).

In XRDB(XRel) [59], descendants are converted to independent records that are stand alone from ancestors. As shown in Figure 2(b), `<creditcard>` node is converted to an individual record (i.e. $elementID = 303$), which is independent from it ancestor `<person>`. To reconstruct a `<person>` node, $C^{-1}_{XRel}()$ only takes the record with $elementID = 293$ (ancestor node) and returns a full person node. Although the descendant node `<creditcard>` is included in the answer, the record $elementID = 303$ is not touched by $C^{-1}_{XRel}()$. In this way, XRel violates condition (2) of Lemma 3, so that we cannot directly implement deep-except operator to support descendant elimination negative rules. When user requests for "`//person`", we are not able to revoke access towards `<creditcard>` child, unless we modify the relational data to the following record for $C^{-1}_{XRel}()$:

DOCID	ELEMENTID	pathID	st	ed
0	NULL	NULL	35832	35988
0	NULL	NULL	36033	36217

However, this is not directly supported in relational algebra or any existing RDBMS.

In Shared-Inlining [52] approach, `<person>` nodes are translated into a table, with each row representing a person, and `<creditcard>` nodes are stored in one of the columns, "`person_credicard`". The relational schema is [33]:

```
Person(PersonId, ParentId, Person, Person_address, Person_address_city, Person_address_country,
Person_address_province, Person_address_street, Person_address_zipcode, Person_creditcard, ......)
```

Here, the ancestor-descendant relationship is kept such that each row represents a "`person`" node, and each cell represents a child node. When $C^{-1}_{Inlining}()$ is called to reconstruct `<person>` nodes, the textual contents of `<credicard>` descendants are retrieved from "`person_credicard`" column. Therefore, to obtain `//person`$\overset{D}{-}_X$`//creditcard`, we just mask "`person_creditcard`" column in the table; and the reconstructed XML tree of "`person`" node will not have corresponding child, i.e., "`creditcard`" node is removed from the XML answer. □

As a conclusion, there is a “semantic gap” between XML and relational data models. XML features a tree structure, where nodes are hierarchically nested, while, relational model only defines a two-dimensional structure. This fundamental difference makes us unable to directly maintain all structural information in X2R conversion. Some conversion approaches store each XML node independently, such as XRel showed above, where descendants are not utilized when converting an ancestor node back to XML. This is different from XML data model, in which ancestor node inherently includes descendants. For those approaches like XRel, descendant elimination negative rules could not be directly enforced through deep-except operator since we have difficulties sweeping off descendants from given node(s). Fortunately, for many other X2R conversion approaches (like shared-inlining), we are able to implement deep-except operator, and then directly enforce descendant elimination negative rules. Moreover, in those approaches where descendant elimination negative rules are not directly supported, we can still use post-processing filtering methods to remove access denied contents from the reconstructed XML answer (more details are provided in the next section).

5 XML Access Control Enforcement in XRDB

In the previous sections we show how XML access control semantics could be converted into relational model to be used in XRDB. However, in real world applications, existing XML access control approaches do not exactly implement the basic semantics shown in Equation 3. A general framework is proposed in [34] to capture different XML access control approaches (as show in row 1 of Figure 3). Now, we extend this framework into XRDB.

As shown in Figures 1 and 3, similar to the framework in native XML databases, there could be three categories of XML access control enforcement mechanisms in XRDB: (1) view-based approach (① ④ in Figures 1 and 3); (2) pre-processing approach (② ⑤ in Figures 1 and 3); and (3) post-processing approach (③ ⑥ in Figures 1 and 3). These approaches enforce access control policy on the document, query and answer, respectively. In this section, we articulate the algebra of these approaches using deep set operators. Then, we briefly describe how they could be converted to their relational counter parts in XRDB.

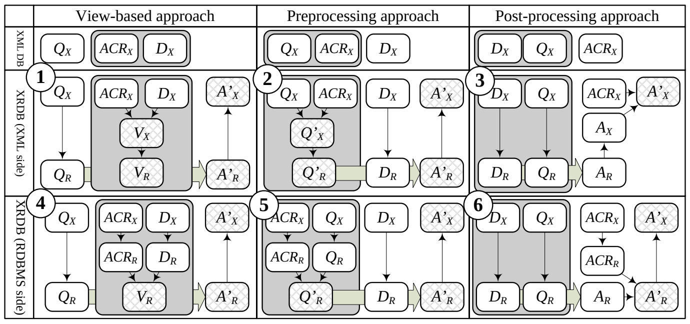


Figure 3: Access control enforcement approaches in XML DB and XRDB.

5.1 View-based approach

When access control is first enforced on XML documents to create *views*, it is the traditional view-based approach. In this model, XML view V_X (or *safe document SD*) is constructed to capture:

$$V_X = [(\langle R^+_{X1}\rangle \overset{D}{\cup}_X \dots \overset{D}{\cup}_X \langle R^+_n\rangle) \overset{D}{-}_X (\langle R^-_1\rangle \overset{D}{\cup}_X \dots \overset{D}{\cup}_X \langle R^-_m\rangle)]$$

And query is evaluated against the view

$$SA = Q\langle V_X\rangle = Q[(\langle R^+_{X1}\rangle \overset{D}{\cup}_X \dots \overset{D}{\cup}_X \langle R^+_n\rangle) \overset{D}{-}_X (\langle R^-_1\rangle \overset{D}{\cup}_X \dots \overset{D}{\cup}_X \langle R^-_m\rangle)]$$

To convert this approach into XRDB, a straightforward approach is to convert each XML view V_X into relational view $V_R = C(V_X)$, as shown in ① of Figure 3:

$$SA = C^{-1}(Q_R\langle V_R\rangle) = C^{-1}(Q_R\langle C(V_X)\rangle)$$

However, this approach suffers from several drawbacks: (1) since views for each role should be materialized, the storage requirement is substantial; and (2) each relational view V_R is independently stored, without any connection to D_R, thus synchronization is difficult to achieve, if not impossible.

Another solution is to employ view support from RDBMS to enforce access control on the relational side of XRDB, as show in ④ of Figure 3:

$$SA = C^{-1}(Q\langle V_R\rangle) = C^{-1}(Q\langle (C(\langle R^+_{X1}\rangle)\ \overset{D}{\cup}_R \dots \overset{D}{\cup}_R\ C(\langle R^+_n\rangle))\ \overset{D}{-}_X\ (C(\langle R^-_1\rangle)\overset{D}{\cup}_R \dots \overset{D}{\cup}_R C(\langle R^-_m\rangle)))\rangle) \quad (8)$$

In implementation of view based approaches, there are three factors to be considered:
Construction of V_R: This issue includes two aspects: (1) the constructed V_R should capture the exact content of access control allowed data, i.e. $V_R \equiv V_X$, as we described in Section 4; and (2) this V_R should be legit to the underlying RDBMS. According to Lemma 3, some X2R conversion algorithms cannot directly support descendant elimination negative rules. Therefore, in the corresponding XRDB systems, we cannot directly employ relational view based approaches to enforce descendant elimination negative rules.
Evaluation of Q_R: Comparing Equation 8 with Equation 4, note that, in Equation 8 deep-intersect operator is replaced by the query evaluation process. Then we need to consider whether query evaluation process conducts the deep intersect semantics. In some XRDB such as XRDB(XRel), the original query translation and evaluation process only conducts intersect semantics, as shown in Example 1. Therefore, we cannot directly employ view based approach, or special treatment is required to implement the deep-intersect semantics.
Reconstruction of SA_X: We still need to mention that, no matter how we tailor D_R into V_R, we need to ensure that the relational answers from V_R is legit to $C^{-1}()$.

5.2 Pre-processing approach

In preprocessing model, *safe query SQ* is constructed as:

$$SQ_X = Q_X \overset{D}{\cap}_X [(\ R^+_{X1}\ \overset{D}{\cup}_X \dots \overset{D}{\cup}_X\ R^+_{Xn})\ \overset{D}{-}_X\ (R^-_{X1} \overset{D}{\cup}_X \dots \overset{D}{\cup}_X R^-_{Xm})]$$

Safe answer is yielded by evaluating safe query against the original document: $SA_X = SQ_X\langle D_X\rangle$. To extend this approach to XRDB, we have two methods:

(1)XML Query Rewriting: as shown in ② in Figure 3, this approach is to convert the safe XML query into SQL, and follow the regular XRDB query evaluation process:

$$SA_X = C^{-1}(SQ_R\langle D_R\rangle) = C^{-1}(C(SQ_X)\langle D_R\rangle)$$

In this approach, we can directly adopt the preprocessing of XML access control mechanisms, such as [35], to generate SA_X. We have to mention that, the generated SA_X should be legit to the X2R conversion algorithm, e.g. most X2R conversion algorithms can only process XPath queries, thus SA_X should only include XPath. However, this requirement may exceed the capability of XML access control mechanisms since XML deep set operators are implemented as user defined functions of XQuery, which is not supported in some X2R conversion algorithms. Therefore, when the safe XML query cannot be expressed as XPath, one cannot directly adopt XML query rewritten approach to enforce access control.

(2)Relational Query Rewriting: As shown in ⑤ in Figure 3, this approach follows regular XRDB query evaluation process to convert user XML query Q_X into SQL Q_R. Then, we conduct query rewriting on Q_R to generate safe query SQ_R:

$$SQ_R = Q_R \overset{D}{\cap}_R [(\ R^+_{R1}\ \overset{D}{\cup}_R \dots \overset{D}{\cup}_R\ R^+_{Rn})\ \overset{D}{-}_X\ (R^-_{R1} \overset{D}{\cup}_X \dots \overset{D}{\cup}_X R^-_{Rm})]$$

We present two approaches to implement this. First, develop an external query rewriting process, which sits as a middle-ware between X2R query conversion and relational query evaluation. Since we have clearly defined relational deep set operators, the implementation is straightforward, although the queries might be complicated. For instance,

```
SELECT   docID, pathid, elementID, st, ed FROM element
WHERE  (elementID IN                                               r∈C(//people::self-or-descendant())
              ( SELECT e0.elementID FROM document d, element e0, pth p0
               WHERE   p0.pathexp LIKE '#%/people%'  AND e0.pathid = p0.pathid AND d.docid = e0.docid )
       AND elementID IN                                            r∈C(//name::self-or-descendant())
              (SELECT e0.elementID FROM document d, element e0, pth p0
               WHERE p0.pathexp LIKE '#%/name%' AND e0.pathid = p0.pathid AND d.docid = e0.docid ))
       AND ( NOT elementID IN                                      r∈C(//people//*)
              (SELECT e0.elementID FROM document d, element e0, pth p0
               WHERE p0.pathexp LIKE '#%/people#/%' AND e0.pathid = p0.pathid AND d.docid = e0.docid)
       OR NOT elementID IN                                         r∈C(//name//*)
              (SELECT e0.elementID FROM document d, element e0, pth p0
               WHERE p0.pathexp LIKE '#%/name#/%' AND e0.pathid = p0.pathid AND d.docid = e0.docid))
```

(a) SQL query for: $//people \overset{D}{\cap} //name$

```
SELECT   e0.docID, e0.elementID, e0.st, e0.ed FROM demo.document d, demo.element e0, demo.pth p0
WHERE   (( p0.pathexp LIKE '#%/people%'  AND e0.pathid = p0.pathid AND d.docid = e0.docid)
AND        (p0.pathexp LIKE '#%/name%' AND e0.pathid = p0.pathid AND d.docid = e0.docid ))
AND        ((NOT p0.pathexp LIKE '#%/people#/%' AND e0.pathid = p0.pathid AND d.docid = e0.docid)
OR         (NOT p0.pathexp LIKE '#%/name#/%' AND e0.pathid = p0.pathid AND d.docid = e0.docid))
```

(b) optimized SQL query

Figure 4: Enforcing XML access control via external pre-processing

Example 4. Let use revisit the previous examples: we manage XMark document in XRDB(XRel). Suppose we have access control rule `(user, //people, read, +)`, and user submits query `//name`. Figure 4(a) shows the relational query for $C(//people) \overset{D}{\cap}_R C(//name)$, which is implemented according to the definition in Equation 7 (we marked up all the sub-queries). Moreover, this query could be further optimized, as shown in Figure 4(b). □

Second method is to use Oracle VPD. Oracle version 8.1.5 introduces a new security feature supporting non-view-based fine-grained access control, namely *Row Level Security* or *Virtual Private Database*. It allows users to control accessibility towards row/cell level. In VPD, to restrict users' access to rows, a policy function is defined to generate additional predicates and attach them to the WHERE clause of the user query. Moreover, VPD allows user to "mask" individual cells to support cell level access control. Other access control mechanism through SQL rewriting or relational views can only work on relations. However, with VPD, we are able to tailor relational data into any shape we want.

To utilize VPD for access control in XRDB, we first construct relational predicates from the converted relational access control rules ACR_R, then define a VPD policy to enforce the predicates on converted SQL queries. Moreover, cell level access control capability of VPD is of special importance to XRDB systems that use schema-based X2R conversion algorithm, such as Inlining. In those XRDB systems, XML nodes are converted to different types of relational objects: tables, rows and cells. In this way, $\langle ACR \rangle$ may not be conventional relations, e.g. it could be arbitrary combinations of columns, rows and/or individual cells.

5.3 Post-processing based approach

In native XML DB, access control through post-processing described as:

$$SA_X = ACR\langle A_X\rangle = ACR\langle Q_X\langle D_X\rangle\rangle = [(\ R^+_{X1} \overset{D}{\cup}_X ... \overset{D}{\cup}_X \ R^+_{Xn}) \overset{D}{-}_X \ (R^-_{X1} \overset{D}{\cup}_X ... \overset{D}{\cup}_X R^-_{Xm})]\langle Q_X\langle D_X\rangle\rangle$$

In XRDB, this approach could be conducted through: (1) XML answer filtering (③ in Figure 3); or (2) relational answer filtering (⑥ in Figure 3). (1) is similar to the postprocessing approach described in [34], while (2) evaluates relational query Q_R to obtain unsafe relational answer, and process ACR_R against the answers:

$$SA_X = C^{-1}(SA_R) = C^{-1}(ACR_R\langle A_R\rangle) = C^{-1}(ACR_R\langle Q_R\langle D_R\rangle\rangle)$$

However, the post-processing filters often require the intermediate answers ($\langle A_R\rangle$ or $\langle A_X\rangle$) to retain additional information of the original paths for ACR to operate on. As an example of this approach, [8] check streaming XML data against both query and ACR at the same time. Since it works in the streaming data environment, full paths are retained. As an counter example, let us look at an XRDB in information pull model. Suppose a user asks for "`//name`", but she is only authorized to access person names, not item names. To enforce access control on $\langle A_R \rangle$ or $\langle A_X \rangle$, we need to be able to distinguish these two types of `<name>` nodes, i.e. recognize the original full path. Unfortunately, as designed in most X2R conversion algorithms, the intermediate answer A_R or A_X does not contain such information. Therefore, postprocessing approaches are not suitable for all applications.

5.4 Descendant elimination negative rules

As we described in Section 3, enforcing descendant elimination negative rules needs the conversion of deep-except operator. Due to the semantic gap between XML and Relational data models, this may not be feasible in all XRDB systems. E.g., in XRel, to enforce descendant elimination negative rules, we need to block access to a descendant node in the `element` table. However, users are still able to retrieve the whole ancestor node (including the "blocked" descendant) since it is stored as an independent record. To avoid security leak, we need to manage these conversion algorithms with extra treatment: an external post processing to enforce DE access control rules.

Let us revisit Example 3 again. Upon user query "`//person`", elements 293 and 299 shown in Figure 2 are included in the relational answer. After the reverse conversion, segments with offsets (35592, 35826) and (35832, 36217) from the XML documents are returned to the user. However, rule 2 restricts access to `//person/creditcard` nodes, thus this answer is not safe. To remove the restricted node from the answer, we first request $\langle Q_R \rangle \overset{D}{\cap}_R \langle ACR_R^- \rangle$ from RDBMS, to yield element 303. Then we remove this segment, i.e. (35989, 36032) from the XML answer.

6 Experimental Validation

To show that the proposed theory and implementations are practical yet efficient, we show our preliminary experimental results.

6.1 Settings

An XML document with 8517 nodes are generated by XMark [50], mimicking online auction scenario. Part of its schema structure is shown in Figure 1. We use XRDB(XRel) [59][2], with Oracle 10g as underlying RDBMS; i.e. we convert XML document into relations using XRel, and manage them in Oracle 10g.

We design five (5) roles, abbreviated as A (administrator), M (manager), RU (registered user), S (sales) and U (unregistered user), respectively. Roles have different levels of accessibility, e.g. U is able to access 5% of total nodes, RU is able to access 40%, and A could access all.

According to Lemma 3, XRDB(XRel) cannot directly handle descendant elimination negative rules, thus we only have positive and node elimination negative rules. As a reference, we also test situations where no access control is enforced – user could access everything.

As we described before, the types of queries that we support totally depend on the X2R conversion algorithm. XRel supports a subset of XPath, with parent-child (/), ancestor-descendant (//) axes, wildcards (*) and predicates. We generate four groups of synthetic XPath queries, each has a different setting of wildcards and predicates.

6.2 Experimental Results

In the XRDB(XRel) environment described above, we convert all access control rules into relational, and enforce them through views and VPD. For a comparison, we also enforce same rule sets on the same XML document in native XML environment. We enforce XML access control rules using QFilter [35], and answer XML queries using Galax. In all the experiments, we use the *query processing time* as an evaluation metric.

[2]We choose XRel because of its available implementations of both Query and Data convertor.

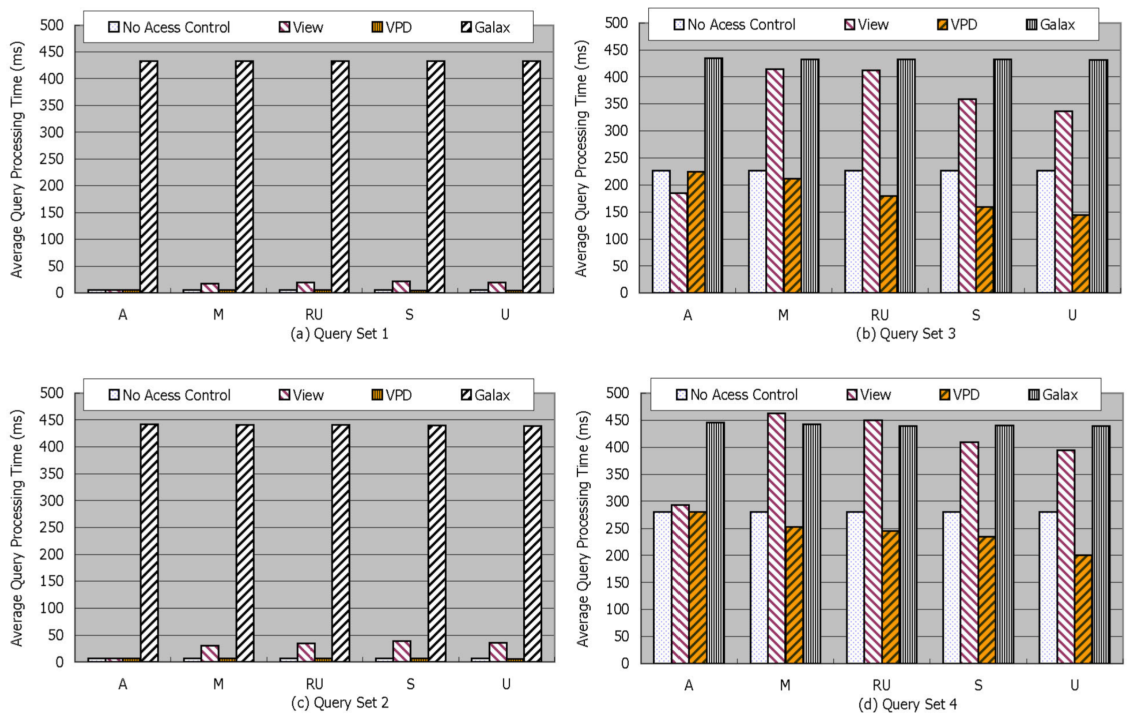


Figure 5: Query processing time for four sets of queries.

Figure 5 shows the result of our experimentation. Comparing both view-based and VPD-based approaches with the reference (no security enforcement), our approaches do not add much overhead for fine-grained access control. Meanwhile, the size of accessible data tends to get smaller after security enforcement. Therefore, querying on smaller set of records is even faster than that on no-security case. XRDB query processing speed is significantly slower for Query Sets 3 and 4. This is because the XML queries have predicates, and they are converted to nested SQL queries under XRel.

With access control enforced, performance of XML querying in XRDB systems or native XML database systems (with QFilter security enforcement) is similar. Note that comparison with Galax is not perfectly fair since Galax is just an XQuery implementation, and does not have storage management or cache. Therefore, Galax take more time to load XML documents from disk to memory. It is just used as a reference.

7 Conclusion

In this paper, we propose a generic analysis to the access control problem in XRDB. We first analyze XML control models to propose a formal description of XML access control using deep set operators. Then we articulate the problem of XML access control in XRDB as essentially the problem of XML/Relational object and operation equivalency and conversion. We show that, equivalent counterparts of deep set operators in relational model are needed to fully implement XML access control in XRDB. We analyze the definition and semantics of each operator, and show how they can be converted to XRDB through two lemmas. Although detailed conversion implementation is connected with the specific X2R conversion algorithm used in XRDB, we propose an algebraic description of these operators.

Moreover, we study possible implementations of XML access control in XRDB. We categorize them into three approaches, and formally describe the semantics of each approach using deep set operators. We also discuss the features and considerations of each approach. Finally, we show the validity of our approaches using experiment results.

We have carefully explored the problem space, proposed theocratical solutions, and discussed implementation approaches. However, there are still open questions in XRDB access control, especially the questions connected with particular implementation methods. E.g. how to enforce XML access control with minimal overhead and alternation upon underlying RDBMS? We leave these as our our future research topics.

References

[1] D. Barbosa, J. Freire, and A. O. Mendelzon. “Designing Information-preserving Mapping Schemes for XML”. In *VLDB*, pages 109–120, Trondheim, Norway, 2005.

[2] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernndez, M. Kay, J. Robie, and J. Simeon. “XML Path Language (XPath) 2.0”. W3C Working Draft, Nov. 2003. http://www.w3.org/TR/xpath20.

[3] E. Bertino and E. Ferrari. “Secure and Selective Dissemination of XML Documents”. *ACM Trans. on Information and System Security (TISSEC)*, 5(3):290–331, Aug. 2002.

[4] Elisa Bertino, Silvana Castano, and Elena Ferrari. Securing xml documents with author-x. *IEEE Internet Computing*, 5(3):21–31, 2001.

[5] Elisa Bertino, Elena Ferrari, and Loredana Parasiliti Provenza. Signature and access control policies for xml documents. In Einar Snekkenes and Dieter Gollmann, editors, *ESORICS*, volume 2808 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2003.

[6] Kevin Beyer, Fatma Ozcan, Sundar Saiprasad, and Bert Van der Linden. DB2/XML: designing for evolution. In *SIGMOD ’05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 948–952, New York, NY, USA, 2005. ACM Press.

[7] S. Boag, D. Chamberlin, M. F. Fernndez, D. Florescu, J. Robie, and J. Simeon. “XQuery 1.0: An XML Query Language”. W3C Working Draft, Nov. 2003. http://www.w3.org/TR/xquery.

[8] L. Bouganim, F. D. Ngoc, and P. Pucheral. “Client-Based Access Control Management for XML Documents”. In *VLDB*, Toronto, Canada, 2004.

[9] T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds). “Extensible Markup Language (XML) 1.0 (2nd Ed.)”. W3C Recommendation, Oct. 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[10] Barbara Carminati, Elena Ferrari, and Elisa Bertino. Securing xml data in third-party distribution systems. In *CIKM ’05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 99–106, New York, NY, USA, 2005. ACM Press.

[11] S. Cho, S. Amer-Yahia, L. V.S. Lakshmanan, and D. Srivastava. “Optimizing the Secure Evaluation of Twig Queries”. In *VLDB*, Hong Kong, China, Aug. 2002.

[12] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. “A Fine-Grained Access Control System for XML Documents”. *ACM Trans. on Information and System Security (TISSEC)*, 5(2):169–202, May 2002.

[13] E. Damiani, S. De Capitani Di Vimercati, S. Paraboschi, and P. Samarati. “Design and Implementation of an Access Control Processor for XML Documents”. *Computer Networks*, 33(6):59–75, 2000.

[14] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Securing xml documents. In Carlo Zaniolo, Peter C. Lockemann, Marc H. Scholl, and Torsten Grust, editors, *EDBT*, volume 1777 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2000.

[15] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. Xml access control systems: A component-based approach. In *DBSec 00: Proceedings of the IFIP TC11/ WG11.3 Fourteenth Annual Working Conference on Database Security*, pages 39–50, Deventer, The Netherlands, The Netherlands, 2001. Kluwer, B.V.

[16] A. Deutsch, M. F. Fernandez, and D. Suciu. “Storing Semistructured Data with STORED”. In *ACM SIGMOD*, Philadephia, PA, Jun. 1998.

[17] Wenfei Fan, Chee-Yong Chan, and Minos Garofalakis. Secure xml querying with security views. In *SIGMOD ’04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 587–598, New York, NY, USA, 2004. ACM Press.

[18] E. Fernandez, E. Gudes, and H. Song. “A Model of Evaluation and Administration of Security in Object-Oriented Databases”. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 6(2):275–292, 1994.

[19] B Finance, S Medjdoub, and P Pucheral. The case for access control on xml relationships. In *CIKM ’05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 107–114, New York, NY, USA, 2005. ACM Press.

[20] D. Florescu and D. Kossmann. “Storing and Querying XML Data Using an RDBMS”. *IEEE Data Eng. Bulletin*, 22(3):27–34, Sep. 1999.

[21] Irini Fundulaki and Maarten Marx. Specifying access control policies for xml documents with xpath. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 61–69, New York, NY, USA, 2004. ACM Press.

[22] Alban Gabillon and Emmanuel Bruno. Regulating access to xml documents. In *DBSec 01: Proceedings of the fifteenth annual working conference on Database and application security*, pages 299–314, Norwell, MA, USA, 2002. Kluwer Academic Publishers.

[23] S. Godik and T. Moses (Eds). "eXtensible Access Control Markup Language (XACML) Version 1.0". OASIS Specification Set, Feb. 2003. http://www.oasis-open.org/committees/xacml/repository/.

[24] P. P. Griffiths and B. W. Wade. "An Authorization Mechanism for a Relational Database System". *ACM Trans. on Database Systems (TODS)*, 1(3):242–255, Sep. 1976.

[25] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. "Flexible Support for Multiple Access Control Policies". *ACM Trans. on Database Systems (TODS)*, 26(2):214–260, Jun. 2001.

[26] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. "A Unified Framework for Enforcing Multiple Access Control Policies". In *ACM SIGMOD*, pages 474–485, May 1997.

[27] S. Jajodia and R. Sandhu. "Toward a Multilevel Secure Relational Data Model". In *ACM SIGMOD*, May 1990.

[28] Mingfei Jiang and Ada Wai-Chee Fu. Integration and efficient lookup of compressed xml accessibility maps. *IEEE Transactions on Knowledge and Data Engineering*, 17(7):939–953, 2005.

[29] M. Kudo and S. Hada. "XML Document Security Based on Provisional Authorization". In *ACM Conf. on Computer and Communications Security (CCS)*, 2000.

[30] Gabriel Kuper, Fabio Massacci, and Nataliya Rassadko. Generalized xml security views. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 77–84, New York, NY, USA, 2005. ACM Press.

[31] D. Lee and W. W. Chu. "Constraints-preserving Transformation from XML Document Type Definition to Relational Schema". In *Int'l Conf. on Conceptual Modeling (ER)*, pages 323–338, Salt Lake City, UT, Oct. 2000.

[32] D. Lee, W.-C. Lee, and P. Liu. "Supporting XML Security Models using Relational Databases: A Vision". In *XML Database Symp. (XSym)*, Berlin, Germany, Sep. 2003.

[33] H. Lu et al. What makes the differences: benchmarking xml database implementations. *ACM Trans. on Internet Technology (TOIT)*, 5(1):154–194, 2005.

[34] B. Luo, D. Lee, W.-C. Lee, and P. Liu. "A Flexible Framework for Architecting XML Access Control Enforcement Mechanisms". In *VLDB Workshop on Secure Data Management in a Connected World (SDM)*, Toronto, Canada, Aug. 2004.

[35] B. Luo, D. Lee, W.-C. Lee, and P. Liu. "QFilter: Fine-Grained Run-Time XML Access Control via NFA-based Query Rewriting". In *ACM CIKM*, Washington D.C., USA, Nov. 2004.

[36] B. Luo, D. Lee, W.-C. Lee, and P. Liu. Deep set operators for xquery. In *ACM SIGMOD Workshop on XQuery Implementation, Experience and Perspectives (XIME-P)*, Baltimore, MD, USA., 2005.

[37] Sriram Mohan, Jonathan Klinginsmith, Arijit Sengupta, and Yuqing Wu. Acxess - access control for xml with enhanced security specifications. In *ICDE '06: Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, page 171, Washington, DC, USA, 2006. IEEE Computer Society.

[38] Sriram Mohan, Arijit Sengupta, and Yuqing Wu. Access control for xml: a dynamic query rewriting approach. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 251–252, New York, NY, USA, 2005. ACM Press.

[39] Sriram Mohan and Yuqing Wu. Ipac: an interactive approach to access control for semi-structured data. In *VLDB'2006: Proceedings of the 32nd international conference on Very large data bases*, pages 1147–1150. VLDB Endowment, 2006.

[40] M. Murata, A. Tozawa, and M. Kudo. "XML Access Control using Static Analysis". In *ACM Conf. on Computer and Communications Security (CCS)*, Washington D.C., 2003.

[41] Makoto Murata, Akihiko Tozawa, Michiharu Kudo, and Satoshi Hada. Xml access control using static analysis. *ACM Trans. Inf. Syst. Secur.*, 9(3):292–324, 2006.

[42] Ravi Murthy, Zhen Hua Liu, Muralidhar Krishnaprasad, Sivasankaran Chandrasekar, Anh-Tuan Tran, Eric Sedlar, Daniela Florescu, Susan Kotsovolos, Nipun Agarwal, Vikas Arora, and Viswanathan Krishnamurthy. Towards an enterprise XML architecture. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 953–957, New York, NY, USA, 2005. ACM Press.

[43] Matthias Nicola and Bert van der Linden. Native XML support in DB2 universal database. In *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, pages 1164–1174. VLDB Endowment, 2005.

[44] Naizhen Qi and Michiharu Kudo. Access-condition-table-driven access control for xml databases. In Pierangela Samarati, Peter Y. A. Ryan, Dieter Gollmann, and Refik Molva, editors, *ESORICS*, volume 3193 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2004.

[45] Naizhen Qi and Michiharu Kudo. Xml access control with policy matching tree. In *ESORICS 2005, 10th European Symposium on Research in Computer Security*, pages 3–23, 2005.

[46] Naizhen Qi, Michiharu Kudo, Jussi Myllymaki, and Hamid Pirahesh. A function-based access control model for xml databases. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 115–122, New York, NY, USA, 2005. ACM Press.

[47] Michael Rys. XML and relational database management systems: inside Microsoft SQL Server 2005. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 958–962, New York, NY, USA, 2005. ACM Press.

[48] P. Samarati, E. Bertino, and S. Jajodia. "An Authorization Model for a Distributed Hypertext System". *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 8(4):555–562, 1996.

[49] R. Sandhu and F. Chen. "The Multilevel Relational (MLR) Data Model". *ACM Trans. on Information and System Security (TISSEC)*, 1(1), 1998.

[50] A. R. Schmidt, F. Waas, M. L. Kersten, D. Florescu, I. Manolescu, M. J. Carey, and R. Busse. "The XML Benchmark Project". Technical Report INS-R0103, CWI, April 2001.

[51] H. Schoning. Tamino - a dbms designed for xml. In *IEEE ICDE*, pages 149–154, Washington, DC, USA, 2001.

[52] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. "Relational Databases for Querying XML Documents: Limitations and Opportunities". In *VLDB*, Edinburgh, Scotland, Sep. 1999.

[53] J. Simeon and M. Fernandez. "Galax V 0.3.5", Jan. 2004. http://db.bell-labs.com/galax/.

[54] Andrei Stoica and Csilla Farkas. Secure xml views. In Ehud Gudes and Sujeet Shenoi, editors, *DBSec*, volume 256 of *IFIP Conference Proceedings*, pages 133–146. Kluwer, 2002.

[55] K.-L. Tan, M. L. Lee, and Y. Wang. "Access Control of XML Documents in Relational Database Systems". In *Int'l Conf. on Internet Computing (IC)*, Las Vegas, NV, Jun. 2001.

[56] Jingzhu Wang and Sylvia L. Osborn. A role-based approach to access control for xml databases. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 70–77, New York, NY, USA, 2004. ACM Press.

[57] M. Winslett, K. Smith, and X. Qian. "Formal Query Languages for Secure Relational Databases". *ACM Trans. on Database Systems (TODS)*, 19(4):626–662, 1994.

[58] Yan Xiao, Bo Luo, and Dongwond Lee. "Security-Conscious XML Indexing". In *Int'l Conf. on Database Systems for Advanced Applications (DASFAA)*, Bangkok, Thailand, 2007.

[59] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. "XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases". *ACM Trans. on Internet Technology (TOIT)*, 1(2):110–141, Nov. 2001.

[60] T. Yu, D. Srivastava, L. V.S. Lakshmanan, and H. V. Jagadish. "Compressed Accessibility Map: Efficient Access Control for XML". In *VLDB*, Hong Kong, China, Aug. 2002.