

Generating Rules for Incremental Maintenance of XML View of Relational Data

Vânia Maria Ponte Vidal
Departamento de Computação
Universidade Federal do Ceará
Fortaleza, CE – Brasil
vvidal@lia.ufc.br

Marco Antonio Casanova
Departamento de Informática
PUC-Rio
Rio de Janeiro, RJ - Brasil
casanova@fplf.org.br

Valdiana da Silva Araujo
Departamento de Computação
Universidade Federal do Ceará
Fortaleza, CE – Brasil
valdiana@lia.ufc.br

ABSTRACT

This work addresses the problem of incremental maintenance of XML views defined on top of relational data. In order to incrementally maintain a XML view, event-condition-rules should be specified for the relational source. Such rules are responsible for correctly modifying the XML view content in order to reflect changes made to the base source.

This work proposes an approach where incremental view maintenance rules are derived from view correspondence assertions, which specify relationships between the view schema and the base source schema. The adoption of correspondence assertions also facilitates the task of formally proving that the derived rules correctly maintain the view.

Categories and Subject Descriptors

H.2.3 [Database Management]: systems – *Relational databases.*

General Terms

Algorithms, Management, Performance, Design.

Keywords

XML, Incremental View Maintenance, Relational Database.

1. INTRODUCTION

The eXtended Markup Language (XML) has quickly emerged as the universal format for publishing and exchanging data on the Web. As a result, data sources often export XML views over base data [3,6]. The exported view can be either virtual or materialized. Materialized views improve query performance and data availability, but they must be updated to reflect changes to the base source.

Basically, there are two strategies for materialized view maintenance: re-materialization and incremental maintenance. In the re-materialization strategy, view data is re-computed at pre-established times. In the incremental maintenance strategy, a

mechanism periodically modifies view data to reflect updates to local sources, which proved to be an effective solution [1, 5].

We address the problem of incremental maintenance of XML views defined on top of relational data. In order to incrementally maintain a XML view, event-condition-rules should be specified for the relational source. The rules are responsible for correctly modifying the XML view content to reflect changes made to the base source.

In our approach, the generation of incremental view maintenance rules consists of three steps:

1. The base source schema is converted to an XML schema. Hence, the view schema and the base source schema are expressed in a “common” data model [10].
2. The correspondence assertions of a view V are generated by matching the view schema and the base source XML schema [10]. The view correspondence assertions formally specify relationships between the view schema and the base source schema.
3. The incremental view maintenance rules are then defined based on the view correspondence assertions identified in Step 2. The correspondence assertions are treated as special types of integrity constraints that must be preserved by update operations on the local sources. We follow the strategy proposed in [4] that describes how constraints can be mapped into rules. The rules required for maintaining a view V with respect to updates on a base source S are those required for maintaining the assertions that specify the relationship between the schemas of V and the source S .

The view maintenance problem has been extensively studied for relational and object-oriented databases [1,7,8,9]. Abiteboul et al [2] proposed an incremental maintenance algorithm for materialized views over semi-structured data, considering the OEM data model and the Lorel query language. El-Sayed et al [5] proposed an algebraic solution approach based on the XAT XML algebra. In [11], we proposed an algorithm for the incremental maintenance of XML views over XML data. To our knowledge, no work addresses the problem of incremental maintenance of materialized views over relational data.

This paper is organized as follows. Section 2 describes how to convert a relational schema to a XML schema. Section 3 discusses the generation of view correspondence assertions. Section 4 presents our approach to generate the incremental view maintenance rules. Section 5 contains the conclusions.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

WIDM '03, November 7-8, 2003, New Orleans, Louisiana.

Copyright 2003 ACM 1-58113-725-7/03/0011...\$5.00.

2. MAPPING RELATIONAL SCHEMA TO XMLS+ SCHEMA

We adopt a graphical notation [10], denoted XMLS⁺ (Semantic XML Schema), to represent the types of a XML schema **S**. XMLS⁺ reduces the complexity of the schema matching process since it allows us to write view correspondence assertions and other formal sentences in a uniform notation. Otherwise, we would have to resort to a mixed notation, catering for XML schemes as well as for relational schemes. Briefly, the notation uses a tree-structured representation for the types of **S**, where bold fonts denote the name of the type, “&” denotes references, “@” denotes attributes and “*” denotes multiple occurrences of an element.

Consider the relational schema for the base source *Bib* shown in Figure 2.1. The corresponding XMLS⁺ schema, **Bib**, generated according to the mapping rules in [10], is shown in Figure 2.2. The root type Troot[Bib] contains an element for each relation scheme of the relational schema. For example, the element authors of type Tauthors corresponds to the relation scheme authors. Tauthors contains a sequence of zero or more tuple_author elements of type Ttuple_author, which in turn contains the attribute ssn and the elements name, email and area. The type ID of the attribute ssn allows a unique identification to be associated with each tuple_author element. The element book_isbn of Ttuple_publication contains a reference &Ttuple_book to a tuple_book element of type Ttuple_book, which is a representation of the referential integrity constraint Publications[book_isbn] ⊆ Books[isbn] in the relational schema.

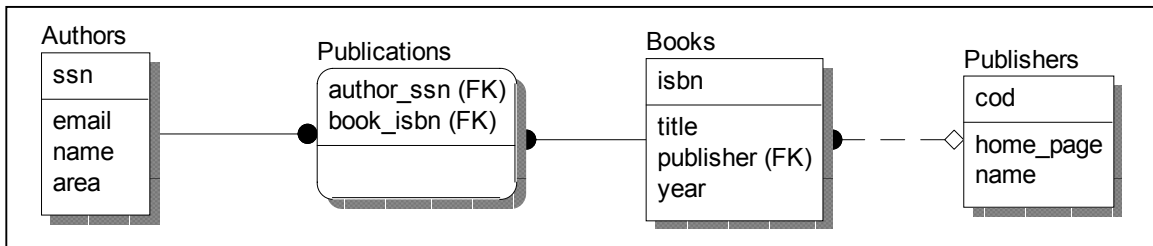


Figure 2.1: Relational Schema for base source *Bib*

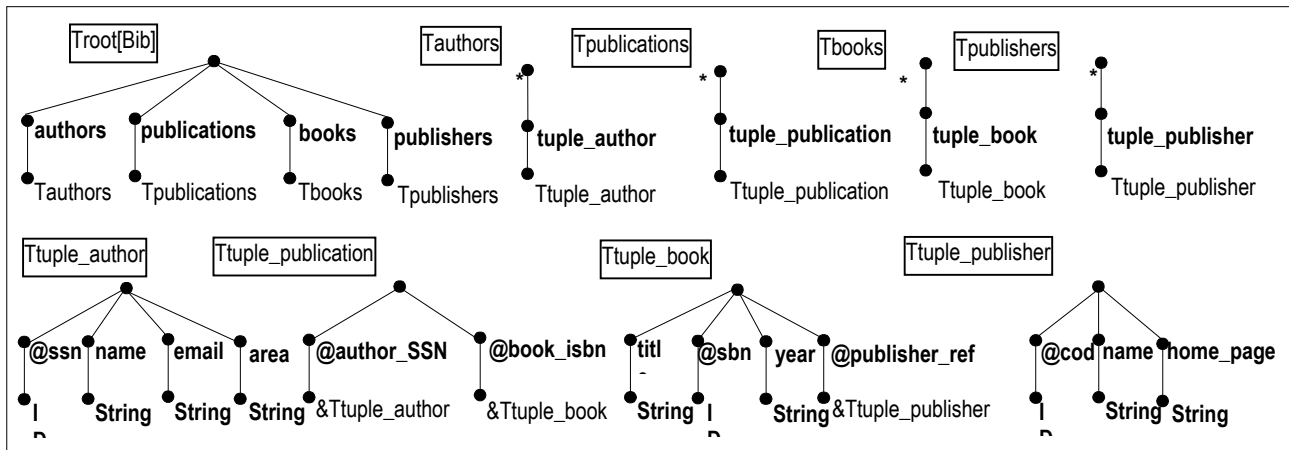


Figure 2.2: XMLS⁺ Schema for *Bib*

3. USING ASSERTIONS

3.1 Notation

A link represents a relationship between XML types. There is a link from type T_1 to type T_2 iff:

- i) T_1 contains an element e whose type is T_2 , denoted $e:T_1 \rightarrow T_2$;
- ii) T_2 contains an element e whose type is T_1 , denoted $e^{-1}:T_1 \rightarrow T_2$;
- iii) T_1 contains an element e whose type is a reference $\&T_2$ to T_2 , denoted $e \rightarrow :T_1 \rightarrow T_2$;
- iv) T_2 contains an element e whose type is a reference $\&T_1$ to T_1 , denoted $(e \rightarrow)^{-1}:T_1 \rightarrow T_2$.

We adopt an extension of XPATH that permits navigating through a reference link. The result of a path expression is a sequence of nodes or primitive values.

Let $l \rightarrow: T_1 \rightarrow T_2$ be a reference link. Given an instance $\$e_1$ of T_1 , the path expression $\$e_1/l \rightarrow$ returns all instances of T_2 referenced in $\$e_1/l$. Conversely, let $l: T_2 \rightarrow T_1$ be a link. Given an instance $\$e_1$ of T_1 , the path expression $\$e_1/l^{-1}$ returns the instance $\$e_2$ of T_2 such that $\$e_1$ in $\$e_2/l$.

Instances of a type T_1 can be related with instances of a type T_n through the composition of two or more links. Consider links $l_k: T_k \rightarrow T_{k+1}$, for $k=1, \dots, n-1$, where T_k are types of an XML schema. Therefore $\delta = l_1 / l_2 / \dots / l_{n-1}$ is a path of T_1 . Given an instance $\$e$ of T_1 , the path expression $\$e/\delta$ selects a set of elements of type T_n . Hence, the type T_δ of the path δ is T_n .

3.2 Specifying View Correspondence

Assertions

In general, we propose to define a view with the help of a view schema, as usual, and a set of view correspondence assertions [10], instead of the more familiar approach of defining a query on the data sources. These assertions axiomatically specify how XML view elements are synthesized from tuples of the base source. In the rest of the paper, consider the schema for the view V in Figure 3.1.

We exemplify the process, described in [10], for generating the correspondence assertions by matching the view XML schema V and the XML base source schema Bib , represented in Figures 3.1 and 2.2. In what follows, let $\$V$, $\$bib$ be global variables corresponding to the XML documents that represent V and Bib , respectively.

Given a XML schema S , the *primary elements* of S are the XML elements in S not nested inside other XML elements in S .

The matching process is top down and consists of two steps:

Step 1: We first define *global collection correspondence assertions* (GCCAs) [10] that match the primary elements of the XML view schema and elements of the XML base schema. For example, the GCCA:

$$\Psi_1: [\$V/author_v] \equiv [\$bib/authors/tuple_author[area = \text{"Database"}]]$$

specifies that $\$V/author_v$ and $\$bib/authors/tuple_author[area = \text{"Database"}]$ denote the same set of real world objects.

Then, we define *matching correspondence assertions* (MCAs) that match elements in two semantically related collections. For example, the MCA:

$$\Psi_2: [Tauthor_v, \{ssn_v\}] \equiv [Ttuple_author, \{@ssn\}]$$

specifies that an author $\$a_v$ in $\$V/author_v$ matches an author $\$a$ in $\$bib/authors/tuple_author$ iff $\$a_v/ssn_v/text() = \$a/@ssn/text()$.

Step 2: We next recursively descend into sub-elements and define other correspondence assertions, including *path correspondence assertions* (PCAs). Figure 3.2 shows the PCAs and MCAs that match the type $Tauthor_v$ with the type $Ttuple_author$. For example, Ψ_6 specifies that, given an instance $\$a_v$, of $Tauthor_v$, if there exists an instance $\$a$ of $Ttuple_author$ such that $\$a_v = \a , then $\$a_v$ contains elements $book_v$ such that $\$a_v/book_v \equiv \$a/(@author_ssn \rightarrow)^{-1}/@book_isbn \rightarrow$. Since $Tbook_v$ is a complex type, we also introduce the MCA Ψ_7 that match instances of $Tbook_v$ with instances of $Ttuple_book$, and the PCAs Ψ_8 , Ψ_9 , Ψ_{10} and Ψ_{11} , obtained by matching $Tbook_v$ and $Ttuple_book$.

4. GENERATING RULES FOR INCREMENTAL VIEW MAINTENANCE

4.1 Terminology

In this section, we study the maintenance of XML views, defined as follows. Consider a view V that contains a set of v elements of type T_v , specified by the GCCA

$$\Psi: [\$V/v] \equiv [\$S/R_b / tuple_R_b[selExp]]$$

where S is the base source, R_b is the base table of the v elements in $\$V$ and $selExp$ is a predicate expression [12]. The GCCA Ψ specifies that $\$V/v$ and $\$S/R_b / tuple_R_b[selExp]$ denote the same set of real world objects. The PCA that matches T_v and $Ttuple_R_b$ specifies the sub-elements of v (see Section 3).

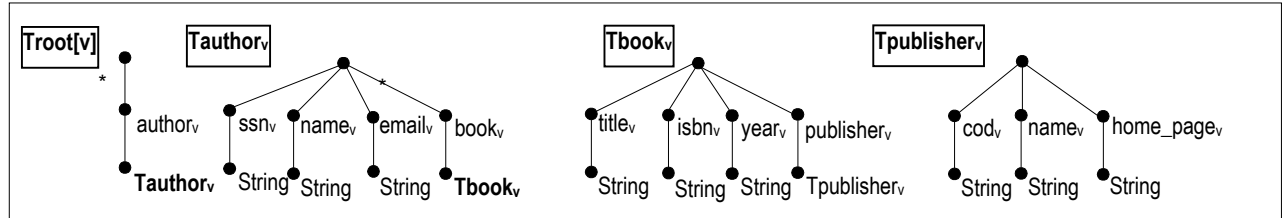


Figure 3.1: XML Schema for View V

<p>PCAs of $Tauthor_v$ & $Ttuple_author$:</p> <p>$\Psi_3: [Tauthor_v/ssn_v] \equiv [Ttuple_author/@ssn]$ $\Psi_4: [Tauthor_v/name_v] \equiv [Ttuple_author/name]$ $\Psi_5: [Tauthor_v/email_v] \equiv [Ttuple_author/email]$ $\Psi_6: [Tauthor_v/book_v] \equiv [Ttuple_author/(@author_ssn \rightarrow)^{-1}/@book_isbn \rightarrow]$</p> <p>MCA of $Tbook_v$ & $Ttuple_book$:</p> <p>$\Psi_7: [Tbook_v, \{isbn_v\}] \equiv [Ttuple_book, \{isbn\}]$</p>	<p>PCAs of $Tbook_v$ & $Ttuple_book$:</p> <p>$\Psi_8: [Tbook_v/title_v] \equiv [Ttuple_book/title]$ $\Psi_9: [Tbook_v/isbn_v] \equiv [Ttuple_book/@isbn]$ $\Psi_{10}: [Tbook_v/year_v] \equiv [Ttuple_book/year]$ $\Psi_{11}: [Tbook_v/publisher_v] \equiv [Ttuple_book/@publisher_ref \rightarrow]$</p> <p>MCA of $Tpublisher_v$ & $Ttuple_publisher$:</p> <p>$\Psi_{12}: [Tpublisher_v, \{cod_v\}] \equiv [Ttuple_publisher, \{@cod\}]$</p> <p>PCAs of $Tpublisher_v$ & $Ttuple_publisher$:</p> <p>$\Psi_{13}: [Tpublisher_v/cod_v] \equiv [Ttuple_publisher/@cod]$ $\Psi_{14}: [Tpublisher_v/name_v] \equiv [Ttuple_publisher/name]$ $\Psi_{15}: [Tpublisher_v/home_page_v] \equiv [Ttuple_publisher/home_page]$</p>
--	---

Figure 3.2: PCAs and MCAs resulting from matching $Tauthor_v$ and $Ttuple_author$

- b) there are paths δ_1 and δ_2 in \mathbf{S} , which may be possibly null, and there is a foreign key FK_1 of R where $\$S / R / \text{tuple}_R$ references $\$S / R_b / \text{tuple}_{R_b} / \delta_1 / \delta_2$ through link FK_1 and $\delta_1 = \delta_{b1} / \dots / \delta_{bn}$ is semantically related to path $\delta_v = \delta_{v1} / \dots / \delta_{vn}$ by PCAs $[T_{R_b} / \delta_{b1}] \equiv [T_v / \delta_{v1}]$ and $[T_{\delta_{bi}} / \delta_{bi+1}] \equiv [T_{\delta_{vi}} / \delta_{vi+1}]$, for $1 \leq i \leq n-1$ (see Figure 4.1), such that

Case 2.1: $\delta = \$S / R_b / \text{tuple}_{R_b} / \delta_1 / \delta_2 / FK_1^{-1}$ and

- (i) if the path δ_1 is not null, then $[T_{\delta_1} / \delta_2 / FK_1^{-1}] \equiv [T_{\delta_v} / v_{n+1}]$
- (ii) if the path δ_1 is null, then $[T_{\text{tuple}_{R_b}} / \delta_2 / FK_1^{-1}] \equiv [T_{\delta_v} / v_{n+1}]$

Case 2.2: there is a path δ_3 , which may be possibly null, and there is a foreign key FK_2 of R such that $\delta = \$S / R_b / \text{tuple}_{R_b} / \delta_1 / \delta_2 / FK_1^{-1} / FK_2 / \delta_3$ and

- (i) if the path δ_1 is not null, then $[T_{\delta_1} / \delta_2 / FK_1^{-1} / FK_2 / \delta_3] \equiv [T_{\delta_v} / v_{n+1}]$
- (ii) if the path δ_1 is null, then $[T_{\text{tuple}_{R_b}} / \delta_2 / FK_1^{-1} / FK_2 / \delta_3] \equiv [T_{\delta_v} / v_{n+1}]$.□

We use \mathcal{P}_R , to denote the set of all paths of $\text{Troot}[\mathbf{S}]$ that satisfy the conditions of Theorem 4.1.

Let δ be the path $\$S / R_b / \text{tuple}_{R_b}$. Then, δ satisfies Case 1 of Theorem 4.1, which implies that δ is relevant to V with respect to insertions into R_b . So, δ can be affected by an insertion into R_b , by Definition 4.5(ii). From the assertion $\Psi: [\$V / v] \equiv [\$S / R_b / \text{tuple}_{R_b}[\text{selExp}]]$, we have that, if the inserted element $\$new$ satisfies the selection condition selExp , then the state of $\$V/v$ is also affected by an insertion into R_b .

When INSERT ON R

Then

If $\text{selExp}(\$new)$ then

{ $\$child = \text{Constructor}_{T_v} \text{tuple}_{R_b}(\$new)$
INSERT $\$child$ as a child of $\$V$ }

Figure 4.2: Rule for maintenance of V with respect to insertion in R for the relevant path $\$S/R/\text{tuple}_{R_b}$.

When INSERT ON R

Then

$Q := \{ \$q \mid \$q \text{ in } \$new / FK_1 / \delta_2^{-1} \}$

$S := \{ \$s \mid \$s \text{ in } \$new / FK_2 / \delta_3 \}$

$T := \emptyset$

For $\$q$ in Q do

$T := T \cup \{ \$v_n \mid \$v_n \text{ in } \$V / v / \delta_v \text{ and } \$v_n \equiv \$q \};$

For $\$s$ in S do

{ $\$child := \text{constructor}_{T_{\delta_{v_{n+1}}}} \& T_{\delta_{bf}}(\$s);$

For $\$target$ in T do

INSERT $\$child$ as a child of $\$target$ }

Figure 4.3: Rule for maintenance of V w.r.t insertion in R for relevant path $\$S / R_b / \text{tuple}_{R_b} / \delta_1 / \delta_2 / FK_1^{-1} / FK_2 / \delta_3$.

Figure 4.2 shows the rule for maintaining V with respect to an insertion into R induced by path $\delta = \$S / R_b / \text{tuple}_{R_b}$.

Let δ be a path of the form $\delta = \$S / R_b / \text{tuple}_{R_b} / \delta'$ and suppose that it satisfies the conditions of Case 2 of Theorem 4.1. Then, δ is relevant to V with respect to insertions into R . Hence, by Definition 4.5, the state of $\delta = \$S / R_b / \text{tuple}_{R_b} / \delta'$ is affected by an insertion into R and δ' is semantically related to a path δ' from T_v . Hence, path $\$V / v / \delta'$ and, consequently, $\$V / v$ can be affected by an insertion into R . (The path $\$V / v / \delta'$ will indeed be affected if the elements affected in $\delta = \$S / R_b / \text{tuple}_{R_b}$ meet the selection condition selExp). Using the correspondence assertions that relate $\delta = \$S / R_b / \text{tuple}_{R_b} / \delta'$ with $\$V / v / \delta'$, we can compute the set of updates required for maintaining the state of $\$V / v / \delta'$ consistent with the new state of the path $\delta = \$S / R_b / \text{tuple}_{R_b} / \delta'$.

Figure 4.3 shows the rules for maintaining V with respect to insertion into R when $\delta = \$S / R_b / \text{tuple}_{R_b} / \delta_1 / \delta_2 / FK_1^{-1} / FK_2 / \delta_3$ (Case 2.2 of Theorem 4.1).

The rule for case 2.1 is a special case of the rule in Figure 4.3, where $S = \{ \$new \}$.

Note that, by adopting the XMLS⁺ notation, we are lead to incremental view maintenance rules expressed over the base XML schema, and not over the base relational schema. This does not really pose a problem, since the translation of rules over one schema into rules over the other is straightforward because one can easily establish an isomorphism between the two schemas. Once again, this apparent drawback is surpassed by the advantages of using a uniform notation, i.e., the XMLS⁺ notation.

Example 4.3: Let us generate the view maintenance rules for V w.r.t insertions into table Authors of Bib.

Step 1: Since Authors is the base table of V (case 1 of Theorem 4.1) then $\$S/R/\text{tuple}_{R_b}$ is the only relevant path.

Step 2: For the relevant path $\$bib/authors/\text{tuple}_{author}$, we generate the rule in Figure 4.4 The rule is fired whenever an insertion in table Authors occurs.

Consider, for example, that tuple $\langle 1546, \text{Kevin}, \text{kevin@oxford.com}, \text{Database} \rangle$ is inserted into table Authors (see Figure 4.5).

According to the rule, since $\$new/area = \text{"Database"}$, the updates required for maintaining V consist of the insertion of $\$child$ as a child of $\$V$, where $\$child$ is an $author_v$ element generated by calling $\text{Constructor}_{T_{author_v}} \text{tuple}_{author}(\$new)$, which creates an $author_v$ element corresponding to the tuple_{author} element $\$new$.

When INSERT ON Authors

Then

If $\$new/area = \text{"Database"}$ then

{ $\$child := \text{Constructor}_{T_{author_v}} \text{tuple}_{author}(\$new)$
INSERT $\$child$ as a child of $\$V$ }

Figure 4.4: Rules for maintaining view V with respect to insertions in Authors

Authors				Publications	
ssn	name	email	area	author_ssn	book_isbn
5467	Alan	alan@oxford.com	Database	5467	1-003-05
1647	Max	max@oxford.com	Artificial Intelligence	1647	1-003-06

Books				Publishers		
title	isbn	year	publisher	cod	home_page	name
Relations	1-003-05	2001	15	15	www.klower.com	Klower
Data Mining	1-003-06	2002	20	20	www.mitpress.com	Mit Press

Figure 4.5: Current state of Bib (Example 4.3)

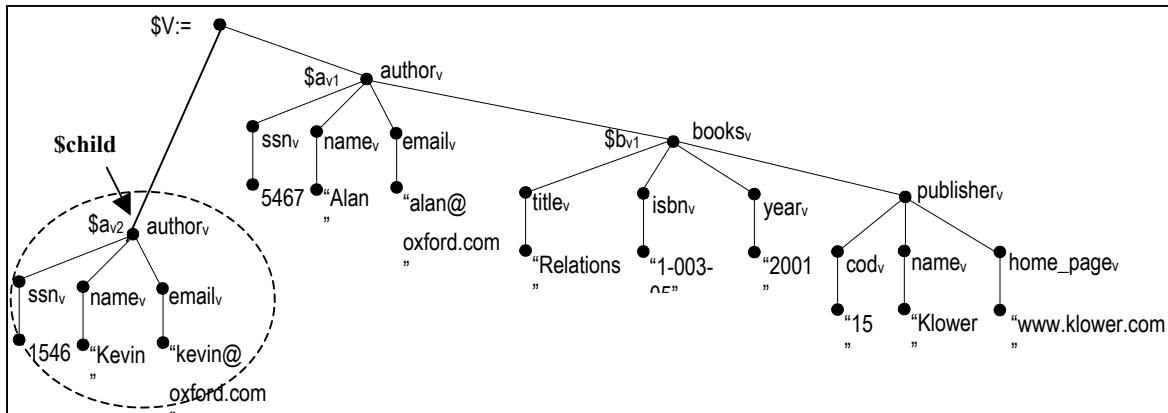


Figure 4.6: The new state of the view V (Example 4.3)

```

Constructorauthor_v_Ttuple_author($new)
return <author_v>
    <ssn_v>$new/@ssn/text()</ssn_v> (from  $\Psi_3$ )
    <name_v>$new/name/text()</name_v> (from  $\Psi_4$ )
    <email_v>$new/email/text()</email_v> (from  $\Psi_5$ )
  </author_v>

```

Figure 4.7: Constructor T_{author_v} _Ttuple_author

```

When INSERT ON Publications
Then
  $q := $new/@author_SSN ->;
  $s := $new/@book_isbn ->;
  $schild:= ConstructorTbook_v_Ttuple_book($s);
  LET $target in $V/author_v where $target $\equiv$ $q do
    INSERT $schild as a child of $target ;

```

Figure 4.8: Rule for maintenance of the view V with respect to insertions in Table Publications of Bib

The procedure Constructor T_{author_v} _Ttuple_author, shown in Figure 4.7, is automatically generated based on the PCAs of T_{author_v} and T_{tuple_author} .

The new state of the materialized view V, after the updates, is shown in Figure 4.6.

Example 4.4: Let us generate the view maintenance rules for V w.r.t insertions into table Publications of Bib.

Step 1: From Example 4.2, we have that $\$bib/publications/tuple_publication$ references the path $\$bib/authors/tuple_author$ through link $@author_ssn \rightarrow$. From PCAs Ψ_6 , we have that:

$$T_{tuple_author}/(@author_ssn \rightarrow)^{-1}/@book_isbn \rightarrow \equiv T_{author_v}/book_v$$

Therefore, we have that the path

$\$bib/authors/tuple_author/(@author_ssn \rightarrow)^{-1}/@book_isbn \rightarrow$ satisfies case 2.2 of Theorem 4.1. Thus, the path is relevant to V.

Step 2: For the relevant path

$\$bib/authors/tuple_author/(@author_ssn \rightarrow)^{-1}/@book_isbn \rightarrow$ we generate the rule in Figure 4.8. Note that this rule is a special case of the rule in Figure 4.3. (Paths δ_1 , δ_2 and δ_3 do not exist and, hence, Q and S are singletons).

Consider, for example, that tuple <1546, 1-003-05> is inserted into table Publications (see Figure 4.9). According to the rule, the updates required for maintaining V consists of the insertion of \$schild as a child of \$target, where:

(i) \$child is a book_v element generated by calling ConstructorTbook_v_Ttuple_book(\$book), which creates an book_v element corresponding to the element \$book. The procedure Constructor_Tbook_v_Ttuple_book, shown in Figure 4.10, is automatically generated, based on the PCAs of Tbook_v and Ttuple_book.

(ii) \$target is the author_v element in \$V/author_v such that author_v≡\$q. From the MCA Ψ₂ we have that author_v≡\$q iff \$target/ssn_v/text()=\$q/ssn_v/text().

The new state of the materialized view V, after the updates, is shown in Figure 4.11.

Authors			
ssn	name	email	area
5467	Alan	alan@oxford.com	Database
1647	Max	max@oxford.com	Artificial Intelligence
1546	Kevin	kevin@oxford.com	Database

Publications	
author_ssn	book_isbn
5467	1-003-05
1647	1-003-06

Books			
title	isbn	year	publisher
Relations	1-003-05	2001	15
Data Mining	1-003-06	2002	20

Publishers		
cod	home_page	name
15	www.klower.com	Klower
20	www.mitpress.com	Mit Press

Figure 4.9: Current state of Bib (Example 4.4)

```

ConstructorTbookv_Ttuple_book($book)
Return <bookv><titlev>$book /title/text()</titlev> (from Ψ8)
        <isbnv>$book /@isbn/text()</isbnv> (from Ψ9)
        <yearv>$book /year/text()</yearv> (from Ψ10)
        <publisherv><codv>$book /@publisher_ref->/@cod/text()</codv> (from Ψ11 and Ψ13)
            <namev>$book /@publisher_ref->/name/text()</namev> (from Ψ11 and Ψ14)
            <home_pagev>$book /@publisher_ref->/home_page/text()</home_pagev> (from Ψ11 and Ψ15)
        </publisherv>
    </bookv>
  
```

Figure 4.10: ConstructorTbook_v_Ttuple_book

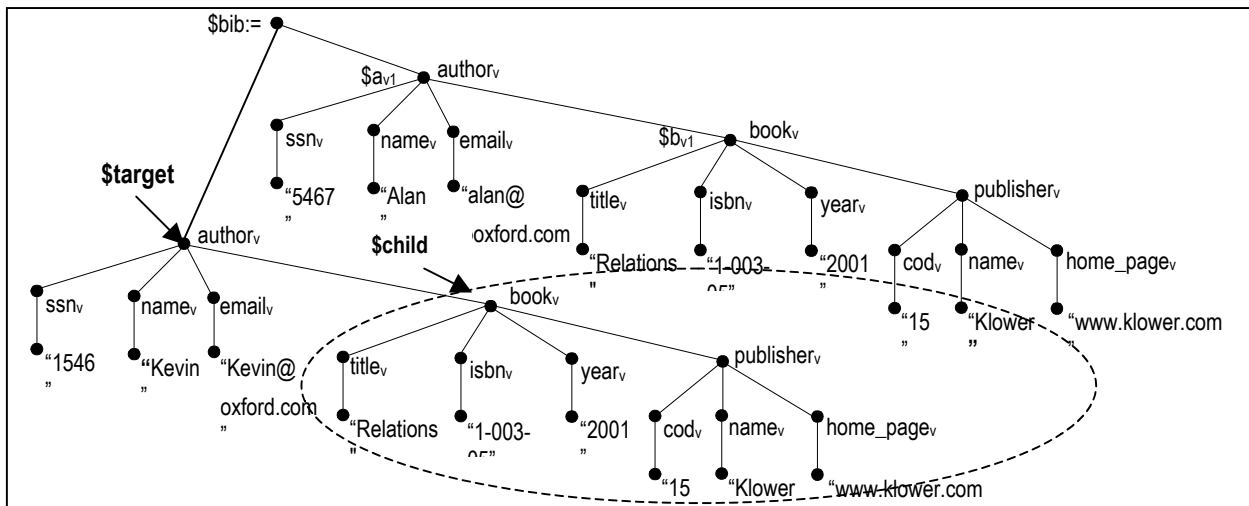


Figure 4.11: The new state of the view V (Example 4.4)

5. CONCLUSIONS

In this paper, we proposed an approach for generating rules for the incremental maintenance of XML views defined on top of relational data. We first described how to generate the XMLS⁺ schema for a relational schema. Then, we discussed how to derive maintenance rules from view correspondence assertions, which formally specify the relationships between the view schema and the XMLS⁺ base source schema.

In more detail, we showed that, based on the view correspondence assertions, we are able to: (i) identify all the paths in the XML base source schema that are relevant for a given update operation; and (ii) define the view maintenance statements required for a given relevant path.

6. REFERENCES

- [1] Ali, M.A., Fernandes, A.A., Paton, N.W.: Incremental Maintenance for Materialized OQL Views. In Proc. DOLAP (2000) 41–48
- [2] Abiteboul, S., McHugh, J., Rys, M., Vassalos, V., Wiener, J.L.: Incremental Maintenance for Materialized Views over Semistructured Data. In Proceedings of the International Conference on Very Large Databases. New York City (1998) 38-49
- [3] Carey, M.J., Kiernan, J., Shanmugasundaram, J., Shekita, E.J., Subramanian, S.N.: XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents. In The VLDB Journal(2000) 646–648
- [4] Ceri, S., Widom, J.: Deriving productions rules for incremental view maintenance. In Proceedings of the International Conference on Very Large Databases (1991) 577-589
- [5] EL-Sayed, M., Wang, L., Ding, L., Rudensteiner, E.: An algebraic approach for Incremental Maintenance of Materialized Xquery Views. In Proceedings of Fourth International Workshop on Web Information and Data Management. McLean, USA (2002)
- [6] Fernandez, M., Morishima, A., Suci, D., Tan, W.: Publishing Relational Data in XML: the SilkRoute Approach. IEEE Trans on Computers, 44(4). (2001) 1–9
- [7] Gupta, A., Mumick, I.S.: Maintenance of Materialized Views: Problems, Techniques, and Applications. In IEEE Bulletin on Data Engineering, 18(2).(1995)3–18
- [8] Gupta, A., Mumick, I.S., Subrahmanian, V.S.: Maintaining Views Incrementally. In SIGMOD (1993) 157–166
- [9] Kuno, H.A., and Rudensteiner, E.A.: Incremental Maintenance of Materialized Object-Oriented Views in MultiView: Strategies and Performance Evaluation. IEEE Transaction on Data and Knowledge Engineering, 10(5):768–792 (1998)
- [10] Vidal, V.M.P., Vilas Boas, R.: A Top-Down Approach for XML Schema Matching. In Proceedings of the 17th Brazilian Symposium on Databases. Gramado, Brazil (2002).
- [11] Vidal, V.M.P., Casanova, M.A.: Efficient Maintenance of XML Views Using View Correspondence Assertions. In Proceedings of the 4th International Conference on Electronic Commerce and Web Technologies. Praga, Czech Republic (to appear).
- [12] World-Wide Web Consortium. XML Path Language (XPath): Version 1.0 (November 1999). See <http://www.w3.org/TR/xpath.html>.