WPI-CS-TR-03-23b Dec 2003

Updating XQuery Views Published over Relational Data

by

Ling Wang and Elke A. Rundesnteiner

Abstract

XML data management using relational database systems has been intensively studied in the last few years. However, in order for such systems to be viable, they must support not only queries, but also updates over virtual XML views that wrap the relational base. While view updating is a long-standing difficult issue in the relational context, the recent invention of the flexible XML data model and nested XML query language poses new challenges for view updating.

In this paper, we first formally model the XQuery View Update Problem. Then the clean extended-source theory for determining the existence of a relational update translation for a given XQuery view update is proposed. We identify new update translation issues, in particular view construction consistency, update granularity and view duplication, which are caused by the distinct features of XML, most notably, the XML hierarchical data structure and nested XQuery expressions. We propose a graph-based algorithm for detecting the occurence of these issues by view definition analysis. The connection between the algorithm with the clean extended-source theory is also established. Finally, the XML view updatability for various update granularity is presented according to different update types.

1 Introduction

XML has become the standard for interchanging data between web applications. Recent XML management systems [6, 10, 19] combine the strengths of the XML data model with the maturity of relational database technology to provide both reliable persistent storage as well as flexible query processing and publishing. To bridge the gap between relational databases and XML applications, such systems typically support creating XML views that wrap the relational base and querying against it. Update operations against such wrapper views, however, are not yet well supported.

The problem of updating XQuery views published over relational data comes with new challenges beyond those of relational [8, 2, 9, 11, 12, 1] or even object-oriented [4] views. We have to address the mismatch between the two distinct data models (the XML flexible hierarchical view model and the flat relational base model). That is the nested structure imposed by an XML view may be in conflict with the constraints of the underlying relational schema. Also the mismatch between the two query and update languages (XQuery nested FLWU updates on the view versus SQL update queries on the base) has to be emphasied. Recent work [15] presents an XQuery update grammar and studies the execution performance of translated updates, assuming that the given view update is indeed always translatable. System specific XML view management solutions are also provided by SQL-Server2000 [13], Oracle [3] and DB2 [7]. Our earlier work [17] discusses the updatability in the context of a particular XML view subproblem, namely, the "round-trip" case. This case is characterized by a pair of inversable loading and extraction queries used for mapping XML to and from the relational database. We show that such views are always updatable by any *valid* update operation.

However, there is no result yet in the literature focus on a general method to assess the *updatability* of arbitrary XML view. That is, given an XML view and an update pair, we need a method to decide whether a translation of the view updates expressed in XQuery into relational updates expressed in SQL exists that satisfies the well-established update correctness properties.

The view updatability decision issue has been a long standing difficulty even in the relational database context. Using the concept of *clean source*, [9] characterizes the conditions under which a relational view over a relational base is updatable. Our work in this paper first adopt the concepts of a "clean source" into the XML context by defining a view-element with various granularity. We then propose a "clean extended-source" concept adjusted to also account for the referential integrity maintenance. Due to the flexibility of XML hierarchical data model and the nested FLWR expressions of the XQuery language, the XQuery views exhibit several distinct features affecting the view update translatability, most notably, including *view construction consistency*, *update granularity* and *view duplication*. That is, the flexible update granularity in XML causes more situations to become untranslatable, especially when the view can be shown to have an inconsistent construction or a duplication. We illustrate the effects of these features on the view updatability by motivating examples in Section 2. Then a graph-based algorithm for detecting the occurrence of these problem features in the view is proposed. Its connection with the clean extended-source basis is also proven. The translatability of different update types on these views with various granularity is finally identified.

A system framework skeleton solving the XML view update problem is provided serving as proof of the viability of our proposed method. We have implemented the system within the XML data management system *Rainbow* [19] and have conducted several experiments to assess various performance characteristics of our update solution [17].

Contributions.

- We characterize the *XQuery view update* problem and identify the new challenges imposed by the XML hierarchical data model and the nested XQuery syntax.
- We propose and prove the *clean extended-source theory* for determining whether a correct view update translation exists.
- We propose a *graph-based* algorithm for detecting the inconsistency and duplication in views, and prove its connection with the clean extended-source theory.
- We analyze the XML view updatability for various update granularity by different update types.
- We propose a *system framework* solving the XML view update problem and implement it in the Rainbow System [19].

Outline. This paper is structured as follows. Section 2 provides several motivating examples to illustrate the challenges of the XQuery view update problem which then is formally modeled by Section 3. In Section 4 we propose the "clean extended-source" theory as our theoretical foundation. Section 5 describes our graph-based algorithm for detecting the consistency and duplication properties of view. The translatability of updates on these views with various granularity is then identified in Section 6. The update system framework for solving the XML view update problem is given in Section 7. Section 8 reviews related work while Section 9 provides the conclusion.

2 Motivating Examples

Basics of update policy. Now we will introduce several examples to illustrate what features of XML cause new view update translation issues. Fig.1 is the example XML document which contains a list of book titles and their optionally prices. Fig.2 describes one possible relational schema and database capturing this XML document. Recent XML systems (XPERANTO [6], SilkRoute [10] and Rainbow [19]) use a basic XML view, called *default XML view*, to define the one-to-one XML-to-relational mapping (Fig.3). On top of this default XML view, a *virtual view* is introduced to define the user-specific views. Such a virtual view can be specified by an XQuery expression [16] called a *view query* (Fig.4). Although W3C is adding update capabilities to the XQuery standard [16], to date there is no one standard update XQuery syntax. For our work, we thus adopt the update XQuery syntax introduced in [15]. Fig.5 shows several examples of view updates.

Note that the *update translation policy* is essential to this view updatability since a given update may translatable under one policy, while not under another. In the examples below, when not stated otherwise, we say an update is translatable under the policy that: (i) translated updates have the same type as the view update and (ii) delete cascading is applied

```
<bib>
  <book>
    <bookid>98001</bookid>
    <title>TCP/IP Illustrated</title>
    <price>
      <amount>63.70</amount>
      <website>www.amazon.com</website>
    </price>
  </book>
  <book>
    <bookid>98002</bookid>
    <title>Programming in Unix</title>
  </book>
  <book>
    <bookid>98003</bookid>
    <title>Data on the Web</title>
    <price>
      <amount>56.00</amount>
      <website>www.amazon.com</website>
    </price>
    <price>
      <amount>45.60</amount>
      <website>www.bookpool.com</website>
    </price>
  </book>
</bib>
```

Fig. 1. Example XML document

book

bookid	title
98001	TCP/IP Illustrated
98002	Programming in Unix
98003	Data on the Web

price

bookid	amount	website
98001	63.70	www.amazon.com
98003	56.00	www.amazon.com
98003	45.60	www.bookpool.com

Legend:
▨ **Primary Key**
☐ **Non Key**

```
CREATE TABLE book(
  bookid VARCHAR2(20),
  title VARCHAR2(100),
  CONSTRAINTS BookPK PRIMARYKEY (bookid))

CREATE TABLE price(
  bookid VARCHAR2(20),
  amount DOUBLE,
  website VARCHAR2(100),
  CONSTRAINTS PricePK PRIMARYKEY (bookid, website),
  FOREIGNKEY (bookid) REFERENCES book (bookid))
```

Fig. 2. Relational database capturing the XML data from Fig. 1

```
<DB>
  <book>
    ...
    <row>
      <bookid>98003</bookid>
      <title>Data on the Web</title>
    </row>
  </book>
  <price>
    ...
    <row>
      <bookid>98003</bookid>
      <amount>56.00</amount>
      <website>www.amazon.com</website>
    </row>
    <row>
      <bookid>98003</bookid>
      <amount>45.60</amount>
      <website>www.bookpool.com</website>
    </row>
  </price>
<DB>
```

Fig. 3. Default XML view of database shown in Figure 2

in maintaining referential integrity of the relational database. For the detail of update policy selection see Section 5.2. Also, we do not indicate the order of the translated relational updates in our description. Using different execution strategy, the correct order will be easily decided.

Example 1 : View construction consistency.
Given the two XQuery views $V1$ and $V2$ in Fig.4, which both represent books from the website "www.amazon.com", but with different XML view hierarchies. Two view updates u_1^V on $V1$ and u_2^V on $V2$ are listed in Fig.5 respectively.

(i) u_1^V is translatable as shown by Fig.6. The translated relational update sequence U^R in Fig.6(b) will delete the first book from the "book" relation by u_1^R, and its price information from the "price" relation through u_2^R. By re-applying the view query $Q1$ on the updated database D' in Fig.6(c), the user would get the updated XML view in Fig.6(d). This view is equal to the expected updated view $V1'$ in Fig.6(a). Hence U^R in Fig.6(b) is a correct translation of u_1^V.

(ii) u_2^V is not translatable as shown by Fig.7. First of all, the relational update u_1^R in Fig.7(b) is generated to delete the book named "TCP/IP Illustrated" from the "book" relation. Then after the existing foreign key from the "price" relation to the "book" relation as shown by relational database schema (Fig.2) is identified, according to our selected update propagation policy, the second update operation u_2^R will be generated by the update translator to keep the relational database consistent. That is, the corresponding price of the deleted book will be deleted as well. By reapplying $Q2$ on this updated database in Fig.7(c), we will produce the updated view in Fig.7(d). This is different than the expected updated view $V2'$ in Fig.7(a). $V2$ is thus said to be not updatable for u_2^V.

This difference in the existence of a correct translation is caused by the mismatch between the XML hierarchical view model and the underlying flat relational base model. This *view construction consistency* property, namely, whether the XML view hierarchy agrees with the hierarchical structure implied in the base relational schema, is one of the key factors for XQuery view updating.

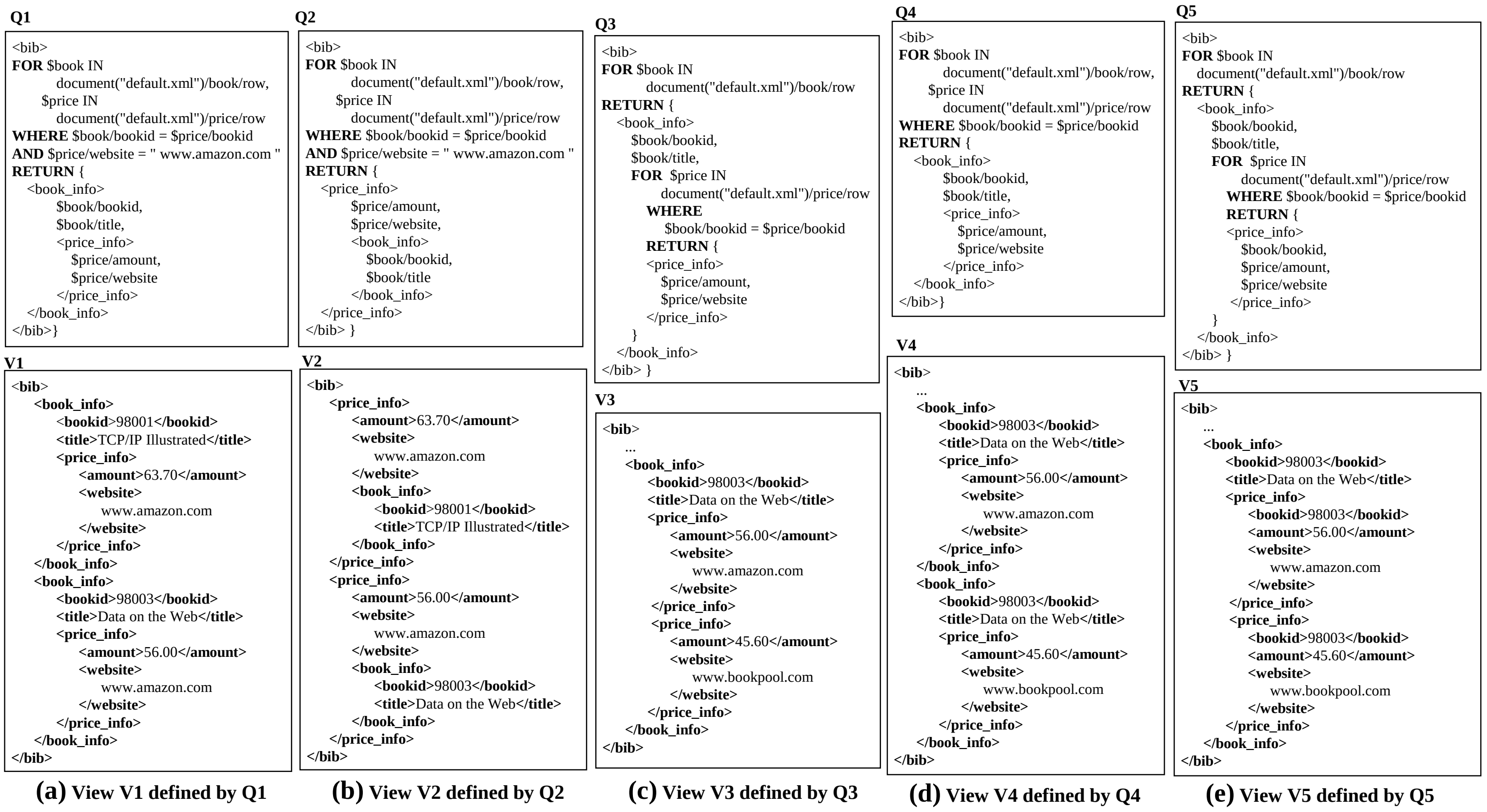
Example 2 : Update granularity.
Compared with the failure of translating u_2^V in Example 1, Fig.8 shows another update operation u_3^V in Fig.5 over the same view $V2$ that is translatable. u_3^V deletes the whole "price_info" element instead of just the subelement "book_info" from $V2$. The translated relational update sequence U^R in Fig.8(b) will delete the first book from the "book" relation by u_1^R, and its prices from the "price" relation through u_2^R. Re-applying the view query $Q2$ on the updated database in Fig.8(c) will generate the XML view in Fig.8(d), which is indeed identical to the expected updated view $V2''$ in Fig.8(a). Hence U^R is a correct translation of u_3^V.

Here the difference in translation existence is not just caused by the shape of the view structure as in Example 1, but also the *granularity* of the update operation. That is the effects of updates on the view. u_3^V has a larger granularity than u_2^V, covering the "top" element of the XML view. It thus "resolves" the inconsistency between the two hierarchies respectively from the view and the relational schema mentioned by Example 1. The XML hierarchical structure offers an opportunity for different update granularity, an issue that never arises for relational views. This flexibility affects update translatability, as will be discussed in Section 6.

Example 3 : View duplication.
Content duplication. *Compare the two XQuery views $V3$ and $V4$ in Fig.4. The book named "Data on the Web" with two prices is exposed twice in $V4$, while only once in $V3$. The update u_5^V in Fig.5 will delete the first book element, which is from the website "www.amazon.com", while keeping the second book element from "www.bookpool.com". Translating update u_5^V is ambiguous since without further communication with the user it is not decidable whether the user desires to delete the book from the "book" table as well. However, update u_4^V in Fig.5 defined on $V3$ is translatable without any ambiguity.*

Note that by further communication with the user, update u_5^V may become translatable. For example, assuming user's desire is to keep the corresponding "book" in the book relation. An extra translation rule: No tuple is deleted if it is still referenced by any other part of the view, would now make the update u_5^V translatable.

Structural duplication. *Given the XQuery view $V5$ in Fig.4, the bookid element is exposed twice by the view query $Q5$. Hence each price within the book will also have a "bookid" element. The update u_8^V in Fig.5, which deletes the first*



Q1
```
<bib>
FOR $book IN
        document("default.xml")/book/row,
     $price IN
        document("default.xml")/price/row
WHERE $book/bookid = $price/bookid
AND $price/website = " www.amazon.com "
RETURN {
   <book_info>
        $book/bookid,
        $book/title,
        <price_info>
           $price/amount,
           $price/website
        </price_info>
   </book_info>
</bib>}
```

V1
```
<bib>
   <book_info>
        <bookid>98001</bookid>
        <title>TCP/IP Illustrated</title>
        <price_info>
           <amount>63.70</amount>
           <website>
              www.amazon.com
           </website>
        </price_info>
   </book_info>
   <book_info>
        <bookid>98003</bookid>
        <title>Data on the Web</title>
        <price_info>
           <amount>56.00</amount>
           <website>
              www.amazon.com
           </website>
        </price_info>
   </book_info>
</bib>
```

(a) View V1 defined by Q1

Q2
```
<bib>
FOR $book IN
        document("default.xml")/book/row,
     $price IN
        document("default.xml")/price/row
WHERE $book/bookid = $price/bookid
AND $price/website = " www.amazon.com "
RETURN {
   <price_info>
        $price/amount,
        $price/website,
        <book_info>
           $book/bookid,
           $book/title
        </book_info>
   </price_info>
</bib> }
```

V2
```
<bib>
   <price_info>
        <amount>63.70</amount>
        <website>
           www.amazon.com
        </website>
        <book_info>
           <bookid>98001</bookid>
           <title>TCP/IP Illustrated</title>
        </book_info>
   </price_info>
   <price_info>
        <amount>56.00</amount>
        <website>
           www.amazon.com
        </website>
        <book_info>
           <bookid>98003</bookid>
           <title>Data on the Web</title>
        </book_info>
   </price_info>
</bib>
```

(b) View V2 defined by Q2

Q3
```
<bib>
FOR $book IN
      document("default.xml")/book/row
RETURN {
   <book_info>
      $book/bookid,
      $book/title,
      FOR $price IN
            document("default.xml")/price/row
         WHERE
            $book/bookid = $price/bookid
         RETURN {
         <price_info>
            $price/amount,
            $price/website
         </price_info>
      }
   </book_info>
</bib> }
```

V3
```
<bib>
   ...
   <book_info>
      <bookid>98003</bookid>
      <title>Data on the Web</title>
      <price_info>
         <amount>56.00</amount>
         <website>
            www.amazon.com
         </website>
      </price_info>
      <price_info>
         <amount>45.60</amount>
         <website>
            www.bookpool.com
         </website>
      </price_info>
   </book_info>
</bib>
```

(c) View V3 defined by Q3

Q4
```
<bib>
FOR $book IN
        document("default.xml")/book/row,
     $price IN
        document("default.xml")/price/row
WHERE $book/bookid = $price/bookid
RETURN {
   <book_info>
        $book/bookid,
        $book/title,
        <price_info>
           $price/amount,
           $price/website
        </price_info>
   </book_info>
</bib>}
```

V4
```
<bib>
   ...
   <book_info>
      <bookid>98003</bookid>
      <title>Data on the Web</title>
      <price_info>
         <amount>56.00</amount>
         <website>
            www.amazon.com
         </website>
      </price_info>
   </book_info>
   <book_info>
      <bookid>98003</bookid>
      <title>Data on the Web</title>
      <price_info>
         <amount>45.60</amount>
         <website>
            www.bookpool.com
         </website>
      </price_info>
   </book_info>
</bib>
```

(d) View V4 defined by Q4

Q5
```
<bib>
FOR $book IN
   document("default.xml")/book/row
RETURN {
   <book_info>
      $book/bookid,
      $book/title,
      FOR $price IN
            document("default.xml")/price/row
         WHERE $book/bookid = $price/bookid
         RETURN {
         <price_info>
            $book/bookid,
            $price/amount,
            $price/website
         </price_info>
      }
   </book_info>
</bib> }
```

V5
```
<bib>
   ...
   <book_info>
      <bookid>98003</bookid>
      <title>Data on the Web</title>
      <price_info>
         <bookid>98003</bookid>
         <amount>56.00</amount>
         <website>
            www.amazon.com
         </website>
      </price_info>
      <price_info>
         <bookid>98003</bookid>
         <amount>45.60</amount>
         <website>
            www.bookpool.com
         </website>
      </price_info>
   </book_info>
</bib>
```

(e) View V5 defined by Q5

Fig. 4. View V1 to V5 defined by XQuery Q1 to Q5 respectively

price of the specified book, will be ambiguous in translation. "bookid" is the key of the "book" relation. Thus we cannot decide whether to delete the book.

In the first case, the "duplication" causing this ambiguity is introduced by the XQuery "FOR" expressions. We call it *content duplication*. This is not unique to XML although it is generated by the nested XQuery expression. It may appear in the relational view context, for example in *Join* views.

However, the *structural duplication*, as illustrated by the second case, is special to XQuery view updating. While it also exists in the relational context, it would not cause any ambiguity in relational update translation. The reason is that in the flat relational data model we always have corresponding *tuple-based* updates. This means that the update operation touches all the duplication parts within a given view tuple instead of just some of them. Thus, it would never generate any inconsistent situation. However, the flexible hierarchical structure of XML allows such a "partial" update instead of enforcing a full update of the XML view. This provides the possibility of some inconsistency between the duplicated parts to arise. See Section 5.4 for details.

3 The XQuery View Update Problem

The XQuery view update problem can be described as follows. Given a relational database and an XQuery view definition over it, can we decide whether an update against the view is translatable into corresponding updates against the underlying relational database without violating any consistency? Intuitively, by consistency, we mean that (1) the requested updates agree with the *XML view schema*, (2) the translated updates against the relational database comply with the *relational schema*, and (3) the XML view reconstructed on the updated relational database using the view definition is exactly the same as the result of directly updating the materialized view, that is without *view side-effects*. And, if it translatable, how would this translation be done.

In this section, we will formally model the *XQuery view update problem* and introduce the notations used for studying its updatability (Table 1).

3.1 Framework of XML View on Relational Database.

The structure of a relation is described by a **relation schema** $\mathcal{R}(\mathcal{N}, \mathcal{A}, \mathcal{F})$, where $\mathcal{N}$ is the name of the relation, $\mathcal{A} = \{a_1, a_2, ..., a_m\}$ is its attribute set, and $\mathcal{F}$ is a set of constraints. An instance of a relation schema or a **relation** for short, denoted by R , is a finite subset of $dom(\mathcal{A})$, a *product* of all the attribute domains. A **relational database schema** $\mathcal{SCH}^{\mathcal{R}}$ is a combination of a set of relation schemes $\mathcal{S}$ and integrity constraints $\mathcal{C}$, where $\mathcal{S} = \{\mathcal{R}_i(\mathcal{N}_i, \mathcal{A}_i, \mathcal{F}_i) | i = 1, ..., n\}$. An instance of a database schema, or a **relational database**, denoted as D, is a finite set of relations $R_1, ..., R_n$ where R_i is a relation instance of relation schema $\mathcal{R}_i$. A **relational update operation** $u^R \in \mho^R$, is a deletion, insertion or replacement on a database D. An **sequence** of relational update operations, denoted by $U^R = \{u_1^R, u_2^R, ..., u_p^R\}$, is modeled as a function $U^R(D) = u_p^R(u_{p-1}^R(..., u_2^R(u_1^R(D))))$.

An XML view instance V, or **XML view** for short, over a given relational database D is defined by a **view definition** DEF^V. The domain of the view is denoted by $dom(V)$. DEF^V is an XQuery expression in our case. Let *rel* be a function to extract the relations referenced by DEF^V in D, then $rel(DEF^V) = \{R_{i_1}, R_{i_2}, ..., R_{i_p}\} \subseteq D$. Let *nest* be a function to nest a set of *FLWR* expressions, then $DEV^V = nest(Exp_0, ..., Exp_m) = Exp_0(nest(Exp_1, ..., Exp_m))$, where Exp_0 is the "outer-most"

u^V_1
```
FOR $root IN document("V1.xml"),
     $book IN $root/book_info
WHERE $book/title/text() = "TCP/IP Illustrated"
UPDATE $root {
     DELETE $book }
```

u^V_2
```
FOR $root IN document("V2.xml"),
     $book IN $root/price_info/book_info
WHERE $book/title/text() = "TCP/IP Illustrated"
UPDATE $root {
     DELETE $book }
```

u^V_3
```
FOR $root IN document("V2.xml"),
     $price IN $root/price_info
WHERE $price/book_info/title/text() = "TCP/IP Illustrated"
UPDATE $root {
     DELETE $price }
```

u^V_4
```
FOR $root IN document("V3.xml"),
     $book IN $root/book_info
WHERE $book/title/text() = " Data on the Web"
AND $book/price_info/website = " www.amazon.com "
UPDATE $root {
     DELETE $book }
```

u^V_5
```
FOR $root IN document("V4.xml"),
     $book IN $root/book_info
WHERE $book/title/text() = " Data on the Web"
AND $book/price_info/website = " www.amazon.com "
UPDATE $root {
     DELETE $book }
```

u^V_6
```
FOR $root IN document("V4.xml"),
     $book IN $root/book_info
WHERE $book/title/text() = " Data on the Web“
AND $book/price_info/website = " www.amazon.com "
UPDATE $root {
     DELETE $book/title }
```

u^V_7
```
FOR $root IN document("V5.xml"),
     $book IN $root/book_info
WHERE $book/title/text() = "Data on the Web"
AND $book/price_info/website = " www.amazon.com "
UPDATE $root {
     DELETE $book }
```

u^V_8
```
FOR $root IN document("V5.xml"),
     $book IN $root/book_info
WHERE $book/title/text() = "Data on the Web"
AND $book/price_info/website = " www.amazon.com "
UPDATE $root {
     DELETE $book/price_info}
```

u^V_9
```
FOR $root IN document("V4.xml")
UPDATE $root {
   INSERT
      <book_info>
         <bookid>"98003"<bookid>
         <title>" Data on the Web "</title>
         <price_info>
            <amount>56.00</amount>
            <website>www.ebay.com</website>
         </price_info>
      </book_info> }
```

u^V_{10}
```
FOR $root IN document("V5.xml")
UPDATE $root {
   INSERT
      <book_info>
         <bookid>"98004"<bookid>
         <title>" Database System "</title>
         <price_info>
            <bookid>"98003"<bookid>
            <amount>56.00</amount>
            <website>www.amazon.com</website>
         </price_info>
      </book_info> }
```

u^V_{11}
```
FOR $root IN document("V5.xml")
LET $book := $root/book_info
WHERE $book/title/text() = " Data on the Web"
UPDATE $book {
  INSERT
     <price_info>
        <bookid>"98003"<bookid>
        <amount>56.00</amount>
        <website>www.ebay.com</website>
     </price_info> }
```

u^V_{12}
```
FOR $root IN document("V5.xml")
LET $book := $root/book_info
WHERE $book/title/text() = " Data on the Web"
UPDATE $book {
  INSERT
     <price_info>
        <bookid>"98004"<bookid>
        <amount>56.00</amount>
        <website>www.ebay.com</website>
     </price_info> }
```

Fig. 5. Update operations on XML views defined in Fig.4

$\mathcal{R}(\mathcal{N},\mathcal{A},\mathcal{F})$	the schema of a relation
R	a relation
$\mathcal{SCH}^{\mathcal{R}}$	the schema of a relational database
D	a relational database
u^R	a relational update operation
U^R	a sequence of relational update operations
$\mho^R$	the domain of relational update operations
$\mathcal{SCH}^{\mathcal{V}}$	the view schema
V	the view instance
DEF^V	the view definition
u^V	a view update
$\mho^V$	the domain of view update operations
v^C	a complete view element
v^P	a partial view-element

Table 1. Notations for XML view update problem

expression. Let $Exp_k^{for}, Exp_k^{let}, Exp_k^{where}, Exp_k^{return}$ denote the FOR, WHERE, LET, RETURN clauses in the kth expression $Exp_k (0 \leq k \leq m)$ respectively. The element structure defined by Exp_0^{return} is named the **complete view-element**, denoted as v^C. Any element or attribute defined inside Exp_0^{return} is called a **partial view-element**, denoted as v^P. For example, Q_3 in Fig.4 can be described as $DEF^V = Exp_0(Exp_1)$, where Exp_0 = "FOR \$book ..." and Exp_1 = "FOR \$price ...". The complete view-element here is *<book> ... < /book>*. All other elements or attributes inside the "book" element, such as *bookid* and *price*, are partial view-elements. A *complete view-element* is actually a special *partial view-element*.

Let $\oplus$ be the operator used to compose view-element into a hierarchy, while $\ominus$ is the operator used to decompose view-elements from a hierarchy. Then two view-elements can construct a "bigger" view-element by the $\oplus$ operator, while a "bigger" view-element can be decomposed into "small" parts using operator $\ominus$. For example, let v_1^P and v_2^P be *amount* and *website* view-elements in $V3$ of Fig.4. Then $v^P = v_1^P \oplus v_2^P$ is the *price* view-element. While $v^P \ominus v_1^P = v_2^P$. A view thus can be viewed as a construction of complete view-elements, denoted as $V = v_1^C \oplus ... \oplus v_m^C$.

An **XML view schema** $\mathcal{SCH}^{\mathcal{V}}$ is extracted from both the view definition DEF^V and the underlying relational database schema $\mathcal{SCH}^{\mathcal{R}}$. It is a combination of a **filtered XML view schema (FSchema)** and an **extracted XML view schema (ESchema)**. An *FSchema*, modeling the constraints extracted from the schema of any relation referenced by DEF_V, is inferred by analyzing the underlying relational database schema and projecting it using the view definition. These constraints may include domain constraints, cardinality constraints, Null constraints and hierarchical constraints. An *ESchema* consisting of constraints that can be inferred from the XML view query specification syntax is extracted by analyzing the view query expression. It includes cardinality constraints, hierarchical constraints and duplication constraints. Figure 9 is the the comparison of the constraints between the original XML schema, XML view schema $\mathcal{SCH}_{\mathcal{V}}$ and the base relational schema $\mathcal{SCH}_{\mathcal{R}}$.

Some constraints in the view schema (domain, etc.) can be used to check if the given view updates are valid. Figure 10 is examples of valid update checking using Null constraints and domain constraints. As shown in Figure 10(a), assuming we have constraints for relational database in Figure 2, then u_{13} in Figure 10(b) is not a valid update because it againsts the NOT NULL constraints for "title" in "book" relation. u_{13} in Figure 10(c) is not a valid update either because it againsts the domain constraints of "price" in "price" relation.

Some constraints may affect the update translatability of the view, in particular, when the constraints extracted from DEF^V and $\mathcal{SCH}^{\mathcal{R}}$ do not match each other. The effect of these constraints in XML view schema on the update translation has been featured by motivation examples in Section 2, and will be discussed in detail in Section **??**.

3.2 XML View Update Operations

Let u^V be an update operation on the view with $\mho^V$ as its domain. Let *type* be a function to identify the type of an update operation to be delete, insert or replace.

Definition 1 *A* **complete update** u^V *on a given view* $V = v_1^C \oplus ... \oplus v_n^C\}$ *is either a complete insertion, deletion or replacement, defined as below.*

Let $type(u^V) = insert$*, and* v^C *be the complete view-element inserted by* u^V*. Then* $u^V(V) = v_1^C \oplus ... \oplus v_n^C \oplus v^C$ *is called a* **complete insertion***.*

Let $type(u^V) = delete$*, and let* v_k^C $(1 \leq k \leq n)$ *be a complete view-element deleted by* u^V*. Then* $u^V(V) = v_1^C \oplus ... \oplus v_{k-1}^C \oplus v_{k+1}^C \oplus ... \oplus v_n^C$ *is called a* **complete deletion***.*

```
<bib>
    <book>
        <bookid>98003</bookid>
        <title>Data on the Web</title>
        <price>
            <amount>56.00</amount>
            <website>www.amazon.com</website>
        </price>
    </book>
</bib>
```

(a) $V1'$

```
u1R:  DELETE FROM book
      WHERE book.ROWID IN (
        SELECT DISTINCT book.ROWID FROM book
        WHERE (book.title = 'TCP/IP Illustrated') )
```

```
u2R:  DELETE FROM price
      WHERE price.ROWID IN (
        SELECT DISTINCT price.ROWID FROM book,price
        WHERE (book.title = 'TCP/IP Illustrated') AND
                   (book.bookid = price.bookid) )
```

(b) U^R

book

bookid	**title**
98002	Programming in Unix
98003	Data on the Web

price

bookid	**amount**	**website**
98003	56.00	www.amazon.com
98003	45.60	www.bookpool.com

Legend:
▨ **Primary Key**
☐ **Non Key**

(c) D'

(d) $Q1(D')$. Same with (a).

Fig. 6. Translation for u_1^V (a) $V1'$: The expected XML view after update $u_1^V(V1)$ (b) U^R: The translated update (c) D': The updated relational database and (d) $Q1(D')$: The regenerated view.

```
<bib>
    <price>
        <amount>63.70</amount>
        <website>www.amazon.com</website>
    </price>
    <price>
        <amount>56.00</amount>
        <website>www.amazon.com</website>
        <book>
            <bookid>98003</bookid>
            <title>Data on the Web</title>
        </book>
    </price>
</bib>
```

(a) $V2'$

(b) U^R. Same with Fig.6(b).

(c) D'. Same with Fig.6(c).

```
<bib>
    <price>
        <amount>56.00</amount>
        <website>www.amazon.com</website>
        <book>
            <bookid>98003</bookid>
            <title>Data on the Web</title>
        </book>
    </price>
</bib>
```

(d) $Q2(D')$

Fig. 7. Translation for u_2^V (a) $V2'$: The expected XML view after update $u_2^V(V2)$ (b) U^R: The translated update (c) D': The updated relational database and (d) $Q2(D')$: The regenerated view.

```
<bib>
    <price>
        <amount>56.00</amount>
        <website>www.amazon.com</website>
        <book>
            <bookid>98003</bookid>
            <title>Data on the Web</title>
        </book>
    </price>
</bib>
```

(a) $V2''$

(b) U^R. Same with Fig.6(b).

(c) D'. Same with Fig.6(c).

(d) $Q3(D')$. Same with (a).

Fig. 8. Translation for u_3^V (a) $V2''$: The expected XML view after update $u_3^V(V2)$ (b) U^R: The translated update (c) D': The updated relational database and (d) $Q2(D')$: The regenerated view.

Let type$(u^V) = $ *replace, and let* v_k^C $(1 \leq k \leq n)$ *be a complete view-element to be replaced, and* v^C *be the complete view-element used as replacement. Then* $u^V(V) = v_1^C \oplus ... \oplus v_{k-1}^C \oplus v_{k+1}^C \oplus ... \oplus v_n^C \oplus v^C$ *is called a* **complete replacement**.

Definition 2 *A* **partial update** u^V *on a given view* $V = v_1^C \oplus ... \oplus v_n^C$ *is either a partial insertion, a partial deletion or a partial replacement, defined as below.*

Let type$(u^V) = $ *insert, and let* v_k^P *be the partial view-element inserted into* v_k^C *by* u^V*. Let* $v_k^{C'} = v_k^C \oplus v_k^P$*. Then* $u^V(V) = v_1^C \oplus ... \oplus v_k^{C'} \oplus ... \oplus v_n^C$ *is called a* **partial insertion**.

Let type$(u^V) = $ *delete, and let* v_k^P $(1 \leq k \leq n)$ *be a partial view-element deleted from* v_k^C *by* u^V*. Let* $v_k^{C'} = v_k^C \ominus v_k^P$*. Then* $u^V(V) = v_1^C \oplus ... \oplus v_k^{C'} \oplus ... \oplus v_n^C\}$ *is called a* **partial deletion**.

Let type$(u^V) = $ *replace, and let* $v_{k_j}^P$ $(1 \leq k \leq n, 1 \leq j \leq m)$ *be a partial view-element to be replaced within* $v_k^C = v_{k_1}^P \oplus ... \oplus v_{k_m}^P$*, and* v^P *be the replacement partial view element. Let* $v_k^{C'} = v_{k_1}^P \oplus ... \oplus v_{k_{j-1}}^P \oplus v^P, v_{k_{j+1}}^P \oplus ... \oplus v_{k_m}^P$*. Then* $u^V(V) = v_1^C \oplus ... \oplus v_k^{C'} \oplus ... \oplus v_n^C$ *is called a* **partial replacement**.

A *complete insertion* adds to while a *complete deletion* removes from the XML view a *complete view-element*. A *complete replacement* replaces a *complete view-element* with a replacing complete view-element. A *partial insertion* adds to while a *partial deletion* removes a *partial view-element* from an XML view. A *partial replacement* replaces a partial view-element with a replacing view-element. A *valid view update* u^V is a partial or a complete view update that $u^V(V)$ satisfies all the constraints in the view schema $\mathcal{SCH}^\mathcal{V}$. For instance, u_2^V in Fig.5 is a complete update while u_3^V is a partial one. Both of them are valid updates since both $u_3^V(V)$ and $u_2^V(V)$ agree with all the constraints of the view schema.

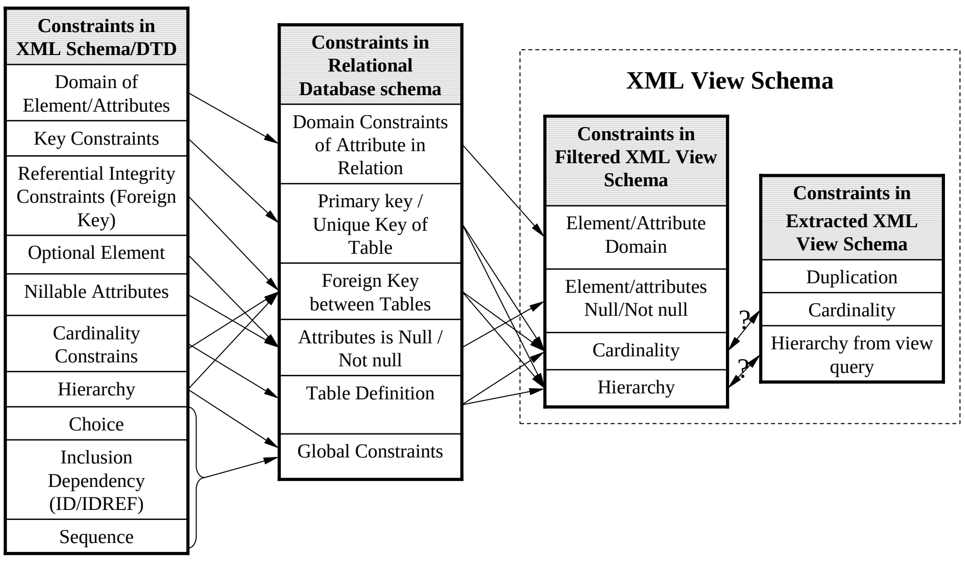

Constraints in XML Schema/DTD
Domain of Element/Attributes
Key Constraints
Referential Integrity Constraints (Foreign Key)
Optional Element
Nillable Attributes
Cardinality Constrains
Hierarchy
Choice
Inclusion Dependency (ID/IDREF)
Sequence
Constraints in Relational Database schema
Domain Constraints of Attribute in Relation
Primary key / Unique Key of Table
Foreign Key between Tables
Attributes is Null / Not null
Table Definition
Global Constraints
XML View Schema
Constraints in Filtered XML View Schema
Element/Attribute Domain
Element/attributes Null/Not null
Cardinality
Hierarchy
Constraints in Extracted XML View Schema
Duplication
Cardinality
Hierarchy from view query
?
?

Fig. 9. Comparison of constraints of XML schema, relational database schema and XML view schema

```
Constraints for Book relation:
   title VARCHAR2(100) NOT NULL

Constraints for price relation:
   amount DOUBLE CHECK (amount > 0.00)
```

(a)

$\mathbf{u^V}_{13}$

```
FOR $root IN document("V1.xml"),
      $book IN $root/book
UPDATE $root {
      DELETE $book/title
}
```

(b)

$\mathbf{u^V}_{14}$

```
FOR $root IN document("V1.xml")
UPDATE $root {
    INSERT
        <book>
            <bookid>"98004"<bookid>
            <title>" Data on the Web "</title>
            <price>
                <amount> -100.00 </amount>
                <website>www.ebay.com</website>
            </price>
        </book>
 }
```

(c)

Fig. 10. Example of valid update checking using constraints in the view schema

3.3 The XQuery View Update Problem

Similar with [2], the correctness criteria of a XML view update translation is defined below.

Definition 3 *Given a relational database D and a view V defined by DEF^V. A relational update sequence U^R is a* **correct translation** *of u^V iff (a) $u^V(DEF^V(D)) = DEF^V(U^R(D))$ and (b) if $u^V(DEF^V(D)) = DEF^V(D) \Rightarrow U^R(D) = D$. Then u^V is said to be* **translatable** *for V, and V is said to be* **updatable** *by u^V.*

First, a correct translation means the "rectangle" rules shown in Fig.11 hold. Intuitively, it implies the translated relational updates "exactly" perform the view update, that is, without view side effects. Second, an update operation does not affect the view, then it should not affect the relational base either. This will guarantee that any modification of the relational base is indeed done for the sake of the view.

$$\begin{tikzcd}
V \arrow[r, "(2)\ u^V"] & u^V(V) \\
S \arrow[u, "(1)\ DEFv"] \arrow[r, "(3)\ \mathbf{U^R}"'] & \mathbf{U^R}(S) \arrow[u, "(4)\ DEFv"']
\end{tikzcd}$$

Fig. 11. Correct translation of view update to relational update

The **XML view update problem** is to determine whether both of the following claims hold: (a) $\forall u^V \in \mho^V$, ((a) $\exists U^R \subset \mho^R$ such that U^R is a correct translation of u^V by Definition 3) and ((b) $\forall U^{R'} \subset \mho^R$, $U^{R'}$ is not a correct translation of u^V). That is whether an update on the view is unambiguously translatable. In our case, the view is defined by XQuery, thus named *XQuery view update problem.*

4 Theoretical Foundation for XQuery View Updatability

In order to bridge the hierarchical feature of XML and flat structure of relational schema in BCNF, the integrity constraints, such as foreign keys, have to be considered. The integrity constraints introduce additional update operations into the translated update sequence besides the "original" update operations. Those additional ones are called *propagated updates.* For example, the second update operation u_2^R in U_2^R (Fig.7(c)) is a propagated update operation. Considering integrity constraints makes the view update problem harder because such generated updates may cause view side effects.

Much work has been done on the existence of a correct translation for various classes of view specifications [2, 9]. [9] shows that a correct translation exists in the case of a "clean source" when only functional dependencies inside a single relation are considered. We now adopt and extend this work in the context of XML views. That is, when functional dependencies exist not only inside a relation, but also between relations, how to decide whether a correct relational translation of a given XQuery view update exists.

Definition 4 *Given a relational database D. Let $V = v_0^C \oplus ... \oplus v_m^C$ be an XML view defined by DEF^V over several relations $rel(DEF^V) \subseteq D$. If $g = \{t_{i_x} \mid t_{i_x} \in R_x$ for each $R_x \in rel(DEF^V)\}$ generates a complete view-element v_k^C by applying the view definition DEF^V, then g is called a* **generator** *of $v_k^C (1 \leq k \leq m)$. $t_{i_x} \in g$ is called a* **source tuple** *in D of v_k^C.*

Further, let v_k^P be a partial view-element inside v_k^C, we have: (1) Let $g^P \subseteq g$ denote the "portion" of g that is the source-tuple of v_k^P. Then g^P is called the **generator** *of the partial view-element v_k^P. (2) $t_{i_y} \in g - g^P$ is an* **extended source tuple** *in D of v_k^P iff $\exists t_{i_x} \in g^P$, $t_{i_x}.a_j$ is a foreign key of $t_{i_y}.a_l$, where $a_j \in \mathcal{R}_x(\mathcal{A})$ and $a_l \in \mathcal{R}_y(\mathcal{A})$. (3) Let $T_e = \{t_{i_y} | t_{i_y} \in g - g^P$ is an extended source tuple of $v_k^P\}$ denote the set of all the extended source tuples of v_k^P. Then $g_e^P = g^P \cup T_e$ is called the* **extended generator** *of v_k^P. The extended generator of a complete view-element is always the same as its generator.*

A *source tuple* corresponds to a tuple that is used to compute a single complete or partial view-element. For instance, in $V1$ of Figure 4, the first complete view-element v_1^C is *book* with "bookid=98001". Let R_1 and R_2 be the *book* and *price* relations respectively, then the generator g of v_1^C is $g = (t_{0_1}, t_{0_2})$ where $t_{0_1} \in R_1$ is the book tuple *(98001,TCP/IP Illustrated)* and $t_{0_2} \in R_2$ is the price tuple *(98001, 63.70, www.amazon.com)*. Let the partial view-element v_1^P be the title of this book. Then the source tuple of v_1^P is t_{0_1}, while t_{0_2} is an extended source tuple of v_1^P. $g^P = (t_{0_1})$ is the generator of v_1^P, while $g = (t_{0_1}, t_{0_2})$ is the extended generator of v_1^P.

Definition 5 *Let V^P be part of a given XML view V. Let $G(V^P)$ be the set of generators of V^P defined by $G(V^P) = \{g \mid g$ is a generator of $v \in V^P$, where v is a complete or a partial view-element of $V^P\}$. For each $g = (t_{i_1}, ..., t_{i_p}) \in G(V^P)$, let $H(g)$ be some nonempty subset of $\{(t_{i_k}; R_k) \mid t_{i_k} \in R_k$ for each $t_{i_k} \in g\}$, then any superset of $\cup_{g \in G(V^P)} H(g)$ is a* **source** *in D of V^P. If $G(V^P) = \emptyset$, then V^P has no source in D.*

Similarly, let $G_e(V^P)$ be the set of extended generators of V^P, any superset of $\cup_{g \in G_e(V^P)} H(g)$ is an **extended source** *in D of V^P, denoted by S_e.*

A *source* includes the underlying relational part of a view "portion" V^P which consists of multiple view-elements. For example, let $V^P = V1$ (Fig.4), $G(V^P) = \{g_0, g_1\}$, where $g_0 = \{$*(98001, TCP/IP Illustrated),(98001, 63.70,www.amazon.com)*$\}$,$g_1 = \{$*(98003,Data on the Web),(98003,56.00,www.amazon.com)*$\}$. That is, $G(V^P)$ includes all the generators for view elements in V^P. Let $H(g_0) = \{$*((98001, TCP/IP Illustrated);book)*$\}$ and $H(g_1) = \{$*((98003, 56.00, www.amazon.com);price)*$\}$. Then $\{$*((98001, TCP/IP Illustrated);book),((98003, 56.00, www.amazon.com);price)*$\}$ is a source of V^P. The extended source is same as its source for V^P here.

Definition 6 *Let relational database $D = \{R_1, ..., R_n\}$. Let V^P be part of a given XML view V and S_e be an extended source in D of V^P. S_e is a* **clean extended source** *in D of V^P iff $(\forall v \in V \ominus V^P)$, $(\exists S_e')$ such that S_e' is an extended source in $(R_1 - S_{e1}, ..., R_n - S_{en})$ of v). Or, equivalently, S_e is a clean extended source in D of V^P iff $(\forall v \in V \ominus V^P)(S_e$ is not an extended source in D of $v)$.*

A *clean source* of a given view element defines a source that is not referenced by any other view part besides the given view element itself. For instance, given the partial view-element v_1^P in $V1$ (Fig.4) representing the title of book with bookid "98001", the extended source $\{$*(98001, TCP/IP Illustrated),(98001, 63.70,www.amazon.com)*$\}$ is not a *clean extended source* since it is also an extended source of the "price" element. Note that for a complete view-element, the *extended source tuple* is equal to its *source tuple*, while the *source* is the same as its *extended source*.

As proven in [9], the concepts of source and clean source are essential for the relational view updating problem. In the XML context, the connection between *clean extended source* and update translatability also exists as illustrated by the following theorems. Although somewhat similar to [9], these theorems actually differ in several ways, most notably the operation on an XML view element with various granularity instead of just a flat relational view tuple. Proofs are available in the appendix. Here we assume view update u^V is a valid view update and the relational update sequence U^R keeps the relational database schema valid. In addition we also choose to map view updates of one type (insertions, deletions) to relational updates of the same type.

Lemma 1, 2 and 3 proved below are used to prove Theorem 1 and Teorem 2.

Lemma 1

(a) S_e is an extended source in D of V^P iff $DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n}) \subseteq V \ominus V^P$.

(b) S_e is a clean extend source in D of V^P iff $DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n}) = V \ominus V^P$.

Proof.

(a) *If.* Suppose $DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n}) \subseteq V \ominus V^P$ but S_e is not an extended source in D of V^P.

Let $G(V^P)$ be the set of generators of V^P. From definition 5, $(\exists (t_{i_1}, ..., t_{i_q}) \in G(V^P)$ be a generator of $v \in V^P)(\forall R_k \in rel(DEF^V))[(t_{i_k}; R_{i_k}) \notin S_e]$. That is, $\forall R_k \in rel(DEF^V), t_{i_k} \in R_i - S_{e_i}$. Thus $v \in DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n})$.
But, $(t_{i_1}, ..., t_{i_p})$ is a generator of $v \in V^P$. That is $v \notin V \ominus V^P$.
Hence, $v \in DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n})$ and $v \notin V \ominus V^P$.
A contradiction with the hypothesis that $DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n}) \subseteq V \ominus V^P$.

Only if. Suppose S_e is an extended source in D of V^P but $DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n}) \not\subseteq V \ominus V^P$.

Then, $\exists v$ such that $(v \in DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n})) \wedge (v \in V^P)$.
This implies that there is a generator $(t_{i_1}, ..., t_{i_p})$ of $v \in V^P$ such that $\{(t_i; R_i) \mid R_i \in rel(DEF^V)\} \cap S_e = \emptyset$, contradicting the hypothesis that S_e is an extended source in D of V^P.

(b) *If.* Suppose $DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n}) = V \ominus V^P$ but S_e is not a clean extended source in D of V^P.

From (a), S_e is an extended source in D of V^P. By Definition 6, $(\exists v \in V \ominus V^P)$ such that there is no source in $(R_1 - S_{e_1}, ..., R_n - S_{e_n})$ of v. By definition 5, this implies that there is no generator $g \in \prod_{R_i \in rel(DEF^V)}(R_i - S_{e_i}$ of v, and hence $v \notin DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n})$, a contradiction.

Only If. Assume that S_e is a clean source in D of V^P.

By (a), $DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n}) \subseteq V \ominus V^P$. Assuming $V \ominus V^P \nsubseteq DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n})$, that is, $(\exists v \in V \ominus V^P) such that (v \notin DEF^V(R_1 - S_{e_1}, ..., R_n - S_{e_n}) \subseteq V - V^P)$. Then there is no generator $g \in \prod_{R_i \in rel(DEF^V)}(R_i - S_{e_i})$ of v. Hence, by Definition 5, there is no source in source in $(R_1 - S_{e_1}, ..., R_n - S_{e_n})$ of $v\}$ and therefor no source in $(R_1 - S_{e_1}, ..., R_n - S_{e_n})$ of $V \ominus V^P$, which contradicts the gypothesis that S_e is a clean source in D of V^P. □

Lemma 2 *Given a view V defined by DEF^V over D. Let u^V and U^R be updates on V and D (respectively). Let $v \in V$. Then (U^R deletes an extended source of v and U^R does not insert an extended source-tuple of v) iff $v \notin DEF^V(U^R(D))$.*

Proof. Let $R_i' = U^R(R_i)$ be the updated relation $R_i \in rel(DEF^V)$. Let $T = D - U^R(D)$.

U^R deletes an extended source of $v \in V$
$\iff T$ is an extended source in D of v
$\iff DEF^V(R_1 - T_1, ..., R_n - T_n) \subseteq V \ominus v$ by lemma 1
$\iff v \notin DEF^V(R_1 - T_1, ..., R_n - T_n)$
$\overset{(1)}{\iff} \nexists\, extended - generator \in \prod_{R_i \in rel(DEF^V)}(R_i \cap T_i)$ of v since $R_i - T_i = R_i \cap R_i'$.

U^R does not insert an extended source-tuple of $v \in V$
$\overset{(2)}{\iff} \forall R_j \in rel(DEF^V)\ \forall t_{i_j} \in R_j' - R_j$
$\nexists\, t_{i_k} \in R_k' - R_k$, $R_k \in rel(DEF^V)$, $k \neq j$,
such that $(t_{i_1}, ..., t_{i_p})$ is an extended-generator of v.

(1) and (2) hold iff there is no extended-generator in D of v. The proposition then follows. □

Lemma 3 *Let u^V, U^R, V, D be as in Lemma 2. Let $v \in dom(V) - V$, where $dom(V)$ be all the possible valid view elements of V. Then U^R inserts a source-tuple of v iff $v \in DEF^V(U^R(D))$.*

Proof.

U inserts an source-tuple of v
$\overset{(1)}{\iff} (\exists R_i \in rel(DEF^V))(\exists t \in R_i' - R_i)$($t$ is a source tuple in $U^R(D)$ of V)
$\overset{(2)}{\iff} (\exists g = (t_{i_1}, ..., t_{i_p}) \in \prod_{R_i \in rel(DEF^V)} R_i')$($g$ is a generator of v).
$\overset{(3)}{\iff} v \in DEF^V(R_1', ..., R_n') = DEF^V(U^R(D))$.

(3) is proved as below:
If. Follow directly from Definition 5.
Only If. Assume that $g = (t_{i_1}, ..., t_{i_p})$ is a generator of v, but that $\forall R_k \in rel(DEF^V)$, $t_{i_k} \in R_i$. Then $g \in \prod_{R_i \in rel(DEF^V)} R_i'$ and so $v \in DEF^V(R_1, ..., R_n) = DEF^V(U^R(D))$, a contradiction. □

Theorem 1 *Let u^V be the deletion of a set of view elements $V^d \subseteq V$. Let τ be a translation procedure, $\tau(u^V, D) = U^R$.*
(a) τ **translates** *u^V to D iff U^R deletes a source of V^d;*
(b) τ **correctly translates** *u^V to D iff U^R deletes a clean extended source of V^d.*

Proof.

(a) Since type(U^R)=delete, $\forall R_i \in rel(DEF^V)$, we have $R_i' \subseteq R_i$. So, $R_i' - R_i = T_i$ or $R_i' - R_i = \emptyset$. Hence, $(\forall v \in V^d)$(U^R does not insert a source-tuple of v). The proposition then follows from lemma 2.

(b) By lemma 1(b), U^R deletes a clean source of V^d
$\iff DEF^V(R_1 - T_1, ..., R_n - T_n) = V \ominus V^d = u^V(V)$
$\iff DEF^V(U^R(D)) = u^V(V)$
$\iff DEF^V(U^R(D)) = u^V(V)$, since $R_i - T_i = R_i'$
$\iff \tau$ exactly translates u^V to U^R. □

Theorem 1 illustrates that a correct delete translation is the one without any view side effect. This is exactly what a clean extended-source deletion achieves by Definition 6.

Theorem 2 *Let u^V be the insertion of a set of view elements V^i into V. Let $V^\cap = V \cap V^i$, $V^- = V^i - V$. Let τ be a translation procedure, $\tau(u^V, D) = U^R$. Then,*
(a) τ **translates** *u^V to D iff $(\forall v \in V^-)$(U^R inserts a source tuple of v);*
(b) τ **correctly translates** *u^V to D iff (i) τ translates u^V and (ii) $(\forall v \in dom(V) - (V^\cap \cup V^-))$($U^R$ does not insert a source tuple of v).*

Proof.

(a) $(\forall v \in V^-)$(U^R inserts a source-tuple of v)
$\overset{(1)}{\iff} V^- \subseteq DEF^V(U^R(D))$, by lemma 3.

Also, since type(U^R) = insert; so, $DEF^V(U^R(D)) \supseteq V$. Hence, we have:

(1) $\iff DEF^V(U^R(D)) \supseteq V \cup V^- \supseteq V^i$
$\iff \tau$ translate u^V to U^R.

(b) By Lemma 3, condition (i) iff $V^- \subseteq DEF^V(U^R(D))$. Also, since $type(u^V) = insert$ and $type(U^R) = type(u^V)$, $DEF^V(U^R(D)) \supseteq V \supseteq V^\cap$.
Hence, $V^\cap \cup V^- \subseteq DEF^V(U^R(D))$.

By Lemma 3, condition (ii) iff $(dom(V) - (V^- \cup V^\cap)) \cap (DEF^V(U^R(D))) = \emptyset$.
Hence, $DEF^V(U^R(D)) \subseteq V^- \cup V^\cap$.

Thus, condition (i) and condition (ii) iff $DEF^V(U^R(D)) = V^- \cup V^\cap = u^V(V)$, that is τ correctly translates u^V to U^R. □

Theorem 2 indicates a correct insert translation is a translation without duplicate insertion and extra insertion. Duplicate insertion (insert a source of V^-) is not allowed by BCNF, while extra insertion (insert source of $dom(V) - (V^\cap \cup V^-)$) will cause a view side effect. For example, for u_9^V in Fig.5, the translated update $U_9^R = \{$*Insert (98003,Data on the Web) into book, Insert (98003,56.00,www.ebay.com) into price*$\}$ is not a correct translation since it inserts a duplicate source tuple into *book*. While $U_9^{R'} = \{$*insert (98003,56.00,www.ebay.com) into price*$\}$ is a correct one.

5 Graph-based Algorithm for Deciding View Updatability

As illustrated in Example 1 of Section 2, the new update translation issues arise due to the distinct features of XML views. In this section, we will identify these characteristics of XML view by using a graph-based algorithm. The connection between these features and the concept of clean extended source introduced in Section 4 are also illustrated.

We assume the relational database is in the BCNF form without any cyclic dependency. This is because we are using functional dependencies and integrity constraints of the relational database (such as keys, foreign keys, etc.) to determine the update propagation strategy. We consider a view update u^V being an insert or a delete operation touching a *single* complete or partial view-element but not a set of view elements at this time.

For the view query DEF^V, we do not consider any aggregation and recursion. These operations make views non-updatable, as enunciated in [11]. Also, the expression Exp^{where} in DEF^V is a conjunction of *non-correlation* or *correlation* predicates defined as below. Let $\$b$ be a variable binding defined in $DEF^V = nest(Exp_0, ..., Exp_n)$ or an XPath which we associate with a default variable binding. Let $a_j \in \mathcal{R}_k(\mathcal{A})$ be the corresponding relational attribute of $\$b$. A *non-correlation predicate* has the form $\$b\ \theta\ c$, where $\theta \in \{=, \neq, <, \leq, >, \geq\}$ and $c \in dom(\mathcal{R}_i.a_j)$ is a constant value. A *correlation predicate* has the form $\$b_1\ \theta\ \$b_2$. For example, *\$price/website = "www.amazon.com"* is a non-correlation predicate while *\$book/bookid = \$price/bookid* is a correlation predicate.

5.1 The Graphic Representation of XQuery Views

Given an XQuery view V defined by DEF^V, we define a *View Trace Graph* $\mathcal{G}_T(V)$ and a *View Relationship Graph* $\mathcal{G}_R(V)$ to represent the hierarchical constraints in the *view schema* and the underlying *relational schema* respectively.

First, we say an attribute $a_j \in R_i(A)$ is said to be **exposed** in $DEF^V = nest(Exp_0, ..., Exp_m)$ by variable $\$b$ if and only if $\exists Exp_k^{return}$ such that $\$b \in Exp_l^{return} (0 \leq k \leq m)$. That is, $\$b$ appears in at least one of the RETURN clauses of the view definition query.

The view relationship graph $\mathcal{G}_R(V)$ of given XML view V is a directed graph with nodes $N_{\mathcal{G}_R}$ and edges $E_{\mathcal{G}_R}$ as follows. For each attribute $a_j \in R_i(A)$, if a_j is exposed in the DEF^V by some variable $\$b$, then there is a *leaf node* in $N_{\mathcal{G}_R}$ labeled by both the attribute name $R_i.a_j$ and the XPath of $\$b$. Every tagger structure in the view definition is represented by an *internal node* annotated by its tagger name. A leaf node is shown by a small circle $\circ$ while an internal node by a triangle $\triangle$. Given two nodes $n_1, n_2 \in N_{\mathcal{G}_R}$, each edge $e(n_1, n_2) \in E_{\mathcal{G}_R}$ with the direction from n_1 to n_2 means n_1 is a parent of n_2 in the view hierarchy. Figures 12 and 13 depict the view relationship graphs for $V1$ and $V2$ in Figure 4.

Definition 7 *We define the* **hierarchy implied in the relational model** *as follows:*

(1) Table vs. attributes. Given a relation schema $\mathcal{R}(\mathcal{N}, \mathcal{A}, \mathcal{F})$, *with* $\mathcal{A} = \{a_i | 1 \leq i \leq m\}$, *then* N *is called the* **parent** *of the attribute* a_i ($1 \leq i \leq m$).

(2) Key vs. foreign key. Given two relation schemas $\mathcal{R}_i(\mathcal{N}_i, \mathcal{A}_i, \mathcal{F}_i)$ *and* $\mathcal{R}_j(\mathcal{N}_j, \mathcal{A}_j, \mathcal{F}_j)$, *with foreign key constraints defined as* $PK(\mathcal{R}_i) \leftarrow FK(\mathcal{R}_j)$, *then* $\mathcal{N}_i$ *is the* **parent** *of* $\mathcal{N}_j$.

The view trace graph $\mathcal{G}_T(V)$ of a given XML view V is a directed graph with nodes $N_{\mathcal{G}_T}$ and edges $E_{\mathcal{G}_T}$ generated as follows. The leaf nodes of $\mathcal{G}_T$ are: (i) the union of all leaves of $\mathcal{G}_R$ and (ii) additional *leaf nodes* representing $\$b_1$ and $\$b_2$ if there is a correlation predicate $\$b_1\ \theta\ \$b_2$ in DEF^V. Specially, a leaf node labeled by the primary key attribute of a relation is called a *key node*. A key node is depicted by a black circle $\bullet$ in $\mathcal{G}_T$. An *internal node*, shown by a triangle $\triangle$, represents a relation. Each edge $e(n_1, n_2) \in E_{\mathcal{G}_T}$ with the direction from n_1 to n_2 means n_1 is a parent of n_2 by the hierarchy definition in the underlying relational database (Definition 7). An edge is labeled by its foreign key condition if it is generated by rule (2) in Definition 7. The view trace graphs of $Q1$ and $Q2$ are identical as shown in Fig.14, since both of them from same relation source.

5.2 Update Translation Policy

Based on the view relationship graph $\mathcal{G}_R$ and view trace graph $\mathcal{G}_T$ defined in Section 5.1, we now use the closure definitions in $\mathcal{G}_R$ and $\mathcal{G}_T$ to indicate the effect of an update on the view and relational database respectively. We notice that the *update translation policy* is essential to the view updatability since a given update may be translatable under one policy, while not under another. Hence the closure definition below is closely related with the selected update policy.

Policy for update type selection. (1) Same type. The translated update always has the same update type as the given view update. (2) Mixed type. Translated updates with different types are allowed.

Policy for deletions in maintaining referential integrity of relational database. (1) Cascade. The directly translated relational update cascades to update the referenced relation also. (2) Restrict. The relational update is restricted to the case where there are no referenced relations, otherwise, reject the view update. (3) No action. The relational update is performed as required, even if referential violation appears.

Policy for deletions in duplication. (1) Restrict. Reject view update if only part of duplications is deleted. (2) Zero reference deletion. No tuple is deleted if it is still referenced by any other part of the view.

Of course, the policy we listed above is not the only possible choice, they are merely the ones that are commonly required in practice. In our discussion below, when not stated otherwise, we will pick the most common used policy, that is, same update type, delete cascading and restrict duplication handle. If a different translation policy is used, such as a restrict policy in maintaining referential integrity, then the definition can be adjusted accordingly.

The *closure* of a node $n \in N_{\mathcal{G}_R}$, denoted by $n^+_{\mathcal{G}_R}$, is defined as all the leaf nodes which have a directed path starting from n. Two leaf nodes $n_i, n_j \in N_{\mathcal{G}_R}, 1 \leq i, j \leq | \mathcal{G}_R |, i \neq j$ are **equal**, denoted as $n_i = n_j$ if and only if the relation attribute labels in their respective node annotations are the same. The *closure* of a node $n \in N_{\mathcal{G}_T}$ is defined in the same manner as in $\mathcal{G}_R$, except for a *key node*. Each key node has the same closure as its parent node. The closure of a set of nodes N, denoted by N^+, is defined as $N^+ = \bigcup_{(n_i \in N)} n_i^+$.

5.3 View Construction Consistency

As illustrated in Example 1 of Section 2, the *construction consistency* is caused by the fact that the hierarchical constraints existing in the *XML view schema* and the *underlying relational schema* may conflict. As a result, $\mathcal{G}_T(V)$ and $\mathcal{G}_R(V)$ of a view V, which capture these two hierarchies respectively, can be distinct from one another. We distinguish between *consistent* and *inconsistent* nodes to describe if an update in $\mathcal{G}_R(V)$ will have the same effect in $\mathcal{G}_T(V)$.

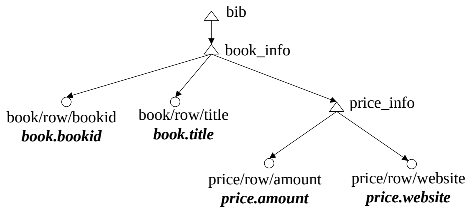


Fig. 12. View relationship graph for Q1

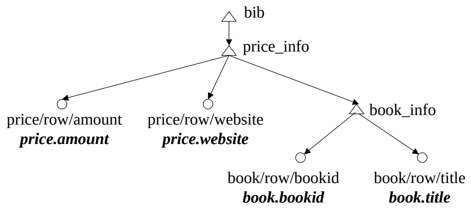


Fig. 13. View relationship graph for Q2

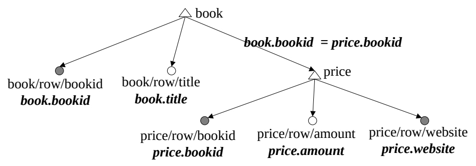


Fig. 14. View trace graphs for both Q1 and Q2

Definition 8 *Let $n \in V_{\mathcal{G}_R}$ be an internal node, with its closure in $\mathcal{G}_R$ denoted by $n^+_{\mathcal{G}_R}$. ($\forall n_i \in n^+_{\mathcal{G}_R}$), let $(n_i)^+_{\mathcal{G}_T}$ be the closure of n_i in $\mathcal{G}_T$. We say n is a* **consistent node** *iff $n^+_{\mathcal{G}_R} = \bigcup_{(n_i \in n^+_{\mathcal{G}_R})} (n_i)^+_{\mathcal{G}_T}$. Further, a view definition DEF^V is called a* **consistent construction** *if and only if all the internal nodes in $\mathcal{G}_R$ are consistent nodes, otherwise DEF^V is called an* **inconsistent construction**.

For a node to be *inconsistent* means that the effect of an update on the view side (node closure in $\mathcal{G}_R$) is different from the effect on the relational side (node closure in $\mathcal{G}_T$) based on the selected update policy (closure definition in $\mathcal{G}_T$). For instance, $V1$ in Fig.4 is a consistent construction, while $V2$ is an inconsistent construction. Because in the view relationship graph $\mathcal{G}_T(V2)$ in Fig.13, $(book)^+ = \{(book/row/bookid), (book/row/title)\}$, while in $\mathcal{G}_T(V2)$ shown in Fig.14, $\{(book/row/bookid)^+ \cup (book/row/title)^+\} = \{(book/row/bookid), (book/row/title), (price/row/bookid), (price/row/amount), (price/row/website)\}$. They are not equal. Hence "book" is not a consistent node. This is the reason why u_2^V is not translatable in Example 1.

The closure comparison between the view relationship graph $\mathcal{G}_R$ and the view trace graph $\mathcal{G}_T$ is important because it indicates the presence of *clean extended sources* defined by Definition 6. This connection between the concept of node consistency and clean extended source is formalized below.

Theorem 3 *Given V be a view defined on a relational database D with view relationship graph $\mathcal{G}_R(N_{\mathcal{G}_R}, E_{\mathcal{G}_R})$ and view trace graph $\mathcal{G}_T(N_{\mathcal{G}_T}, E_{\mathcal{G}_T})$. Let $Y \subseteq N_{\mathcal{G}_R}$, and $X \subseteq N_{\mathcal{G}_T}$. ($\forall$ generators $g, g' \in \Pi_{R_i \in rel(DEF^V)} R_i$ of view elements v and v' respectively, $g[X] = g'[X] \Rightarrow v[Y] = v'[Y]$) iff $X^+_{\mathcal{G}_T} = Y^+_{\mathcal{G}_R}$.*

Proof.

(1) *If.* Suppose $X^+_{\mathcal{G}_T} = Y^+_{\mathcal{G}_R}$. We shall prove the proposition by induction on the distance d from X to the root of the view trace graph. Assuming the depth of view trace graph is m.

Basis Step. For $d = m$, X includes the deepest leave nodes in view trace graph. $\forall y \in Y$, there are two cases:
(i) y is a leaf node. According to the node generation rules of view relationship graph and view trace graph, let $x \in X$ be the node in $\mathcal{G}_T$ corresponding to y. Then since $g[x] = g'[x]$, $v[y] = v'[y]$ follows trivially.
(ii) y is an internal node if all its children already in Y. Let $L_y \subseteq Y$ be all the leaf node rooted in y. Then according to (i), is true. $\forall n \in Y - L_y$, z is an internal node, which is a tag structure. Thus $v[z] = v'[z]$ is *true*. Hence we have $v[y] = v'[y]$ holds.
This complete the basis step.

Induction Hypothesis. Assume the proposition is true for all $d < k$. That is, if $Y^+_{\mathcal{G}_R} = X^+_{\mathcal{G}_T}$, then $\forall g, g'$, if $g[X] = g'[X]$, then $v[Y] = v'[Y]$ is true.

Induction Step. We shall demonstrate the proposition is true for $d = k$. $\forall x \in X$, there are two cases:

(i) x is a leaf node. According to the node generation rules of view relationship graph and view trace graph, let $y \in Y$ be the node in $\mathcal{G}_R$ corresponding to x. Then since $g[x] = g'[x]$, $v[y] = v'[y]$ follows trivially.

(ii) x is an internal node. Let $C_x = \{x_{i_1}, ..., x_{i_p}\}$ be the set including x's children. Then $x^+ = C^+_x$. Because of the induction hypothesis, the proposition is true for all the children of x. Then the proposition is true for x.

Hence, the proposition follows for all nodes with $d = k$.

(2) *Only If.* Suppose $X^+_{\mathcal{G}_T} \neq Y^+_{\mathcal{G}_R}$. According to the definition of view relationship graph and view trace graph, $\exists Y' \subseteq N_{\mathcal{G}_R}$ such that $Y'^+_{\mathcal{G}_R} = X^+_{\mathcal{G}_T}$. By (1), $\forall g, g'$, $g[X] = g'[X] \Rightarrow v[Y'] = v'[Y']$. Let $Y^\cap = Y \cap Y'$, then there is two cases:

(i) $Y^\cap \neq \emptyset$. Let $Y^- = Y - Y'$, then $Y^- \cap Y' = \emptyset$. This becomes same with case (ii).

(ii) $Y^\cap = \emptyset$. We make a "richness assumption". That is, the domain of the relational attributes are "rich" enough, so that their cardinalities do not impose any additional constraints on the view instance. By this assumption, we can find two generators g, g', $g[X] = g'[X]$ but $v[Y] \neq v'[Y]$. A contradiction with the proposition. □

Theorem 3 indicates that two equal generators always produce the identical view elements if and only if the respective closures of these two view element nodes in $\mathcal{G}_R$ and $\mathcal{G}_T$ are equal. Theorem 3 thus now enables us to produce an algorithm for detecting the *clean extended sources* S_e of a view element as stated below.

Theorem 4 *Let $D, V, \mathcal{G}_R, \mathcal{G}_T, Y$ be defined as in Theorem 3. Given a view element $v \in V(Y)$, there is a clean extended source S_e of v in D iff ($\exists X \subseteq N_{\mathcal{G}_T}$) such that $X^+_{\mathcal{G}_T} = Y^+_{\mathcal{G}_R}$.*

Proof.

(1) *If.* Suppose $X^+_{\mathcal{G}_T} = Y^+_{\mathcal{G}_R}$. Let $g[X]$ be an extended generator of v. By theorem 3, if $\exists v' \in V$ ($g[X]$ is an extended generator of v'), then $v'[Y] = v[Y]$. Hence, by Definition 6, $g[X]$ is a clean extended source of v.

(2) *Only if.* Suppose there is a clean extend source S_e of v in D, but $(\forall X \subseteq N_{\mathcal{G}_T})(X^+_{\mathcal{G}_T} \neq Y^+_{\mathcal{G}_R})$.

Then by theorem 3, there exist extended generators $g = \{t_{i_1}, ..., t_{i_p}\}, g' = \{t'_{i_1}, ..., t'_{i_p}\}$ of view tuples v, v' such that $g[X] = g'[X]$ but $v[Y] \neq v'[Y]$.
Let R_k be one of the relations referenced by $X^+ - Y^+$. Consider the database instance $D' = \{R'_{i_1}, ..., R'_{i_p}\}$ where R'_{i_j} defined as: (i) $R'_{i_j} = \{t_{i_j}\}$ if $R_{i_j}(A) \cap X \neq \emptyset$ when $R_{i_j} \in rel(DEF^V)$. That is some of attributes of R_{i_j} is in X. (ii) $R'_{i_j} = \{t_{i_j}, t'_{i_j}\}$ otherwise when $R_{i_j} \in rel(DEF^V)$.
Clearly, $S_e = \{(t_i; R'_i) \mid R'_i \in D'\}$ is an extended source in D' of v.
But S_e is not a clean extended source, because: when R'_k is

generated by case (i), $R'_k - S_{e_k} = \emptyset$, and so there is no source in $\{R'_{i_1}, ..., R'_{k-1}, R'_k - S_{e_k}, R'_{k+1}, ..., R'_{i_p}\}$ of v'. Further, since $R'_k = \{t_k\}$, there is no other source to consider.

Hence, there is no clean extended source S_e of v. A contradiction with the proposition. □

Theorem 4 indicates that a given view element has a clean extended source if and only if the corresponding portion in the view relationship is a consistent construction.

5.4 View Duplication

As illustrated by Example 3 in Section 1, duplication is another important factor influencing view updatability. There are two kinds of duplications that affect the updatability of a view, called **content duplication** and **structural duplication**. Example 3 illustrates both cases.

Definition 9 *A view definition DEF^V with view relationship graph $\mathcal{G}_R(N_{\mathcal{G}_R}, E_{\mathcal{G}_R})$ is free of* **structural duplication** *iff any two leaf nodes $n_i, n_j \in N_{\mathcal{G}_R}$ are not equal.*

Definition 10 *Given a structural duplication free view V, we say V is free of* **content duplication** *iff $(\forall v, v' \in V, v \neq v'$, and their source tuples $t_i, t'_i \in R_i)(t_i \neq t'_i)$.*

As discussed in Example 3, The structural duplication is not restricted to XML views only, but will not cause any ambiguity in relational context. The checking for structural duplication in an XML view can be comprehensive using the view relationship graph. A structural duplication arises as long as two nodes in the view relationship graph are equal (Fig. 15).

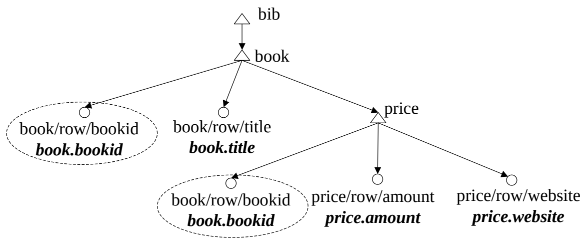


Fig. 15. VRG for Q5

Content duplication is not unique to the XQuery view update problem, although in the XML context it is generated by the nested XQuery syntax of a view definition. The same duplication may appear in the relational context for *Join* views. The *correlation predicate* in a view definition, however, does not necessarily generate content duplication. The generation of content duplication depends on filter conditions as well. $Q4$ in Example 3 is an example of content duplication. However, as noticed, $Q1$ and $Q2$ in Fig.4 do not generate any content duplication, although they also have nested FOR clauses. The reason is that we are using the schema knowledge of the relational database, that is "bookid" and "website" together serve as a primary key of relation *Price*. Hence the non-correlation predicate "*\$price/website = www.amazon.com*" in both $Q1$ and $Q2$ will filter the duplication.

By Definition 6, it is straightforward that a view element involved in any duplication does not have a clean extended source in the relational base. Updates on these view elements may cause problems as illustrated in the next section.

6 Translate View Updates with Various Granularity

Given the above connection we have established between the concepts of a clean extended source and the features of XQuery view definitions in Section 5, we now consider translatability for *valid updates* with various granularity.

6.1 Translatability of Deletions

The translatability of deletions in different XQuery views is described by the following observation. The cases are divided by using our graph-approach introduced in Section 5. The updatability of each case is now analyzed by using the extended clean-source deletion theorem (Theorem 1). The intuition behind is still the "clean extended-source" concept.

Observation 1 *Given a XQuery view definition DEF^V.*
Case 1*: If DEF^V is an inconsistent construction by Definition 8, then (a) A complete deletion is always translatable. (b) A partial deletion applied on any sub-tree rooted in an inconsistent node of VRG is not translatable.*
Case 2*: If DEF^V is a view with content duplication by Definition 10, then (a) A complete deletion is not translatable. (b) A partial deletion is not translatable if it touches only part of the duplications instead of all.*
Case 3*: If DEF^V is a view with structural duplication by Definition 9, then (a) A complete deletion is always translatable. (b) A partial deletion is not translatable if it touches any of the duplication.*
Case 4*: Otherwise, the set of translatable deletions is equal to the intersection of considering the above cases one by one.*

Example 4 *For case 1 in Observation 1. Given $Q2$ in Fig.4 defining a view with an inconsistent construction, then the complete deletion u_3^V in Fig.5 is translatable, while the partial deletion u_2^V, which is applied on the sub-tree rooted in an inconsistent node "book_info", is not translatable.*

Example 5 *For case 2 in Observation 1. Given $Q4$ in Fig.4 defining a view with a content duplication. The complete deletion u_5^V in Fig.5 is not translatable. The partial deletion u_6^V in Fig.5 is not translatable either since it touches duplication "\$book/title". However, if the delete operation is changed to delete "\$book/price" (no longer a duplicate part), then this partial update operation is translatable.*

Example 6 *For case 3 in Observation 1. Given $Q5$ in Fig.4 defining a view with a structural duplication of "bookid", the complete deletion u_7^V (Fig.5) is translatable, while the partial deletion u_8^V in Fig.5 is not translatable since it only touches one but not all parts of the duplication.*

6.2 Translatability of Insertions

The translatability of insertions is affected by an additional feature of XQuery views, namely, the **exposition completeness**. Similar to the relational view update problem [11], an updatable XML view requires the exposition of *primary key*, *non-correlation* and *correlation predicates* as explained below.

(1) Key exposition. The primary keys of the relations referenced by the view have to be exposed. This is because, for a partial insertion, the inserting partial view-element often includes just part of the attributes required by a relational tuple when mapping to the base relation. It has to be extended to become a "complete" tuple in order to be insertable into a relation. The primary key attributes have to be specified explicitly by the insert operation instead of taking the respective default values. Thus they have to be exposed.

(2) Predicate exposition. Attributes involved in a *non-correlation* or in a *correlation* predicate have to be exposed. Without predicates exposition, the validity of the insertion cannot be examined. We refer to these requirements as *complete exposition* as stated below.

Definition 11 *An XML view definition DEF^V is called a* **complete exposition** *iff the following hold: (i) $\forall R_i \in rel(DEF^V)$, let $PK(R_i)$ denote the set of primary key attributes of relation R_i, then this key $PK(\mathcal{R}_\rangle)$ is exposed in DEF^V. (ii) All attributes involved in any non-correlation or correlation predicate are exposed in DEF^V.*

Based on Theorem 2, we have following observations.

Observation 2 *An XQuery view defined by a query with an incomplete exposition is not insertable.*

Observation 3 *Given an XQuery view definition DEF^V with a complete exposition, we have:*
Case 1*: If DEF^V is a view with content duplication by Definition 10, then (a) A complete insertion is always translatable. (b) A partial insertion is not translatable if it touches any part of the duplication instead of all.*
Case 2*: If DEF^V is a view with structural duplication by Definition 9, then (a) Given a complete insertion with consistent values for the duplications, it is translatable. (b) Given a partial insertion, if it touches the duplication and has consistent values for all the duplications, then it is translatable.*
Case 3*: Otherwise, the set of translatable deletions is equal to the intersection of considering the above cases one by one.*

Example 7 *For case 1 in Observation 3. Given Q4 in Fig.4 defining a view with content duplication, the complete insertion u_9^V in Fig.5 is translatable. Partial insertion of only price element will be translatable, otherwise, not translatable.*

Example 8 *For case 2 in Observation 3. Given Q5 in Fig.4 defining a view with structural duplication on "bookid", the complete insertion u_{10}^V in Fig.5 is translatable since it has the same value for "bookid" within each "book_info" element. The partial insertion u_{11}^V is translatable since it also provides a consistent value for the duplicated "bookid" in all the "book_info" elements. The partial insertion u_{12}^V, however, is not translatable since it provides bookid="98004", which is inconsistent with existing bookid="98003" within the "book_info" element.*

7 An XQuery View Update System Framework

In general, the procedure of translating an update operation through an XML view can be divided into three separate but consecutive processes:

- *Information Preparation.* This process analyzes the XQuery view definition to provide us with prior knowledge about the relationship of the view with the relational database, that is, extracting the view schema. It also performs pre-checking of updates issued on the view to reject invalid or un-translatable updates using the *graph-based algorithm for deciding view updatability* proposed in Section 5.

- *Update Decomposition.* This is the key process of the XML update translation to bridge the XQuery model and the relational query model. The given XML update request is decomposed into a set of valid database operations, with each being applied to a single relation.

- *Global Integrity Maintenance.* Because of the structural model of the relational database with its integrity constraints, the database operations resulting from the *decomposition* process may need to be propagated globally throughout the base relations to assure the consistency of the relational database.

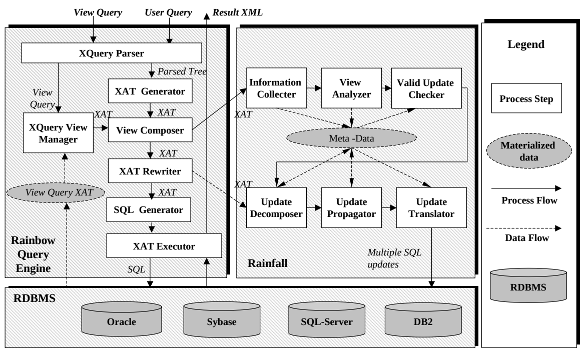


Fig. 16. Architecture of Rainbow query engine with update extension.

We hence call it a *decomposition-based update strategy.* The result of this paper is used in the pre-checking within the information preparation step. We have built an XQuery view update system named *Rainfall*, which is an extension of the base XML query engine *Rainbow* [19]. *Rainbow* is an XML data management system designed to support XQuery processing and optimization based on an XML algebra with the underlying data store being relational. Figure 16 depicts the architecture of the *Rainbow Query Engine* with our now proposed update extension named *Rainfall.* We have also finished several experiment on this system to address the performance in different scenarios, as presented in [17].

8 Related Work

[9] is the first work for studying the view updatability problem in the relational context. Based on the notion of *clean source*, it presents an approach to determine the existence of update translations by performing a careful semantic analysis of the view definition. The XQuery update problem discussed in this paper is more complex than that of a pure relational view update. Not only do all the problems in the relational context still exist in the XML semantics, but we also have to address the new update issues introduced the XML hierarchical data model and the nested XQuery syntax as described in Section 2.

[11, 12, 1] study the view update translation mechanism for SPJ queries on relations that are in Boyce-Codd Normal Form. These works have been further extended for object-based views in [4]. However, the XQuery FLWU expressions are different from SQL expressions used in these works. As an algebraic approach, the update translation strategy used by our system [18] bridges the gap between distinct query languages, and thus provide us with a clear solution for the view update translation task.

[15] presents an XQuery update grammar. It also studies the performance of executing a given update translation, assuming that the view is indeed translatable and has in fact already been translated using a fixed inlining shredding technique [14]. Instead of assuming the update is always translatable, our work addresses how the updatability is affected by the XML nested structure. And our solution is also not limited to any specific loading strategy.

[13] introduces the XML view update in SQL-Server2000, based on a specific *annotated schema* and update language called *updategrams*. Our update system thoroughly explores the characteristics of XML view update problem instead of focusing on a system-specific solution, although we have also implemented our ideas in the context of the Rainbow system [19] to check their feasibility.

The most recent work [5] studies the updatability of XML views using a *nested relational algebra*. By assuming the algebraic representation of view does not include any *Unnest* operators, while the *Nest* operator occurs only as last operator and would never affect the view updatability. However, the use of *Unnest* operator is unavoidable in an XQuery view definition over the default XML view mechanism. In addition, the order inside *Nest* operators will affect the view updatability since it decides the view hierarchy. In our paper, this issue is identified by the "view construction consistency". Its effects on view updatability are illustrated using the view trace graph and view relationship graph in Section 5.3.

9 Conclusion

In this paper, we have characterized the *XQuery View Update* in the context of XML views being published over relational databases. The *theoretical foundation* for translation existence is proposed by using the *extended clean-source* concept. The new update translation issues, caused by the characteristics of the XML hierarchical structure as well as nested XQuery syntax, have been identified. A graph-based algorithms for detecting the occurence of these issues in view definition have also been proposed. Its connection with the clean extended-source basis is also proven. The updatability of XQuery view on various update granularity is concluded according to different update types. As proof of viability, a system framework solving the XQuery view update problem has also been proposed and implemented within the XML data management system *Rainbow* [19]. Several experiments are also been conducted to assess various performance characteristics of our update solution [17].

Future extensions to our work could include considering element order in XML view updates, and studying index techniques for optimizing the efficiency in locating the updated nodes in the view, or locating the related tuples in the underlying database.

References

[1] A. M. Keller. The Role of Semantics in Translating View Updates. *IEEE Transactions on Computers*, 19(1):63–73, 1986.

[2] F. Bancilhon and N. Spyratos. Update Semantics of Relational Views. In *ACM Transactions on Database Systems*, pages 557–575, Dec 1981.

[3] S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, and R. Murthy. Oracle8i - The XML Enabled Data Management System. In *ICDE*, pages 561–568, 2000.

[4] T. Barsalou, N. Siambela, A. M. Keller, and G. Wiederhold. Updating Relational Databases through Object-Based Views. In *10th ACM SIGACT-SIGMOD*, pages 248–257, 1991.

[5] V. P. Braganholo, S. B. Davidson, and C. A. Heuser. On the Updatability of XML Views over Relational Databases. In *WEBDB*, 2003.

[6] M. J. Carey, J. Kiernan, J. Shanmugasundaram, E. J. Shekita, and S. N. Subramanian. XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents. In *The VLDB Journal*, pages 646–648, 2000.

[7] J. M. Cheng and J. Xu. XML and DB2. In *ICDE*, pages 569–573, 2000.

[8] S. S. Cosmadakis and C. H. Papadimitriou. Updates of Relational Views. *Journal of the Association for Computing Machinery*, pages 742–760, Oct 1984.

[9] U. Dayal and P. A. Bernstein. On the Correct Translation of Update Operations on Relational Views. In *ACM Transactions on Database Systems*, volume 7(3), pages 381–416, Sept 1982.

[10] M. F. Fernandez, A. Morishima, D. Suciu, and W. C. Tan. Publishing Relational Data in XML: the SilkRoute Approach. *IEEE Data Engineering Bulletin*, 24(2):12–19, 2001.

[11] A. M. Keller. Algorithms for Translating View Updates to Database Updates for View Involving Selections, Projections and Joins. In *Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 154–163, 1985.

[12] A. M. Keller. Choosing a View Update Translator by Dialog at View Definition Time. In *VLDB*, pages 467–474, 1986.

[13] M. Rys. Bringing the Internet to Your Database: Using SQL Server 2000 and XML to Build Loosely-Coupled Systems. In *VLDB*, pages 465–472, 2001.

[14] J. Shanmugasundaram, G. He, K. Tufte, C. Zhang, D. DeWitt, and J. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB*, pages 302–314, September 1999.

[15] I. Tatarinov, Z. G. Ives, A. Y. Halevy, and D. S. Weld. Updating XML. In *Proceedings of the ACM SIGMOD International Conference*, pages 413–424, May 2001.

[16] W3C. XQuery: A Query Language for XML. http://www.w3.org/TR/xquery/, February 2001.

[17] L. Wang, M. Mulchandani, and E. A. Rundensteiner. Updating XQuery Views Published over Relational Data: A Round-trip case study. In *XML Database Symposium(VLDB Workshop)*, pages 223–237, 2003.

[18] L. Wang, M. Mulchandani, and E. A. Rundensteiner. Updating XQuery Views Published over Relational Data. Technical Report WPI-CS-TR-03-23b, Computer Science Department, WPI, 2003.

[19] X. Zhang, K. Dimitrova, L. Wang, M. EL-Sayed, B. Murphy, L. Ding, and E. A. Rundensteiner. RainbowII: Multi-XQuery Optimization Using Materialized XML Views. In *Demo Session Proceedings of SIGMOD*, 2003.