RDB 上の XSLT 実体化ビューのインクリメンタルな更新手法

An Incremental Update Method for Materialized XSLT Views on RDBs

石川 佳治♡ 宮坂 集策♢
北川 博之♡

Yoshiharu ISHIKAWA Shusaku MIYASAKA
Hiroyuki KITAGAWA

XML の利用形態の一つとして，RDB 上に XML 出版機能を実現するアプローチがある．そこでは，XML データは RDB に対する一種のビューとして位置づけられる．本研究では，RDB 上の仮想的な XML ビューに対し，クライアントが XSLT を用いて導出ビューの定義を行う環境を想定する．導出ビューの定義に基づいて，実体化された XML データがクライアントに配信され，XML 実体化ビューとしてクライアント側で管理される．この場合，RDB に更新が発生したときの実体化ビューの更新が重要となる．そこで本稿では，RDB の更新の際の XML 実体化ビューの効率的な更新手法の提案を行う．提案手法では，与えられた XSLT によるビュー定義と RDB のスキーマおよび更新パターンの情報を解析し，XML 実体化ビューをインクリメンタルに更新するためのスクリプトを生成し，効率的な更新処理を実現する．

In information systems which provide XML documents, RDBs are often used for data storage and XML generation. XML documents in these systems can be seen as database views. In this paper, we assume an environment such that a client can define XML views using XSLT over a remote relational database and XML views are materialized on the client. We propose an efficient method for updating materialized XML views in an incremental manner. In our approach, the view management system analyzes a database schema and XSLT view definitions, and generates update scripts. When a new update occurs, the scripts are executed for XML view updates.

1. はじめに

XML の利用形態の一つとして，RDB 上に XML 出版機能を実現するアプローチがある．XML データは必要に応じて RDB のデータをもとに加工される．出版された XML データがクライアント側で永続的に蓄積・管理される場合，この XML データは RDB 上の一種の実体化ビューとみなすことができる．しかし，RDB における実体化ビューの管理と同様に，このようなアプローチにおいては更新処理への対応が問題となる．直接的なアプローチとしては，RDB に対する更新が発生するたびに XML 実体化ビューの再構築を行いクライアントに配信する手法が考えられるが，

- 1 回の配信当たりのデータ生成・転送量が大きい．
- RDB に対する更新が XML 実体化ビューに影響を与えない場合でも，XML 実体化ビューの再構築が行われる．

という問題点がある．したがって，XML 実体化ビューの効率的な更新処理手法の開発は重要な研究課題である．

本研究では，XML ビュー定義のために XSLT [1] を用いる．XSLT では，XSLT スタイルシートにより XML の変換処理を記述する．本研究で想定する環境では，クライアントはサーバから提供されるデフォルト XML ビュー（RDB 上に仮想的に提供される XML ビュー）に対して，それを加工する XSLT スタイルシートを記述する．XSLT スタイルシートを XML デフォルトビューに適用した結果が XML 実体化ビューとなる．実際には，XSLT スタイルシートと XML 実体化ビューの定義をもとに，サーバ上で RDBMS への問合せが生成され，その結果をもとに XML 実体化ビューが生成され，クライアントに配信される．

提案手法では，XSLT スタイルシートと RDB のスキーマおよび更新パターンを事前に解析することにより，RDB が更新されたときに実行される更新処理スクリプトを生成する．RDB に更新がなされたとき，その更新の内容に応じて，クライアントに差分データを配信することにより，効率的な XML 実体化ビュー更新を実現する．

RDB 上の XML ビューの構築に関する研究としては [2, 3, 4, 5] が挙げられる．そのうち [4, 5] は問合せ言語として XSLT の利用を想定している．[4] は XSLT スタイルシートの内容を解析し SQL 文を生成することで RDB 上に XML ビューを構築する．[5] は XSLT スタイルシートに対して解析を行った結果を独自の代数に変換し，XSLT の処理内容をできるだけ RDB にプッシュする手法を提案している．これらのいずれにおいても，実体化した XML ビューの更新については述べられていない．一方，RDB の実体化ビューのインクリメンタルな管理手法として [6] がある．実体化補助ビューの概念を用いることにより，サーバ内のデータベースに対する差分データの情報のみを利用して，クライアントが保持する実体化ビューの効率的な更新管理処理を実現するものである．クライアントは差分データの情報と補助実体化ビューの情報をもとに，インクリメンタルに実体化ビューの更新を行うことができる．本研究では，この研究を先に述べた XML 実体化ビューの更新処理のために拡張する．

2. 概要

2.1 システム環境

想定するシステム環境は，図 1 のようなクライアント・サーバ環境である．クライアントは XML に関する処理機能のみを持っているものとする．XML 実体化ビューの更新処理の流れは以下のようになる．

[ビュー定義登録時]

1. サーバは，データベースのスキーマ情報をもとにマッピングを行い，RDB 中のデータからデフォルト XML ビュー（実際には実体化されない）を生成しクライアントに提供する．
2. クライアントは提供されたデフォルト XML ビューに対して XML 導出ビューの定義を記述した XSLT スタイルシートを作成し，サーバに登録する．
3. サーバは登録された XML ビュー定義の内容を解析し，その時点における XML 実体化ビュー，補助ビュー，及び更新用スクリプトを作成し，クライアントに配信する．
4. 更新情報配信システム内において，それぞれのクライアントの要求に応じたビュー更新プランを構築する．

[データ更新発生時]

1. サーバはビュー更新処理プランをもとに，それぞれの 各クライアントの XML 実体化ビューに対応した更新用の差分データを配信する．

♡ 正会員 筑波大学システム情報工学研究科
{ishikawa,kitagawa}@cs.tsukuba.ac.jp
♢ 学生会員 筑波大学システム情報工学研究科
syusaku@kde.is.tsukuba.ac.jp

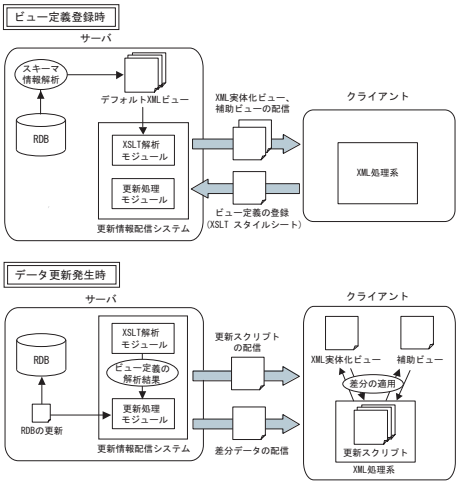

図 1: システム環境
Fig. 1: System Environment

2. クライアントは差分データを受け取ると，対応する更新スクリプトを用いて，まず補助ビューの更新を行い，次いで，XML 実体化ビューに対して差分データの内容を反映する．

2.2 具体例

2.2.1 データベースとデフォルト XML ビューの例

チェーン店の売上げ情報を持つデータベースを考える．図 2 はサーバの RDB 内に保持されるリレーション群である．下線はキー属性を示す．リレーション `store` は，それぞれの店舗の責任者と店舗のある地区の情報を保持している．`sale` は，1 つの売上げに対して 1 つの情報を，その年の情報とともに持つ．1 つの売上げは複数の商品を含み，レシートの 1 行に 1 つの商品が記載される．`line` はこのような情報を，ある売上げにおいて売れた商品それぞれについて保持する．また，`item` は商品に関する情報を保持している．さらに，このデータベースには，$sale.store\_id \Rightarrow store.store\_id$, $line.sale\_id \Rightarrow sale.sale\_id$, $line.item\_id \Rightarrow item.item\_id$ の 3 つの参照整合性制約が存在するとする．

```
store(store_id, state, manager)
sale(sale_id, store_id, year)
line(line_id, sale_id, item_id, sales_price)
item(item_id, name, category)
```

図 2: 想定するデータベース
Fig. 2: Sample Database

なお，本稿ではこのデータベースにおいて発生し得るデータ更新について，データの追加のみを考える．他の更新パターン（タプルの削除・更新）については今後の検討課題とする．

このデータベースに対する，あるデフォルト XML ビューの DTD を図 3 に示す．デフォルト XML ビューを導出するためのマッピング処理については，2.3 節でその概要を述べる．

```
<?xml version="1.0"?>
<!ELEMENT db (store*)>
<!ELEMENT store (store_id, state, manager, sale*)>
<!ELEMENT sale (sale_id, year, line*)>
<!ELEMENT line (line_id, sales_price, item_id, name, category)>
```

※ その他の要素は内容が#PCDATAとなる．ここでは省略する．

図 3: デフォルト XML ビューの DTD
Fig. 3: DTD for a Default XML View

2.2.2 ビュー定義の例

あるクライアントシステムのユーザが,「2004 年のカリフォルニアの店舗の玩具の売上げ」について興味を持っており，この情報を含む XML 実体化ビューを常時手元に保持していたいとする．このとき，ユーザは図 3 のデフォルト XML ビューをもとに，図 4 のような XML 実体化ビュー定義（XSLT スタイルシート）を記述し，サーバに登録することになる．

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="db/store">
    <xsl:if test="state='CA'">
      <store>
        <xsl:copy-of select="manager" />
        <xsl:for-each select="sale/line[../year='2004' and category='toy']">
          <sale>
            <xsl:copy-of select="../sale_id" />
            <xsl:copy-of select="line_id" />
            <xsl:copy-of select="sales_price" />
            <xsl:copy-of select="item_id" />
            <xsl:copy-of select="name" />
          </sale>
        </xsl:for-each>
      </store>
    </xsl:if>
  </xsl:template>

  <xsl:template match="/">
    <result><xsl:apply-templates /></result>
  </xsl:template>
</xsl:stylesheet>
```

図 4: ビュー定義の例
Fig. 4: Example of View Definition

このビュー定義では，3 行目の `xsl:apply-template` 文と 4 行目の `xsl:if` 文によって，`state` 要素の値が “CA” である店舗それぞれに対して処理を行うことを示している．7 行目の `xsl:for-each` 文によって `year` 要素の値が “2004” であり，かつ `category` 要素の値が “toy” であるような売上げデータを，店舗の責任者名や売上げ ID，販売価格等といった情報と共に出力することを指示している．

2.2.3 XML 実体化ビューと補助ビュー

図 4 のビュー定義をもとに，サーバは図 5 の XML 実体化ビューと図 6 の DTD で示される補助ビューを構築し，クライアントに配信する．

```
<?xml version="1.0"?>
<result>
  <store>
    <manager>Jim</manager>
    <sale>
      <sale_id>s3</sale_id><line_id>l4</line_id>
      <sales_price>48</sales_price><item_id>i2</item_id>
      <name>pinball</name>
    </sale>
  </store>
  <store>
    <manager>Jody</manager>
  </store>
</result>
```

図 5: XML 実体化ビューの例
Fig. 5: Example of Materialized XML View

```
<?xml version="1.0"?>
<!ELEMENT aux_root (aux_store*, aux_sale*, aux_item*)>
<!ELEMENT aux_store (store_id, manager)>
<!ELEMENT aux_sale (sale_id, store_id)>
<!ELEMENT aux_item (item_id, name)>
```

※ その他の要素は内容が#PCDATAとなる．ここでは省略する．

図 6: 補助ビューの例
Fig. 6: Example of Auxiliary View

図 6 の補助ビューにおいて，リレーション `line` の実体化がなされていないことに注意する．これは，2.2.1 節で述べた参照整合性制約によって，既存の `line` タプルが他のリレーションに結合されて追加されるという更新が発生しないことが保障されるためである．すなわち，`line` タプルが追加されるのは，新規の売上げデータが追加されるときのみである．この補助ビューの役割等については後述する．

2.3 デフォルト XML ビューへのマッピング

RDB から XML へのマッピングのために，参照整合性制約の情報を用いる．参照整合性制約の情報は，非巡回有向グラフ（directed acyclic graph，DAG）として表される（グラフに閉路がある場合は本研究では考慮しない）．たとえば，前節で示した売上げデータベースにおける参照整合性制約に対する DAG は，*line* → *sale*, *sale* → *store*, *line* → *item* という 3 つの有向辺からなる．

図 7 にマッピングのアルゴリズムを示す．このアルゴリズムを先の売上げデータベースに対して適用すると，図 3 の DTD が作成される．ただし，例のデータベースに対する DAG には末端ノードとして `item` ノードも存在するため，本アルゴリズムを適用すると，`item` を最上位要素とした DTD も生成される．サーバは，アルゴリズムを適用することによって得られるすべてのデフォルト XML ビューをクライアントに提供する．クライアントも自分の要求に合わせて適切なデフォルト XML ビューを選択することによって，ビュー定義を容易に行うことが可能となる．

マッピングアルゴリズム

DAG の各末端ノード（有向辺の終点ノード）*n* について以下を行う．

1. *n* から DAG を走査し，すべてのノードを含む木 *T* を作成する．走査の際には，一時的に DAG を無向グラフとみなして処理する．
2. 木 *T* の辺の集合を *edges* とする．ただし，*edges* の各要素は $(rel_1, rel_2, type)$ という三つ組みである．*type* は元の DAG における辺の向きを表すのに用いる．たとえば，例のデータベースに対する DAG を `store` から走査した場合，*edges* = {(*store*, *sale*, 1), (*sale*, *line*, 1), (*line*, *item*, 2)} となる．
3. *create_element*(*root*, *true*) を呼びだす．

function *create_element*(*rel*, *flag*)

```
1.  // flag が true の場合，rel に対する XML 要素を出力
2.  // false の場合，rel に対する XML 要素のリストを返す
3.  elems に rel に対応するリレーションの属性集合を代入する;
4.  for edges 中の (rel, r, t) というパターンの辺の各々について do
5.    if t ≡ 1 then
6.      elems := elems ∪ {r + "*"};
7.      create_element(r, true);
8.    else
9.      elems := elems ∪ create_element(r, false);
10.   end
11. end
12. if flag ≡ true then
13.   output "ELEMENT #rel (#elems)";
14. else
15.   return elems;
16. end
```

[注] 6 行目の "+" は文字列の連接を表す．**output** 文中の "#. . ." は，その評価結果を文字列として埋め込むことを意味している．なお，内容が#PCDATA となる要素の出力については省略している．また，実際には重複する属性の削除や同じ XML 要素を複数回出力しないための配慮も必要であるが，ここでは省略する．

図 7: マッピングのアルゴリズム
Fig. 7: Mapping Algorithm

3. 更新処理の具体例

新たな売上げがあり，サーバの RDB に図 8 で示されるタプルがリレーション `sale`, `line` にそれぞれに追加された場合を考える．`sale` の追加タプル中の `store2` は Jim が店長を務めるカリフォルニアの店舗の ID とする．`line` の追加タプル中で，`i5` のみが玩具に対応しているとする．

タプルの追加が発生すると，サーバは，あらかじめ作成されていた更新スクリプトの実行を行う．図 4 の XSLT ビューに対しては，この更新内容のうち，リレーション `sale` の (`s5`, `store2`, `2004`)，リレーション `line` の (`l8`, `s5`, `i5`, `8`) をクライアント側の XML 実体化ビューに反映する必要がある．`line` の残り 2 つのタプルは，XML 実体化ビューには影響を与えないため，配信は不要である．すなわち，サーバは以下のような処理を行う．

1. XSLT ビューの定義に基づき，追加されたタプルの中で，実体化ビューに関連するもののみを選択する．なお，選択の際には，不要な情報を削減するため射影演算を適用する．
2. ステップ 1 で抽出した更新情報を XML 化して，更新データとしてクライアントに配信する．

Δsale

sale_id	store_id	year
s5	store2	2004

Δline

line_id	sale_id	item_id	sales_price
l7	s5	i10	20
l8	s5	i5	8
l9	s5	i18	29

図 8: 追加タプルの例
Fig. 8: Example of New Tuples

この例の場合，クライアントに配信される差分データは図 9 のようになる．

```
<insert>
  <diff_sale>
    <sale_id>s5</sale_id>
    <store_id>store2</store_id>
  </diff_sale>
  <diff_line>
    <line_id>l8</line_id>
    <store_id>store2</store_id>
    <item_id>i5</item_id>
    <sales_price>8</sales_price>
  </diff_line>
</insert>
```

図 9: 差分データの例
Fig. 9: Example of Differential Data

差分データを受け取ると，クライアントは事前にサーバから送られていた更新スクリプトを起動する．この例の場合，まず補助ビューの `aux_sale` 要素の更新が行われ，(`s5`, `store2`) が追加される．次に，追加された `line` の情報 (`l8`, `s5`, `i5`, `8`) に関する更新がなされる．この XML 実体化ビューでは，`line` に該当する商品名の情報も合わせて提示するため，補助ビューの `aux_item` から商品 `i5` の名前を抽出し，図 10 のような XML 実体化ビューの差分データのフラグメントを生成する．このフラグメントを図 5 の，`store_id` が "`store2`" である店舗の売上げデータの最後尾の要素として挿入することによって XML 実体化ビューの更新処理が完了する．なお，参照整合性制約とこの更新手続きの流れにより，`line` に対する追加情報を挿入すべき「`store_id` が "`store2`" である店舗の売り上げデータ」を表す XML 要素はすでに作成されている．よって，追加対象となる XML 要素が存在しないという問題は発生しない．

```
<sale>
  <sale_id>s5</sale_id>
  <line_id>l8</line_id>
  <sales_price>8</sales_price>
  <item_id>i5</item_id>
  <name>playing cards</name>
</sale>
```

図 10: ビュー更新のための差分フラグメント
Fig. 10: Differential Fragment for View Update

4. 更新処理手法

提案手法では，以下の点を考慮して更新処理を実現する．

1. サーバからクライアントへは最小限の差分データを配信する：これにより XML ビュー全体の再構築や不要なデータの配信を避ける．
2. クライアントにおける更新処理は，クライアントが有する XML 処理機能（XSLT など）のみで実現可能とする．

以下では本手法の概要を述べる．

4.1 ビュー定義の解析

XML ビュー定義（XSLT スタイルシート）の解析には [4] の XSLT スタイルシートのの変換手法を拡張して用いる．まず，XML ビュー定義が登録された際に，サーバの RDB から必要な情報を抜き出して XML 実体化ビューを構築するための SQL 問合せおよび XML 生成スクリプトを作成する（詳細は省略）．それと同時に，RDB に新たなタプルが追加された際にタプルから情報を抽出・加工し，更新処理用の XML データを作成するためのスクリプトを作成する．

たとえば，図 4 のビュー定義の 9 行目の `xsl:copy-of` に対して解析を行った場合，以下のような中間結果が得られる．

```
¶db/store[state = 'CA']/sale[year = '2004']/line
  [category = 'toy']/{CE}sale/sale_id{select : sale_id}
```

これは，[4] において *T-expression* (Translation Expression) と呼ばれる表記であり，直感的には，デフォルト XML ビューのルートから選択処理などを行いながら木を辿り，出力対象の XML 要素へ到達するまでのパスの情報を示している．`{CE}sale` は XSLT スクリプト内で現れる `<sale>` タグの挿入を表し，`{select:sale_id}` は，最終的に抽出すべき要素を示している．この情報から以下ようなリレーショナル代数式を導くことで，RDB のデータと XML 実体化ビューのデータの対応付けを行う．

$$\pi_{store\_id, sale\_id, line\_id, item\_id, sales\_price}\sigma_{state='CA'}(store) \bowtie \sigma_{year='2004'}(sale) \bowtie line \bowtie \sigma_{category='toy'}(item)$$

実際に必要なのは `sale_id` だけであるが，抽出した `sale_id` を出力する XML データのどこに埋め込むかというコンテクスト情報も出力する必要があるため，上記のような射影が必要となる．このような処理を図 4 の他の行の処理についても同様に適用することによって，図 8 の差分データが図 9 の形式へと変換される．

4.2 補助ビュー

補助ビュー（auxiliary view）の概念は，データウェアハウス環境において，リモートサイトに位置する実体化ビューの効率的な更新のために提案された [6]．本研究では，この概念を XML 実体化ビューの更新に拡張する．

更新情報配信システムでは，ビュー定義の登録時に補助ビューを構築するための XSLT スタイルシートを生成する．これをデフォルト XML ビューに適用することにより，図 6 のような補助ビューを構築し，クライアントに配信する．補助ビューは RDB に保存されている元データと比較して容量が非常に小さく，また，その更新は後にサーバから配信される差分データと更新スクリプトによって効率よく行われる．

4.3 クライアントの更新スクリプト

クライアント側の更新スクリプトは，サーバから配信される差分データをクライアントの補助ビュー及び XML 実体化ビューに適用するためのものである．クライアントはこれを保存しておき，更新データの配信の際に適用する．更新スクリプトは以下からなる．

(1) 差分データを補助ビューに適用するための XSLT スタイルシート

(2) 更新された補助ビューと差分データを利用して XML 実体化ビューの更新を行うための XSLT スタイルシート

(1) は，差分データを補助ビューの構造へ変換する処理を行う．(2) は，データ間の参照整合性制約に基づいた結合処理を行い，その結果を XML 実体化ビューの構造にフォーマットするという処理を行う．たとえば，図 9 の差分データが配信された際に適用され，図 10 のビューの差分フラグメントを生成する (2) の XSLT スタイルシートは図 11 のようになる．

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:variable name="auxi" select="document('aux_view.xml')/aux_root/aux_item" />

  <xsl:template match="/insert">
    <xsl:for-each select="diff_line">
      <xsl:variable name="l" select="item_id" />
      <sale>
        <xsl:copy-of select="../diff_sale/sale_id" />
        <xsl:copy-of select="line_id" />
        <xsl:copy-of select="sales_price" />
        <xsl:copy-of select="item_id" />
        <xsl:copy-of select="$auxi[item_id=$l]/name" />
      </sale>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

図 11: 更新スクリプトに含まれる XSLT スタイルシート (2)
Fig. 11: XSLT Stylesheet (2) included in a Update Script

5. まとめ

本稿では RDB 上で XSLT によって定義された XML 実体化ビューのインクリメンタルな更新処理方式のアプローチについて述べた．更新パターンに応じた事前の解析，補助ビューの利用による効率化が特徴である．今後の課題としては，更新処理方式の詳細についての検討，システムの実装と評価実験などがある．

[謝辞]

本研究の一部は，日本学術振興会科学研究費 (C)(2) (16500048)，旭硝子財団研究助成，文部科学省科学研究費特定領域研究 (2)(16016205) 及び CREST 「自律連合型基盤システムの構築」による．

[文献]

[1] XSLT, XSL Transformations version 1.0. `http://www.w3.org/TR/xslt`.

[2] 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 *Proc. VLDB*, pp. 646–648, 2000.

[3] M. F. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima, and W. C. Tan. SilkRoute: A Framework for Publishing Relational Data in XML. *ACM TODS*, Vol. 27, No. 4, pp. 438–493, 2002.

[4] J. Liu and M. Vincent. Querying Relational Databases through XSLT. *Data & Knowledge Engineering*, Vol. 48, No. 1, pp. 103–128, 2004.

[5] G. Moerkotte. Incorporating XSL Processing into Database Engines. In *Proc. VLDB*, pp. 107–118, 2002.

[6] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making Views Self-maintainable for Data Warehousing. In *Proc. PDIS*, pp. 158–169, 1996.

石川 佳治　Yoshiharu ISHIKAWA
筑波大学システム情報工学研究科コンピュータサイエンス専攻助教授．データベース，データ工学，情報検索などに興味を持つ．日本データベース学会，情報処理学会，電子情報通信学会，人工知能学会，ACM，IEEE CS 各会員．

宮坂 集策　Shusaku MIYASAKA
筑波大学システム情報工学研究科在学中．XML データベースの研究に従事．情報処理学会，日本データベース学会学生会員．

北川 博之　Hiroyuki KITAGAWA
筑波大学システム情報工学研究科コンピュータサイエンス専攻教授．異種情報源統合，データマイニング，文書データベース，WWW の高度利用などの研究に従事．日本データベース学会，情報処理学会，電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE CS 各会員．著書に「データベースシステム」（昭晃堂）など．