RDB 上の XSLT 実体化ビューのインクリメンタルな更新について

宮坂集策†,　　石川佳治†,　　北川博之†

概要

近年，ビジネスにおけるデータ交換用ファイル形式のデファクトスタンダードとして XML が広く用いられている．XML を利用するアプリケーションの背後には RDB がデータストレージとして用いられることが多く，その場合 XML データは RDB に対する一種のビューとして位置づけることができる．本研究では特に，データベースサーバに対して，クライアントが XSLT を用いた XML ビュー定義を行う環境を想定する．XSLT による XML 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，クライアント側で管理される．このような環境では，RDB に更新が発生したときの XML 実体化ビューの更新が重要な課題となる．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．

Incremental Update for Materialized XSLT Views on RDBs

Shusaku MIYASAKA†,　　Yoshiharu ISHIKAWA†,　　Hiroyuki KITAGAWA†

abstract

Today, XML is widely used as a general-purpose format for information exchange on the Web. In information systems which provide XML documents, RDBs are often used for data storage and XML generation. In this sense, 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 over a remote relational database and XML views are materialized on the client. We propose an efficient method for updating materialized XML views defined by XSLT in an incremental manner. In our approach, the view management system analyzes a database schema and XSLT view definitions, and generates an update script. When new updates occur, the script is executed for XML view updates.

1　はじめに

XML [1] はインターネット上の汎用の情報交換フォーマットとして広く用いられているが，近年では情報配信･データ放送におけるストリーム的な配信情報の記述にも利用されている．様々なアプリケーションの中で XML をビューとして用いることにより，データベース内のデータに対してより深い意味情報を付加したり，用途に応じて構造を柔軟に組み替えることが可能となる．そのようなシステムでは RDB がデータストレージとして用いられることが最も一般的であるが，これには 2 つのケースが考えられる．1 つは元データが XML 形式であり，効率の良いデータアクセスのために RDBMS を用いる場合，もう 1 つは元データがリレーショナルデータであって，データ交換・公開の目的のために XML を用いる必要がある場合である．後者は XML 出版（XML Publishing）と呼ばれることもある．

XML 出版では，XML データは RDB への問合せ結果をもとに XML 形式に加工され，クライアントに配信される．この意味で，出版される XML データは RDB に対する一種のビューと捉えることができる．特に，生成された 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．直接的な手法としては，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ってクライアントに配信することが考えられるが，この手法には以下の問題点がある．

†筑波大学システム情報工学研究科
Graduate School of Systems and Information Engineering,
University of Tsukuba

- 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

従って，XML 実体化ビューの効率的な更新処理手法を構築することは重要な研究課題である．

本研究においては，XML ビューの構造を定義するための問合せ言語として XSLT [2] を用いる．XSLT では，それ自体が XML 文書である XSLT スタイルシートにより XML の変換処理を記述する．それを XSLT プロセッサが処理することで，変換結果のテキストが生成される．

本研究では特に，データベースサーバのデータが XML 実体化ビューとしてリモートのクライアントに配信されている状況における XML 実体化ビューの効率的な更新処理手法の構築を目指す．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．

2 関連研究

RDB 上の XML ビューの構築に関する代表的な研究として [3]，[4]，[5]，[6] が挙げられる．その中でも [5]，[6] は問合せ言語として XSLT を用いることを想定している．[5] は XSLT スタイルシートの内容を解析して複数の SQL 文に変換することによって XML ビューの構築を行うものである．[6] 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，スタイルシートの処理内容をできるだけ RDB 側にプッシュする手法である．しかし，これらのいずれの手法についても，実体化した XML ビューの更新を効率的に行う手法については述べられていない．

また，リレーショナルデータの実体化ビューのインクリメンタルな管理手法として [7] がある．これは，サーバ・クライアント型のシステムにおいて，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というものである．本研究では，この研究を先に述べた XML 実体化ビューのインクリメンタルな管理に拡張する．

3 概要

本節では提案手法による XML 実体化ビューの更新処理の概要を述べる．

3.1 システムの構成

本研究で想定しているシステム環境は次の図 1 のような，サーバとクライアントがネットワークを介して接続されているものである．また，クライアントの処理能力としては，XML に対する処理機能のみを持っているものとする．なお，補助ビュー等の図中に示されている各要素の役割及び詳細については後述する．

3.2 更新処理の流れ

図 1 における 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 RDB 内に保持しているデータを，データベースのスキーマ情報をもとに一定のルールに従ったマッピングを行うことにより**デフォルト XML ビュー**を生成しユーザに提供する．
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 ビュー定義を記述した

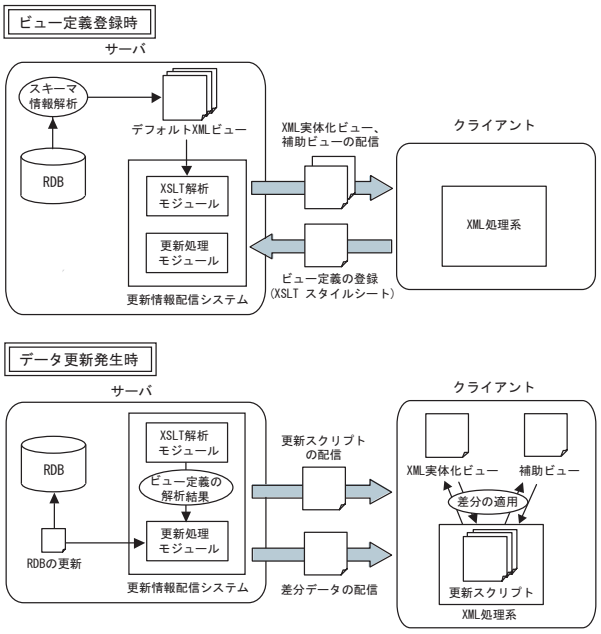

図 1: システム環境

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 実体化ビュー及び補助ビューを構築し，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 XML 実体化ビューの構造に対応した更新用の差分データをクライアントに配信する．また，データ更新のパターンが現在までの更新と異なるものであった場合には，そのパターンに対応した更新スクリプトを生成し，クライアントに配信する．配信された更新スクリプトはクライアントで蓄積・管理される．

2. クライアントは受け取った差分データに対し，そのデータ更新のパターンに対応した更新スクリプトを利用して，まず補助ビューに対しての更新を行い，その後 XML 実体化ビューに対して差分データの内容を反映することで，XML 実体化ビューの更新を行う．

次節において，本稿で用いる具体例についての説明を行う．

3.3 具体例

3.3.1 RDB 上のデータ

ここでは，あるチェーン店の売上げ情報を持つデータベースを考える．次の図 2 はサーバの RDB 内に保持されるリレーション群である．これは [7] で用いられた例をもとにしたものである．

```
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: 売上げデータベースのリレーション群

下線のある属性は，それぞれのリレーションのキー属性である．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．リレーション `sale` は 1 つの売上げに対して 1 つの情報を，売上げがあった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 つの商品が当てはまる．リレーション `line` はこのような情報を，ある売上げに対して売れた商品それぞれについて保持する．また，リレーション `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，次のようなリレーション間の参照整合性制約が存在するとする．

1. $sale.store\_id \Rightarrow store.store\_id$

2. $line.sale\_id \Rightarrow sale.sale\_id$

3. $line.item\_id \Rightarrow item.item\_id$

$S.B \Rightarrow R.A$ とは，任意のタプル $s \in S$ に対して $s.B = r.A$ となる $r \in R$ が必ず存在するということである．

また，本稿では発生し得るデータ更新のパターンについて，簡単化のためデータの追加（insertion）のみを考える．その理由としては，先に述べたようなサーバ・クライアント型の情報配信的なシステム環境においては，タプルの追加に伴う更新処理が最も頻繁に生じると考えられることが挙げられる．本節で挙げた売上げデータベースの例に対しても，新たな売上げデータの追加による更新が最も多くなると考えてよい．他の更新パターン（タプルの削除，

タプルの内容の更新）については今後の検討課題とする．

3.3.2 デフォルト XML ビュー

既に述べたように，サーバは RDB 上のデータを XML データの形式でクライアントに対してデフォルト XML ビューとして提供する．これにより，クライアントシステムのユーザはサーバに保持されているデータを XML データとして捉えることができ，それぞれの要求に応じて XSLT スタイルシートによる問合せ（ビュー定義）を行うことができる．デフォルト XML ビュー自体は仮想的なものであり，実体化はなされない．前節で挙げた売上げデータに対する，あるデフォルト XML ビューの DTD は次の図 3 のようになる．

```
<?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: 売上げ情報を示すデフォルト XML ビューの DTD

要素 db はこの XML データのルート要素であり，子要素として要素 store のシーケンスを持つ．要素 store はその子要素に RDB 上のリレーション store の各属性とリレーション sale の情報を持つ要素 sale のシーケンスを含む．同様に要素 sale は子要素としてリレーション line の情報を持つ要素 line のシーケンス及びリレーション item の情報を持つ．

RDB 上のリレーショナルデータをこのような XML データとして表現するためには一定の **XML マッピングルール**を定める必要がある．3.3 節において我々が提案するマッピングルールの概要について述べる．

3.3.3 ビュー定義

ここでは，クライアントシステムのユーザが，あるチェーン店のカリフォルニア地区の玩具担当の責任者であるとし，「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いるとする．そこで，この情報を含むような XML 実体化ビューを常時保持していたいという状況を考える．このとき，前節の図 3 のデフォルト XML ビューをもとに，たとえば図 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．

```
<?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: 売上げ情報に対する XSLT ビュー定義例

図 4 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する処理を行うようにしており，さらに 8 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ものである．

3.3.4 XML 実体化ビューと補助ビュー

サーバは図 4 のビュー定義を受信すると次の図 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: 得られる XML 実体化ビューの例

```
<?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

図 6 の補助ビューについて，サーバに保存されている元データのリレーション line に関するデータは実体化する必要はない．それは，図 7 の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と結合して追加されるという更新は発生しない．すなわち，追加される line タプルは新規の売上げデータであることが保証されるためである．補助ビューの役割等については後述する．

3.4 XML マッピングルール

リレーショナルデータベースから XML へのマッピングには，リレーショナルデータベースの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考慮しない）．例として，前節で挙げた売上げデータベース内に存在する参照整合性制約を DAG を用いて表現すると次の図 7 のようになる．

以下にマッピングルールのアルゴリズムを示す．

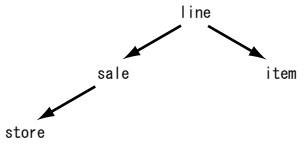

図 7: 売上データの参照整合性制約を示す DAG

Algorithm マッピングのアルゴリズム
DAG の各末端ノード（有向辺の終着ノード）n について以下の処理を行う．

(a) 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．ただし，走査の際には，一時的に DAG を無向グラフと考えて処理する．
(b) 木 T の辺の集合を $edges$ とする．ただし，$edges$ の各要素は $(rel1, rel2, type)$ 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，図 7 の DAG を store から走査した場合，
$edges = \{(store, sale, 1), (sale, line, 1), (line, item, 2)\}$
となる．
(c) $create\_element(root, true)$ 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
4 $edges$ 中の (rel, r, t) というパターンの辺の各々について do
5 $if\ t == 1$
6 $elems := elems \cup \{r + "*"\}$
7 $create\_element(r, true)$
8 $else$
9 $elems := elems \cup create\_element(r, false)$
10 end
11 end
12 $if\ flag == true$
13 $output "ELEMENT\ \#\{rel\}\ (\#\{elems\})$
14 $else$
15 $return\ elems$
16 end

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．

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，まず図 7 の DAG には末端ノードとして item ノードも存在する．よって，本アルゴリズムを適用すると，item を上位要素とした DTD も生成される．サーバシステムではこのアルゴリズムを適用することによって得られる全てのデフォルト XML ビューをクライアントに提供する．これにより，クライアントの多様な要求に対応することができ，クライアントも適切なデフォルト XML ビューを自分の要求に合わせて選択することによって，XSLT ビュー定義を容易に行うことが可能となる．

4 更新処理の具体例

ここではデータの更新の例として，新たな売上げがあり，サーバの RDB に対して次の表 1 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ものを考える．
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．

このような追加が生じると，サーバは，XSLT ビュー定義を解析することで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

sale

sale_id	store_id	year
s5	store2	2004

line

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

表 1: 追加タプルの例

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．よって本質的には，サーバは以下の処理を行う．

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．
2. 選択の際に，適切な射影演算を適用する：この場合，sale のタプル (s5, store, 2004) の情報のうち，クライアントでは 2004 年の情報のみをビューで管理しているため，売り上げ年の情報は必要としていない．そこで，(s5, store2) という射影された情報を配信すればよい．
3. (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．

以上の処理によってクライアントに配信される差分データは次の図 8 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 8: クライアントに配信される差分データ例

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．この例については，更新スクリプトはまず補助ビューの aux_sale 要素の更新を行う．具体的には (s5, store2) を追加する．次に，追加された line の情報 (l8, s5, i5, 8) に関する更新を行う．XML 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必要があるため，補助ビューの aux_item に含まれる情報から商品 i5 の名前を抽出し，図 9 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 に対するこの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，追加場所の 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 9: ビューの差分フラグメント

5 更新処理手法

提案手法では以下の点を考慮した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）で実現可能とする．

以下では本手法の概要を述べる．

5.1 ビュー定義の解析

XSLT スタイルシートによって記述された XML ビュー定義の解析には [5] で述べられている XSLT の変換手法を拡張して用いる．[5] では RDB 上に

定義された XSLT ビュー定義を SQL 問合せに変換する方式について述べているが，本手法では追加されたタプルを選択・加工するために変換処理を行う点が異なる．本アプローチでは，まず [5] の手法をベースとして，XML ビュー定義の登録時に，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（これについては省略する）．また，[5] の手法の拡張により，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 データを作成するスクリプトを作成する．

例として，図 4 のビュー定義の 10 行目の $xsl:copy{-}of$ の処理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，[5] において T-expression（Translation Expression）と呼ばれるものである．直感的には，この式は XML デフォルトビューのルートから順にたどって，選択処理などを行いながら，出力対象となる XML 要素へ到達するまでのパス情報を示している．なお，{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．この情報から次のようなリレーショナル代数式を導くことで RDB 内のデータを XML 実体化ビュー化した際の対応付けを行う．

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

実際に必要とされるのは 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を出力する必要があるため，上記のような射影が行われる．以上のこの処理を図 4 の他の行の $xsl:copy{-}of$ 文の処理についても同様に行うことによって，表 1 の差分データから 図 8 の形式への変換を行う．

5.2 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 [7]．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．[7] では，ビュー定義とリレーショナルデータベースの参照整合性制約の情報から導かれるリレーション間の依存関係を利用し，補助ビューの構築を行う SQL 問合せを生成する．本手法では，リレーショナルデータベースをデフォルト XML ビューに対応付ける点，SQL 問合せの代わりに追加タプルを更新するスクリプトを生成する点，リレーションの形式ではなく XML 形式での補助ビューの管理などの面でアプローチが異なる．

更新情報配信システムでは，クライアントによるビュー定義の登録時に，以上のような手法に基づいて解析を行うことによって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 の例に示されるような，ビュー定義の内容に対応した補助ビューを構築し，クライアントに送信する．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行われるため．補助ビューの配信・更新処理における負荷は小さい．

5.3 更新スクリプト

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．サーバにおいて，登録されたビュー定義の内容を解析することにより，発生し得る更新パターン毎に作成され，クライアントに配信される．クライアントではこれを保存しておき，後にサーバから配信される差分データに対し，その更新パターンに応じた更新スクリプトを適用することによって更新処理を行う．更新スクリプトは以下のようなファイルからなる．

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) では，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) では，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が行われる．例として，クライアントに対して図 8 の差分データが配信された際に適用され，図 9 のビュー

の差分フラグメントを生成する (2) の XSLT スタイルシートは次の図 10 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 10: 更新スクリプトに含まれる XSLT スタイルシート (2)

6 まとめと今後の課題

本稿では RDB 上の XSLT によって構造が定義された 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. サーバ・クライアント型の XML 出版システムにおいて, データベースサーバに対して更新が発生する度に, 新たにビューの構築を行ってクライアントに送信するのではなく, ビュー定義の内容に対して解析を行い, それぞれのクライアントが保持する XML 実体化ビューの内容に応じたインクリメンタルな更新を行うことにより, 効率的なビュー更新処理の実現を目指している.

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を行った後, システムの実装と評価実験を行っていく必要がある.

謝辞

本研究の一部は, 日本学術振興会科学研究費 (C)(2) (16500048), 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.

参考文献

[1] XML, Extensible Markup Language.
http://www.w3.org/XML/

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

[3] M. F. Fernandez, Y. Kadiyska, D. Suciu, A. Morishima, W. C. Tan, ”SilkRoute: A Framework for Publishing Relational Data in XML”, TODS 27(4), pp. 438-493, 2002.

[4] M. J. Carey, J. Kiernan, J. Shanmugasundaram, E. J. Shekita, S. N. Subramanian, ”XPERANTO: Middleware for Publishing Object-Relational Data as XML Documents”, VLDB pp. 646-648, 2000.

[5] J. Liu, M. Vincent, ”Querying Relational Databases through XSLT”, Data & Knowledge Engineering 48, pp. 103-128, 2004.

[6] G. Moerkotte, ”Incorporating XSL Processing into Database Engines”, VLDB 2002.

[7] D. Quass, A. Gupta, I.S. Mumick, J. Widom, ”Making Views Self-Mainteinable for Data Warehousing”, IEEE PDIS 1996.