リレーショナルデータベース上の XML ビューに対する外部関数を考慮した問合せ処理

川田 純 † 石川 佳治 ‡ 北川 博之 ‡

† 筑波大学 システム情報工学研究科 ‡ 筑波大学 電子・情報工学系

kuro@kde.is.tsukuba.ac.jp {ishikawa, kitagawa}@is.tsukuba.ac.jp

1 はじめに

ネットワーク技術の発展に伴い，XML はインターネット上でのアプリケーション同士の共通のデータ交換様式となりつつある．一方，実際のデータはリレーショナルデータベース等に格納されていることが多く，アプリケーションはデータ交換のためにデータを XML に変換する必要がある．

このような問題点に対し，様々な組織で利用されているリレーショナルデータベースに焦点を当て，独自のミドルウェアを用いてリレーショナルデータベース (例・図 1) からユーザの要求に合致した XML ビュー (例・図 2) を効率的に生成するための研究が行われている [1][2].

施設

施設ID	施設名	市ID
0015	Aモール	C0100
0016	H公園	C0100
0017	M研究所	C0100
0018	Bモール	C0101
0019	Cモール	C0102
0020	D飛行場	C0103
…		

位置情報

施設ID	X座標	Y座標
0015	1	1
0015	4	1
…	…	…
0016	11	2
0016	15	2
…		

市

市ID	市名	人口（万人）
C0100	つくば市	16
C0101	土浦市	13
C0102	竜ヶ崎市	7
C0103	阿見町	4
…		

図 1: リレーショナルデータベースの例

```
01: <施設一覧>
02:   <施設 id="0015">
03:     <施設名>A モール</>
04:     <所在>
05:       <市名>つくば市</>
06:       <人口（万人）>16</>
07:     </>
08:     <位置情報>
09:       <座標><X 座標>1</><Y 座標>1</></>
10:       ...
11:   </></>
12:   ...
13: </>
```

図 2: XML ビュー

XPERANT グループ [2] は，リレーショナルデータベース上の XML ビューに対する XML 問合せ言語 XQuery[4] を用いた問合せ (例・図 3) 処理を，リレーショナルエンジンに可能な限り演算をプッシュダウンすることで，XML 問合せ処理の効率化を図る基本アプローチを提案している．

```
01: <結果>{
02:   for $施設 in view("施設一覧")/施設一覧/施設
03:   where isWider($施設/位置情報, 10, "km")
04:   return <施設>$施設/施設名</>
05: }</>
```

図 3: ユーザ問合せ XQuery

他方で，XQuery をはじめいくつかの XML 問合せ言語では，図 3 に示す *isWider()* のような利用者定義の外部関数の利用が検討あるいは実現されている [5]．これらの外部関数は XML 文書要素フラグメントの存在とメモリ上での何らかのその表現形式を前提に汎用プログラミング言語等で記述されるため，上記のようなリレーションデータに対する XML ビューに対して用いられた場合には，問合せ処理上考慮する必要がある．

本稿では，SQL を処理可能な一般的 RDBMS とその上位に位置するミドルウェアの連携により，リレーショナルデータベース上の XML ビューに対する外部関数を含む問合せを処理する方式を提案する．具体的には，異なる問合せ実行方式をもつ二つの問合せ処理方式を提案し，それらの処理効率の比較も行う．本提案方式における問合せ計画では，上記の XPERANT における基本アプローチをミドルウェアにおける外部関数処理を考慮するように拡張して用いる．

2 外部関数処理へのアプローチ

XPERANT[2] や SilkRoute[1] のような RDB 上の XML ビューを扱う先行研究では，述語演算など可能な限りの演算を RDBMS に委譲し，タグ付けのみをミドルウェアで実行する事で，RDBMS の問合せ処理能力を最大限に利用する．しかし，図 3 に示すような外部関数は，問合せ対象となる XML フラグメントの存在と，そのメモリ上での DOM(Document Object Model) などの表現形式を前提に Java などの汎用プログラミング言語で記述される．そのため外部関数処理を行う際には，予めリレーションタプルとして得たデータにタグ付けを行い XML フラグメントを生成した後，その XML フラグメントをメモリ上の表現形式に変換して外部関数を評価する必要がある．従って，単純に RDBMS に外部関数演算を委譲することは難しいため，図 3 に示すような外部関数を用いた選択条件を持つ問合せをこの方式で処理することは困難である．

この問題に対して，XPERANT グループは上記基本アプローチの拡張として，RDBMS 内部に XML フラグメントの生成機能とそれに対する外部関数評価機能を追加し，外部関数を含めた問合せ処理を委譲するというアイデアを [2] で簡単に触れている．この拡張アプローチの問題点としては，RDBMS 内部に専用の追加機能を実装する必要があり，一般の様々な RDBMS に対しては適用できない点があげられる．

そこで本稿では，ミドルウェアにおいて外部関数処理を行うことと，一般的な RDBMS とミドルウェアの連携による RDB 上の XML ビューに対する外部関数を含む問合せ処理方式を 2 種類 (図 4，図 5) 提案する．

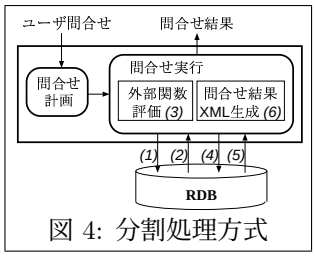

図 4: 分割処理方式

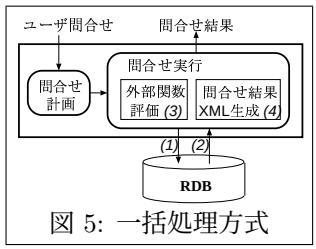

図 5: 一括処理方式

それぞれの処理方式では，まずユーザから与えられる XML 問合せ言語をもとに問合せ計画が立てられる．この結果，XML フラグメント生成に必要なリレーションタプルを取得するための SQL 問合せと，取得されたタプルに XML タグ付けを行う演算子群が抽出される．

問合せ計画以降の問合せ処理は，**分割処理方式**では以下のようになる．

(1) 外部関数評価に必要なタプル取得のための SQL 問合せを発行.

(2) 外部関数評価用のタプルを取得.
(3) 取得したタプルから XML フラグメントを生成し外部関数評価を行う.
(4) 外部関数評価結果に基づく条件を付加した，問合せ結果 (XML) 生成用のタプル取得のための SQL 問合せを発行.
(5) 問合せ結果生成のためのタプルを取得.
(6) 問合せ結果 XML を生成する.

図 3 の外部関数を含む XQuery 問合せを例とした場合の処理の流れは図 6 となる.

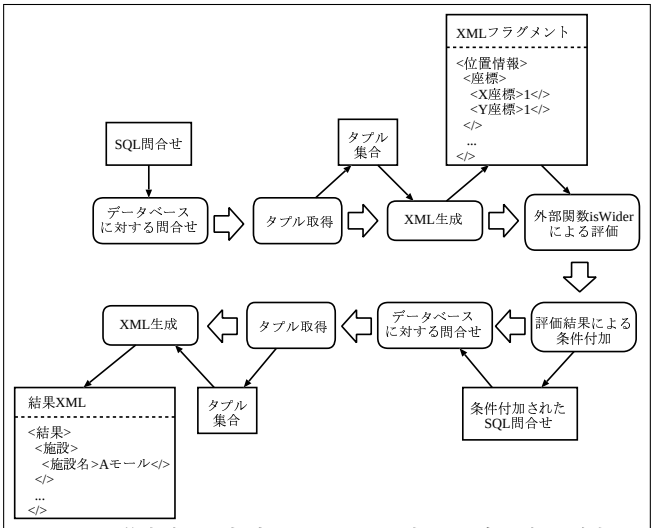

図 6: 分割処理方式における一連の問合せ処理例

また，**一括処理方式**では以下のようになる.

(1) 外部関数評価用と問合せ結果 (XML) 生成用のタプルを一括に取得する SQL 問合せを発行.
(2) 外部関数評価用のタプルと問合せ結果生成のためのタプルを取得.
(3) 取得したタプルから XML フラグメントを生成し外部関数評価を行う.
(4) 評価結果によりタプルをさらに選択した上で，問合せ結果 XML を生成する.

図 3 の外部関数を含む XQuery 問合せを例とした場合の処理の流れは図 7 となる.

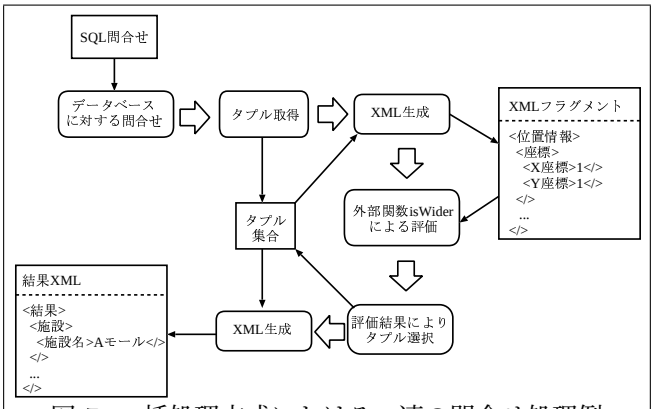

図 7: 一括処理方式における一連の問合せ処理例

これらの処理方式の詳細は 5 節で述べる．次節ではこれらの処理方式における問合せ計画を構築する上でベースとした，XPERANT における XML ビュー問合せ処理方式の概略を説明する.

3 XPERANT における基本アプローチ

XPERANT では，予め図 1 のリレーショナルデータベースのテーブル構造を直接表現した図 8 のようなデフォルト XML ビューを構築する．そして，提供されるデフォルト XML ビューから図 9 に示すビュー定義 XQuery によって，図 2 のような XML ビューを定義する.

```
01: <db>
02:   <施設>
03:     <row>
04:       <施設 ID>0015</施設 ID>
05:       <施設名>A モール</施設名>
06:       <市 ID>C0100</市 ID>
07:     </row>
08:     ...
09:   </施設>
10:   <市>
11:     <row>
12:       <市 ID>C0100</市 ID>
13:       <市名>つくば市</市名>
14:       <人口>16</人口>
15:     </row>
16:     ...
17:   </市>
18:   <位置情報>
19:     <row>
20:       <施設 ID>0015</施設 ID>
21:       <X 座標>1</X 座標>
22:       <Y 座標>1</Y 座標>
23:     </row>
24:     ...
25:   </位置情報>
26: </db>
```

図 8: デフォルト XML ビュー

```
01: create view 施設一覧 as (
02: <施設一覧>
03:   for $施設 in view("default")/施設/row
04:   return
05:   <施設 id={$施設/施設 ID/data()}>
06:     {$施設/施設名}
07:     <所在>
08:       for $市 in view("default")/市/row
09:       where $施設/市 ID=$市/市 ID
10:       return
11:       {$市/市名}
12:       {$市/人口}
13:     </>
14:     <位置情報>
15:       for $位置 in view("default")/位置情報/row
16:       where $施設/施設 ID=$位置/施設 ID
17:       return
18:       <座標>
19:         {$位置/X 座標}
20:         {$位置/Y 座標}
21:     </></>
22: </></>
23: )
```

図 9: XML ビュー定義 XQuery

XML ビューに対する XQuery 問合せ処理は次の三つから構成される．(1) **問合せ解析**:XQuery 問合せを問合せ内部表現に解析，(2) **問合せ変換**:問合せ内部表現を用いた問合せ変換，(3) **問合せ実行**:変換により取得された SQL 問合せとタグ付け演算子による問合せ実行．(1) と (2) をまとめて**問合せ計画**と呼ぶ．以下では問合せ計画を中心に説明する.

3.1 問合せ解析と XQGM

XML ビュー定義 XQuery とユーザが与える XQuery 問合せは **XQGM (XML Query Graph Model)** と呼ばれる問合せ中間形式に変換される．それぞれを**ビュー定義 XQGM** と**ユーザ問合せ XQGM** と呼ぶ.

この XQGM を操作することで，リレーショナルデータベースから結果 XML 文書生成に必要なタプルを取得する

SQL 問合せと，それらのタプルから XML 文書生成を行うタグ付け演算子群の抽出を行う．これら SQL 問合せとタグ付け演算を実行することで，リレーショナルデータベース上の XML ビューに対する問合せ処理を実現する．

3.2 問合せ変換

導出された XQGM を変換し SQL 問合せとタグ付け演算子に分割する．この変換手順の概要は次のようになる．

1. **ビュー合成**: 生成されたビュー定義 XQGM の XML フラグメント生成用の関数群と，ユーザ問合せ XQGM の XML フラグメント操作用の関数群の合成を行い，余分な XML フラグメント生成コストを削減する．
2. **演算子プッシュダウン**: RDBMS で処理可能な演算子を XQGM のリーフ側にプッシュダウンする．これは，RDBMS に発行する SQL 問合せに可能な限り演算を委譲し，ミドルウェアの負荷を軽減するための処理である．
3. **タグ付け演算子プルアップ**: XML フラグメント生成用のタグ付け演算と RDBMS で処理可能な演算を分離する．ミドルウェアで処理するタグ付け演算子は XQGM のルート方向にできるだけ移動し，リーフ側に残された演算子群から RDBMS に対するいくつかの SQL 問合せを生成する．
4. 最後に，ミドルウェアにおける XML フラグメント生成コストを抑制し，少ないメモリ領域でタグ付け演算処理を実現するために，生成された SQL 問合せ群を *OUTER UNION* を用いて出力タプルストリームをまとめ，出力 XML 構造に適した形式となるように再構成する [3].

以上のように XPERANT では，ビュー合成によって余分な XML フラグメント生成を抑制し，演算子プッシュダウンとタグ付け演算子プルアップによって，合成された XQGM からリレーショナルデータベースに対する SQL 問合せとミドルウェアによるタグ付け演算を抽出する．

しかし，論文 [2] では，これらのアプローチを簡単な例に適用するにとどまっている．特に演算子プッシュダウンに関して，問合せ結果に副作用を及ぼさない単純な場合のみを扱っており，プッシュダウンによって問合せ結果に副作用が現れるような複雑な場合については述べられていない．また，XQuery における問合せ条件を記述する **where** 節と結果 XML 構造を指定する **return** 節の性質の違いについても十分に考慮されていないため，例えば，XQGM の操作によって，結果 XML 生成のための **return** 節で必要となる XML フラグメント生成用の関数が **where** 節の XML フラグメント操作用の関数により，ビュー合成中に失われるといった問題がある．

そこで本研究では，このような問題点と外部関数評価に必要となる機能を考慮して，上記の XPERANT のアプローチに対して拡張を行う．次節でこの拡張について詳述する．

4 外部関数処理のための機能拡張

3 節で述べた XPERANT の基本アプローチでは，**where** 節の選択演算は全て RDBMS にプッシュダウン可能であることを前提としており，プッシュダウンが困難な外部関数処理や前節の最後に述べた事項については考慮されていない．そこで本稿では，XPERANT の基本アプローチに機能拡張を行う．

まず，本稿で想定する XML ビュー，ビュー定義問合せやユーザ問合せにおける XQuery 記述，ユーザ問合せ中で利用される外部関数に関する前提条件を述べる．特に XQuery は一般的に高度な問合せ記述が可能であるが，本稿では以下に述べるような制約を設ける．

- XML ビューに関する前提条件
  - XML ビュー中の各文書要素の構築に必要なタプル集合を特定するため，キー集合は予め分かっており，各文書要素が与えられたときにはその値を具体的に決定できる．
- ビュー定義問合せ XQuery とユーザ問合せ XQuery に関する前提条件
  - **for** 節では単一のパス式のみ記述可能であり，パス式中では条件によるフィルタリングを扱わない．
  - ユーザ問合せの **return** 節では副問合せを指定しない．
- ユーザ問合せ中で利用される外部関数
  - **where** 節に述語として一つ指定される．
  - 引数には文書要素，定数，文字列型のリテラルが指定可能である．
  - 引数にとる各文書要素は，ある一つのリレーション中のタプル集合からのみ構成される．

4.1 問合せ解析

基本的に 3.1 節の処理をベースとする．図 2 を定義する図 9 のビュー定義 XQuery 問合せをビュー定義 XQGM に変換すると図 10 のように解析される．解析された XQGM は表 1 に示す演算子群で定義され，これらの演算子中では図 10 の *12* で示すように，表 2 の XML 関数群を利用して演算処理を行う．また，図 10 の *10:correlated join* は相関関係にある副問合せからの入力を結合する演算であり，位置情報と市テーブルのタプルリストを施設テーブルのタプルリストに関連付ける．

演算子	内容
Table	リレーショナルデータベースのテーブルを表す
Project	入力に基づいた結果の計算
Select	入力を選択
Join	二つ以上の入力を結合
Groupby	集約関数適用，グルーピング
Orderby	属性値によってソート
Union	二つ以上の入力を UNION
Unnest	入力をアンネストしたリストを返す
View	ビューを表す
Function	XQuery 関数を表す

表 1: XQGM 演算子

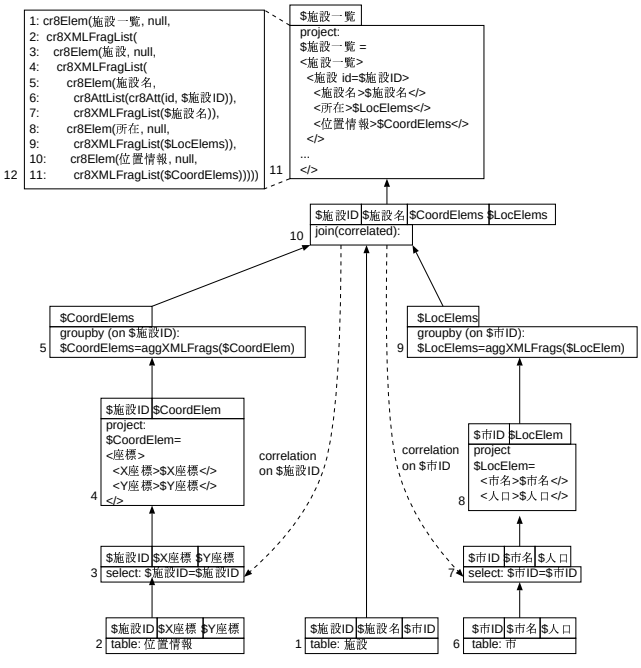

図 10: XML ビュー定義 XQGM

一方，ユーザ問合せ XQuery が図 3 であるとき，XQGM への変換結果は図 11 となる．このとき，XQuery 問合せ

	関数	内容	演算子
1	cr8Elem(Tag, Atts, Clist)	要素名 Tag，属性リスト Atts，要素内容 Clist からなる要素生成	Project
2	cr8AttList(A_1,...A_n)	パラメータの属性群からなる属性リスト生成	Project
3	cr8Att(Name, Val)	属性名 Name，属性値 Val からなる属性生成	Project
4	cr8XMLFragList (C_1,...C_n)	要素内容 (要素やテキスト) パラメータから XML フラグメントリストを生成	Project
5	aggXMLFrags(C)	入力から XML フラグメントリストを作成する集約関数	Groupby
6	getTagName(Elem)	要素 Elem の要素名を返す	Project, Select
7	getAttributes(Elem)	要素 Elem の属性リストを返す	Project, Select
8	getContents(Elem)	Elem の要素内容 (要素やテキスト) の XML フラグメントリストを返す	Project, Select
9	getAttName(Att)	属性 Att の属性名を返す	Project, Select
10	getAttValue(Att)	属性 Att の属性値を返す	Project, Select
11	isElement(E)	E が要素であれば true を，さもなくば false を返す	Select
12	isText(T)	T がテキストであれば true を，さもなくば false を返す	Select
13	unnest(List)	リスト List をアンネストする	Unnest

表 2: XML 関数とその関数が出現する演算子

の **where** 節の持つ述語条件は *11:correlated join* と *existential quantification* によって実現する．

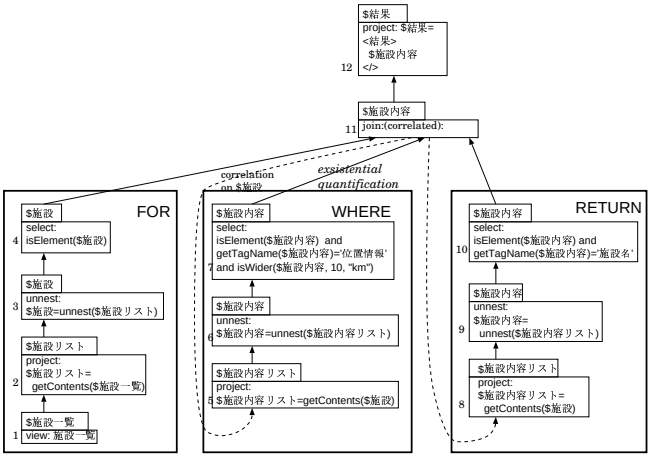

図 11: ユーザ問合せ XQGM

4.2 問合せ変換

3.2 節の処理をベースに拡張する．まず，ビュー合成に関する拡張として，XPERANT の基本アプローチではビュー合成後に行われていた相関関係の分離操作を，ビュー合成の前処理として行う．これによって，5 節で詳述する本稿の提案手法で外部関数評価を行う際に，外部関数評価用 XML フラグメント生成パスと，問合せ結果に必要な XML フラグメント生成パスを独立して扱うことが可能になる．

この相関関係分離の結果，図 10 のビュー定義 XQGM は図 12 のようになる．一方，ユーザ問合せ XQGM では，**for** 節からの相関関係を持つパスを **where** 節と **return** 節中のそれぞれのパスに接ぎ木し，**where** 節の *exsistential quantification* の制約条件を考慮して，*correlated join* を *left(right) outer join* に変換する．この解析の結果，図 11 の演算 *1〜4* は *5* と *8* に接ぎ木され，図 13 の *1〜5*，*8〜12* となり，図 11 の *11:join* は *exsitential quantification* の条件を等価に表現する図 13 の *15:left outer join* となる．

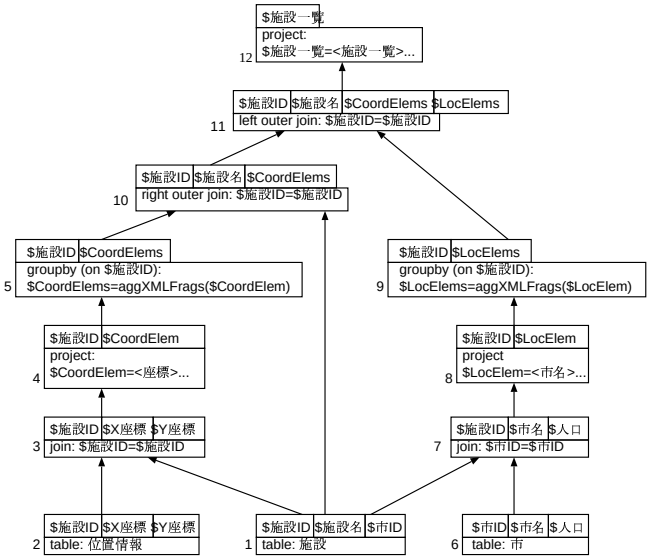

図 12: ビュー定義 XQGM の相関関係分離

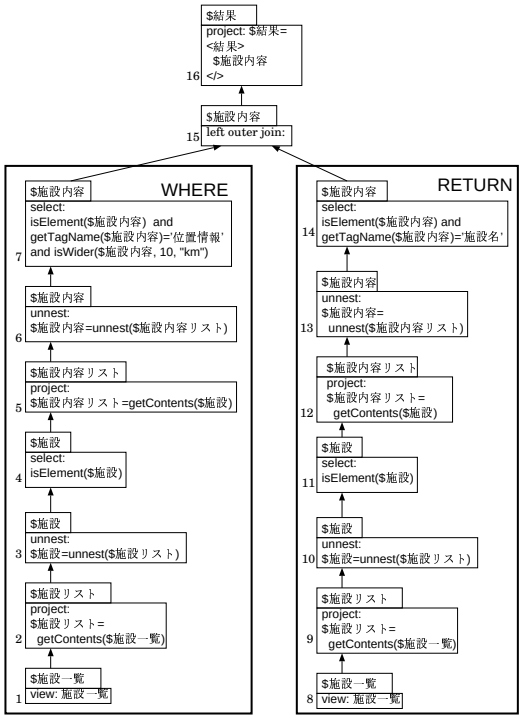

図 13: ユーザ問合せ XQGM の相関関係分離

次に，図 13 の *1,8:view* 演算を図 12 に展開した後，関数合成ルール表 3 に従って，ビュー定義 XQGM の XML フラグメント生成関数とユーザ問合せ XQGM の XML 操作関数を合成すると，図 14 の XQGM が生成される．このとき，前述のように前処理として相関関係の分離を行ったことで，外部関数を含む **where** 節と **return** 節を独立したパスとしてビュー合成を行うことができる．

ビュー合成後は RDBMS で処理可能な演算を XQGM のリーフ側にプッシュダウンする．このとき，3.2 節の演算子プッシュダウンステップの拡張を行い，XML フラグメントを前提として外部関数評価を行うため，外部関数演算子は *project* 演算によって評価に必要な XML フラグメントが生成された直後に配置する．

	関数	合成対象	合成後
1	getTagName	cr8Elem(Tag, Atts, Clist)	Tag
2	getAttributes	cr8Elem(Tag, Atts, Clist)	Atts
3	getContents	cr8Elem(Tag, Atts, Clist)	Clist
4	getAttName	cr8Att(Name, Val)	Name
5	getAttValue	cr8Att(Name, Val)	Val
6	isElement	cr8Elem(Tag, Atts, Clist)	True
7	isElement	cr8Elem 以外	False
8	isText	PCDATA	True
9	isText	PCDATA 以外	False
10	unnest	aggXMLFrags(C)	C
11	unnest	cr8XMLFragList($C_1, ..., C_n$)	$C_1 \cup ... \cup C_n$
12	unnest	cr8AttList($A_1, ..., A_n$)	$A_1 \cup ... \cup A_n$

表 3: 関数合成ルール

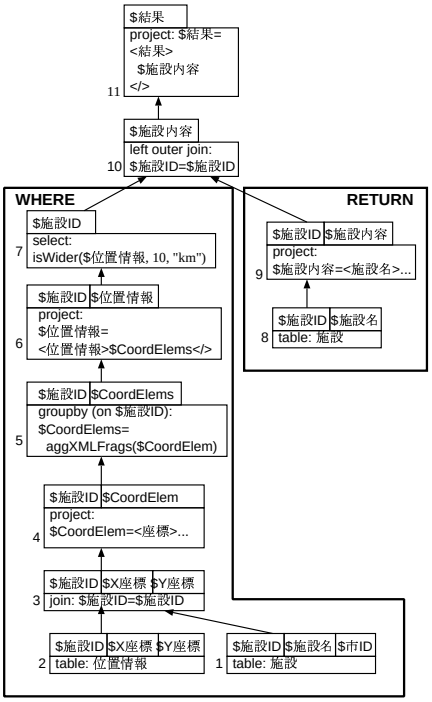

図 14: ビュー合成後 XQGM

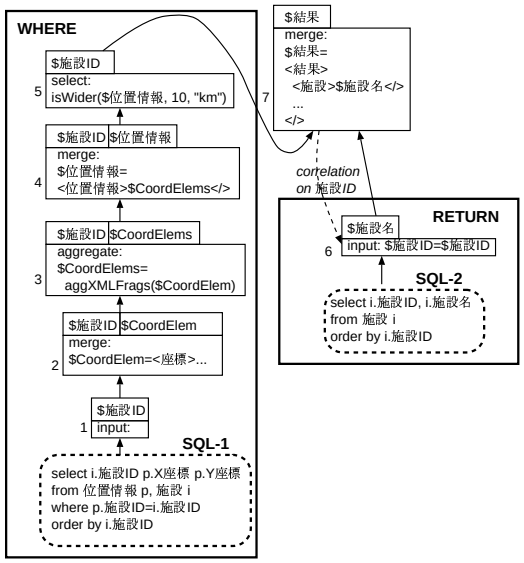

図 15: タグ付け演算子プルアップ後 XQGM

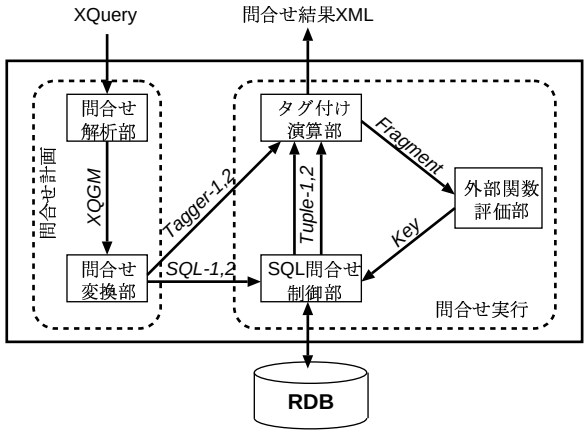

図 16: 分割処理方式

図 14 では，*7:select* の外部関数を評価するために *6:project* で生成される位置情報要素が必要となるので，それ以上リーフ側にプッシュダウンすることはない．

最後に，タグ付け演算子プルアップを行い図 15 を取得する．図 15 で利用される演算子 *Merge* は順位付けされた入力ストリームをマージ，*Aggregate* は集約演算を行う．このタグ付け演算子プルアップでも **where** 節と **return** 節は独立したパスとして扱うので，それぞれのパスから *SQL-1* と *SQL-2* が生成される．これらの SQL 問合せは 3.2 節で述べたように，*OUTER UNION* によって XML 生成に適した形式に再構築されるが，次節で述べるようにその形式は本稿で提案する二つの問合せ処理方式で異なる．

5 問合せ処理方式

本節では，本稿で提案する二つの問合せ処理方式について述べる．

5.1 分割処理方式

分割処理方式は，中間的に外部関数評価を行うことで結果 XML 文書生成に必要なタプルを絞込むことが可能になる．そのため，外部関数評価によるタプル絞込み効率が高い場合に有効であると考えられる．概略図を図 16 に示し，全体の処理の流れを説明する．

問合せ解析部では，与えられた XQuery 問合せを XQGM 表現へ変換し，生成された XQGM 表現から問合せ変換部によって XML フラグメント生成用のタグ付け演算子と SQL 問合せを抽出する．

図 15 に示す外部関数評価用 XML フラグメントを生成する *1*〜*4* のタグ付け演算子 (*Tagger-1*) と SQL 問合せ (*SQL-1*)，問合せ結果 XML を生成する *6*〜*7* のタグ付け演算子 (*Tagger-2*) と SQL 問合せ (*SQL-2*) は，それぞれタグ付け演算部と SQL 問合せ制御部に渡される．

SQL 問合せ制御部は，この *SQL-1* を用いて RDB から XML フラグメント生成用のタプル群 (*Tuple-1*) を取得しタグ付け演算部に渡す．そして、タグ付け演算部では *Tagger-1* によって *Tuple-1* に対するタグ付け演算を行い，外部関数評価用 XML フラグメント (*Fragment*) を生成する．

次に、外部関数評価部では *Fragment* を引数とする外部関数評価を行い，条件を満たした文書要素に対応するタプルのキー値 (*Key*) を SQL 問合せ制御部に渡す．SQL 問合せ制御部は，図 17 に示すように *SQL-2* の WHERE 節に *Key* による条件を付加し，RDB から結果タプル群 *Tuple-2* を取得する．

最後に，タグ付け演算部で *Tagger-2* によって *Tuple-2* にタグ付け演算を行い，問合せ結果 XML を生成する．

```
01:  -- SQL-2: 結果 XML 生成用 --
02: select i. 施設 ID, i. 施設名
03: from 施設 i
04: where i. 施設 ID
05: in  '外部関数評価により取得された施設 ID の集合'
06: order by i. 施設 ID
```

図 17: 分割処理方式で利用する SQL 問合せ

5.2 一括処理方式

一括処理方式は，外部関数評価用 XML フラグメントと結果 XML フラグメント生成に必要なタプル群を一括して取得することで，RDB に対する SQL 問合せ回数を削減する．この処理方式は，外部関数評価によるタプル絞込み効率が低い場合に有効であると考えられる．概略図を図 18 に示し，全体の流れを説明する．

解析部・変換部の処理の流れは分割処理方式と共通である．SQL 問合せ制御部では，問合せ変換部から受け取った *SQL-1* と *SQL-2* を，RDB に一括問合せするために *OUTER UNION* を用いて合成する．この操作には論文 [3] で提案されている SQL 生成アルゴリズムを，問合せ結果タプルに対して外部関数評価用と結果 XML 生成用を区別するための識別子を付加するように拡張する．図 20，図 21，図 22 に示すこの拡張アルゴリズムによって，図 19 のような SQL 問合せが生成される．

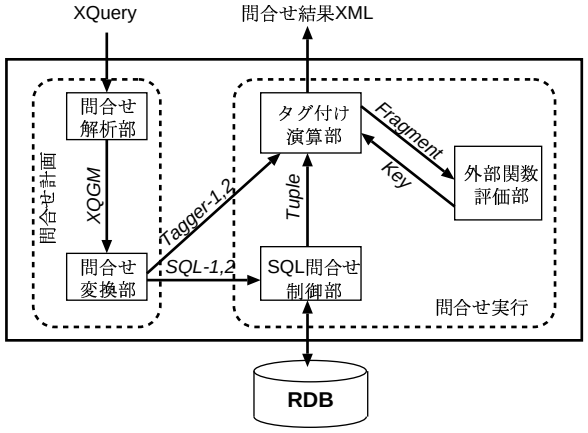

図 18: 一括処理方式

```
01: with 施設テーブル (施設 ID integer, 施設名 varchar(20),
02:                    市 ID integer ) as (
03:   select i. 施設 ID, i. 施設名, i. 市 ID
04:   from 施設 i
05: ),
06: 位置情報テーブル (施設 ID integer,
07:                  X 座標 integer, Y 座標 integer) as (
08:   select i. 施設 ID, p.X 座標, p.Y 座標
09:   from 施設テーブル, 位置情報 p
10:   where i. 施設 ID = p. 位置情報 ID
11: ),
12: outerUnion (target varchar(20), type integer,
13:             施設 ID integer, 施設名 varchar(20),
14:             X 座標 integer, Y 座標 integer) as (
15:   select 'isWider 評価用', 0, 施設 ID, null, X 座標, Y 座標
16:   from 位置情報テーブル
17:     union all
18:   select ' 結果 XML 用', 0, 施設 ID, 施設名, null, null
19:   from 施設テーブル
20: ),
21:  -- Outer Union の結果をタグ付けのためにソートする主問合せ --
22: select target, type, 施設 ID, 施設名, X 座標, Y 座標
23: from outerUnion
24: order by 施設 ID
```

図 19: 一括処理方式で利用する SQL 問合せ

この合成された SQL 問合せによって，RDBMS から外部関数評価と結果 XML 生成に必要なタプルを一括に取得可能となる．タグ付け演算部は外部関数評価に必要な XML フラグメントを生成し，これを外部関数評価部で評価する．そして，評価結果として取得された *Key* を用いて，結果 XML 生成用タプルの取捨選択を行い，条件を満たすタプルから問合せ結果 XML を生成する．

```
01: Function buildBatchSQL( SQL 問合せ SQL-1, SQL 問合せ SQL-2,
02:           外部関数名 extfuncname ) return SQL 問合せ文字列
03: {
04:   // 一括処理方式で利用する SQL 問合せ例の 1〜11 行に相当する.
05:   resultstring = buildPath( SQL-1, SQL-2 );
06:
07:   // SQL 問合せ例の 12〜20 行に相当する.
08:   resultstring += buildOuterUnion( SQL-1, SQL-2,
09:                     extfuncname );
10:
11:   // OUTER UNION の結果ストリームを順序づける主問合せを生成
12:   // 一括処理方式で利用する SQL 問合せの例, 22〜24 に相当する.
13:   resultstring += "select " + 最終結果として出力される属性名;
14:   resultstring += "from outerUnion";
15:   resultstring += "orderby ";
16:
17:   // XML フラグメント生成に適したソーティング
18:   for( each subquery in SQL-1,SQL-2 中の全問合せ ) {
19:     resultstring += subquery の各 ID を取得;
20:   }
21:
22:   // 構築された SQL 問合せ
23:   return resultstring;
24: }
```

図 20: 一括型 SQL 問合せ生成アルゴリズム 1

```
01: Function buildPath( SQL 問合せ SQL-1, SQL 問合せ SQL-2 )
02:           return SQL 問合せ文字列
03: {
04:   // 主問合せとは XML ビューのトップレベル要素を構成する
05:   // テーブルへの問合せであり,
06:   // 出力例では施設テーブルへの問合せ 1〜5 行に相当する.
07:   mainquery = SQL-1 と SQL-2 から主問合せを取得;
08:   resultstring =  "with " + mainquery の名前 + "("
09:                 + 出力される属性名と型 + ") as (";
10:   resultstring += "select " + 結果ストリーム操作用キー属性
11:                 + ", " mainquery の XML フラグメント生成用属性;
12:   resultstring += "from " + mainquery の from 節;
13:   resultstring += "where " + mainquery の述語;
14:   resultstring += ")";
15:
16:   // 主問合せ以外の問合せである副問合せを OUTER JOIN で結合
17:   // 出力例では 6〜11 行に相当する.
18:   while( SQL-1 と SQL-2 に次の副問合せが存在する ) {
19:     subquery = SQL-1 と SQL-2 から次の副問合せを取得;
20:     resultstring += "," + subquery の名前 + "("
21:                   + 出力される属性名と型 + ") as (";
22:
23:     // 出力される属性を指定
24:     // 出力例でキー属性は施設に相当する.
25:     resultstring += "select " + 主問合せテーブルのキー属性
26:                   + ", " + subquery の XML フラグメント生成用属
性;
27:
28:     // 主問合せテーブルとの結合
29:     resultstring += "from " + 主問合せの名前
30:                   + ", " + subquery の from 節;
31:     resultstring += "where " + subquery の述語;
32:     resultstring += ")";
33:   }
34:   return resultstring;
35: }
```

図 21: 一括型 SQL 問合せ生成アルゴリズム 2

6 評価実験

二つの処理方式に関して簡単な評価実験を行い，それぞれの問合せ処理効率を比較した．

実験用システムは Java で実装し JDK1.2 を用いた．データベースとして PostgreSQL 7.1 を対象とし，データベースとの通信には JDBC2.0 を利用した．また，実験環境のプラットフォームは Celeron 300MHz, メモリ 196MB，OS は Microsoft Windows 2000 Service Pack 2 である．

PostgreSQL に格納されるリレーションは，これまで論文中で扱ってきた市，施設，位置情報テーブルを利用する．このリレーション間の関係は各市ごとに四つの施設が存在し，それぞれの施設に四つの位置情報が対応するものとし，市の数をパラメータで調整することでリレーションのタプル数を変化させる．

```
01: Function buildOuterUnion( SQL 問合せ SQL-1, SQL 問合せ SQL-2,
02:           外部関数名 extfuncname ) return SQL 問合せ文字列
03: {
04:   // OUTER UNION で主問合せから副問合せまでをつなげる.
05:   reuststring = ",outerUnion (" + 出力属性と型 + ") as (";
06:
07:   // 外部関数評価用
08:   for( index = 0; index < SQL-2 の問合せ要素数; ++index ) {
09:     if( index > 0 )
10:       resultstring += "union all";
11:
12:     // 主・副問合せを表す type 属性をパスに追加
13:     resultstring += "select " + extfuncname + index + ", ";
14:
15:     // 他の属性を追加し, from 節を作成
16:     currsubquery = SQL-2 の index 番目の問合せ要素を取得;
17:     resultstring += currsubquery の全属性と
18:                     必要に応じて null を追加;
19:     resultstring += "from " + currsubquery の名前;
20:   }
21:   // 結果 XML 生成用
22:   for( index = 0; index < SQL-1 の問合せ要素数; ++index ) {
23:     resultstring += "union all";
24:
25:     // 主・副問合せを表す type 属性をパスに追加
26:     resultstring += "select " + "結果 XML 用" + index + ", ";
27:
28:     // 他の属性を追加し, from 節を作成
29:     currsubquery = SQL-1 の index 番目の問合せ要素を取得;
30:     resultstring += currsubquery の全属性と
31:                     必要に応じて null を追加;
32:     resultstring += "from " + currsubquery の名前;
33:   }
34:   resultstring += "),";
35:
36:   return resultstring;
37: }
```

図 22: 一括型 SQL 問合せ生成アルゴリズム 3

6.1 実験内容

分割処理方式の全体の問合せ処理コストは，次の各処理コストの合計からなると考えられる．

- SQL 問合せ *SQL-1*
- XML フラグメント生成
- 返値となるキー値リスト生成を含む外部関数処理
- SQL 問合せ *SQL-2*
- 問合せ結果 XML 生成

また，一括処理方式の問合せ処理コストに関しては次のようになる．

- *OUTER UNION* によって合成された SQL 問合せ *SQL-3*
- XML フラグメント生成
- 外部関数処理
- 問合せ結果 XML 生成

これらの処理の中で，XML フラグメント生成と問合せ結果 XML 生成コストは両処理方式で必要となるので二つの問合せ処理方式の処理能力の明確な差として現れないと考えられる．そこで，本実験ではこれら共通の問合せ処理機能は評価対象外として，SQL 問合せ *SQL-1,2,3* と外部関数処理を中心に二つの処理能力を比較した．本実験に用いた外部関数は，実際に利用される XML フラグメントを対象とした複雑な処理を行うものではなく，指定された選択率にもとづいて，入力されたリレーションタプルを選択する単純な機能を実装したものである．

6.2 実験結果

図 23 は分割処理方式実験結果の合計処理時間に関するグラフである．ここでタプル数は左から市，施設，位置情報テーブルを表す．タプル数と選択率の上昇と共に急激に処理時間が増加することが分かる．

図 24 は一括処理方式実験結果の合計処理時間に関するグラフである．合計処理時間は外部関数選択率にはあまり影響されず，対象となるリレーションタプル数に比例することが明確に示されている．

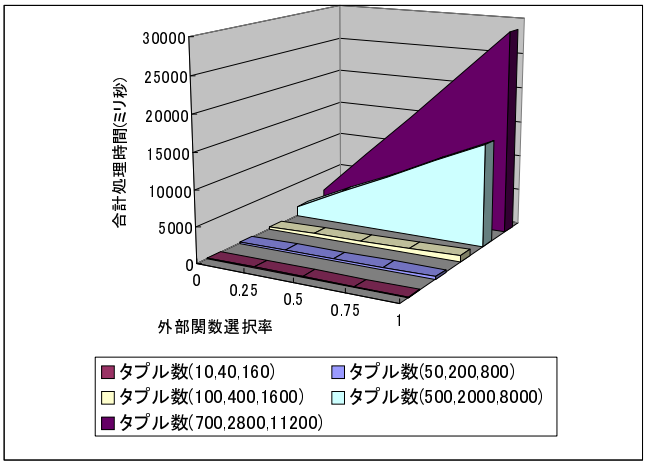

図 23: 分割処理方式実験結果グラフ

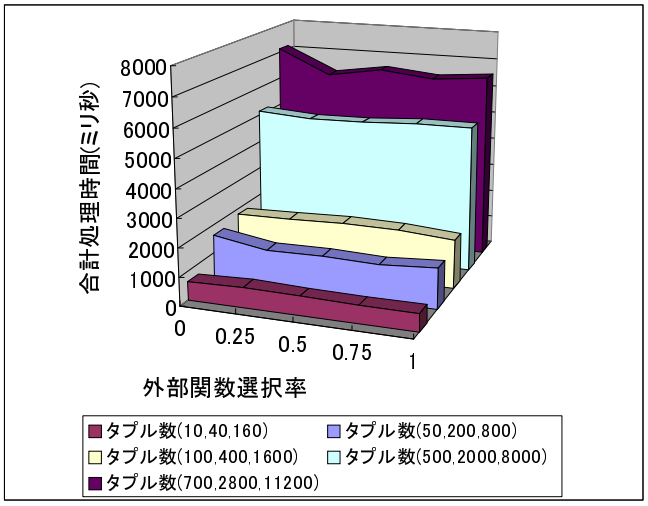

図 24: 一括処理方式実験結果グラフ

図 25 は市テーブル 100 タプル，施設テーブル 400 タプル，位置情報テーブル 1600 タプルにおける分割処理方式と一括処理方式の合計処理時間の比較グラフである．外部関数の全ての選択率において分割処理方式の方が処理効率が高いことが分かる．

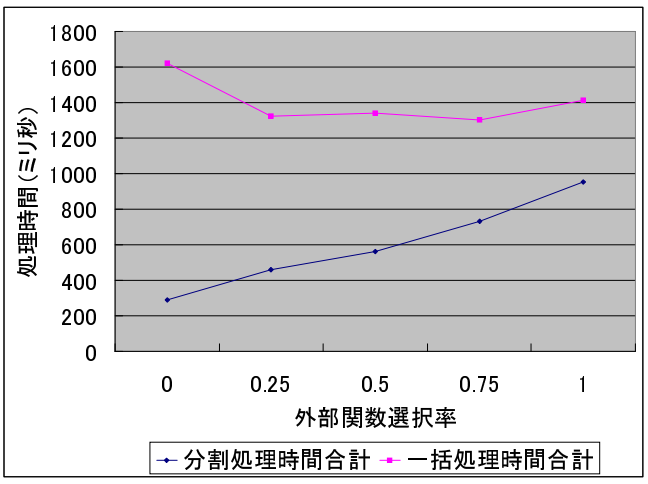

図 25: 比較グラフ (1)

図 26 は市テーブル 500 タプル，施設テーブル 2000 タプル，位置情報テーブル 8000 タプルにおける分割処理方式と一括処理方式の合計処理時間の比較グラフである．選択率 0.25 までは分割処理方式の方が効率的であるが，それより選択率が高くなると一括処理方式の方が効率的である．

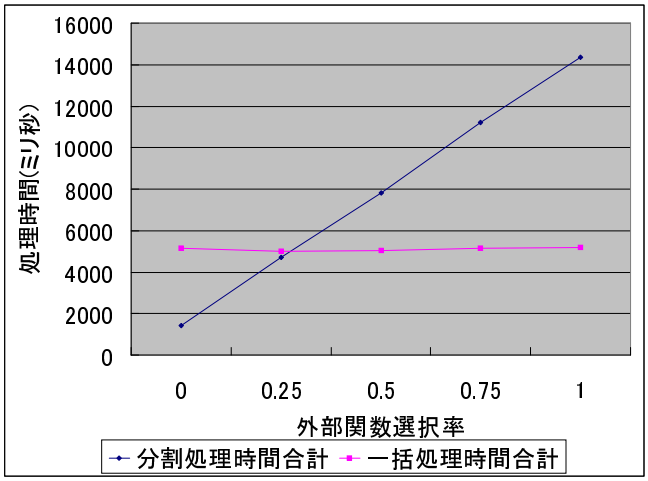

図 26: 比較グラフ (2)

図 27 は市テーブル 700 タプル，施設テーブル 2800 タプル，位置情報テーブル 11200 タプルにおける分割処理方式と一括処理方式の合計処理時間の比較グラフである．図 26 と比較して，さらに広い範囲の選択率において一括処理方式の方が効率的であることが分かる．

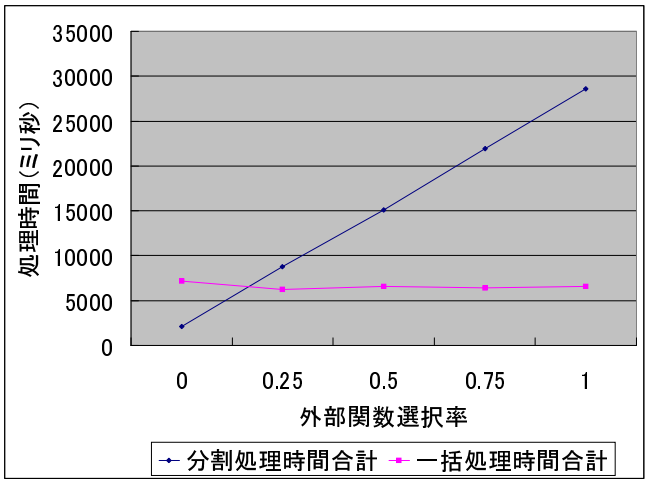

図 27: 比較グラフ (3)

6.3 実験の考察

6.2 節の実験結果から次のようなことが考察される．

今回実験で用いた外部関数はタプルの取捨選択のみを対象とし，その評価コストは非常に小さかった．しかし，分割処理方式における外部関数評価では，返値となるキー値リスト生成のための文字列操作が必要となる．そのため，高い選択率のときには大規模なキー値リストを生成する必要があり，このコストが外部関数評価時間に影響していると考えられる．一方，一括処理方式では，外部関数評価はタプルストリームからの逐次的なタプルの取捨選択のみを対象とするので，選択率によって評価コストに大きな変化は現れなかった．

また，図 25，図 26，図 27 からも明らかな様に，分割処理方式では，比較的処理対象となるタプルが少ない場合，一括処理方式に比べて効率的な処理が可能となるが，処理対象のタプル数が大量になると，外部関数の選択率の上昇と共にキー値生成操作を含む外部関数評価コストと，外部関数によって取得された大量のキー値による選択条件を持つ *SQL-2* 問合せ処理コストが急激に増加する．

一方，一括処理方式では，処理対象タプル数が少ない場合には，*OUTER UNION* によって合成された一括型の SQL 問合せ処理コストが，分割処理方式の単純な複数の SQL 問合せ処理コストの合計よりも高くなるが，処理対象となるタプルが多い場合には，外部関数処理コストと SQL 問合せコストが分割処理方式よりも安定しているので，急激な処理コストの増加が起こらず，結果的に分割処理コストよりも効率的な問合せ処理が可能となる．

従って，分割処理方式は処理対象となるリレーショナルデータベースのタプル数が少なく外部関数の選択率が低い場合には有効となる．他方で，一括処理方式はタプル数と外部関数の選択率に大きく依存することはなく比較的安定した問合せ処理コストとなるので，タプル数が多く外部関数の選択率が高い場合に分割処理方式よりも有利になることが明らかになった．

7 まとめと今後の課題

本稿では，SQL を処理可能な一般的 RDBMS とその上位に位置するミドルウェアの連携により，リレーショナルデータベース上の XML ビューに対する外部関数を含む問合せを処理する方式を提案した．具体的には，XML フラグメント生成と外部関数評価に留意した問合せ計画の方式を提案し，また分割処理方式と一括処理方式の二つの問合せ処理方式を示した．さらに，これらの二つの問合せ処理方式の特徴について，実験による検証を行った．

今後の課題としては，より一般的な外部関数処理コストを考慮した更に詳細な実験を行う必要がある．また，問合せや各種データベースパラメタに応じて分割処理方式と一括処理方式を使い分ける複合型処理方式の実現についても検討する予定である．

謝辞

本研究の一部は，日本学術振興会科学研究費基盤研究 (B)(12480067)，奨励研究 (A)(12780183)，および文部科学省科学研究費特定領域研究 (C)(13224008) による．

参考文献

[1] Mary F. Fernandez, Atsuyuki Morishima, and Dan Suciu ”Efficient Evaluation of XML Middle-ware Queries.” *Proc. ACM SIGMOD 2001*, May 2001.

[2] Jayavel Shanmugasundaram, Jerry Kiernan, Eugene Shekita, Catalina Fan, and John Funderburk ”Querying XML Views of Relational Data.” *Proc. 27th VLDB Conference*, Roma, Italy, 2001.

[3] Jayavel Shanmugasundaram, Eugene J. Shekita, Rimon Barr, Michael J. Carey, Bruce G. Lindsay, Hamid Pirahesh, and Berthold Reinwald, ”Efficiently Publishing Relational Data as XML Documents”, *The VLDB Journal*, pp 65-76, 2000.

[4] Scott Boag, Don Chamberlin, Mary F. Fernandez, Daniela Florescu, Jonathan Robie, Jérôme Simèon, and Mugur Stefanescu, XQuery 1.0: An XML Query Language http://www.w3.org/TR/2001/WD-xquery-20011220/

[5] Norihide Shinagawa and Hiroyuki Kitagawa, ”Extension Mechanism in Extensible XML Query Language X^2QL”, *Proc. 2nd International Conference on Web Information Systems Engineering (WISE 2001)*, Kyoto, Japan, December 2001.