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[††]

XMLの普及にともない，リレーショナルデータベース中のデータを仮想的なXMLビューとして提供し，XQueryなどのXML問合せ言語によるアクセスを可能とする手法に対する関心が高まっている．特に，RDBMSとミドルウェアの連携のもと，問合せ処理の多くをRDBMSに委譲することで効率的処理を行う方式が研究されている．一方，XML問合せにおいてドメイン依存の処理機能を導入する手法として，利用者定義の外部関数を支援するアプローチが知られている．一般にこれらの外部関数は，XMLフラグメントに対する固有の処理を汎用プログラミング言語などで記述したものである．外部関数を直接RDBMSで評価することは一般に困難なため，このような問合せを既存のアプローチでそのまま処理することはできない．そこで本論文では，RDBMSとミドルウェアの連携による，XMLビューに対する外部関数を含む問合せの処理方式を提案する．本方式では，与えられた問合せに含まれる処理をできるだけRDBMSへ委譲することを基本とするが，外部関数の評価については，評価に必要なデータをRDBMSから獲得した後ミドルウェア自身が行うことで，汎用的なRDBMS上での処理を可能とする．具体的には3種類の連携方式を提案し，評価実験によりそれらの処理効率について検討する．

Processing Queries Including User-defined Foreign Functions on XML Views over Relational Databases

Jun Kawada,[†,☆] Yoshiharu Ishikawa[††] and Hiroyuki Kitagawa[††]

With the increased popularity of XML, XML publishing of RDBs has been attracting a lot of research interests. One of typical approaches is to use a middleware system to render XML views over RDBs and to allow users to access data with XML query languages such as XQuery. The query processing is done efficiently by making the best of the querying power of RDBMSs. Namely, XML queries are translated into SQL queries and tagging operations, which are processed by the RDBMSs and middleware, respectively. In some XML query languages including XQuery, use of user-defined foreign functions is enabled or planned as an extension feature to cope with domain dependent semantics. Foreign functions are defined for XML fragments, and their implementations are often given by codes in a general programming language. The existing query processing schemes on XML views do not consider cases where foreign functions are included in XML queries. In this paper, we propose extended schemes to process XML queries in such cases. In the proposed schemes, the middleware takes care of processing foreign functions as well as tagging operations. Therefore, the proposed schemes are applicable to XML views on commonly available RDBMSs. Three types of query processing schemes are proposed, and their performance is studied with experiments.

1. はじめに

XMLは急速にインターネット上での各種データの標準的な交換様式となりつつあり，XMLを支援するためのデータベース技術の開発がますます重要となってきている[1),6)]．一方，基幹業務データなどのデータはリレーショナルデータベース（以下，RDB）に格納されていることが多く，RDB中のデータをどのようにXMLの形で外部に提供するかは近年重要な研究テーマの1つとなっている．

このような問題に対し，ミドルウェアを用いてRDB上にXML形式のビューを提供し，XML問合せ言語を用いた問合せを可能とするアプローチがいくつか研

† 筑波大学システム情報工学研究科
Graduate School in Systems and Information Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
☆ 現在，株式会社リコー
Presently with Ricoh Co. Ltd.

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
…	…	…

位置情報

市ID	X座標	Y座標
C0100	100	400
C0100	100	200
…	…	…
C0101	110	250
C0101	150	200
…	…	…

施設

施設ID	施設名	市ID
I0015	Aモール	C0100
I0016	H公園	C0100
I0017	M研究所	C0100
I0018	Bモール	C0101
I0019	Cモール	C0102
I0020	D飛行場	C0103
…	…	…

図 1 リレーショナルデータベースの例
Fig. 1 An example of relational database.

```
01: <市一覧>
02:  <市 id="C0100">
03:   <市名>つくば市</市名>
04:   <人口>16</人口>
05:   <位置情報>
06:    <座標><X 座標>100</X 座標><Y 座標>400</Y 座標></座標>
07:    <座標><X 座標>100</X 座標><Y 座標>200</Y 座標></座標>
08:    ...
09:   </位置情報>
10:   <施設一覧>
11:    <施設 id="I0015"><施設名>A モール</施設名></施設>
12:    <施設 id="I0016"><施設名>H 公園</施設名></施設>
13:    ...
14:   </施設一覧>
15:  </市>
16:  ...
17: </市一覧>
```

図 2 XML ビューの例
Fig. 2 An example of XML view.

```
01: <結果>{
02:   for $ 市 in view("市一覧")/市一覧/市
03:   where isWider($ 市/位置情報, 10000, "km")
04:     and $ 市/人口 >= 10
05:   return <市内容>
06:            $ 市/市名
07:            $ 市/施設一覧
08:          </市内容>
09: }</結果>
```

図 3 ユーザ問合せの例
Fig. 3 An example of user query.

究されている[3)~5),8),9),12)]．たとえば，**図 1** のようなRDBのテーブルに対して，**図 2** のようなXMLビューを提供することが可能である．

XPERANTOグループは，このように定義されたRDB上のXMLビューのための問合せ処理手法の開発を行っている[8)]．XMLビューに対しXML問合せ言語XQuery[2)]に基づく問合せ（例：**図 3**）が与えられたとき，問合せをRDBMSで処理可能な形式にできるだけ変換しプッシュダウンすることで効率的に処理するアプローチを提案している．

一方，XQueryをはじめいくつかのXML問合せ言語では，図3に示す問合せ例の中で用いられている`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[2),10),11)]．これらの外部関数は対象XML文書要素フラグメントのメモリ上での表現を前提に汎用プログラミング言語などで記述されるため，上記のようなXMLビュー上で用いられた場合には問合せ処理上の配慮が必要である．

本論文では，SQLを処理可能な一般的なRDBMSとその上位に位置するミドルウェアの連携により，RDB上のXMLビューに対する外部関数を含む問合せの処理方式を提案する．このため，上記のXPERANTOにおけるアプローチを外部関数処理を考慮して拡張する．具体的には3種類の問合せ処理方式を提案し，それらの処理効率を実験に基づいて比較する．

本論文の構成は以下のようになる．2章では本提案方式の概要について述べる．3章では本提案方式のベースとなったXPERANTOのアプローチについて簡単に説明する．4章では，XPERANTOのアプローチに対して外部関数処理などのために本研究で独自に拡張した部分を中心に，問合せ変換方式について述べる．5章では，4章で述べた問合せ変換結果に基づいて実際に問合せを処理するための，3種類の問合せ処理方式について説明する．6章ではPostgreSQLを用いた評価実験を示し，その分析を行う．最後に7章でまとめと今後の課題について述べる．

2. 本提案方式の概要

XPERANTO[3),8),9)]やSilkRoute[4),5)]などのRDB上のXMLビューに対する問合せ処理の先行研究では，与えられたXML問合せからSQLで記述可能な処理をできるだけ抽出しRDBMSに委譲することで，問合せ処理の効率化を図っている．理想的にはRDBから取得されたタプル集合をXML形式に変換するためのタグ付け処理のみがミドルウェアで実行され，それ以外の処理はRDBMSが担当することになる．これにより，RDBMSの問合せ処理能力を有効に活用することができる．

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フラグメントがメモリ上にDOM（Document Object Model）などの形式で存在することを前提に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，外部関数の評価を行う際には，RDBからタプルとして得たデータにタグ付けを行いメモリ上でXMLフラグメントをまず生成する必要がある．一般にRDBMS内ではこのような処理を行うことは難しいため，図3に示すような外部関数を含むXML問合せを先行研究における方式で処理することは困難である．

この問題に対してXPERANTOグループは，RDBMS内部にXMLフラグメントの生成機能とそれに基づく外部関数評価機能を追加し，外部関数を含

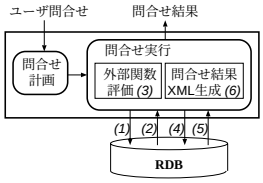

図 4 分割処理方式
Fig. 4 Two-step processing method.

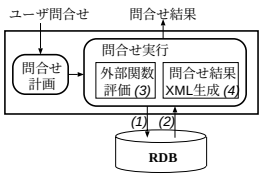

図 5 一括処理方式
Fig. 5 One-step processing method.

んだ問合せの処理を実現するというアイデアを論文 8) の今後の課題の中で簡単に触れている．しかし，このようなアプローチでは RDBMS 内部に専用の追加機能を実装する必要があり，一般の RDBMS に対して容易に適用できないという問題点がある．

そこで本論文では，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の処理方式を提案する．提案方式では外部関数の評価はミドルウェアで行うものとし，RDBMS に特別の拡張はいっさい必要としない．具体的な問合せ処理方式としては，大きく分けて**図 4** の**分割処理方式**と**図 5** の**一括処理方式**の 2 種類を考える．

各処理方式では，まずユーザから与えられるユーザ問合せ（XQuery で記述）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．

問合せ計画に基づく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) 外部関数評価結果をもとにデータをさらに絞り込むための条件を付加し，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．

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** のようになる．実際には，上記 (4) を実行するための方法として，一時テーブルを用いた方式と用いない方式の 2 種類を検討する．

一括処理方式での問合せ処理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うえで，問合せ結果 XML を生成．

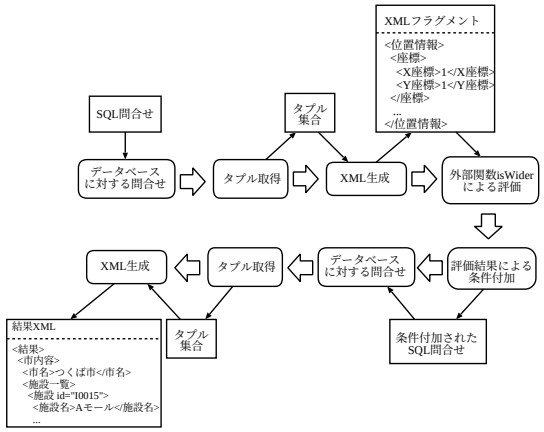

図 6 分割処理方式における問合せ処理の流れ
Fig. 6 Query processing with two-step processing method.

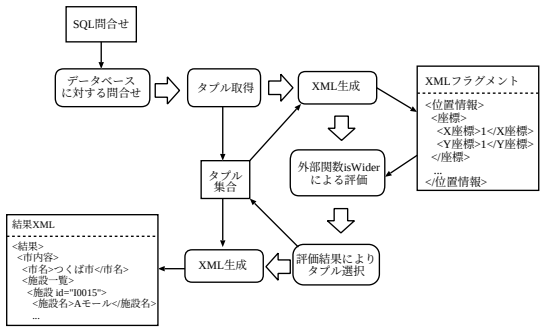

図 7 一括処理方式における問合せ処理の流れ
Fig. 7 Query processing with one-step processing method.

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** のようになる．

これらの処理方式の詳細は 5 章で述べる．次章ではこれらの処理方式における問合せ計画を構築するうえでベースとした，XPERANTO における XML ビュー問合せ処理方式の概略について説明する．

3. XPERANTO における基本アプローチ

3.1 アプローチの概要

XPERANTO では，あらかじめ，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** のような仮想的なデフォルト XML ビューを自動的に構築する．ユーザは，このデフォルト XML ビューに対し，XQuery を用いてビュー定義や問合せを行うこ

```
01: <db>
02:   <市>
03:     <row>
04:       <市ID>C0100</市ID>
05:       <市名>つくば市</市名>
06:       <人口>16</人口>
07:     </row>
08:     ...
09:   </市>
10:   <位置情報>
11:     <row>
12:       <市ID>C0100</市ID>
13:       <X座標>100</X座標>
14:       <Y座標>400</Y座標>
15:     </row>
16:     ...
17:   </位置情報>
18:   <施設>
19:     <row>
20:       <施設ID>I0015</施設ID>
21:       <施設名>A モール</施設名>
22:       <市ID>C0100</市ID>
23:     </row>
24:     ...
25:   </施設>
26: </db>
```

図 8 デフォルト XML ビュー
Fig. 8 Default XML view.

```
01: create view 市一覧 as (
02:   <市一覧>
03:     for $ 市 in view("default")/市/row
04:     return
05:       <市 id=$ 市/市ID>
06:         <市名>$ 市/市名</市名>
07:         <人口>$ 市/人口</人口>
08:         <位置情報>
09:           for $ 位置 in view("default")/位置情報/row
10:           where $ 市/市ID = $ 位置/市ID
11:           return
12:             <座標>
13:               <X座標>$ 位置/X座標</X座標>
14:               <Y座標>$ 位置/Y座標</Y座標>
15:             </座標>
16:         </位置情報>
17:         <施設一覧>
18:           for $ 施設 in view("default")/施設/row
19:           where $ 市/市ID = $ 施設/市ID
20:           return
21:             <施設 id=$ 施設/施設ID>
22:               <施設名>$ 施設/施設名</施設名>
23:             </施設>
24:         </施設一覧>
25:       </市>
26:   </市一覧>
27: )
```

図 9 XML ビュー定義 XQuery
Fig. 9 XQuery for the XML view definition.

とになる．たとえば，**図 9** に示すようなビュー定義 XQuery を用いることで，図 2 に示した XML ビューが定義できる．このアプローチにより，ユーザは下位に位置する RDB を意識する必要がなく，XQuery を用いて一貫した XML データベース操作が行える．

XQuery 問合せ処理は以下のステップからなる．

(1) **問合せ解析**：XQuery 問合せを解析し，問合せ内部表現に変換する．
(2) **問合せ変換**：ビュー定義に基づいて問合せ内部表現を変換し，RDBMS に発行する SQL 問合せとミドルウェアで実行するタグ付け演算子群を生成する．
(3) **問合せ実行**：SQL 問合せを RDBMS に発行し，RDBMS から得られた問合せ結果にタグ付け演算子群に基づく XML タグ付けを行う．

(1) と (2) をまとめて**問合せ計画**と呼ぶ．以下では，この問合せ計画を中心に説明する．

3.2 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，それぞれ **XQGM**（XML Query Graph Model）と呼ばれる問合せ内部表現に変換される．それぞれ**ビュー定義 XQGM**，**ユーザ問合せ XQGM** と呼ぶ．

図 10 に，図 9 のビュー定義から生成されたビュー定義 XQGM を示す．XQGM は**表 1** に示す拡張リレーショナル代数演算子をノードとする有向グラフである．図 10 では，番号 1 から 11 のノードがそれぞれ拡張

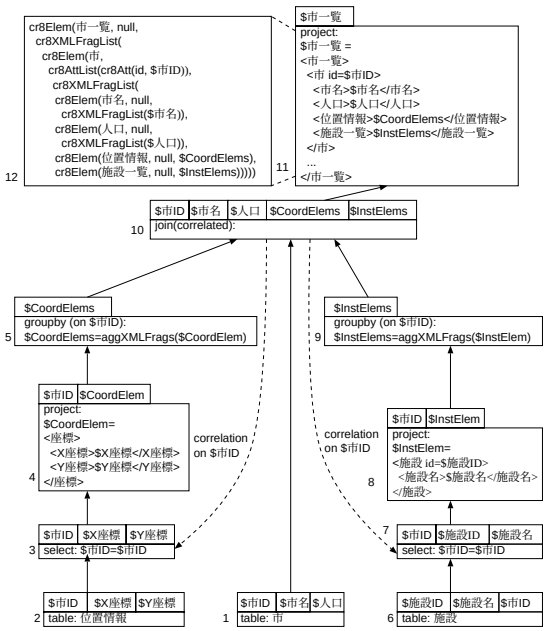

図 10 XML ビュー定義 XQGM
Fig. 10 XQGM for XML view definition.

表 1 XQGM 演算子
Table 1 XQGM operators.

演算子	機能
table	リレーショナルテーブルを指定
project	射影演算
select	選択演算
join	結合演算
groupby	集約関数の適用
orderby	属性値に基づくソート処理
union	和集合演算
unnest	入力をアンネストしたリストを返す
view	ビューを指定

リレーショナル代数の演算子に対応しており，実線の矢印は演算子間の入出力関係を示している．

なお，ノード 10 は *correlated join* 演算を表している．correlated join 演算とは，入力として副問合せを受け取っている結合演算で，その副問合せが他の入力に対し相関（correlation）を有しているもののことである．図 10 では，ノード 2〜5 およびノード 6〜9 が，各市に対し位置情報および施設情報を求める副問合せにあたり，ノード 3，7 の select 演算では，ノード 1 の市テーブルの市 ID 属性の値を参照し，選択を行っている．このような参照部分が相関にあたり，点線の矢印はその対応関係を示している．XML 文書を生成する際には，ある XML 要素に対し関連する XML 要素をその子として埋め込む処理が頻繁に現れるため，correlated join 演算が頻出することになる．

XQGM の演算子中では，XML のタグ付け操作などのための XML 関数を呼び出すことが可能である．**表 2** に各 XML 関数と，その関数を呼び出せる演算子を示している．その使用例として，図 10 のノード 5 を見てみる．このノードは XML 関数 aggXMLFrags によるグループ化の処理を表しており，ノード 4 の出力である XML 要素「座標」とその座標値が対応する市 ID のペアの集合を受け取り，それらを市 ID でグループ化し，XML フラグメントリストを生成する．例として，図 1 の「つくば市」（市 ID：C0100）に対して作成される XML フラグメントリストを**図 11** に示す．

なお，図 10 のノード 11 は，ノード 10 以下で得られたデータをもとにタグ付けする処理を略記したものであり，実際のタグ付け処理はノード 12 の XML 関数群の呼び出しにより実現される．cr8Elem，cr8Attr 関数により，XML の要素および属性が作成され，cr8XMLFragList，cr8AttList 関数により要素内容リストおよび属性リストが作成されることが分かる．

次に，ユーザ問合せ XQGM について説明する．

表 2 XML 関数とその関数を呼び出す演算子
Table 2 XML functions and their corresponding operators.

関数	機能	演算子
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
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
cr8Att(Name, Val)	属性名 Name，属性値 Val からなる属性生成	project
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
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
getTagName(Elem)	要素 Elem の要素名を返す	project, select
getAttributes(Elem)	要素 Elem の属性リストを返す	project, select
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
getAttName(Att)	属性 Att の属性名を返す	project, select
getAttValue(Att)	属性 Att の属性値を返す	project, select
isElement(E)	E が要素であれば true を，さもなくば false を返す	select
isText(T)	T がテキストであれば true を，さもなくば false を返す	select
unnest(List)	リスト List をアンネストする	unnest

```
01: <座標>
02:   <X座標>100</X座標>
03:   <Y座標>400</Y座標>
04: </座標>
05: <座標>
06:   <X座標>100</X座標>
07:   <Y座標>200</Y座標>
08: </座標>
09: ...
```

図 11 中間結果の XML フラグメントリスト
Fig. 11 XML fragment list of an intermediate result.

図 12 は，図 3 のユーザ問合せに対するユーザ問合せ XQGM である．ビュー定義 XQGM がリレーショナルテーブルをもとに XML ビューを導出する過程を表しているのに対し，ユーザ問合せ XQGM は図 12 のノード 1 で示されるデフォルト XML ビューから問合せ結果を生成する過程を表している．四角で囲った部分グラフは，元の XQuery 問合せのそれぞれ `for` 節，`where` 節，`return` 節に対応している．なお，図 12 のノード 11 からノード 18 への辺は存在限量の依存関係があることを表している．これは，ノード 18 でそれぞれの市に対する副問合せ結果を統合する処理において，ノード 11 以下の `where` 節の条件が成立した場合にのみ，その市に対する統合結果を生成することを意味している．

なお，本研究では，外部関数の評価や問合せ記述の

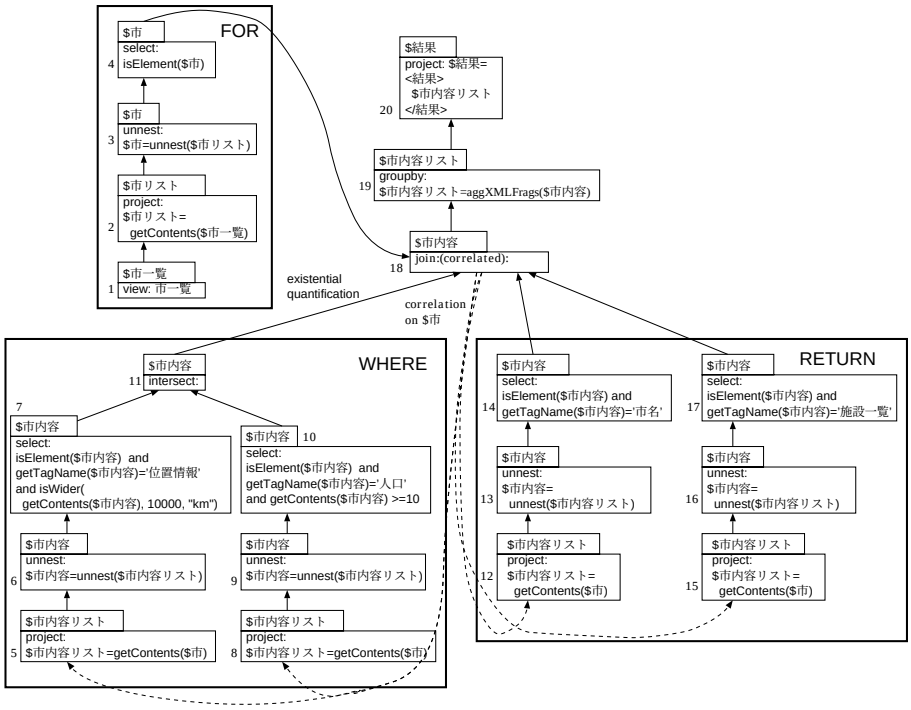

図 12 ユーザ問合せ XQGM
Fig. 12 XQGM for user query.

一般化などのため，XPERANTO グループにより提案された XQGM の演算子や構成方式について拡張を図っている．特に，図 12 に示されるようなユーザ問合せ XQGM における拡張が大きいが，その具体的な箇所については後述する．

3.3 問合せ変換

このステップでは，ユーザ問合せ XQGM をビュー定義 XQGM と合成し，変換を行うことで SQL 問合せとタグ付け演算子群を生成する．この変換手順の概要は次のようになる．

(1) **ビュー合成**：生成されたユーザ問合せ XQGM とビュー定義 XQGM を合成する．先の例でいえば，図 12 のノード 1 を図 10 のノード 11 以下のグラフで置き換えたグラフを作成することに相当する．次いで，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 フラグメントの生成を削減する．

関数合成のためのルールを**表 3** に示す．先の例では，図 12 のノード 2 の `getContents($ 市一覧)` が図 10 のノード 12 のトップレベルの `cr8Elem(市一覧, null, cr8XMLFragList(...))` と合成されることで `cr8XMLFragList(...)` に縮約されることになる．このような処理が繰り返し適用されることでグラフが簡略化される．

表 3 関数合成ルール
Table 3 Function composition rules.

関数	合成対象	合成結果
getTagName	cr8Elem(Tag, Atts, Clist)	Tag
getAttributes	cr8Elem(Tag, Atts, Clist)	Atts
getContents	cr8Elem(Tag, Atts, Clist)	Clist
getAttName	cr8Att(Name, Val)	Name
getAttValue	cr8Att(Name, Val)	Val
isElement	cr8Elem(Tag, Atts, Clist)	True
isElement	cr8Elem 以外	False
isText	PCDATA	True
isText	PCDATA 以外	False
unnest	aggXMLFrags(C)	C
unnest	cr8XMLFragList($C_1, \ldots, C_n$)	$C_1 \cup \cdots \cup C_n$
unnest	cr8AttList($A_1, \ldots, A_n$)	$A_1 \cup \cdots \cup A_n$

(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

```
01: for $ 市 in view("市一覧")
02: where $ 市一覧/市/人口 >= 10
03: return $ 市
```

図 13 XPERANTO が想定する問合せの例
Fig. 13 Sample query assumed in XPERANTO.

子は XQGM のルート方向にできるだけ移動し，リーフ側に残った演算子群から RDBMS に対する SQL 問合せ群を生成する．

(4) 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処理を実現するため，生成された SQL 問合せ群の出力を outer union 演算により統合し，出力 XML の構造に合わせて再構成する[9]．

4. XPERANTO アプローチの拡張

4.1 基本的なアプローチ

この章では，本研究における XPERANTO のアプローチの拡張と，処理対象として想定する XQuery 問合せについて述べる．XPERANTO の論文 8) では，**図 13** のような単純な形式の XQuery 問合せしか考慮しておらず，以下のような問題点が存在する．

- トップレベルのリレーション（本論文の例では市テーブル）に対する検索条件として処理できる検索条件しか考慮していない：この場合，ユーザ XQGM グラフの形式が単純になり，変換処理も容易となる．たとえば図 13 の例では，トップレベルの市テーブルに対する「人口 >= 10」という条件を，子要素（選択されたそれぞれの市に対応する位置情報や施設）の選択のために，親から子へ一方向に伝播させればよい．しかし，子要素側にも条件が存在する場合，子の選択条件により得られた絞り込み結果（例：施設名 = "A モール" という条件により得られた市 ID）を親の要素の選択処理に伝播させる必要がある．しかし，論文 8) ではこのような処理は考慮されていない．
- 図 13 のように，where 節に条件があるだけのトップレベル要素の単純な選択処理しか考えられておらず，図 3 のような return 節に出力指定を含む場合は考慮されていない：この場合，図 12 のように where 節と return 節の処理を分けて考える必要はなく，処理は簡単であるが，return 節に出力指定を含む問合せを扱うことができず一般性に欠ける．
- where 節の条件指定も単一の条件のみを含む単純なものであり，図 3 のように複数の条件を含む場合については考慮されていない．

また，本研究に固有の問題点として外部関数の問題がある．XPERANTO の基本アプローチでは，where 節の選択演算はすべて 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についてはまったく考慮されていない．そこで本研究では，上記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，XPERANTO のアプローチに対する拡張を行っている．図 12 に示したユーザ問合せ XQGM はこの拡張を反映したものである．XPERANTO との相違点は，return 節の処理内容を反映した部分グラフの生成と，where 節における複数の条件指定を統合するための intersect 演算の導入（図 12 のノード 11）である．4.2 節でこの拡張に基づく問合せ変換について述べる．

次に，本研究で想定する 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，および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とその導入理由について述べる．

XML ビューに関する前提条件 XML ビューについては以下のような前提条件を設ける．

- XML ビュー中の各文書要素の構築に必要なタプル集合を，キー値に関する条件を用いて特定できる：たとえば図 2 の XML ビューの例では，各「市」文書要素の構築に必要なデータは，位置情報テーブルおよび施設テーブルの市 ID 属性（市テーブルの外部キーを保持する）を用いて特定可能である．この条件は，本研究のみならず，XML データを複数のリレーション中のタプルから組み立てる場合に必要となる前提条件である．

XQuery 問合せに関する前提条件 XQuery は非常に表現能力の高い問合せ言語であるため，ビュー定義 XQuery 問合せおよびユーザ XQuery 問合せとして複雑な問合せが記述可能である．そこで，本研究では実際によく用いられる以下のような問合せ形式に制限して問合せ処理を考える．

- 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．
- where 節に複数の選択条件がある場合は，論理積による結合のみを許す．
- ユーザ問合せの return 節では副問合せを指定しない．

後述の問合せ変換アルゴリズムはこれらの条件を前提としている．

外部関数に関する前提条件 外部関数に関しても，簡略化のために以下のような制約をおく．

- where 節に述語としてただ 1 つ指定される：where

節に複数の外部関数が現れる場合，外部関数の実行順序やXMLフラグメントの生成方式について，多数の選択肢が生じうる．また，真偽値を返す述語ではなく，一般の関数も許した場合，関数の評価結果の扱いが問題となる．そこでこのような制約を課して処理の簡略化を図る．

- 引数にとる各文書要素は，ある1つのリレーションのタプル集合からのみ構成される：外部関数が引数としてとるXMLフラグメントが複数のリレーションのタプルから構成される場合には，XMLフラグメントの構成処理が複雑化することがこの条件の理由である．
- 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：これは，ミドルウェア上で外部関数の引数を生成する処理において，外部関数の適用対象データを前もって特定のクラスのオブジェクトに加工するなどの処理を行わないことを意図している．

外部関数に関する以上の前提は，本研究の本質的なねらいを失わずに問題を簡単化するためのものである．

XQueryは高度な問合せ記述が可能であることから上記の制約はかなり強いものであるが，これらは少なくとも論文8)のXPERANTOのアプローチに対する拡張となっている．これらの一般化については今後の検討課題としたい．

4.2 問合せ変換

以下では，3.3節で述べたXPERANTOのアプローチを拡張した問合せ変換方式の概略について述べ，その詳細については付録A.1で補足する．4.1節でも触れたが，XPERANTOの問合せ変換のアプローチに対する本研究での拡張は，おもに以下の3点である．

- `where`節における外部関数評価への対応．
- `return`節における出力指定への対応．具体的には，後述するように`where`節と`return`節のパスを独立して変換することにより対処する．
- `where`節の複数の選択条件とサブリレーションへの演算子プッシュダウンを考慮．

4.2.1 相関関係分離

まず，本提案手法における問合せ変換の第1ステップである相関関係分離（decorrelation）について述べる．図10と図12のように，問合せ変換の対象となるXQGMグラフは，副問合せと相関関係を有するcorrelated join演算を一般に含んでいる．このcorrelated joinをそのまま実行することは非常に効率が悪いことから，XQGMグラフを変形し相関関係を解消することが必要となる[9]．XPERANTOの基本アプローチでは，相関関係分離の処理はユーザ問合せXQGMとビュー定義XQGMのビュー合成の後でなされていた．しかし本提案手法では，変換処理の見通しを良くするため，相関関係の分離処理をビュー合成の前処理として行う．

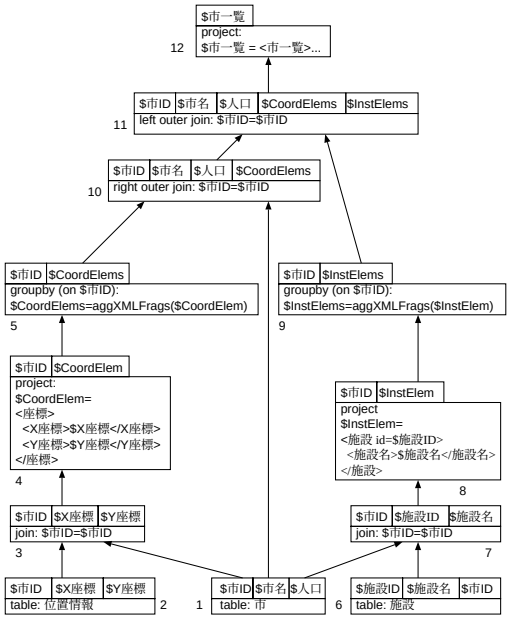

図14 ビュー定義XQGMの相関関係分離
Fig. 14 Decorrelated view definition XQGM.

相関関係分離の結果，図10のビュー定義XQGMは**図14**のように変形される．これは，図10中のcorrelated join演算を変換し，市テーブルを位置情報テーブル，施設テーブルとそれぞれ結合することで実現される．なお，図で示したように，元の問合せのセマンティクスを保持するためにはleft/right outer join演算の導入が必要となる[8]．この理由について例を用いて簡単に説明する．たとえば，データベース中に施設が1つも登録されていない市があったとする．そのとき，その市の市IDを持つタプルは当然施設テーブルには存在しない．そのため，図14のノード7の結合処理の結果にはその市の市IDは含まれず，よって施設を持たないその市の市IDはノード8，9には伝わらないことになる．ここでノード11がleft outer joinでなくただの結合処理であったとすると，施設を持たないその市の情報はノード12に伝わらなくなってしまう．しかし，もともとの図10のノード10のcorrelated joinは，「市に施設がもしあれば，その情報を結合する」という意味を表しているため，異なるセマンティクスとなってしまう．そこで，図14のノード11をleft outer joinとすることでこの問題点を解

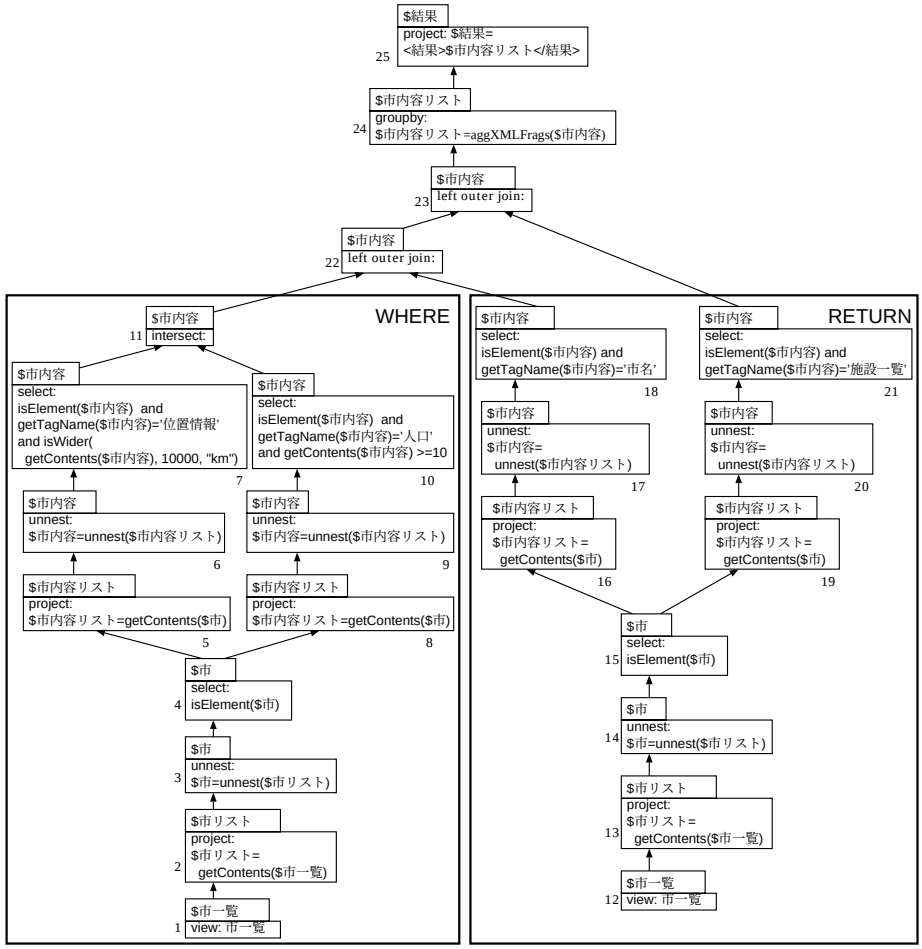

図 15 ユーザ問合せ XQGM の相関関係分離
Fig. 15 Decorrelated user query XQGM.

消することができる．ノード 10 を right outer join にする理由も同じである．

一方，図 12 のユーザ問合せ XQGM については，ノード 4 以下の `for` 節に関する相関関係を持つ部分グラフを `where` 節と `return` 節に対応する各部分グラフに接ぎ木する形で相関関係分離を行う．また，`where` 節の存在限量制約を考慮して，correlated join 演算を left outer join のシーケンスに変換する．その結果が **図 15** である．left outer join が必要となる理由についてはノード 23 を見ると分かりやすい．ノード 23 では，左側の入力は条件を満たした市に関する情報であり，右側の入力は施設に関する情報である．ここで，`where` 節の条件は満たすが施設を持たない市があったとする．この場合，ノード 23 が通常の結合処理であると，そのような市の情報は結合結果から削除されてしまい，本来のセマンティクスが維持できないことになる．そのため，left outer join 演算を用いて，図 12 のノード 11 からノード 18 への存在限量の有向辺のセマンティクスを反映する．同じ処理を適用することで，ノード 22 も left outer join となる．ただし，この場合については，結合対象の右側の入力が市名であり，`where` 節の条件を満たす市に対して対応する市名が存在しないことはありえないので，実際には通常の結合処理を用いても例外的に正しい結果が得られる．

4.2.2 ビュー合成

次にビュー合成を行う．図 15 のノード 1 と 12 を図 14 の内容で置き換えた後，表 3 の関数合成ルールに従って，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．これにより，**図 16** の XQGM が得られる．外部関数を含む `where` 節と `return` 節が独立したパスとしてビュー合成されていることが分かる．

4.2.3 演算子プッシュダウン

3.3 節で述べたように，ビュー合成の次のステップ

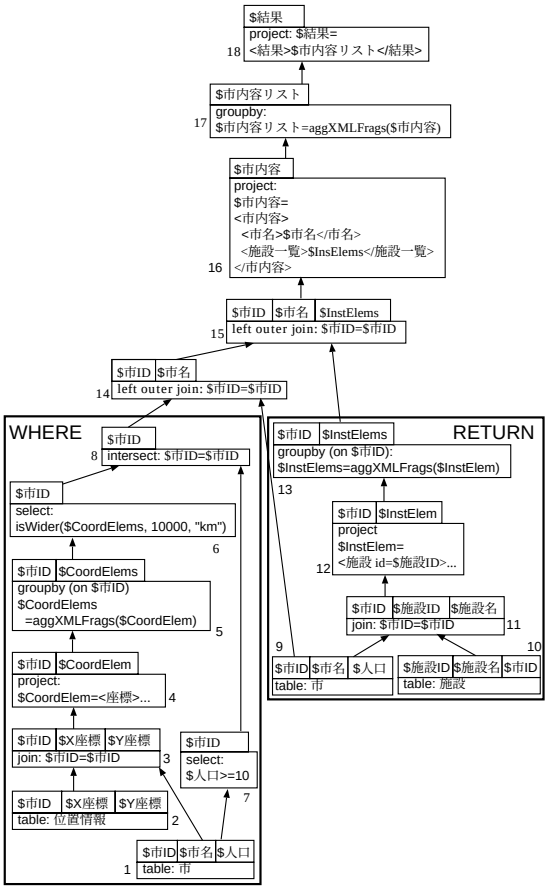

図 16 ビュー合成後 XQGM
Fig. 16 XQGM after view composition.

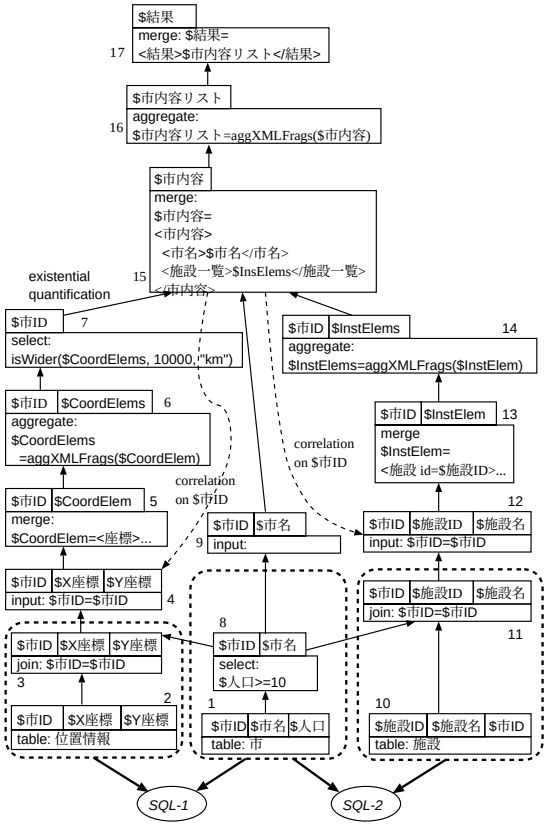

図 17 タグ付け演算子プルアップ後 XQGM
Fig. 17 XQGM after tagger pull-up.

として，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．ただし，本研究では問合せ処理時において外部関数がミドルウェア上で評価されることを想定しているため，XPERANTO のプッシュダウン処理に拡張が必要となる．外部関数の評価の際には，外部関数の適用対象である文書要素が XML フラグメントとしてミドルウェア上で実体化されることを想定しているため，外部関数呼び出しの演算は，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ことになる．たとえば図 16 では，ノード 6 中の外部関数 `isWider()` を評価するためにノード 5 で生成される位置情報の XML フラグメントが必要となるため，外部関数呼び出しをそれ以上リーフ側にプッシュダウンすることはない．

4.2.4 タグ付け演算子プルアップ

最後に，タグ付け演算子プルアップを行い**図 17** を得る．その概略は次のようになる．まず，図 16 においてノード 7 の intersect 演算のセマンティクスを考慮し，ノード 1，2，3，7，8 を図 17 のノード 1，2，3，8 に変形する．また，`where` 節におけるテーブル「市」への制約条件（ノード 7）を `return` 節に伝播させ，さらに先の `where` 節の変形と統合することで，ノード 9 を図 17 のノード 1，8 に変形する．さらに，project，groupby などの XML 関数を後述するタグ付け演算子に置き換える．そして，ノード 14，15 の left outer join のセマンティクスを考慮し，相関関係を表す辺を生成する．

図 17 中の演算子 merge，aggregate，input は XPERANTO ではタグ付け演算子と呼ばれており，ミドルウェアレベルでの実際の問合せ処理を実行する．input は RDBMS からタプルを獲得するための演算子である．merge は順位付けされた入力ストリームをマージする演算であり，XML 関数 cr8Elem，cr8Att，cr8XMLFragList，cr8AttList を実現している．これにより，図 17 のノード 15 においては，人口と面積に関する条件を満たす各市について，**図 18** に示すような XML フラグメントが生成される．aggregate は集約処理を行う演算子であり，XML 関数 aggXMLFrags

```
01: <市内容>
02:   <市名>つくば市</市名>
03:   <施設一覧>
04:     <施設 id="I0015">A モール</施設>
05:     <施設 id="I0016">H 講演</施設>
06:     <施設 id="I0017">M 研究所</施設>
07:   </施設一覧>
08: </市内容>
```

図 18 中間結果の XML フラグメントリスト
Fig. 18 XML fragment list of an intermediate result.

を実現し，ノード 16 では，市内容要素のリストを生成する．

なお，図 17 中で点線の枠で示した部分はリレーショナル代数演算子のみからなる部分グラフであり，これらが RDBMS にプッシュダウンされることになる．図 17 に示すように，本研究ではプッシュダウンする SQL 問合せとして，`where` 節に対応するパスから得られる *SQL-1* と `return` 節に対応するパスから得られる *SQL-2* の 2 つを考える．これらを順次 RDBMS に発行するか，一括して発行するかで大きく 2 つの問合せ処理が可能となる．これら *SQL-1*，*SQL-2* の問合せ処理方式については次章で詳しく述べる．

5. 問合せ処理方式

5.1 分割処理方式

分割処理方式では，まず *SQL-1* を RDBMS に発行して外部関数評価対象の XML フラグメントを生成し，外部関数評価を行う．次いで，その評価結果を反映した *SQL-2* により，結果 XML 文書生成に必要なタプルを RDBMS から取得する．このため，外部関数評価によるタプル絞り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 19** に示す．

問合せ解析部では前章で述べた方式によって問合せ変換を行い，*SQL-1* と *SQL-2* を生成する．また，XML タグ付け処理式として，図 17 のノード 4〜6 に対応する *Tagger-1* とノード 12〜17 に対応する *Tagger-2* を生成する．これらはそれぞれ，外部関数評価用の XML フラグメントの生成と問合せ結果 XML の生成のために用いられる．

SQL 問合せ制御部は，まず *SQL-1* を用いて RDB から XML フラグメント生成用のタプル集合 *Tuple-1* を取得しタグ付け演算部に渡す．タグ付け演算部では *Tagger-1* に基づいて *Tuple-1* に対するタグ付け処理を行い，外部関数評価用 XML フラグメント *Fragment* を生成する．外部関数評価部では *Fragment* を引数として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キー値（前述の例では市 ID に対応）の集合

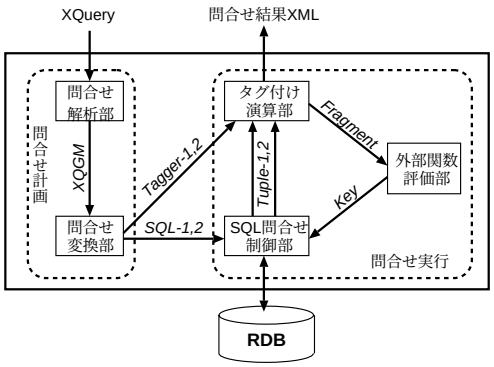

図 19 分割処理方式の流れ
Fig. 19 Flow of two-step processing method.

Key を SQL 問合せ制御部に渡す．SQL 問合せ制御部は，*Key* を反映した問合せ *SQL-2* を RDBMS に発行し，結果タプル集合 *Tuple-2* を取得する．最後に，タグ付け演算部で *Tagger-2* に基づいて *Tuple-2* にタグ付け処理を行い，問合せ結果 XML を生成する．

問合せ *SQL-2* の生成方式として，本研究では以下の 2 種類を考える．

- **分割処理方式 (where)**：*SQL-1* の評価で得られた外部関数を満たすキー値の集合 *Key* を，*SQL-2* の `where` 節に “`where 施設ID in Key`” のように制約条件として追加することで，外部関数の評価結果を利用する方式．
- **分割処理方式 (tmp)**：得られたキー集合 *Key* の各要素を含む 1 カラムの一時テーブルを作成し，この一時テーブルとの結合処理を *SQL-2* に含める方式．

分割処理方式にはこのほかにビットマップ索引を用いる手法なども考えられるが，一般の RDBMS で広く利用できる手法として，本研究ではこの 2 方式を検討の対象とする．

5.2 一括処理方式

一括処理方式では，*SQL-1* と *SQL-2* を統合して発行し，外部関数評価と結果 XML 生成に必要なタプル集合を一括して取得することで，RDBMS 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り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 20** に示す．

問合せ変換部の役割は分割処理方式と同じである．SQL 問合せ制御部では，*SQL-1* と *SQL-2* を合成した問合せ *SQL-3* を問合せ変換部から受け取り，RDB に一括問合せを行う．この合成手法は XPERANTO の論文 9) の手法に基づいており，RDB から得られるタプル集合 *Tuple* 中の各タプルが外部関数評価用の

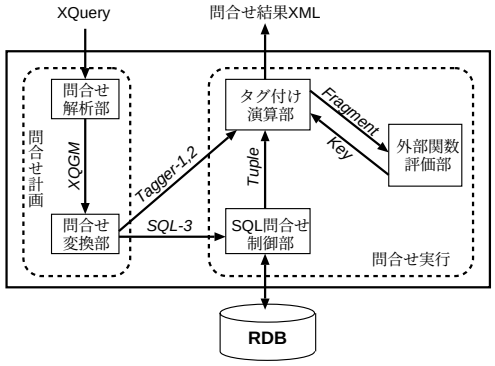

図 20 一括処理方式の流れ
Fig. 20 Flow of one-step processing method.

タプルであるか結果 XML 生成用のタプルであるかを区別するための識別子を付与するよう拡張している点のみが異なる．タグ付け演算部は，タプル集合 *Tuple* から外部関数評価に必要な XML フラグメント *Fragment* を生成し，これが外部関数評価部で評価される．最後に，外部関数の評価結果として得られたキー集合 *Key* を用いて結果 XML 生成用タプルの選択をミドルウェア上で行い，条件を満たすタプルから問合せ結果 XML を生成する．

この章では本研究で提案する問合せ処理方式について説明した．実際に RDBMS に発行される SQL 問合せの具体例は次章で示すことにする．

6. 評価実験

3 種類の処理方式について評価実験を行い，問合せ処理効率を比較した．本章ではその概要と実験結果について述べる．

6.1 実験の概要

実験は，Linux OS が稼動する PC（Celeron 300 MHz，メモリ 196 MB）上で PostgreSQL 7.1 を用いて行った．図 1 に示した「市」,「位置情報」,「施設」の 3 つのテーブルを，以下のタプル数の設定で構築した．

- 市テーブルのタプル数：$N = 1{,}000$
- 位置情報テーブルのタプル数：$10N$ および $100N$
- 施設テーブルのタプル数：$10N$ および $100N$

なお，市，位置情報，施設の各テーブルの市 ID 属性上には索引を構築している．

実験では，以下の 4 つの問合せについて，各種パラメータを設定して問合せを実行した．

- **Q1**：面積 X 以上の市について，市名と市内の施設情報を検索．
- **Q2**：面積 X 以上の市について，市名と市の位置情報，および市内の施設情報を検索．
- **Q3**：面積 X 以上かつ人口 Y 以上の市について，市名と市内の施設情報を検索（$X = 10{,}000$, $Y = 10$ とおいたものが図 3 の問合せ例に対応する）．
- **Q4**：面積 X 以上かつ人口 Y 以上の市について，市名と市の位置情報，および市内の施設情報を検索．

「面積 X 以上の市について」という条件が `isWider()` による外部関数呼び出しに対応しており，「人口 Y 以上」は通常の選択条件に対応する．それぞれの選択率は

- 面積による市の選択率：$S_a = 0.1, 0.3, 0.5, 0.7, 0.9$
- 人口による市の選択率：$S_p = 0.1, 0.3$

の値をとるものとした．

性能を測る指標に関しては次のように考えた．分割処理方式の全体の問合せ処理コストは，主として以下の各処理コストの合計からなる．

(1) SQL 問合せ *SQL-1* の処理
(2) 外部関数評価用 XML フラグメント生成
(3) 外部関数の評価
(4) SQL 問合せ *SQL-2* の処理
(5) 問合せ結果 XML 生成

一方，一括処理方式の問合せ処理コストについては次のようになる．

(1) *SQL-1*，*SQL-2* を合成した SQL 問合せ *SQL-3* の処理
(2) 外部関数評価用 XML フラグメント生成
(3) 外部関数の評価
(4) 問合せ結果 XML 生成

これらの処理の中で，外部関数評価用 XML フラグメント生成と問合せ結果 XML 生成は比較的コストが小さい処理であり，両方式でほぼ同じ処理が実行される☆．また，外部関数の評価についても両方式でコストはほぼ等しいと考えられる．よって，本実験ではこれらを除いたコストについて考え，分割処理方式については *SQL-1* の処理時間と *SQL-2* の処理時間の和を，一括処理方式については *SQL-3* の処理時間を評価指標とする．

6.2 SQL 問合せの具体例

問合せ Q3 の場合を例として，実験に用いた SQL 問合せを示す．SQL 問合せの生成方式については，付

☆ 実際には一括処理方式の方がコストが高い処理であるが，両手法のコストの差異は，本研究における実験の範囲では他の高コストの処理に比べると無視できる程度である．詳しくは 6.4 節で分析結果を示す．

```
01: create view citytable as
02:   select c.市ID, c.市名
03:   from 市 c
04:   where c.人口 >= 10;
05: create view coordtable as
06:   select c.市ID, p.X座標, p.Y座標
07:   from citytable c, 位置情報 p
08:   where c.市ID = p.市ID;
09: create view insttable as
10:   select c.市ID, i.施設ID, i.施設名
11:   from citytable c, 施設 i
12:   where c.市ID = i.市ID;
13: create view outerUnion1(target, type, 市ID,
14:                         X座標, Y座標) as
15:   select INTEGER'0' as target, INTEGER'0' as type,
16:          市ID, X座標, Y座標
17:   from coordtable;
18: select *
19: from outerUnion1
20: order by 市ID, target, type;
```

図 21 分割処理方式 (where) に対する *SQL-1*
Fig. 21 *SQL-1* for two-step (where) processing method.

録 A.2 にその概略を示す．

6.2.1 分割処理方式 (where)

図 21 に，外部関数評価に必要となるタプルを検索するための *SQL-1* 問合せを示す．図 17 に示したタグ付け演算子プルアップ後 XQGM からの機械的な変換を想定しているため，記述は若干冗長となっている．まず 1〜12 行目では，問合せ対象となる RDB 上のタプルを特定する一時的なビューを定義している．これらは図 17 のノード 1，2，3，8，10，11 に対応する．13〜17 行目は実際に検索すべきタプルを設定するためのビュー定義であり，図 17 のノード 4 に対応する．outer union 演算や `target` および `type` 属性の役割については，以下の *SQL-2* の説明において述べる．18〜20 行目が最終的に RDBMS に発行される問合せである．20 行目の `order by` の役割についても後述する．

5.1 節で述べたように，*SQL-1* を RDBMS に発行して得られるタプル集合をもとに外部関数評価が行われ，その結果がキー値集合 *Key* に得られるものとする．**図 22** に示す *SQL-2* はこの *Key* をもとに作成される．1〜9 行目では，結果 XML 生成に必要となる市テーブルと施設テーブルの情報を outer union 演算を用いて設定しており，図 17 のノード 12 に対応する．5 行目の `citytable`，9 行目の `insttable` は *SQL-1* の処理時に定義したビューの名前である．図 21 と異なり `target` 属性の値が 1 に設定されているが，これはこの outer union のタプルが結果 XML 生成用であることを示すためのフラグである（実際にはこのフラグは一括処理方式でのみ利用される）．`type` 属性の値は，outer union の各タプルが市テーブルからのものか，施設テーブルからのものかを判定するためのフラグとして用いられる．10〜13 行目が実際に実行される SQL 問合せである．12 行目の `in` 演算子の右辺には *Key* の要素が展開され，外部関数評価の結果による検索タプルの絞り込みが行われる．13 行目の `order by` は outer union タプルをソーティングするためのもので，結果 XML 生成におけるタグ付け処理の効率化のために有効な手法である[9]．

```
01: create view outerUnion2(target, type, 市ID, 市名,
02:                         施設ID, 施設名) as
03:   select INTEGER'1' as target, INTEGER'0' as type,
04:          市ID, 市名, null, null
05:   from citytable
06:     union all
07:   select INTEGER'1' as target, INTEGER'1' as type,
08:          市ID, null, 施設ID, 施設名
09:   from insttable;
10: select *
11: from outerUnion2 o
12: where o.市ID in ('キー値1', 'キー値2', ...)
13: order by 市ID, target, type;
```

図 22 分割処理方式 (where) における *SQL-2*
Fig. 22 *SQL-2* for two-step (where) processing method.

上に示した *SQL-1*，*SQL-2* ではビュー定義をできるだけ用いて一時テーブルの作成を省くアプローチをとったが，一時テーブルを利用することで *SQL-1* の処理の中間結果を再利用できる可能性がある．これは以下で述べる分割処理方式 (tmp) にも共通する．具体的には，図 21 の 5 行目の “`create view coordtable`” を “`create temp table coordtable`” に変更することが考えられる．9 行目の `create view insttable` についても同様である．ただし，いくつかの実験の結果，ビュー定義を一時テーブルに置き換えるアプローチで必ずしも性能が向上するとは限らないという結果が得られたため，以下の性能評価ではビュー定義に基づくアプローチについてのみ触れる．

6.2.2 分割処理方式 (tmp)

分割処理方式 (tmp) では，*SQL-1* は分割処理方式 (where) の場合と同一であり，外部関数評価結果のキー集合 *Key* の扱いにのみ違いがある．**図 23** に *SQL-2* 問合せを示す．1〜5 行は *SQL-2* の準備段階であり，一時テーブルを作成して *Key* の要素を挿入し，索引付けを行う処理を示している．17〜19 行の問合せでこの一時テーブルとの結合処理を行うことで，検索タプルの絞り込みを実現する．

6.2.3 一括処理方式

図 24 に示すように，一括処理方式では *SQL-1*，*SQL-2* を合成した *SQL-3* により RDBMS に問合せを一括して発行する．結果として得られたタプル集合から `target` 属性の値が 0 であるものを選択し，外部関数評価のための XML フラグメントを生成して外部

```
01: create tmp table keytable(市ID text);
02: insert into keytable('キー値1');
03: insert into keytable('キー値2');
04: ...
05: create unique index keytableindex on keytable(市ID);
06: ------------------------------------------------------
07: create view outerUnion2(target, type, 市ID, 市名,
08:                         施設ID, 施設名) as
09:   select INTEGER'1' as target, INTEGER'0' as type,
10:          市ID, 市名, null, null
11:   from citytable
12:     union all
13:   select INTEGER'1' as target, INTEGER'1' as type,
14:          市ID, null, 施設ID, 施設名
15:   from insttable;
16: select *
17: from outerUnion2 o, keytable k
18: where o.市ID = k.市ID
19: order by 市ID, target, type;
```

図 23 分割処理方式 (tmp) における *SQL-2*
Fig. 23 *SQL-2* for two-step (tmp) processing method.

```
01: create view citytable as
02:   select c.市ID, c.市名
03:   from 市 c
04:   where c.人口 >= 10;
05: create view coordtable as
06:   select c.市ID, p.X座標, p.Y座標
07:   from citytable c, 位置情報 p
08:   where c.市ID = p.市ID;
09: create view insttable as
10:   select c.市ID, i.施設ID, i.施設名
11:   from citytable c, 施設 i
12:   where c.市ID = i.市ID;
13: create view outerUnion(target, type, 市ID, 市名,
14:                        X座標, Y座標, 施設ID, 施設名) as
15:   select INTEGER'0' as target, INTEGER'0' as type,
16:          市ID, null, X座標, Y座標, null, null
17:   from coordtable
18:     union all
19:   select INTEGER'1' as target, INTEGER'0' as type,
20:          市ID, 市名, null, null, null, null
21:   from citytable
22:     union all
23:   select INTEGER'1' as target, INTEGER'1' as type,
24:          市ID, null, null, null, 施設ID, 施設名
25:   from insttable;
26: select *
27: from outerUnion
28: order by 市ID, target, type;
```

図 24 一括処理方式における *SQL-3*
Fig. 24 *SQL-3* for one-step processing method.

関数評価を行うものとする．外部関数の評価結果が真になったキー値の集合を用いて，先に得られた *SQL-3* の結果で `target` 属性が 1 であるものをフィルタリングすることで，結果 XML 生成に必要なタプル集合が得られることになる．

6.3 実験結果

前述のように，実験では市テーブルのタプル数を $N = 1{,}000$ で固定して実験を行った．位置情報テーブルと施設テーブルのタプル数は，それぞれ $10N$，$100N$ のいずれかの値をとる．以下では，たとえば施設テーブルのタプル数が $10N$ であり，位置情報テーブルのタプル数が $100N$ である場合を「$10N$，$100N$ の場合」などと簡略化して呼ぶ．

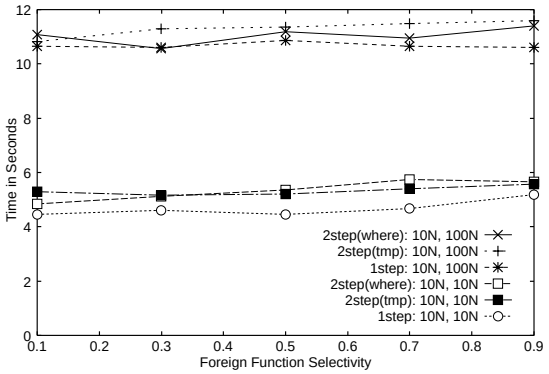

図 25 問合せ処理時間（Q3($S_p = 0.3$): $10N$, $10N/100N$）
Fig. 25 Elapsed time (Q3($S_p = 0.3$): $10N$, $10N/100N$).

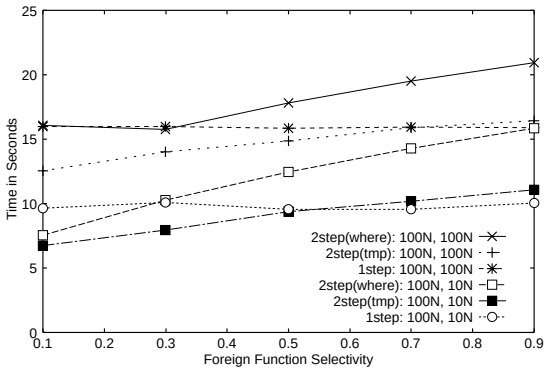

図 26 問合せ処理時間（Q3($S_p = 0.3$): $100N$, $10N/100N$）
Fig. 26 Elapsed time (Q3($S_p = 0.3$): $100N$, $10N/100N$).

6.3.1 Q3（$S_p = 0.3$）の場合

まず，問合せ Q3 において人口に関する条件の選択率が $S_p = 0.3$ のとき，施設テーブルのタプル数を $10N$ とし，位置情報テーブルのタプル数を $10N$ および $100N$ とした場合の実験結果を**図 25** に示す．横軸は外部関数の選択率を表し，縦軸は問合せに要した時間を表している．3 種類の問合せ処理方式とも外部関数の選択率に関係なくほぼ一定で同程度の処理コストとなっているが，$10N$，$10N$ の場合にはわずかながら一括処理方式が良い結果を示している．

次に，施設テーブルのタプル数を $100N$ に増やした場合の実験結果を**図 26** に示す．施設情報は Q3 の問合せ結果に含まれなければならないため，施設テーブルのタプル数の増加は Q3 の結果のサイズが増加することを意味している．図を見て分かるとおり，外部関数の選択率が小さい場合には絞り込みが効くことから分割処理方式が有利であることが分かる．特に分割処理方式 (tmp) は広い範囲において一括処理方式に

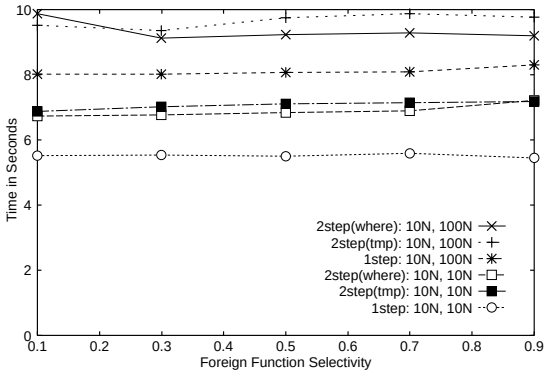

図 27 問合せ処理時間（Q3($S_p = 0.1$): $10N$, $10N/100N$）
Fig. 27 Elapsed time (Q3($S_p = 0.1$): $10N$, $10N/100N$).

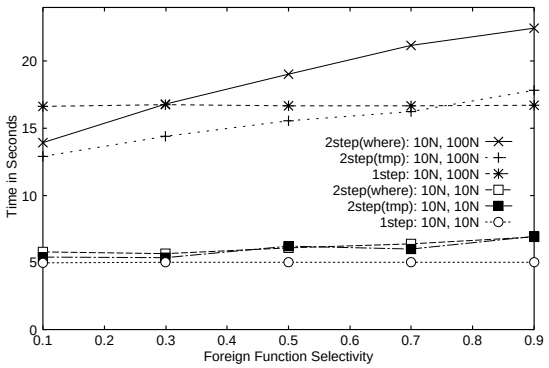

図 28 問合せ処理時間（Q4($S_p = 0.3$): $10N$, $10N/100N$）
Fig. 28 Elapsed time (Q4($S_p = 0.3$): $10N$, $10N/100N$).

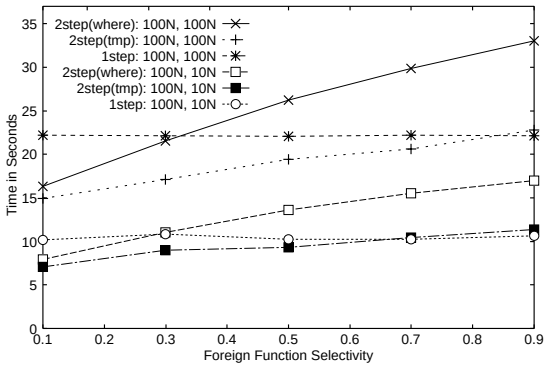

図 29 問合せ処理時間（Q4($S_p = 0.3$): $100N$, $10N/100N$）
Fig. 29 Elapsed time (Q4($S_p = 0.3$): $100N$, $10N/100N$).

比べ優れている．また，$100N$，$10N$ の場合，$100N$，$100N$ の場合ともに分割処理方式 (where) は分割処理方式 (tmp) に劣っている．これは，外部関数評価結果のキー値集合を *SQL-2* の条件節に展開するという分割処理方式 (where) のアプローチが，大量のキー値集合が得られた場合に非効率であるためと考えられる．

6.3.2 Q3（$S_p = 0.1$）の場合

次に，人口に関する選択率が $S_p = 0.1$ と小さい場合について実験結果を示す．**図 27** に $10N$，$10N$ および $10N$，$100N$ の場合を示す．この場合には RDBMS にプッシュダウンされる人口に関する選択条件が有効に機能し，取得データの量が削減されることから，一括処理方式が効率的である．外部関数の選択率によらず，分割処理方式の 80〜85%の実行時間となっている．

施設テーブルのタプル数を増やして $100N$，$10N$ および $100N$，$100N$ とした場合も同様の傾向が見られた．図 26 の場合とは異なり，この設定では人口に関する条件で大幅な絞り込みが可能であるため，3 つの問合せ処理方式ともほぼ同程度の処理時間であるが，一括処理方式の性能が若干良いという結果になっている．グラフは省略する．

6.3.3 Q4（$S_p = 0.3$）の場合

次に問合せ Q4 について，人口に関する条件の選択率が $S_p = 0.3$ の場合の実験結果を示す．Q3 と Q4 の違いは，Q3 が市名と施設情報のみの出力を求めていたのに対し，Q4 ではそれに加えて市の位置情報の出力も求めている点である．**図 28** は $10N$，$10N$ および $10N$，$100N$ の場合の結果である．$10N$，$100N$ の場合は出力されるべき位置情報のサイズが増加することより，外部関数の選択率が小さい範囲では分割処理方式が有利となっている．特に分割処理方式 (tmp) は外部関数の選択率のほとんどの範囲において，他の 2 者と同等もしくは優れているという実験結果となっている．

図 29 に $100N$，$10N$ および $100N$，$100N$ の場合を示す．特に $100N$，$100N$ の場合には分割処理方式 (tmp) の優位性が高いことが分かる．

6.3.4 Q4（$S_p = 0.1$）の場合

人口に関する検索条件の選択率を小さくした場合には，Q3（$S_p = 0.1$）の場合と同様，**図 30** に示すように一括処理方式が優位となる．これは $10N$，$10N$ および $10N$，$100N$ の場合であるが，$100N$，$10N$ および $100N$，$100N$ の場合にも同様の傾向が見られる．ただし，3 種類の手法の差異は縮まる傾向にある．

6.3.5 Q1，Q2 の場合

Q1，Q2 については概要だけを触れておく．3 つの問合せ手法のうち，分割処理方式 (where) は他の 2 つに比べ処理時間が大きく，その差は外部関数の選択率が大きくなるにつれて増える傾向にあった．前述のように，これは *SQL-2* の条件節に外部関数評価結果のキー値集合を展開するアプローチが非効率的であることに起因すると考えられる．分割処理方式 (tmp) と一括処理方式に関しては，外部関数の選択率が小さい場

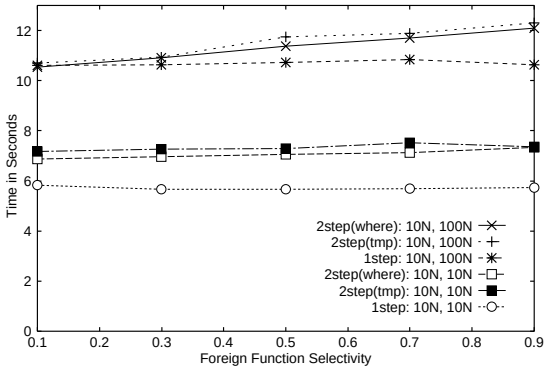

図 30 問合せ処理時間（Q4($S_p = 0.1$): $10N$, $10N/100N$）
Fig. 30 Elapsed time (Q4($S_p = 0.1$): $10N$, $10N/100N$).

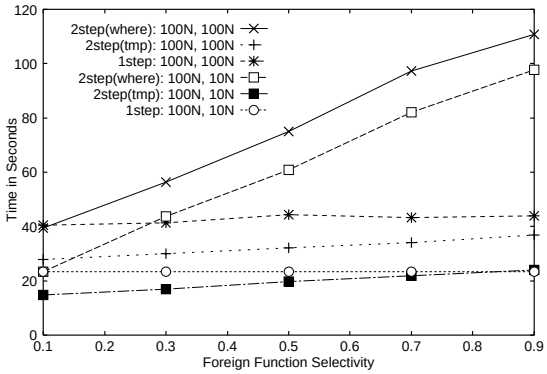

図 31 問合せ処理時間（Q1: $100N$, $10N/100N$）
Fig. 31 Elapsed time (Q1: $100N$, $10N/100N$).

合には分割処理方式 (tmp) が良いという傾向が一般に見られた．施設テーブル，位置情報テーブルのタプル数が少ない場合には優劣が逆転することもあるが，その差はわずかであった．**図 31** がその一例であり，他のパラメータ設定でも同様のグラフが得られた．

6.4 問合せ処理コスト全体の分析

6.1 節で述べた問合せ方式の比較実験の方針では，全体の処理コストに対する影響が小さいと考えられる XML 生成コストや，3 種類の手法で処理内容が同一である外部関数評価コストは比較の対象項目から外していた．しかし，この方針が妥当であるかどうかについては検証の必要がある．以下ではそれぞれのコストの分析結果について述べる．

6.4.1 XML の生成コストについて

外部関数評価用の XML フラグメント生成時間および問合せ結果 XML 生成時間については，処理コストが比較的小さく，また，3 種類の問合せ処理方式のいずれにおいてもほぼ同じ処理が実行されると考え，比較対象の項目からは外していた．この仮定が実際に成り立つかどうかが明らかでないことから，簡単な検証を行った．

例として，前節で述べた問合せ Q3 を分割処理方式で処理する場合について，実際に XML フラグメントを生成する関数を C 言語で構築し，その処理時間を測定した．XML フラグメントは，主記憶上にすでに存在するデータを用いて，主記憶上に生成されるものとした．まず，外部関数評価用の XML フラグメント生成時間に関しては，位置情報テーブルのタプル数（$10N$ および $100N$）や人口による市の選択率（$S_p = 0.1$ および 0.3）によらず，0.1 秒程度の処理時間であった．一方，結果 XML フラグメントの生成時間については，条件によって多少変化するが，0.3 秒程度の処理時間であった．この実験では，分割処理方式の場合を調べたが，一括処理方式もほぼ同じ処理内容であるため，所要時間はほぼ等しいと考えられる．以上の結果より，XML フラグメント生成と結果 XML の生成時間の和は，問合せ Q3 については 0.5 秒以内であり，前節で述べた分析結果にはほとんど影響を与えないといえる．

Q3 以外の問合せについても同様のことがいえる．Q4 は Q3 とほぼ同じ処理であるため，処理コストの傾向も同様であると考えられる．Q1，Q2 については，人口による市テーブルの選択がないことから，外部関数評価用 XML フラグメント生成と結果 XML 生成時間が Q3，Q4 に比べ増加するが，Q1，Q2 の場合には SQL 問合せの処理時間自体も大きいことから，全体の処理時間に与える影響は同様に小さいと考えられる．

6.4.2 外部関数の評価コストについて

6.1 節では，外部関数評価はどの問合せ処理方式でも同じ処理時間を要することから，各手法の性能の違いを図る指標の中にはこれを含めなかった．しかし，外部関数評価に要するコストがほかに比べ非常に大きく，全体の問合せ処理コストの大半を占めてしまうような状況が生じると，3 種類の問合せ処理手法の違いは実質的にはなくなってしまうことになる．そこで，前節の実験の設定において，外部関数の評価コストがどの程度の値となるのかを検証するための実験を行った．

まず，外部関数適用対象の XML フラグメントのリストを作成した．これは，それぞれの市について市 ID と市の領域を表す点のシーケンスを XML で表現し，それらをあわせたリストを XML データとして表現したものである．外部関数評価プログラムは Java 言語で作成されており，まず，ファイルから XML フラグメントのリストを読み込み，XML の DOM 表現に変換する．次いで，それぞれの市の位置情報を表す DOM の部分木に対し，多角形の面積を計算する外部関数を適用する．外部関数の適用結果として面積を表す数値

が得られるが，これを対応する市 ID とペアにして出力する（ただし，実際の処理時間の測定では出力時間は含めていない）．

Q3，Q4 の場合についての実験結果について簡単にまとめる．たとえば，Q3 で位置情報のタプル数が $10N$（つまり多角形が 10 個の点から構成される）で人口に関する選択率が 0.3 である場合（図 25 および図 26 の $10N$，$10N$ および $100N$，$10N$ の場合に相当），その処理時間は 1.2 秒であった．一方，同じ設定で位置情報のタプル数だけ $100N$ にした場合（図 25 および図 26 の $10N$，$100N$ および $100N$，$100N$ の場合に相当）は 1.8 秒の処理時間であった．よって，外部関数の処理時間は無視できないものの，前節での実験結果に大幅な変化をもたらすものではないことが分かった．

また，Q1，Q2 については，人口による条件での絞り込みがないため，外部関数の処理コストが大きくなる．それが最大となるのは位置情報のタプル数が $100N$ の場合である．この場合には，1,000 個の市のそれぞれについて，100 個の点からなる多角形の面積を計算することになり，実験してみると約 3 秒の実行時間を要した．しかし，図 31 の $100N$，$100N$ の場合を例にとると，Q1 における SQL 問合せの処理時間は最も小さい場合でも 20 秒程度を要するため，問合せ処理コストの傾向を大幅に変えることはないことが分かった．一方，位置情報のタプル数が $10N$ の場合には 1.5 秒ほどの実行時間となり，この場合にも同様の議論が成り立つ．

他のパラメータ設定についても実験を行ったが，同様の結果を得た．なお，注意が必要な点として，Java プログラムの実行においては，実際には何も処理を行わない場合でも 1 秒程度の時間を要したという事実がある．これは，Java 仮想マシンの起動やクラスのロードに要する時間と考えられる．本実験では，評価対象の市とその位置情報のリストを 1 つの XML ファイルにまとめ，これを 1 回の Java プログラムの起動により処理したが，各市ごとに XML フラグメントを作成してそれらを評価する Java プログラムを毎回起動するアプローチをとってしまうと，処理時間が大幅に増大し，全体の処理コストが悪化すると考えられる．実際には，上述のように Java プログラムの 1 回の実行で外部関数評価を処理できるため，このアプローチをとる必要はない．

最後に，今回の実験では面積を計算する外部関数を実装し利用したが，外部関数によってはその処理時間自体が大きいものがあり，最悪の場合，実質的な処理時間の大半を外部関数処理が占める状況が生じうる．このような場合には，問合せ処理方式の実質的な差異は小さくなることになる．

6.5 実験結果のまとめと考察

6.3 節の実験結果は以下のようにまとめることができる．

- 分割処理方式 (where) はほとんどすべての状況において分割処理方式 (tmp) に劣っている：これは，SQL の `where` 節に `in` 演算子で大量のキー値集合を与えるアプローチに対し PostgreSQL が対応しておらず，非効率的な処理になっているためと考えられる．
- 一括処理方式の処理時間は外部関数の選択率には依存しない：処理方式を考えればこれは自明であるが，一括処理方式では RDB から必要なタプルを抽出した後は残りの処理をミドルウェアで行うため，問合せ *SQL-3* の処理時間には外部関数の選択率は影響を与えない．
- Q1，Q2 のような外部関数の条件のみを含む問合せに関しては，一般に分割処理方式 (tmp) が優れている：これは特に外部関数の選択率が小さい場合に顕著である．外部関数の選択率が 1 に近くなると，一括処理方式とほぼ同等の処理コストとなる．
- Q3，Q4 のように外部関数以外の他の検索条件が存在する場合には，それぞれの選択率の大小に応じて得失がある：外部関数の選択率が小さく他の検索条件の選択率が大きい場合には分割処理方式 (tmp) が優れ，逆の場合には一括処理方式が優れているという傾向がある．
- RDBMS における処理時間が小さい場合には一括処理が有利になる傾向がある：このような状況は，もともとのタプル数が少ない場合や，Q3，Q4 において人口による選択で問合せサイズが小さくなる場合に見られる．全体の処理コストが小さくなるため，分割処理において必要となる 2 回の RDBMS とのやりとりのオーバヘッドなどが全体の処理時間に影響するのがその理由ではないかと考えられる．

以上をまとめると，Q1，Q2 のような問合せに関しては分割処理方式 (tmp) を用いることが有効であると考えられる．Q3，Q4 に関しては，外部関数の選択率や他の検索条件の選択率を考慮して，分割処理方式 (tmp) と一括処理方式のいずれかを選択することが必要である．ただし，処理時間が小さい場合には一括処理が有利になる傾向にあることも考慮する必要がある．

7. まとめと今後の課題

本論文では，RDB上に構築されたXMLビュー機能を支援するミドルウェアにおいて，ドメイン依存外部関数を含んだXML問合せが与えられた際の問合せ処理方式を提案した．XMLビュー上で定義される外部関数の評価はミドルウェア上に対象のXMLフラグメントを実体化して行われることから，RDBMSによるXMLビュー機能支援に関する既存のアプローチを直接的に適用することは困難である．そのため本研究では，XPERANTOグループにより提案されたXMLビュー支援のアプローチを拡張することで外部関数を含んだXML問合せの効率的な問合せ処理の実現を図った．

外部関数を含んだXML問合せの処理方式として，ミドルウェアからRDBMSに2回のSQLを発行する分割処理方式と，1回のSQLでまとめて情報を取得する一括処理方式を提案し，さらに前者に対しては，外部関数の評価結果を2回目に発行するSQLの`where`節に反映する分割処理方式(where)と，一時テーブルの利用で対処する分割処理方式(tmp)を提案した．実際のRDBMSを用いた実験によりこれらの手法の性質を調べ，有用性について比較検討を行った．

今後の課題としては以下があげられる．

- 支援可能なXML問合せの一般化：4.1節で示したユーザXML問合せに対する制約を緩和し，より広い問合せのクラスを支援することが求められることから，4.2節で述べた問合せ変換手法をより汎用性の高い手法に洗練する必要がある．緩和すべき制約の例としては，たとえば，以下のような項目があげられる．
  (1) 4.1節で述べたように，ユーザ問合せ，ビュー定義問合せとして想定するXQuery問合せについては，`for`節，`where`節，`return`節に制限を加えていた．このうち，`where`節において複数の検索条件が現れる場合にそれらが論理積で結ばれるという想定は，実際の使用における大きな制約となりうる．よって，論理和による検索条件の結合などについても検討したいと考えている．
  (2) 同じく4.1節で述べたように，外部関数に関しても本研究ではいくつかの想定を行った．このうち，`where`節に外部関数がただ1つだけ現れるという制約については，その制限を緩めることが重要な課題となる．`where`節に含まれる外部関数の数が一般にk個存在した場合にも，本論文で提案した一括処理方式と分割処理方式を適用することは可能である．前者の場合は，SQL問合せを1回だけRDBMSに発行し，残りの処理はすべてミドルウェアで行うことになる．また，後者の場合は外部関数評価用XMLフラグメント生成のためのSQL問合せをk回，最終的な結果XML生成のためのSQL問合せを1回の計$k+1$回のSQL問合せの発行となる．しかし，実際にとりうる問合せ処理方式はこれだけにとどまらず，一部の外部関数群の評価に必要なデータを一括して入手するためのSQL問合せを発行し，残りの処理は個別に行うなどのハイブリッド手法も考えられる．また，複数個の外部関数が存在する場合は，外部関数の選択率や評価コストなどを考慮した，外部関数の評価順の最適化なども必要であると考えられる．
- 問合せの最適化：図21から図24の図に示したような変換結果のSQLは，求める問合せ結果を得るための機械的な変換処理を重視したものであり，明らかに冗長な表現が含まれている．このような冗長性は，問合せを受け取ったRDBMS側の最適化機能により一部は解消されると考えられるが，ミドルウェア側での問合せの簡略化なども効率化のために有効であると考えられる．RDBMS上のミドルウェアにおけるXML問合せの最適化に関しては現在さかんに研究が進められていることから[5),8)]，今後の研究の成果を柔軟に取り込んでいくことも必要と考えられる．
- 他の問合せ処理手法の開発：本論文では，実験のプラットフォームであるPostgreSQLで利用でき，他のRDBMSでも一般に提供されている機能のみを利用して，3種類の問合せ処理手法を提案した．しかし，ビットマップ索引の利用によるキー値に基づくフィルタリングの効率化や，ビュー定義の代わりにSQL:1999[7)]で導入された`with`文を利用することなど，商用RDBMSで広く導入されつつある新機能を利用することも重要と考えられる．
- 問合せ処理方式の選択方式の開発：実験において述べたように，分割処理方式(tmp)と一括処理方式の効率の優劣は，与えられた問合せや選択率などのさまざまな要因に依存している．状況に応じて最適な問合せ処理方式を選択する手法の開発が，問合せ処理コストの削減のために必要であると考えられる．

その他，より規模の大きいデータベースを対象とした実験や，今回実験を行ったPostgreSQL以外の

RDBMSにおける実験などが今後の課題としてあげられる．

謝辞 査読者の方々からは貴重なコメントをいただきました．ここに感謝いたします．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），若手研究（B）（14780316），および文部科学省科学研究費特定領域研究（14019009）による．

参 考 文 献

1) Aberer, K. (Ed.): Special Section on Advanced XML Data Processing, *ACM SIGMOD Record*, Vol.30, No.3 (2001).
2) Boag, S., Chamberlin, D., Fernandez, M.F., Florescu, D., Robie, J., Simèon, J. and Stefanescu, M.: XQuery 1.0: An XML Query Language W3C Working Draft (2001). http://www.w3.org/TR/xquery/
3) Carey, M., Florescu, D., Ives, Z., Lu, Y., Shanmugasundaram, J., Shekita, E. and Subramanian, S.: XPERANTO: Publishing Object-Relational Data as XML, *Proc. WebDB 2000* (*Informal Proceedings*), pp.105–110 (2000).
4) Fernández, M., Suciu, D. and Tan, W.-C.: SilkRoute: Trading between Relations and XML, *Proc. WWW Conference* (*WWW9*), pp.723–745 (2000).
5) Fernandez, M., Morishima, A. and Suciu, D.: Efficient Evaluation of XML Middle-ware Queries, *Proc. ACM SIGMOD*, pp.103–114 (2001).
6) Halevy, A. (Ed.): Special Issue on XML Data Management, *Bulletin of the TCDE*, IEEE CS, Vol.24, No.2 (2001).
7) Melton, J. and Simon, A.R.: *SQL:1999 — Understanding Relational Language Components*, Morgan Kaufmann (2001).
8) Shanmugasundaram, J., Kiernan, J., Shekita, E., Fan, C. and Funderburk, J.: Querying XML Views of Relational Data, *Proc. 27th VLDB Conference*, pp.261–270 (2001).
9) Shanmugasundaram, J., Shekita, E., Barr, R., Carey, M., Lindsay, B., Pirahesh, H. and Reinwald, B.: Efficiently Publishing Relational Data as XML Documents, *The VLDB Journal*, Vol.10, No.2-3, pp.133–154 (2001).
10) Shinagawa, N., Kitagawa, H. and Ishikawa, Y.: X^2QL: An eXtensible XML Query Language Supporting User-defined Foreign Functions, *Proc. ADBIS-DASFAA Symposium on Advances in Databases and Information Systems*, pp.251–264 (2000).
11) Shinagawa, N. and Kitagawa, H.: Extension Mechanism in Extensible XML Query Language X^2QL, *Proc. 2nd International Conference on Web Information Systems Engineering* (*WISE 2001*) (2001).
12) 赤堀正剛，有澤達也，遠山元道：SuperSQLによる関係データベースとXMLデータの統合利用，情報処理学会論文誌：データベース，Vol.42, No.SIG 8(TOD 10), pp.66–95 (2001).

付 録

A.1 問合せ変換方式の詳細

A.1.1 XQuery 問合せの XQGM グラフへの変換

XQGMはXPERANTOグループにより提案された問合せの中間形式であるが，論文8)では，XQuery問合せからXQGMグラフを導出する方式については触れていない．また，本研究で扱う問合せは論文8)よりも一般的なものである．そこで，以下に本研究における変換方式を示す．

本研究では，ビュー定義XQuery問合せおよびユーザXQuery問合せに，4.1節で述べたような制約を課している．XMLビュー定義XQuery問合せ（例：図9）およびユーザXQuery問合せ（例：図3）を，それぞれ図10，図12のようなXQGMグラフに変換するため，以下のような手順をとる．

（1） `for`節を処理し，対応する部分グラフを作成する：4.1節の制約条件により，`for`節にはフィルタリング条件を含まない単一のパス式のみが現れることに注意する．変換処理は直接的である．実テーブルもしくはビューを表すノードを開始ノードとし，パスを1ステップたどるごとに要素内容リスト取得のための`getContents`関数を含む`project`演算子，要素内容リストからの要素取得のための`unnest`演算子，`isElement`関数による要素かどうかのチェック（必要であれば，`getTagName`関数を用いた要素名のチェック）を含む`select`演算子からなるシーケンスを繰り返す．たとえば，図3の`for`節は図12のXQGMグラフのノード1から4に変換される．なお，図9の`for`文に現れる“`view("default")/市/row`”のようなパスについては，これが実テーブル（市テーブル）の行を参照していることから，図10のノード1のように，実テーブルを参照するノードに変換する．

（2） correlated join演算子のノード（図10のノード10，図12のノード18）を作成し，`for`節に対応するグラフからそのノードへの有向辺を作成する．

（3） `where`節が存在する場合，対応する部分グラフを作成する：`where`節中の各条件について，`for`節の

場合と同様，パスをたどって対象の要素を得るための部分グラフを作成する．ただし，図12のノード7，10で示されるように，選択条件を select 演算子のノード中に含める．また，correlated join 演算子のノードから各条件に対する部分グラフの開始ノードへ，相関関係を表す点線の有向辺を作成する．where 節中に条件が複数ある場合（4.1 節の制約により，本研究では複数の条件は論理積で結合されると想定している），図12のノード 11 のように intersect 演算子を導入し，各条件に対する部分グラフと接続する．

(4) return 節に対応する部分グラフを作成する：ユーザ XQuery 問合せについては，4.1 節で述べたように return 節に副問合せが現れないという制約があるため，for 節，from 節の場合と同様の処理となる．ビュー定義 XQuery 問合せについては，return 節に含まれる副問合せごとに部分グラフを作成する．このような副問合せは，最終結果となる XML 文書に入れ子状に部分 XML 文書を埋め込むためのものであるため，主問合せと相関関係を有していることに注意する．変換結果の一般形は，図10のノード 2から5，およびノード 6から9のように，table 演算子による実テーブルの指定，select 演算子による相関関係を反映した選択処理，タグ付け用の XML 関数群を含む project 演算子による部分 XML 要素の生成，groupby 演算子による XML 部分要素のグループ化というステップからなる．最後に，相関関係に対応する select 演算子のノードへ，correlated join 演算子のノードから点線の有向辺を生成する．

(5) 出力 XML のための部分グラフを作成する：この部分グラフは，correlated join からの有向辺を入力とする．ユーザ XQuery 問合せに対しては，correlated join の結果の XML 文書の集合をリストに変換し，最終結果を表す XML タグに埋め込めばよい．図12のノード 19, 20 がその処理に相当する．ビュー定義 XQuery 問合せについては，return 節の記述内容に従って，correlated join の結果にタグ付けするための XML 関数のシーケンスからなるノード（図10のノード 11）を生成する．

A.1.2 相関関係分離の処理手順

相関関係分離の処理は，4.2.1 項で述べたとおりであるので省略する．

A.1.3 ビュー合成の処理手順

A.1.3.1 キー値による結合処理のための変形

ビュー合成を行う前の前処理として，まず，相関関係分離結果のユーザ問合せ XQGM グラフを，キー値を用いた結合処理のために変形する．図15の where 節に対する部分グラフ（ノード 1〜11）を変形した例を**図 32** に示す．ノード 15 に示されるように，キー値「市 ID」を出力し，その後の結合処理（where 節の条件を満たすキー値でのフィルタリング処理）に備える．それにともない，キー値を取得するための処理をグラフに加える．図32では，ノード 5，12 でキー値である属性 id の値を取得し，その値を上位ノード 15 まで保持するよう，変形が加えられている．なお，ここでは後の変換処理の見通しを良くするため，図32に見られるように重複したパスを複製し，独立した複数のパスとしておくが，実際の処理の対象は重複したパスでも差し支えない．

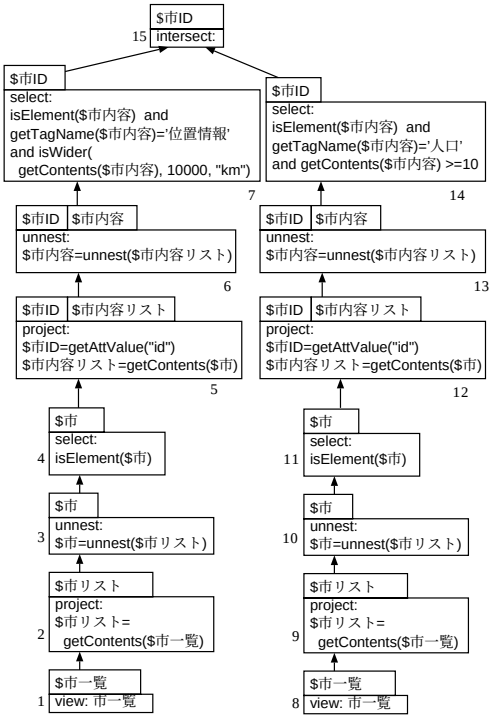

図 32 キー値による結合処理のための変形
Fig. 32 Translation for key-based joins.

A.1.3.2 関数合成ルールの適用と冗長な結合処理の削除

次に，4.2.2 項で述べたように，相関関係分離結果のユーザ問合せ XQGM グラフとビュー定義 XQGM グラフを合成する．図32のノード 1〜7 の部分グラフに図14を接ぎ木し，表3の関数合成ルールを繰り返し適用した中間結果を**図 33** に示す．

この図においてノード 12 は，ノード 11 から得られるそれぞれの市 ID および，各市 ID に対する，市名，人口，位置情報，施設一覧の XML フラグメントの和をとった内容（市内容）をノード 13 に渡してい

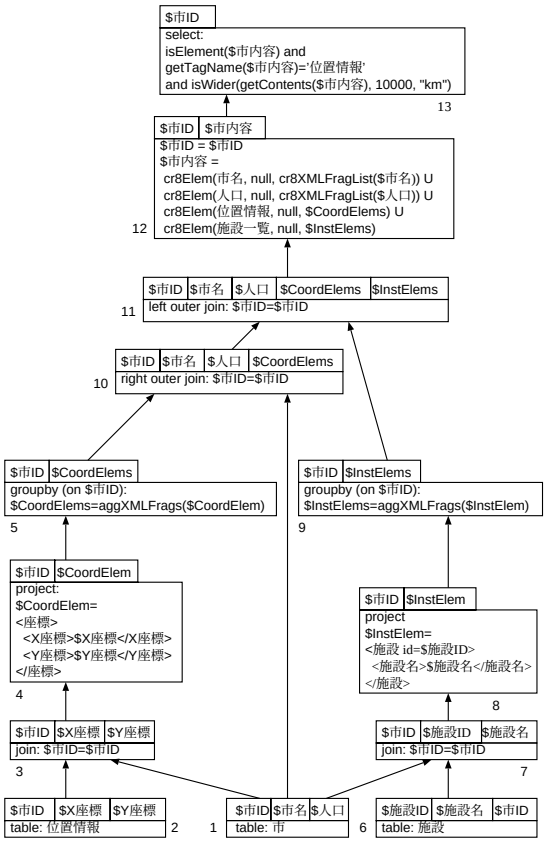

図 33 ビュー合成の中間結果
Fig. 33 Intermediate result of view composition.

る．しかし，ノード 13 ではこのうち，市 ID および市内容の中の位置情報しか使用しない．よって，このことをノード 13 からリーフ方向に伝播させ，冗長な結合処理を削除する．具体的には，施設情報および市に関する情報を結合し提供するためのノード 10，11 をそれぞれ削除し，それに関連する子ノード 6，7，8，9 を削除する．その結果のグラフに再度関数合成ルールを適用すると，図 16 のノード 1〜6 の部分グラフが得られる．同様のアプローチで図 32 のノード 8〜14 の部分グラフも変形でき，図 16 のノード 1，7 の部分グラフが得られる．

`return` 節の部分グラフに関するビュー合成の処理も同様であるが，キー値のみ返せばよい `where` 節と異なり，`return` 節の場合には，キー値以外に出力 XML の部分要素となる属性値および XML フラグメントを返す必要があることを考慮し，キー値以外の出力もそのまま部分グラフの出力に含めたままで変形し，図 16 のノード 9〜13 の部分グラフを得る．

ビュー合成の最後のステップとしては，結合処理から結果生成までのステップを表す部分グラフの修正がある．これには，図 15 のノード 22，23 の結合処理を図 16 のノード 14，15 で示されるキー値に基づく結合処理に変換することや，図 16 のノード 16 のように，`return` 節に対するサブグラフで行っていたタグ付け処理を最終処理段階に移行するためのグラフ修正などがあげられる．これらの処理の詳細については省略する．

入力：タグ付け演算子プルアップ後 XQGM グラフ G
出力：問合せ SQL-1
1) G 中の各 table ノード t についてビューを定義する．ただし，
 - t を始点とするプッシュダウン対象部分グラフ中の join ノード，select ノード中の結合条件および選択条件を，ビュー定義 SQL の where 節に含める．
 - t を始点とするプッシュダウン対象部分グラフの出力属性を調べ，ビュー定義 SQL の select 節に含める．
2) where 節に対応するプッシュダウン対象部分グラフについて outer union 演算を用いたビューを定義する．その際，この部分グラフの出力属性に加え，target および type 属性を outer union に追加し，両者の属性値を 0 に設定する．
3) 2) で作成した outer union ビューの内容を出力する SQL 文を生成する．ただし，キー属性，target，type という属性の並びでソート出力するよう，order by 節を追加する．

図 34 分割処理方式 (where) における *SQL-1* の生成アルゴリズム
Fig. 34 *SQL-1* generation algorithm for two-step processing method (where).

A.1.4 演算子プッシュダウンとタグ付け演算子プルアップの処理手順

演算子プッシュダウンとタグ付け演算子プルアップの処理については，基本的に XPERANTO のアプローチ[8] に従う．ただし，XPERANTO では相関関係分離を演算子プッシュダウンの処理の一部に含めているのに対し，本研究のアプローチでは，相関関係分離を別途ビュー合成の前に済ませてある点に違いがある．また，4.2.3 項で述べたように，外部関数評価を考慮してプッシュダウン処理に一部制限を加える点も異なる．タグ付け演算子プルアップについては，4.2.4 項に述べたとおりである．

A.2 SQL 問合せ生成アルゴリズムの概略

図 17 に示したようなタグ付け演算子プルアップ後 XQGM グラフから，問合せ処理方式に応じた SQL 問合せを生成するアルゴリズムの概略を示す．

まず，**図 34** は，分割処理方式 (where) において問合せ *SQL-1* を生成するためのアルゴリズムである．このアルゴリズムにおいて，「プッシュダウン対象部分グラフ」とは，たとえば図 17 では点線で囲んだ各部分グラフに相当する．このアルゴリズムの結果として図 21 に示したような SQL 問合せ *SQL-1* が得られる．

次に，分割処理方式 (where) において問合せ *SQL-2*

入力：タグ付け演算子プルアップ後 XQGM グラフ G
出力：問合せ SQL-2
1) return 節に対応する各プッシュダウン対象部分グラフの結果を統合するための，outer union 演算を用いたビューを定義する．ただし，
 - 各部分グラフの結果を生成するための SQL 問合せを作成し，それらを和演算（union all）により統合する．
 - 和演算による統合対象の上記 SQL 問合せでは，target 属性の値が 1 となるように出力指定する．また，type 属性の値については，各テーブルを識別するための数値を設定する．
2) 1) で作成した outer union ビューの内容を出力する SQL 文を生成する．ただし，where 節の条件として in 述語を用いたキー値集合によるフィルタリング条件を付与する．キー値集合自体はこの時点では未定のため，キー値集合の位置にはダミー記号を与えておく．また，キー属性，target，type という属性の並びでソート出力するよう，order by 節を追加する．

図 35 分割処理方式 (where) における *SQL-2* の生成アルゴリズム
Fig. 35 *SQL-2* generation algorithm for two-step processing method (where).

を生成するためのアルゴリズムを**図 35** に示す．その結果得られるのが図 22 のような SQL 問合せ *SQL-2* であるが，この時点では in 述語を用いたキー値集合によるフィルタリング条件には，具体的なキー値集合ではなくダミー記号が与えられることに注意する．実際のキー値集合の値設定は，*SQL-1* 問合せの実行後に SQL 問合せ制御部によりなされる．なお，ここでは *SQL-1* と *SQL-2* の生成アルゴリズムを別個に示したが，実際の処理においてはアルゴリズム間で共有する情報が存在するため，両者は 1 つのプログラムとして実現される．

分割処理方式 (tmp) における *SQL-1* の生成アルゴリズムは図 34 とまったく同じであり，*SQL-2* の生成アルゴリズムも図 35 と同様となるので省略する．後者は，一時テーブルの作成，一時テーブルに対する索引付け処理指定，および，in 述語ではなく一時テーブルとの結合を用いた SQL 問合せを作成する点が分割処理方式 (where) の場合と異なる．

一括処理方式における *SQL-3* 問合せ生成アルゴリズムについても，図 34，図 35 に示した分割処理方式 (where) のアルゴリズムから容易に類推できるので省略する．

(平成 14 年 3 月 29 日受付)
(平成 14 年 9 月 17 日採録)

(担当編集委員 国島 丈夫)

川田 純（正会員）

1976 年生．2000 年筑波大学第三学群情報学類卒業．2002 年同大学大学院システム情報工学研究科修士課程修了．現在，株式会社リコーソフトウェア研究開発本部・研究統括センター所属．XML データ利用方式等に興味を持つ．

石川 佳治（正会員）

1965 年生．1989 年筑波大学第三学群情報学類卒業．1994 年同大学大学院博士課程工学研究科単位取得退学．同年奈良先端科学技術大学院大学情報科学研究科助手．1999 年より筑波大学電子・情報工学系講師．博士（工学）（筑波大学）．2000 年度山下記念研究賞受賞．文書データベース，空間データベース，情報検索等に興味を持つ．ACM，IEEE-CS，電子情報通信学会，日本ソフトウェア科学会，日本データベース学会各会員．

北川 博之（正会員）

1955 年生．1978 年東京大学理学部物理学科卒業．1980 年同大学大学院理学系研究科修士課程修了．日本電気（株）勤務の後，1988 年筑波大学電子・情報工学系講師．同助教授を経て，現在，筑波大学電子・情報工学系教授．理学博士（東京大学）．異種情報源統合，XML とデータベース，WWW の高度利用等の研究に従事．著書「データベースシステム」（昭晃堂），「The Unnormalized Relational Data Model」（共著，Springer-Verlag）等．ACM，IEEE-CS，電子情報通信学会，日本ソフトウェア科学会，日本データベース学会各会員．