情 報 学 基 礎 70−3
デジタル・ドキュメント 38−3
（２００３．３．２８）

RDB と ODB の融合技術と代数学的考察

小 松 誠[†,††]

BI をはじめとする企業情報システムにおいて，いまや情報系のみならず基幹系にもその基盤技術として XML-DB が浸透しつつある．一方でレガシーシステムが抱える資産を RDB から ODB へ移行するのは容易でなく，このことが XML-DB を中核とする次世代システム手法への移行の障壁となっている．

XMLPGSQL は経済産業省の補助を受けて著者によって開発された XML-DB ミドルウェアである．当初，低コストで容易に導入・運用ができることを目指して開発された XMLPGSQL であったが，実際には前述のようなシステム移行の障壁を取り除き，レガシーシステムである RDB と XML ネイティブな ODB の双方の優れた特徴を併せ持ち両者をシームレスに結合できる新しい特性を生み出した．

これまで XMLPGSQL の実装や性能面ばかりに目を向けてきたが，あらためて理論的裏づけを行い，RDB と ODB の融合技術から何が生まれるかを考察する．

Fusion technology between RDB and ODB and study with algebra

Makoto KOMATSU[†,††]

There has been used to developed XML-DB, on many corporate information systems such as Business Integrations. However, the problem is clear that is difficult to port the resources from the legacy systems on the RDB to new systems on the ODB like a XML-DB.

Then, the XMLPGSQL is a middleware developed by the author subsidized by METI. It was aimed to be able to use as low cost, easy to install, easy to use XML-DB. But it's brought forth the special feature to remove the former obstacle and to unite the superiority of both RDB and ODB.

1. はじめに

XML データベースは XML 文書というオブジェクトインスタンスを永続化させるための手段であるが，それと同時に，安全に管理されたシステムのもとで，XML 文書オブジェクトにアクセスするための方法を提供する最良の方法である．

企業文書やプライバシー情報の XML 化が急速に進展する中，これらの XML 文書を安全にかつ確実に管理する基盤ソフトウェアとして XML データベースのニーズは高まりつつある．

著者は平成 13 年度未踏ソフトウェア創造事業[☆]において，XML データベースミドルウェア XMLPGSQL を開発したが，それはまさしくこうしたニーズに応えるためであった．

XMLPGSQL はオープンソースデータベースとして世界で幅広く利用されている PostgreSQL に XML ネイティブな機能[☆☆]を付加するものであり，PostgreSQL が持つ堅牢性や運用性を引出しながら，XML 文書へ自由にアクセスすることを可能にしている．

しかしそれだけではなく，既存システムのデータベーススキーマに影響を与えず，なおかつ既存のデータリレーションと自由に結合させてシステムを構築できるという画期的な特徴を有している．

それはあたかもリレーショナルなデータモデルと XML データモデルとを融合したような新しいデータモデルをなしているといえよう．

ここで XML データモデルとは一般にオブジェクトデータベースが有している木構造のデータモデルと同

† 株式会社メディアフロント (MediaFront Inc.)
http://www.mediafront.co.jp/
†† 日本 PostgreSQL ユーザー会
(Japan PostgreSQL Users Group)
http://www.postgresql.jp/

☆ 情報処理振興事業協会 (略称：IPA，経済産業省外郭団体) 実施

☆☆ ここでは XML 文書に DOM 手順で直接アクセスできることを示している

等のものであるが，リレーショナルなデータモデルとXML データモデルとは，本質的に違うスキーマに基づくものであることから，これらを一体的に取り扱うためにこれまで多くの努力がなされてきているところである．それらの多くでは XML 文書をどのようにリレーションへマッピングするかという点に工夫がなされ，本質的に両者を一体として取り扱う方法論は取られていない．

しかし XMLPGSQL においては，はじめから両者のデータモデルを一体化した新しいデータモデルを想起して，いわば融合データモデルに基づいて問題の解決を図ろうとしている点が画期的である．

このような融合データモデルをどのように理論づけるのか，あるいはそこにどのような可能性が見出せるのかについては，残念ながら，これまで全く論じられてこなかった．

そこで本稿では関係代数学的視点から，あらためて XMLPGSQL のデータモデルについて検証し，その有用性について評価することを目的とする．

2. XML 文書の空間定義

2.1 XML 文書と集合

XML 文書 D は，文書スキーマ D^* によって規定される XML 文書インスタンスのひとつである．

XML 文書 D は任意の要素とテキストから構成されており，この文書 D を構成する要素集合 $E\{\}$ は $E\{E_0,\ldots,E_k\}$ と表され，要素 E_m に含まれるテキスト集合 $T\{\}$ は $T\{T_1^m,\ldots,T_l^m\}$ と表される．

このとき，E_m を要素ノード，T_l^m を要素 E_m に属するテキストノードという．$m=0$ のとき，E_0 はルートノード直下の特別な要素ノード (最上位要素ノード) を表す．

すべての E が属する空間を $\prod_E$ とする．またこのとき要素 E_k が規定する空間を $\prod_{E_k}$ と表す．

明らかに $\prod_E \equiv \prod_{E_0}$ である．

ある E_i が他のいずれかの要素ノード E_j に属するとき，これを $E_i \in E_j$ と表す．

定義から，

$$\{[E_i,E_j]|E_i \notin E_j \wedge E_j \notin E_i\} \Rightarrow \{E_i \cap E_j = \emptyset\}$$

および

$$\{E|E_{k>0} \in E_0\} \wedge \{T|T_* \in E_0\}$$

であるのがわかる．

2.2 XML 文書が規定する集合空間

ある任意の E_i，E_j，E_k において，

$$\{[E_i,E_j]|E_i \in E_k \wedge E_j \in E_k\} \wedge \{E_i \cap E_j = \emptyset\}$$

であるとき，E_i と E_j は互いに独立であるという．先に述べたように $\prod_{E_k}$ は要素 E_k が規定する空間を表す．

ここで $\prod_{E_i},\prod_{E_j},\prod_{E_k}$ には，以下の関係が成り立つ．

$$\left\{\prod_{E_i} \subset \prod_{E_k}\right\} \wedge \left\{\prod_{E_j} \subset \prod_{E_k}\right\}$$

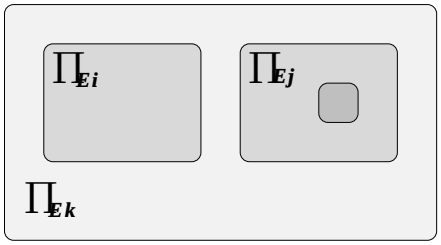

図 1 要素が規定する空間

さて，空間の横の広がりについてわかってきたところで，次に縦の広がりつまりは高さあるいは深さについて検討しよう．

XML 文書をツリー構造でとらえた場合に，ルートノードから見て，より遠く，枝が伸びるにしたがって，慣例的に「深くなる」と表現する．

ここで深くなるとは，平面的に見ると，つまりは多くの要素集合に含まれていくことを示している．

ある E_p について，

$$\left\{\prod_{E_i} \subset \prod_{E_p}\right\} \wedge \left\{\prod_{E_p} \subset \prod_{E_k}\right\}$$

が成り立つとき，E_p は E_k よりも深く，E_i よりも浅いという．

そこでこの相対的な深さの差を「深度の差」として，たとえば E_i と E_k の「深度の差」を $\Delta_i^k(E_i \in E_k)$ と表すこととする．

もしも E_i の「深度」そのものを表す必要があれば，E_0 を用いて，Δ_i^0 とするのが相当であり，この場合は特に Δ_i と略することとする．

$\Delta_i^k = \Delta_j^k = 1$ であれば，E_i と E_j をひとくくりにして，E_t で置き換えることができる．このとき，$\prod_{E_t} = \prod_{E_i} \bigcup \prod_{E_j}$ という結合則が成立する．

このように，単一の XML 文書が表現する空間 $\prod_E$ とは，

$$\prod_E \equiv \prod_{E_0} = \prod_{E_1} \bigcup \prod_{E_2} \bigcup \prod_{E_3} \cdots \prod_{E_k}$$

ととらえることができる．

3. XML 文書ノードとリレーション

3.1 文書ノード集合

要素ノード集合 $E\{E_0,\ldots,E_k\}$ とテキストノード集合 $T\{T_1^m,\ldots,T_l^m\}$ とを合わせて，文書ノード集合☆(または単に文書ノード) $N\{N_0,\ldots,N_n\}$ とする．

このとき，$E_0 \equiv N_0$，また，定義から $n = k + \sum_{e=1}^{k} l$ となっている．

3.2 文書ノード集合と和両立

あるデータベース DB における，2つのリレーション R と S を $R(A_1,\ldots,A_n), S(B_1,\ldots,B_m)$ と表す．

ここで $A_1,\ldots,A_n$，$B_1,\ldots,B_m$ はリレーションを構成する属性である．

このときリレーションが持つ属性の数を「次数」という．また各属性の意味する概念をドメインという．

いま，2つのリレーション R と S に関して次の条件を満たすとき，これら2つのリレーションは和両立であるという．

(1) R と S の次数が等しい

(2) 各 $i(1 \le i \le n)$ について，A_i と B_i のドメインが等しい $(dom(A_i) = dom(B_i))$

和両立であることは，集合の和演算，差演算，積演算 (共通演算) を定義可能とするための必要条件とされている．

前節において述べたとおり，任意の XML 文書が表現する空間はその文書に属する全ての要素が規定する集合空間の和で表される．

よって任意の要素が規定する空間 $\prod_{E_i}, \prod_{E_j}$ について，これらをリレーションで置き換えたときに，これらのリレーションが和両立であるかを確認する．

任意の要素ノード E_i，E_j について，E_i が規定する空間 $\prod_{E_i}$ をリレーション $N^{E_i}(X_1,\ldots,X_n)$ と表現し，E_j が規定する空間 $\prod_{E_j}$ をリレーション $N^{E_j}(X_1,\ldots,X_n)$ と表現する．

ここで $X_1,\ldots,X_n$ はリレーション N^E における属性を示す．

明らかに，N^{E_i}，N^{E_j} について，

(1) N^{E_i} と N^{E_j} の次数は等しい

(2) 各 $i(1 \le i \le n)$ について，A_i と B_i のドメインが等しい

すなわち2つのリレーション $N^{E_i}(X_1,\ldots,X_n)$ と $N^{E_j}(X_1,\ldots,X_n)$ は相互に和両立である．

さらには任意のテキストノード T_m^k についても，$N^T(X_1,\ldots,X_n)$ と表現できれば，ここにすべての文書ノードを相互に和両立なリレーションとして表現できることとなる．

これらを整理して，$N(X_1,\ldots,X_n)$ としよう．すなわち，リレーション $N(X_1,\ldots,X_n)$ は XML 文書を表現し，そのタプルは全て，ある XML 文書に属する文書ノードである．これを XML 文書リレーションと呼ぶことにする．

任意の XML 文書リレーション $N(X_1,\ldots,X_n)$ は，相互に和両立である．

3.3 文書ノード集合と和演算

2つの和両立の関係にあるリレーション，

$R(A_1,\ldots,A_n), S(B_1,\ldots,B_m)$

において，R と S の和は以下のように定義される．

$$R \cup S = \{t | t \in R \vee t \in S\}$$

XML 文書リレーション $N(X_1,\ldots,X_n)$ においては，和両立の関係にあるリレーション $R(A_1,\ldots,A_n)$ もまた XML 文書リレーションである．

ここでリレーション N と R が互いに独立であるとき，$\prod_N = \prod_{E_i}, \prod_R = \prod_{E_j}$ とすると，

$$E = E_0 \cup N \cup R | \Delta_i = \Delta_j = 1$$

$$\Delta_i^k = \Delta_j^k = 1$$

さらに，リレーション N と R が独立でないときには，$\{N \in R \vee R \in N\}$ であり，

$$N \cup R = N | R \in N$$

または

$$N \cup R = R | N \in R$$

である．

3.4 文書ノード集合と差演算

2つの和両立の関係にあるリレーション，

$R(A_1,\ldots,A_n), S(B_1,\ldots,B_m)$

において，R と S の差は以下のように定義される．

$$R - S = \{t | t \in R \wedge \neg t \in S\}$$

XML 文書リレーション $N(X_1,\ldots,X_n)$ においては，和両立の関係にあるリレーション $R(A_1,\ldots,A_n)$ もまた XML 文書リレーションである．

ここでリレーション N と R が互いに独立であるとき，定義から明らかに，$N - R = R - N = \phi$ である．

さらに，リレーション N と R が独立でないときには，$\{N \in R \vee R \in N\}$ であり，

$$N - R = N | R \in N$$

または

$$N - R = \phi | N \in R$$

である．

☆ 実際には XML 文書ノードとして取り扱われるものはこれだけではない．本稿ではあくまでも簡略化して述べている

3.5 文書ノード集合と共通演算

2 つの和両立の関係にあるリレーション，

$R(A_1,\ldots,A_n), S(B_1,\ldots,B_m)$

において，R と S の共通は以下のように表される．

$R \cap S = \{t | t \in R \wedge t \in S\}$

これは差を利用して次のようにも表現可能である．

$R \cap S = R - (R - S)$

XML 文書リレーション $N(X_1,\ldots,X_n)$ においては，和両立の関係にあるリレーション $R(A_1,\ldots,A_n)$ もまた XML 文書リレーションである．

ここでリレーション N と R が互いに独立であるとき，定義から明らかに，$N \cap R = \phi$ である．

さらに，リレーション N と R が独立でないときには，$\{N \in R \vee R \in N\}$ であり，

$N \cap R = R | R \in N$

または

$N \cap R = N | N \in R$

である．

3.6 文書ノード集合と直積演算

2 つのリレーション $R(A_1,\ldots,A_n), S(B_1,\ldots,B_m)$ において，直積演算は，以下のように表される．

$R \times S = \{r * s | r \in R \wedge s \in S\}$

ここで $r * s$ はタプルの連結を表す．

XML 文書リレーション $N(X_1,\ldots,X_n)$ においては，あらゆる他のリレーションと自由に直積を取ることが可能である．

3.7 文書ノード集合と選択演算

3.7.1 θ-比較可能

XML 文書リレーション $N(X_1,\ldots,X_n)$ について，任意のノードであるタプル t の属性に対して $t[X_i]\ \theta\ t[X_j]$ を考える．

ここで，θ とは，$\{=, <, >, \leq, \geq, \neq\}$ のいずれかの比較演算子を示す．

リレーション $R(A_1,\ldots,A_n)$ において，属性 A_i と A_j が比較可能であるのは，以下の条件が成立するときである．

(1) $dom(A_i) = dom(A_j)$

(2) R の任意のタプル t に対して，$t[X_i]\ \theta\ t[X_j]$ の真偽が常に定まる

XML 文書リレーション $N(X_1,\ldots,X_n)$ についても $t[X_i]\ \theta\ t[X_j]$ が可能であり，θ-比較可能である．

3.7.2 θ-選択演算

選択演算は正確には θ-選択演算といわれる．リレーション $R(A_1,\ldots,A_n)$ における R の属性 A_i と A_j での θ-選択を $R[A_i \theta A_j]$ と書き，これは次のように表現されるリレーションである．

$R[A_i\ \theta\ A_j] = \{t | t \in R \wedge t[A_i\ \theta\ A_j]\}$

ただし，A_i，A_j ともに空値を取らないことが必要である．

また空でない定数 C との比較についても，

$R[A_i\ \theta\ C] = t | t \in R \wedge t[A_i]\ \theta\ C$

である．

XML 文書リレーション $N(X_1,\ldots,X_n)$ についても θ-選択が可能であり，

$N[X_i\ \theta\ X_j] = t | t \in N \wedge t[X_i\ \theta\ X_j]$

$N[X_i\ \theta\ C] = t | t \in N \wedge t[X_i]\ \theta\ C$

である．

3.8 文書ノード集合と結合演算

3.8.1 θ-結合演算

XML 文書リレーション $N_1(X_1,\ldots,X_n)$ とリレーション $R(A_1,\ldots,A_m)$ において，X_i と A_j が θ-比較可能であるとき，θ-結合 $N[X_i\ \theta\ A_j]R$ を考える．

結合演算の定義から，結果リレーションは以下のように表される．

$N[X_i\,\theta\,A_j]R = \{(t,u) | t \in R \wedge u \in S[A_i]\,\theta\,u[B_j]\}$

また，直積演算と θ-選択演算を用いれば，次のようにも表される．

$N[X_i\ \theta\ A_j]R = (N \times S)[N.X_i\ \theta\ R.A_j]$

4. リレーションにおける XML 文書の表現

リレーション $N(X_1,\ldots,X_n)$ は XML 文書空間を表し，そのタプルは文書ノードに対応させることができる．

これを実現するために必要十分な属性の組合せ $X_1,\ldots,X_n$ はどのようなものだろうか．

当然に要素ノード集合を表現する $N^E(X_1,\ldots,X_n)$ と，テキストノード集合を表現する $N^T(X_1,\ldots,X_n)$ とでは，必要十分なリレーション属性の組合せは違ったはずである．

そこでこれらを分けて整理してみる．

4.1 要素ノードを表現するリレーション属性

XML 文書において，要素とはすなわちタグのことである．タグはさらに，タグ名と属性名，そして属性値から成っている．

つまり，リレーション属性としては次のように考えられる．

- 要素名
- 属性名
- 属性値

4.2 テキストノードを表現するリレーション属性

テキストノードを表現するのは非常にシンプルであり，ただテキスト本体があればよい．

- テキスト

4.3 メタ情報を表現するリレーション属性

各文書ノードはそれらが属する単一の文書を示すことが必要となる．つまりは文書名が必要になる．

さらに，各ノードの種類が要素ノードであったりテキストノードであったりするので，これを区分するための情報も必要である．

- 文書名
- ノード種類

4.4 従属関係を表現するリレーション属性

文書ノードはそれぞれ独立しているのではなく，それぞれに従属関係があって全体として木構造をなしている．

これはすなわち，親ノードと子ノードのリンク情報を必要とすることを意味する．

さらには共通の親を持つ文書ノードに順序付けを行うための情報も必要である．

- 親子関係情報
- 兄弟順序情報

4.5 リレーション属性のまとめ

以上の検討をもとにして，以下のように案をまとめる．

- 文書名
- ノード種類
- 要素名
- 属性名
- 属性値
- 親子関係情報
- 兄弟順序情報
- テキスト

5. XMLPGSQL の実装

では XMLPGSQL においては実際どのように文書ノードをリレーション化しているのかを説明する．

5.1 固有 ID による効率化

XMLPGSQL においては，文書ノードはすべて 16 桁の整数からなる文字列の固有 ID で管理されている．複数の XML 文書を同一のデータベースで管理できるように，文書が異なってもノード ID がかぶらないように工夫されている．

さらには文書名や要素名も固有 ID 化され，それぞれ別のリレーションで実体が管理されている．すなわち第 2 正規形をなしている．これらの固有 ID もすべて 16 桁の整数からなる文字列で管理されている．

5.2 最低限のリンク情報

親子関係と兄弟順序については，3 つの属性にまとめられており，親，最初の子，次の兄弟だけがわかる構成になっている．

前の兄弟がすぐに辿れないが，SQL を応用すると問題なく辿れることがわかる．

5.3 属性情報の分離

属性名と属性値については 2 種類のアプローチをとっている．

(1) 属性名と属性値を式のままテキストとして管理
(2) 属性について文書ノードと管理テーブルを分離
属性ノードを作成して属性名と属性値を管理．
属性名はさらに固有 ID を振って別テーブルで管理．

前者は 1.x 系列，後者は 2.x 系列においてそれぞれ採用されている．

5.4 リレーション実装

XMLPGSQL2.x 系列では以下の 6 つのリレーションを用いている．

- xml_document . . . 文書名を管理する文書バインダ
- xml_tag . . . 要素名を管理するタグ辞書
- xml_attribute . . . 属性名を管理する属性名辞書
- xml_namespace . . . 名前空間名を管理する名前空間辞書
- xml_pi . . . 実行命令を管理する実行命令辞書
- xml_node . . . 文書ノード本体

5.5 DOM インターフェース

XMLPGSQL2.x 系列では以下のような DOM level2 準拠のインターフェースを SQL 関数として実装し，XML-DB 内部の XML 文書に直接アクセスできる．

- create_element(親ノード, 要素名)
- create_text_node(親ノード, テキスト)
- set_attribute(親ノード, 属性名, 属性値)
- get_attribute(親ノード, 属性名)

6. XMLPGSQL データモデルと XPath

6.1 XML 文書リレーションと軸

XPath1.0 では以下の 13 軸が定義されている．

- ancestor
- ancestor-or-self
- attribute
- child
- descendant
- descendant-or-self
- following
- following-sibling
- namespace

- parent
- preceding
- preceding-sibling
- self

XML 文書リレーションでこれらの軸を表現できるか検証する．

まず 13 軸を分類し，{self, attribute, namespace, parent, child, ancestor, descendant, preceding, following} の 9 種類に整理する．残る 4 つの軸は和集合によって求めることができる．

6.1.1 self

self 軸は文脈ノード自身であるので，文書リレーション N 自身を示す．すなわち $\{R = N\}$ である．

6.1.2 attribute

attribute 軸は N に従属する属性ノードを示す．すなわち $\{t \in R | t.kind = ATTRIBUTE \wedge t.parent = N\}$ である．

6.1.3 namespace

attribute 軸は N に従属する名前空間ノードを示す．すなわち $\{t \in R | t.kind = NAMESPACE \wedge t.parent = N\}$ である．

6.1.4 parent

parent 軸は N を子とする要素ノードを示す．すなわち $\{R | N.parent = R\}$ である．

6.1.5 child

child 軸は N を親とする，属性ノードと名前空間ノードを除いたものを示す．すなわち $\{t \in R | t.kind \notin (ATTRIBUTE, NAMESPACE) \wedge t.parent = N\}$ である．

6.1.6 ancestor

ancestor 軸は少し複雑である．まず従属関係にある任意の文書リレーション $(R_i, R_j) | R_i \in R_j$ を考える．当然に $R_i.parent = R_j \wedge R_i.kind \notin (ATTRIBUTE, NAMESPACE)$ である．次に文書リレーションの結合 $R_i[parent = id]R_j$ を考える．ここで，ある i について，$R_i.id = N.id$ である．このとき，$\{R | R_{n+1}.parent = R_n.id \wedge R_n.kind \neq ROOT$ である．実際には再帰クエリを用いて結果セットを得ることになる．

6.1.7 descendant

descendant 軸も ancestor 軸と同様に再帰手続きを必要とする．$\{R | R_n.id = R_{n+1}.parent \wedge R_n.kind \neq ROOT$

6.1.8 preceding

preceding 軸は構造的な文書ノードの位置によるのではなく，文書順位 (ドキュメントオーダー) によって定められる．しかし現在の XMLPGSQL は文書順位を定めていないために，preceding 軸が定義できない．しかし考え方は平易であるので，別途文書順位を管理することで preceding 軸を処理することが可能である．

6.1.9 following

following 軸についても preceding 軸と同様，文書順位を必要とするので，現状では定義することができない．

7. 文書モデルと考察

以上のことから，XMLPGSQL の採用するデータモデルは，リレーショナルなデータモデルでありながら，さらに XML に適したツリー構造のデータモデルとしての側面も合わせ持つことがわかった．

さらに DOM インターフェースや XPath 式との親和性も合わせ持つ柔軟性のあるデータモデルであることが発見できた．

今後はこのデータモデルをベースとして，系の標準化と問題解決の体系化に取り組む考えである．

XMLPGSQL は DOM，XPath ともにサポートする本格的なオープンソースの XML-DB としてその地位を固めつつある．

今回文書リレーションモデルを確立したことで，今後は学術分野での活用やユーザ自身による様々な機能拡張を期待したい．

8. おわりに

まず今回の発表に際して，迷っていた著者の背中を押して下さった野村直之氏 (法政大学エクステンションカレッジ) に心から感謝したい．

また，慣れない TEX のことで丁寧にご対応頂いた情報処理学会研究会事務局の方にも御礼申し上げる．

参 考 文 献

1) 増永 良文: リレーショナルデータベース入門 [新訂版], サイエンス社 (2003).
2) Peter M.D.Gray, 訳．田中 穂積，徳永 健伸: 論理・代数・データベース, 産業図書 (1993).
3) 吉川正俊: データベースの観点から見た XML の研究.
4) 小松 誠：RDB と ODB を融合する XML-DB フレームワーク, 未踏ソフトウェア創造事業 (2002).