平成 13 年度未踏ソフトウェア創造事業
テーマ名（RDB と ODB を融合する XML-DB フレームワーク）

Framework for XML-DB which is applied from harmonizing RDB with ODB

小松 誠 [1)]
Makoto KOMATSU

1) 有限会社メディアフロント（〒730-0015 広島市中区橋本町 2 番 19 号　E-mail: eurah@mediafront.co.jp ）

ABSTRACT. XML which is increasing presence as a means of the data expression in the Internet. Also in the back end of Web application or intranet, the practical use range has spread. On the other hand, isn't that there are few solutions suitable for a minor scale user the hindrance of spread, either? Although PostgreSQL which is RDBMS of the open-source product currently used for the user of present many is permeating such a minor scale user, software XMLPGSQL which extends the PostgreSQL and gives the function of XML-DBMS is developing it as open-source product too. It has been the epoch-making trial which can cooperate with the system of XML base, inheriting RDB property.

1．背景

Web アプリケーションでは多くの場合, Web サーバとデータベースサーバが協調して動作しており, その協調動作を記述するプログラムとデータがサーバ側に収められています.

そのデータベースとしては, Oracle や SQLServer 等の商業製品だけでなく, MySQL や PostgreSQL などのオープンソース製品も数多く使われています. これらは表形式のデータベースを関係モデル（RelationModel）構成しているところから, リレーショナルデータベース管理システム（RDBMS）とも呼ばれています. これらの RDBMS では SQL という標準的な手続言語の利用によって自由に操作できるので, イントラネットやインターネットの Web アプリケーションでも多く利用されるようになっています.

一方で最近では, 高い可搬性や記述性を有するデータの規格として XML がよく用いられるようにもなっています. XML は HTML と同様にタグ付けされたテキスト文書という体裁でありながら, 高度な意味付けがなされたデータ構造を実現でき, また文書定義が明らかであれば異なるシステム間でも移植･搬送が可能であるために, これからの普及が確実視されています.

こうした流れから, 当然に XML を効率良く管理操作できるシステムが求められるところですが, RDBMS には本質的に XML を取り扱いにくいという弱点がありました.

それは, RDBMS が表形式のデータ操作に向いているのに対して, XML は木形式のデータ構造をとっているという事情によるものです.

このことから現状 Web アプリケーションにおいて XML を取り扱うには, DBMS によらない方法, 例えば XML 形式のテキストファイルを OS のファイルシステムに依存して操作する方法などがよく用いられています. しかしセキュリティを考慮した場合に, こうしたデータが生のまま取り扱われることは本来リスクがあり, 専用の高価な DBMS

が用いられるケースもあります.

2．目的

私はオープンソース RDBMS として広く普及した PostgreSQL を用いてこれを拡張するソフトウェアを追加することで, それまでの RDBMS としての特性を損なうことなく新たに XML を操作できる機能を付加する方法を考案し, これを発表いたしました.

最も画期的なことは, Web ソリューションとしてすでに実績のある PostgreSQL を用いながら, 必要に応じて XML-DB を取り扱えるよう拡張が可能で, 他のシステムには何ら影響を及ぼすことなく, アプリケーションに機能を追加できるという点です. しかも内部的には XML ノードツリーを擬似的に表現しているため, データに制約が加えられることもなく, 原理的にはマルチメディアデータも取り扱うことが可能です.

3．Web アプリケーションと XML

Web アプリケーションは拡張の容易さやメンテナンスコストの低減といった優れた特徴を備えながらさらなる展開をしつつありますが, 多くの Web アプリケーションが世に出るにつれて, いくつかの問題点が見えてきました.

その 1 つがアプリケーション間でのデータの可搬性と互換性です. 元来データベースサーバに格納されているデータは直接インターネットにさらされていないのが普通ですから, アプリケーションインターフェースを通さずにデータの受け渡しをすることは不可能です. また, アプリケーションが受け渡しするデータは通常 Web 記述言語である HTML で書かれているわけですが, これは再利用性の低いデータであり, 扱いにくいものです.

そこでこのようなデータの受け渡し, そしてそれのみならず, 内部のデータ構造そのものを記述できるものとして, XML が注目されるようになりました.

XML は HTML と同じくタグ型言語でありながら, データの意味付けを行いながら記述でき, 高い再利用性を持ちます. またデータの種類によっては, 特定の属性記述があったりなかったりという曖昧さを持つ場合があり, こうしたデータについては通常のリレーショナル型データベースよりも木構造で記述できる XML のほうが記述性が高い場合があります. さまざまなアプリケーションが多くの場所で使われるようになれば, こうした長所を活かせる機会も増すことでしょう.

こうした理由から, 今後は XML を活用した Web アプリケーションが増えていくものと思われます. このように Web アプリケーションのうち XML を活用したものを特に XML アプリケーションと呼ぶことにします.

4．XML アプリケーションの特徴と分類

XML アプリケーションは, その機能的特徴からいくつかに分類できます.

(1) 扱う XML の種類による分類

XML 文書には, 大きく 2 つの分類があります. 1 つは文書構造の整合性を保証した整形式文書 (well-formed document). もう 1 つは, タグの種類や使い方をも規定した妥当文書 (valid document) です.

整形式文書は, 文書構造が規約どおり正しく組み立てられてさえいれば, タグ名の種類や使い方を自由に定義できて, それらを自由に追加したり削除したりできます. つまり柔軟な文書システムを構築するのに優れています.

その反面，文書内で決められたタグはその文書内でのみ通用するものですから，外部の文書やシステムと適合させるためには特別なしくみを用意して，タグの置き換えを行う必要があります．あるいは適合させることさえ困難になることもあります．これは難点です．

他方の妥当文書というものは，タグの使い方や種類をあらかじめ決めておいて，このルールを元に文書を作成します．ルールさえ共通にしておけば，外部の文書やシステムと適合させることも容易です．これはすなわち情報の流通性が高まるということを意味しています．異なる企業間で文書決済を行うような場合には有利な点です．

その代わりに，はじめにほんのちょっぴり手間をかける必要があります．文書のルール作りが必要なのです．このルールを記述したものを，DTD と呼んでいます．DTD は，基本的には最初に規定すれば変更されることはありません．つまり途中でタグの種類を増やしたり減らしたりということは自由にできないわけです．

(2) 流通するものか記録するものか

XML アプリケーションでの XML 文書は，通常サーバ側に文書情報を保持しているはずです．その文書情報に基づいて，アプリケーションのシステムは動作を決めます．

サーバに収められた XML 文書は，ただそこに記録され管理されるだけではなく，インターネットを通じて，特定のサーバ間を自由に通信することが可能です．蓄えられた XML 文書もその一部または全部を，多種多様な方法で流通させることができます．ただし XML 文書を流通させる上で重要なことは，文書中のタグの意味を明らかにしておくこと，あるいはそれらを他のサーバシステムと共通にしておくことです．

(3) サーバかクライアントか

最後の区分は，具体的に XML 文書を加工したり検索したり操作したりといった作業を主にどこで行うかという点です．

多くの Web アプリケーションは多層構造といって，データベースサーバ，Web サーバ，Web ブラウザなどの複数のコンピュータシステムから構成されています．

この場合，ユーザが実際に使うのは Web ブラウザが動作するパソコンなどの情報端末です．多くの場合は HTML によって書かれた Web ページを 1 枚ずつサーバが生成し，クライアント側である Web ブラウザがそれを画面に組み合わせて表示します．あるいは逆にユーザの入力した情報を Web ブラウザが編集してサーバへ返すということもあります．

こうした作業の中では文書処理のほとんどはサーバに依拠しています．クライアントは HTML という制約のもとにあって，あまり複雑なことはできません．

しかし最近 Web ブラウザが XML にネイティブな対応を進めてきた結果，サーバから必ずしも HTML を送らなくても，直接 XML を送ればよいという選択も現れました．XML には XSLT というデータ操作規約によって動的に出し方を変えたり，スタイルシートによって見せ方を変えたりする方法も用意されています．最も単純な方法としては，データベースをまるごと XML 形式でクライアントへダウンロードして，クライアント側に組み込まれたスクリプトによってデータを検索操作したり印刷するという使い方です．これならネットワークでなくても，たとえば CD-ROM で XML 文書をデータベースとして提供するア

プリケーションも可能になります.

ただしブラウザがデータをすべて持っているということは, 誰でもそれを閲覧する方法があるということです. 機密や権限が必要なアプリケーションには適さない方法です.

むしろ大量あるいは複雑な XML 文書がサーバ側に格納されていて, サーバ側アプリケーションによってその一部が加工されて切り出され, それがクライアントへ渡されるというのが賢明な方法だと考えられます.

たとえば在庫状況を検索するシステムがあって, Web で検索を行った結果は表形式で出るけれども, 簡単な操作で帳票形式にもなるようであれば使い勝手は向上します. そのうえ受け取ったデータは表計算ソフトやワープロへ入力可能ですから再利用も容易になります.

5. Web アプリケーションとデータベース

一般的に Web アプリケーションは三層型アプリケーションと呼ばれます. クライアント側から見ると, Web ブラウザ, Web サーバ, データベースサーバという順番で構成されているためです. データベースサーバの代わりにアプリケーションサーバの場合もあります. (図1)

実際にはアプリケーション部分も Web サーバが行っている場合が多いのですが, その後ろ側では必ずといっていいほどデータベースサーバが動いています. サーバアプリケーションが要求するデータの管理や処理をデータベースサーバが引き受けています.

PostgreSQL も多くの Web アプリケーションのバックエンドで使われています. PostgreSQL の機能性や柔軟性といった理由だけでなく, 経済的な理由もあります. 商用データベースでは接続クライアントのライセンス数に応じた価格体系を取ることが多いものですが, インターネット利用の場合, ライセンス数が無制限となるために高コストになることがあります. こういったケースでは PostgreSQL の採用による大きなメリットがあります.

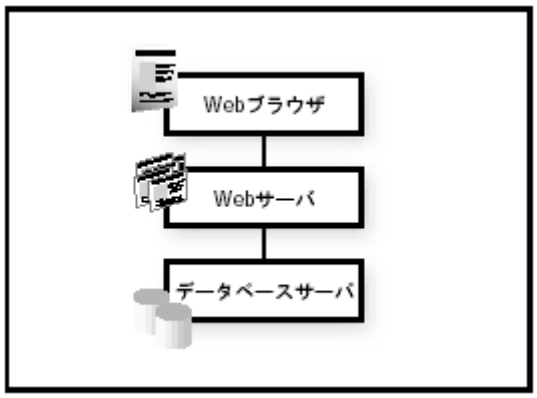

図1：Web アプリケーションの構成

6. XML とリレーショナルデータベース

リレーショナルとは関係あるいは関連づけのことです. リレーショナルデータベースとはテーブル形式のデータベースを複数用意して, データベーステーブル間の関連づけを定義したものです.

これは図 2 のように表されます. この例はあるくだもの屋さんの販売管理データベースの一部を表しています. このように, 表形式で記述できるデータの管理にはリレーショナルデータベースが適しています.

次に図 3 の例は, 食品の特性を記述したデータベースの一部です. ここでは「いちご」について記述されています. これは表形式ではありません. 実はこれが XML 形式になっています. このデータは木構造といって, 階層関係に基づいて書かれているので, 表形式にするのは難しいのです.

さて, 2 番目の例で扱われた XML ですが, これを表形式にするのが難しいのは, 食品ごとに必要な情報が

違っているからです. それは, いちごはそのまま食べたり潰して加工したりしますが, 小麦粉は練ったり固めたりして加熱する必要があるからです.

これらを情報を表に収めるためにはこうした項目や属性の違いをすべて吸収できるように, 可能な限り項目(フィールド)を追加してやらなければなりません. 結果として大変効率の悪いデータベースになってしまいます.

一方でXMLを用いると必要な情報だけを適宜追加していくことができます.

このようにXMLデータをアプリケーションで取り扱うことの価値は, XMLであれば柔軟に拡張ができ, 外部とのデータ交換も可能になるということでしょう.

さらにシステムの拡張や利用価値の変化によって, 新たなシステム設計を行う際にも速やかにかつスムーズにデータの移行が可能となるのも大きなメリットです.

図2:表形式のデータベース例

品名	単価	数量
みかん	１００	１０
りんご	２００	５
バナナ	１５０	２０

仕入先	品名	数量
○○商事	みかん	20
××物産	りんご	50
○○商事	バナナ	15

```
<果物>
  <いちご>
    <色>赤</色>
    <味>甘酸っぱい</味>
    <成分>
      <糖質>7.5g</糖質>
      <ビタミンC>80mg</ビタミンC>
      カルシウム
      食物繊維
    </成分>
  </いちご>
</果物>
```

図3:XMLデータの例

7．文書ノードの表現

全てのXML文書は木構造の文書ツリーで表現できるとされています. このような木構造は葉や節を表すノードとノードのつながりを表すパスで図式化することが可能です.(図4)

●表形式と木構造

たとえばC++やPascalのプログラミングでは, もともと構造体とポインタを使ったり, あるいはオブジェクト指向の概念を用いて木構造のデータ表現を行います.

同じことがデータベースでもできるのではないだろうか？これが着想のきっかけでした.

●親, 子, 兄弟

図5のように, 親ノード, 子ノードへのリンクを保持し, 兄弟関係を扱いやすくするために次の子へのリンクを保持します. こうすると, ノード間を楽に行き来することができるようになります. 親ノードからは長男(第1子)のみにリンクしていますから, 弟ノードから兄ノードへいくのは多少苦労を伴いますが, XMLの性質上, こういった遷移の必要性は低いでしょう.

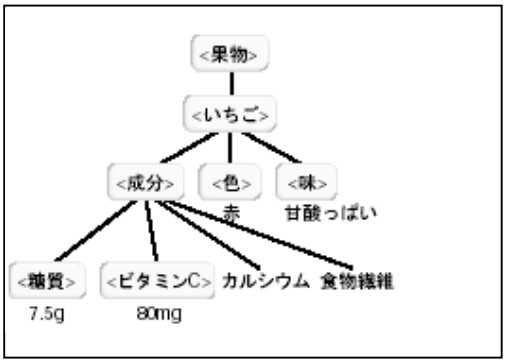

図4:木構造によるXMLデータの図式化

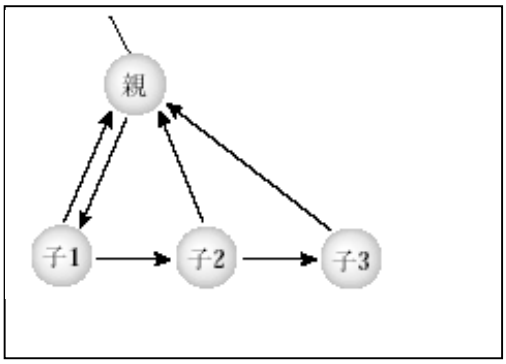

図 5:文書ノードの表現

●ノードとエレメント

さてノードをつなぐ木構造はできました. ところで, ここで気をつけておくべきことがあります.

XML では基本的な単位を要素(エレメント)といいます. XML 要素は開始タグで始まり終了タグで終わります. 先ほどのノードツリーが階層になることは XML 要素が入れ子になることに対応します.

では, 開始タグと終了タグに挟まれた要素中のテキストはどうなるのでしょうか？

このテキストは要素の一部ですから, この要素に対応するノードに入れるべきでしょうか？ ならば複数のテキストと入れ子要素が出現する場合はどうしましょうか？

もともと xmlpgsql の最初のバージョンでは,「エレメント＝ノード」であり, エレメントが保持するテキストは, 対応するノードに入れていました. 結果として, 今の問いかけに答えられなくなりました.

「1 エレメント＝1 ノード」という考え方にはおのずから限界があります.

そこで, 単なるテキスト(実際には数字かも誰かの名前かもしれませんが)については, 特別なノードを割り当てて対応することにしました. これをテキストノードと呼びます. またタグに対応するノードをタグノードと呼ぶことにしました. テキストノードはテキスト専用のフィールドにテキストデータを保持します.

こうすると, すべてのエレメントは何らかのノードの組み合わせで表現されることになります.

空要素の場合はテキストノードを持たないタグノードとして表現されます.

さて次に問題になったのは属性です. 属性は必ず開始タグの一部として,「属性名="属性値"」として定義されます. そこでこの属性定義は開始タグを含むノードへ保持しておくことにしましょう.

ちょうどテキストノードでテキストデータを保持するのに利用したフィールドが空きますから, タグノードについてはこれを属性定義を保持するのに使うと都合が良いのです.

しかしこのことは, 実際に運用する中でちょっとした問題を引き起こしています.

というのも, XML 文書においては, すべての属性値を検索していくような処理を必要とすることがあります. そこで xmlpgsql の次期バージョンでは, 属性は別管理テーブルにしてあります.

次はタグ名について. タグ名はいちいちノードに記録しては非効率なので, タグ辞書を作ってそこへ書いておくことにします.

同様に文書名は文書バインダを作っておいて, そこへ入れておくと便利です. これで複数の XML 文書をすべて1 つのデータベースに入れておくことが可能になったのです.

8. テーブル設計

ではこれまでのことを整理して, フィールドとテーブルの設計を行います.

xmlpgsql では 3 つのテーブルを使用します.

文書バインダの役割を行う xml_document と, タグ辞書となる xml_tag, それにノードの本体となる xml_node の 3 つに加え, 属性名を保持する xml_attribute, 名前空間を表す xml_namespace, そして処理命令要素を表す xml_pi です.

```
CREATE TABLE "xml_document"(
        "id" char(16) primary key,
        "file" text unique);
CREATE TABLE "xml_tag" (
        "id" char(16) primary key,
        "name" text unique);
CREATE TABLE "xml_node" (
        "id" char(16) primary key,
        "kind" int2 not null,
        "document" char(16) not null,
        "tag" char(16) not null,
        "parent" char(16),
        "child" char(16),
        "next" char(16),
        "content" text);
```

(1) xml_document

id はデータベース中で固有の XML 文書 ID です. 主キーをつけておきます.

file はデータベース中で固有の XML 文書名です. すべてのノードはいずれかの XML 文書に属するので文書 ID に結び付けられています. これによって複数の XML 文書を 1 つのデータベースに収納することが可能になります.

(2) xml_tag

id はデータベース中で固有のタグ ID です. これに主キーをつけます.

name はデータベース中で固有のタグ名です.

タグ名は複数文書で同じものが使われていてもシステム上は支障ありません. もし明確に使い分ける必要があれば, 違うタグ名にするか, または名前空間を使って, nsi:tagname のようにすると良いでしょう. ただし現在の xmlpgsql は名前空間を認識しませんから, 「nsi:tagname」タグと「tagname」タグを同一に扱うようなことはできません. 両者は明確に違うタグとして扱われます.

(3) xml_node

id はデータベース中で固有のノード ID です. 主キーをつけます.

kind はノードの種類を表しています. 1 ならばタグノード, 2 ならばテキストノード, 3 ならばコメントノードです.

document は文書ノードが所属する XML 文書の文書 ID です.

tag は使用するタグのタグ ID です. テキストノードの場合はタグがありませんから空にします.

parent は親ノードのノード ID です. ルートノードの場合は親がいませんから NULL になります.

child は子ノードのうち 1 番目のノード ID です. テキストノードなど, 子ノードを持たないノードの場合は NULL になります.

next は兄弟ノードがあるときは次の子ノード ID です. 末っ子には次の兄弟がありませんから NULL とします.

content には可変長の文字列が入ります. タグノードの場合には, 属性リストが半角空白で分けられて入ります. テキストノードの場合にはテキストデータが入ります.

9. PostgreSQL の拡張

(1) 拡張の方法

XML 文書を扱うためのテーブルは用意できました. フィールド設計が終わったので, 次はこれらを操作するた

めの SQL 関数を整備します.

何しろ普通のリレーショナルデータベースとは違い, データはノードごとにバラバラになっていますから, データ列を取るにもひと苦労します.

PostgreSQL はその柔軟な拡張性が魅力のひとつです. ユーザ関数やユーザ型の定義, 演算子の定義などが自由にできます.

拡張コードの多くは C++ などで書かれますが, PostgreSQL には PL/pgSQL という内部組込の言語も用意されています.

それで現在の XMLPGSQL では, PL/pgSQL版と C++版, そして C 関数版という3つの実装が用意されています.

これから各拡張関数の説明を進めていきますが, プログラムリストは以下の URL にアップされています.

http://xmlpgsql.domestic.jp/archive/

ついでに, サンプル用の XML データファイルを登録する SQL ファイルも用意してありますので, それもダウンロードしておくと便利です.

では psql を起動して, プログラムリストをインクルードします. psql の詳しい使い方は割愛しますので, 各ドキュメントや参考書などを参照してください.

(2) 現在の関数

現在定義されている主な関数は以下のようなものです.

Create_New_Document(text)
新しい文書の生成
Get_Doc_ID(text)
登録した文書 ID の取得
Drop_Document(text)
指定文書をまるごと削除
Create_New_Tag(text)
要素タグの登録
Get_Tag_ID(text)
指定タグの ID を取得
Create_New_Node(integer,char(16),char(16),text)
新しい文書ノードを生成
Add_Child(char(16),char(16))
親子関係を作る. 兄弟関係は更新される
Leave_Child(char(16))
親と兄弟を切り離す
Delete_Node_Tree(char(16))
ノード以下の文書ツリーを削除する
Get_Parent_Tag_Name(char(16))
親要素のタグ名を取得
Get_Child_Contents(char(16))
要素の子となるテキストを抽出する

(3) 今後の予定

XML 文書をより標準的方法で取り扱うには, 原始的なツリー操作だけでなく, XPath によるノードリスト取得や DOM 操作のサポートが必要です.

これらは現在開発途上にあり, 間もなく発表できる段階に達するはずです.

ノードの生成や高速化については日々改善されていて, 現在数万ノードの生成が数分, ノードの検索は 10 万ノード程度であれば数秒というところです. ほとんどは PostgreSQL の基本性能に依拠しています*.

これらのより詳しい情報は Web サイト等をご覧頂きたいと思います.

10. おわりに

今回の発表にあたっては, 未踏ソフトウェア創造事業のプロジェクトマネージャである湯淺先生やをはじめ, 多くの未踏関係者のご協力を頂きました. この場を借りて感謝申し上げます.

XMLPGSQL は低コストで手軽に XML アプリケーションを構築しうる基盤ソフトウェアとして，今後も発展していくものと期待されています．

これからも，みなさんのご支援とご活用，そしてご叱責などなど，末永くお付き合いいただきたいと願っています．

11. 参考文献

[1] 『オブジェクト指向データベース』J.G.ヒューズ著／石丸知之訳／植村俊亮監訳／サイエンス社／ISBN4-78190854-3.

[2] 『PC UNIX ユーザのための PostgreSQL 完全攻略ガイド』石井達夫著／技術評論社／ISBN4-7741-0890-1.

[3] 『標準 XML 完全解説』XML/SGML サロン著／技術評論社／ISBN4-7741-0584-8.

[4] 『XML 構築ガイド』サイモン・ノース，ポール・ハーマン著／船切誠，小林聡史訳／ピアソン・エデュケーション／ISBN4-89471-220-2.

[5] 『XML Press vol.1, 2』技術評論社／ISBN4-7741-1061, 4-7741-1123-6.