PostgreSQLで作るXMLデータベース.

油井 誠 yuin@bb.din.or.jp

平成 16 年 6 月 18 日

概 要

Software Design 誌の 2004 年 5 月号の記事の元原稿に関して公開の許可を技術評論社様から戴きましたので，本文書を公開致します．SD 誌記事中のイラストや表現はオリジナルのものより洗練されていますので，そちらも是非併せて参照下さい．

http://www.gihyo.co.jp/magazines/SD/archive/200405

また，理論的な事については，こちらの論文を参照下さい．

http://www.ipsj.or.jp/members/Trans/Jpn/02/2003/4412/article003.html

1 はじめに/導入

XML はコンピュータ環境におけるデータ及び文書の表現形式の一標準として，EDI・Web サービスといった場面をはじめ，その利用が急速に進んでいます．

かつては一部の先端ユーザや研究者の興味の対象であった XML ですが，今やエンジニアにとって身近な技術となったと言えるでしょう．

今後，XML の浸透による応用の拡散により，管理すべき XML データが増えていくことが予想されます．市場動向の調査/分析でも，XML の利用の本格化を受けて，XML データベース市場の成長が指摘されています[1]．

これまで企業の持つ情報資産は多くの場合，リレーショナルデータベースを用いて管理されてきました．そうしたデータのいくつかが，XML として蓄積されるケースが出てきたわけです．

しかし，実際に XML ストレージの運用に際してみると，現在多く用いられているファイルベースの XML プロセッシングでは，大規模の XML データを処理するのに限界が生じます．大容量の XML データや，数千の XML 文書から目的の要素を DOM や SAX で抽出することを想像してみてください．

大規模 XML データを取扱う環境で，構造化された XML データに対する問い合わせ機構を持ち，大量の XML 文書を効率的に管理することができる XML データベースは不可欠な存在です．特に近年，データの入出力を XML で行なうという案件も増えています．

1.1 XML を用いた情報統合

現在，データベース技術のエンタープライズコンピューティング領域への応用テーマとして，XML をベースとした情報統合が俄かに脚光を浴びています．様々な形態で存在する情報資源に対して，XML に対する標準の照会言語である XQuery/XPath を用いて，ひとつのデータソースとして透過的にアクセスする手段を提供するといったものです．

IBM の Xperanto 技術がその代表格です．Microsoft の InfoPath も，ビジネスプロセスにおける情報統合, 情報共有を狙った技術と捉えることができるでしょう．

こうした情報統合の中でも，もっとも重要なのが，XML データとリレーショナルデータベース上のデータの情報統合です．これまで，企業の持つ情報資産の多くはトランザクション処理機能などデータの管理機能に優れた RDBMS を用いて管理されてきました．

RDBMS に格納された顧客データや販売実績など，過去のデータを見過ごすことはできません．こうした時に，RDB に格納されたデータと XML のシームレスな情報統合が求められます．

RDB の XML 拡張は，そうしたニーズに応えた技術のひとつです．RDB の XML 拡張[2]では，XML データは RDBMS のエンジンを用いて管理されます．ここで，XML データと RDB に格納したデータとの自然な情報統合が期待できます．

[1] 過去にもされていた感もありますが，製品が揃って来たこと，何よりユーザニーズが先行開発に追いついたことが，以前と異なる状況であると私は理解しています

[2] 以下，XML-RDB と呼称します

また，RDB 技術者の持つ運用ノウハウを活かせるというのも，XML-RDB を用いる利点の一つです．大抵の企業では，RDBMS の運用ノウハウや資格を持った技術者を有していることでしょう．かくいう私も新人研修として，オラクルの講習を受ける機会がありました．

XML データベースを導入して，性能問題や運用トラブルが発生した時，対処できる姿勢が整っているかというのは，見えにくいコストではありますが，重要な問題です．

1.2 XpSQL とは

さて，導入が長くなりましたが，ここからが本題です．今回私が紹介する XpSQL は，オープンソースの RDBMS である PostgreSQL をベースに XML 拡張を施した，XML データベース環境です．

XpSQL は平成 14 年度の IPA 未踏ユースソフトウェア創成事業[3]による支援を受け，開発しました．

プロジェクト発足当初は XMLPGSQL (http://xmlpgsql.domestic.jp/)[4]に対する XPath 機能のアドオンというスタンスを取っていましたが，XPath 処理系を開発する課程で，性能を出すためには XPath ストレージとしてスキーマから見直す必要がでてきました．

形態としては XMLPGSQL よりフォークという形となっていますが，現在では内部設計，インタフェースもかなり異なるものとなっています．この後の解説では，XMLPGSQL との違いについても焦点を充てていきます．

現在は，プロジェクトは PostgreSQL 関連のソフトウェアホスティングサイト GBorg で，オープンな開発を行なっています．

http://gborg.postgresql.org/project/xpsql/

詳しくは以降の文章で述べていきますが，XpSQL の特長としては，DOM Level2 Core のサブセット及び，XPath 1.0 をサポートしていること（全ての XPath Axis を利用可能），そして細粒度の更新が可能なことがあります．

[3]http://www.ipa.go.jp/archive/NBP/14nendo/14youth/

[4]本誌 2002 年 4 月号の記事「XMLPGSQL で広がる XML-DB システム」を参照のこと

なかなか離陸しなかった感もある XML データベース市場ですが，ユーザニーズの波に押されて，再び活性化してきた感があります．XML データベースが，目新しいものから，パフォーマンス要件や開発効率から必然に迫られて選択されているケースが増えているのでしょう．

---

2 XpSQL のアプローチ

前節の導入で説明したように，XpSQL はオープンソースの RDBMS である PostgreSQL をストレージバックエンドとする，PostgreSQL の XML 拡張です．

XpSQL では，XML ネイティブデータベース（NXDB）とは違い XML を独自の形式で格納することはしません．PostgreSQL のリレーショナルストレージを用いて XML データを管理し，RDBMS が提供するインデックス機構，統計情報，SQL 処理系といったものを最大限に活用するアプローチを取っています．

XML データを RDBMS に格納する方法は，大きく別けると 2 つに分類することができます．

（α）CLOB/BLOB などに XML 文章を丸ごとアトミックな値として格納する手法

（β）XML 文章を分解し，データベーススキーマに写像する手法

XpSQL では，後者の分解する手法を取っています．分解する手法を選択した主な理由としては，正規化により重複データを除くことで，全体のデータ量を抑えられること，XML 文書の部分更新機能を提供できることがあります．

PostgreSQL の contrib に付属する XML モジュールなどは，正に（α）の手法と言えます．このモジュールは Gnome の XML エンジンである libxml2(http://www.xmlsoft.org/) を PostgreSQL のユーザ定義関数を用いて DB エンジンに組み込んだものです．

DB2 XML Extender の XML Column 方式は手法（α）の改良で，サイドテーブルに検索用のインデックスを貼るというものです．このような XML 文章に対してインデックスを貼ることで検索速度の高速化を図るといったアプローチは Apache Xindice など一部の XML Native Database(NXDB) でも同様の手法が用いられています．

手法（α）の欠点としては，データの更新時に文書全体にロックを欠ける必要がある，アトミックなデータの塊ごと更新する必要があるっといったことがあります．逆に，有利な点は既に XML データがある分けですから，XPath などで文書の一部を抽出する場合に，検索結果として返す XML の再構成を比較的スムーズに行なうことができるということです．

手法（β）は，更に細かく分類すると，構造写像アプローチ，モデル写像アプローチと言われる 2 つの手法に別けることができます．

- **構造写像アプローチ**（Structure Mapping）
  XML のスキーマ情報をデータベーススキーマに反映させる手法，DTD 情報など基本的にマッピング定義を必要とするアプローチ（図 1参照）．
- **モデル写像アプローチ**（Model Mapping）
  XML の木構造[5]を静的なデータベーススキーマを用いて表現する手法．XML 木を表現する代表的な手法としては，Range と呼ばれる範囲を用いた手法，文書順を用いた手法があります（図 2参照）．

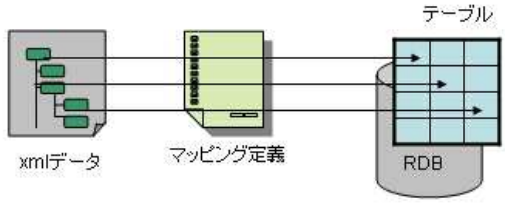

図 1: 構造写像アプローチ

DAD というマッピング定義を用いる DB2 XML Extender の XML Collection 方式などは構造写像アプローチに当てはまります．

XpSQL や XMLPGSQL は，固定したデータベーススキーマを用いる手法を取っています．XMLPGSQL では，XML の木構造を parent，child，next の 3 つのリンクによって表します（図 3参照）．XpSQL については後程詳しく説明しますが，初期設計において XMLPGSQL をベースとしたこともあり，XMLPGSQL と同様にこれらのリンク情報を保持します[6]．

固定したデータベーススキーマを用いる手法の利点としては，DTD のない整形式の XML 文書に対応できること，

[5]XML 木と呼称します
[6]将来のリリースでは後述する Dewey Order に一本化します．

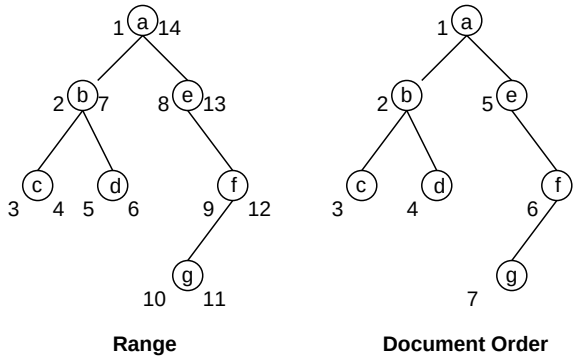

図 2: 代表的な XML 木のラベリング手法

ユーザにマッピング操作を課せる必要が無いため利用面でのハードルが低くなることがあります．

新興 NXDB である，NeoCore XMS などもこのようなアプローチを取っています．スキーマを用いたバリデーションはデータの整合性を保つ面では，必要な機能ですが同時に更新時の不可が高く付きます．

固定したスキーマを用いるアプローチでは，一般的に XML を分解する前に，既存のバリデータを用いたバリデーションが行なわれます．用途にもよりますが多くのケースでは，こうしたデータ投入前のバリデーションや，ユーザアプリケーションレベルでのデータチェックで事が足りるのではないでしょうか．

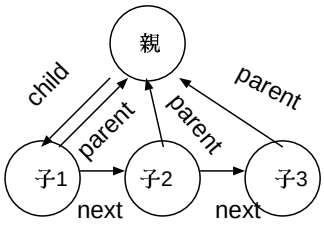

図 3: XMLPGSQL のノード管理

3 XpSQL のアーキテクチャ

XML 文書はその構造を木構造で表すことができます．XpSQL では，XML 木をノード単位に分解し，複数のリレーションを用いて管理します．

このノード指向のアプローチを採用した理由としては，ノード単位にデータを扱うことでロックの粒度や更新の影響範囲を下げることができるということがあります．

商用の製品を含めて NXDB の種類によっては，更新時にテーブルロックに相当する文書データ全体やインデックスファイルをファイル単位でロックする事が必要となるものが見られます．ロックの粒度は，マルチユーザ環境では得に重要になってきますので，XpSQL に限らず，XML データベースを選択する際は抑えておきたいポイントです．

リレーショナル・モデルを使用して木構造を
表現するアイデアを最初に一般に紹介したのは，
Joe Celko 氏の著作,SQL for Smarties:
Advanced SQL Programming (Morgan Kayfmann
Publishers,Inc / 1995) ISBN:1558605762 です．
この著作では，Range を用いた木のラベリング
手法が提案されています．
XML-RDB に限らず，XML データベース一般に，
XML 木に対するラベリング方法は，性能面での
肝となる重要なテーマです．

3.1 XPath データモデル

XpSQL において，永続化管理する XML データは XPath 1.0 データモデル[7] に沿ったものとなっています．

XPath とは簡単にいうと，XML の木構造を辿り要素のアドレッシングを行なう仕様です．Unix のファイルシステムでは，/usr/local//src といったディレクトリの指定ができますが，それと似たイメージを持って戴ければ分り易いかもしれません．

XPath 1.0 データモデルでは，ツリーを構成するノードが 7 種類定義されています．それぞれ，XpSQL におけるノードの種別との対応状況を以下に示します．

- **ルートノード** kind 番号:0
  ルートノードとは文書要素（トップレベルの要素）の仮想的な親です．
- **要素ノード** kind 番号:1
- **テキストノード** kind 番号:2
  開始タグと終了タグで挟まれた文字列データ．
  XPath データモデルでは，展開されたエンティティ情報は存在しません．これらは単に文字列として扱われます．
- **コメントノード** kind 番号:3
- **処理命令ノード** kind 番号:4
- **属性ノード** kind 番号:5
- **名前空間ノード** kind 番号:6
  属性ノード，名前空間ノードは要素ノードを親としてもつが，要素の子ではない，と表現されています．これらは要素ノードのプロパティのしての位置付けであり，従属ノードとも表現されます[8]．

XMLPGSQL は DOM データベースではありますが，将来の XPath への対応を見込んでノード種別は XPath データモデルに従っています．従って，XpSQL においてもノード種別については，XMLPGSQL の扱い方をそのまま継承しています．

XPath データモデルでは，XML 木を辿る基本的な順序を文書順（Document Order）として規定しています．文書順とは読んで字のごとく，XML 文書内でのノードの出現順のことです（図 2参照）．XPath を処理するに当っては，要素間の順序情報が不可欠です．

XMLPGSQL ではノード間の関係は，リンクを用いて管理されてきました（図 3参照）．これは XMLPGSQL が基本的に DOM 操作に特化している DOM データベース[9]であるからです．

XpSQL が XPath によるアドレッシングを提供するに当って，XMLPGSQL が提供しているリンク情報だけではリンクを巡航的（Navigational）に辿る必要が出て来る為，パフォーマンスが出せないという問題がありました．

/dogs/dog[101] といった XPath 式の評価や Descendant(子孫) ノードをリンク情報で求めるケースを想像して貰うと分るかもしれませんが，リレーションで処理する都合，ノードテーブルの等結合（Self-Join）の数がふくれ上がり，性能が期待できません．

そこで，XpSQL では XMLPGSQL のスキーマに対して，次の 2 つの情報を追加することで，XPath の検索処理における性能問題を解決を図りました．

- Dewey Order を用いたノードのラベル情報
- ルートからのパス情報

ここで，Dewey Order とは Dewey Decimal Classification(DDC) に基づいた木の順序付け手

[7] XPath Version 1.0 Spec - http://www.w3.org/TR/xpath

[8] 技術評論社 改訂版 XML 完全解説（下）ISBN4-7741-1302-6 C3055

[9] しばしば，これらは永続化 DOM(PersistentDOM, PDOM) と呼ばれます．

法です．DDC は従来，主に図書分類に用いられてきた分類法です．

XpSQL では，この Dewey Order を XML 木に対して順序情報のラベリングを行なうのに用います．Dewey Order を用いたラベリングでは，各ノードのラベルは親のラベルの後ろに兄弟間の順序を足したものとなっています (図 4参照)．

Dewey Order の特徴としては，Range など他の手法 (図 2参照) と比較して XPath による検索時の性能とデータ更新時の性能のバランスがよいことが知られています．

図 5は，DOM の InsertBefore 操作によるノードの追加を行なった場合に順序木の再編性が必要となる，更新の影響範囲を示しています．Dewey Order ではノードを追加をした際にラベルの張り替えが必要となる影響範囲は，挿入するノードの After-Sibling(弟ノード) とその子孫ノードです．

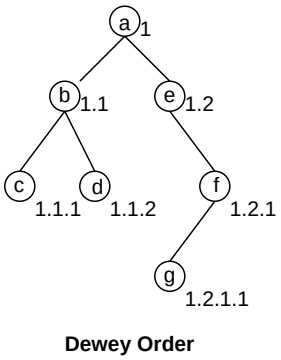

図 4: Dewey Order を用いたノードのラベリング

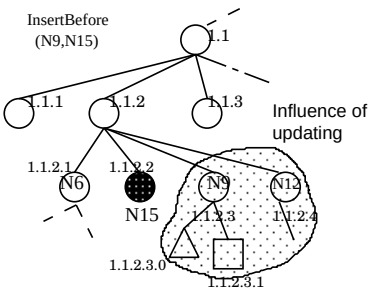

図 5: Dewey Order における更新の影響範囲

図 6に先に挙げた文書順と Range による場合を載せましたので比較してみてください．

eXist などは順序付けこそ文書順によるシンプルなものですが，ノードの採番に空き (余裕) を持たせることで，将来来たるいくつかのノードの追加に備えるという工夫を取っています．このアプローチは，特定の位置で子要素を足す操作をくり返した場合などには弱いですが，通常いくらかの効果が見込めます．

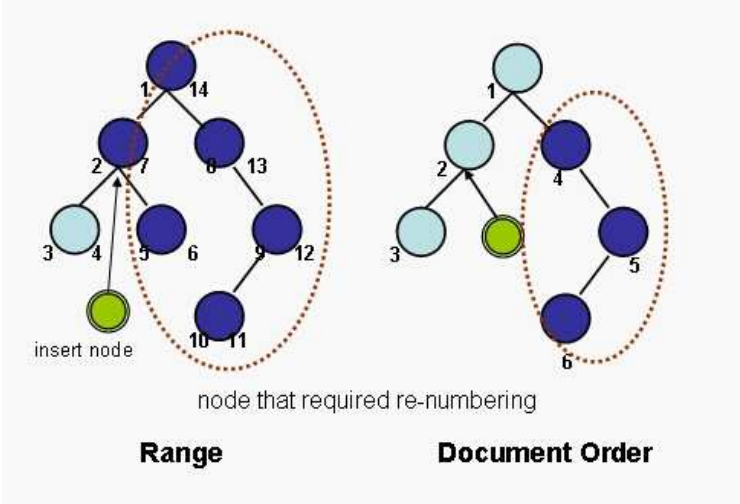

図 6: 他の順序付け手法の更新影響範囲

3.2 XML データのリレーションへの写像

ここでは，サンプル XML 文書を用いて，実際にどのように XML データがリレーションに写像 (格納) されるか見て行きましょう．ここでは，図 8の XML 文書をサンプルとして例にとって，説明していきます．

まず，XpSQL のデータベーススキーマは図 7に示すように 7 つのテーブルからなります．

それぞれのテーブルの意図する所は次のようになります．

- xml_document テーブル
  XML の文書情報を管理する．
- xml_node テーブル
  XML のノード情報を管理する．
- xml_path テーブル
  ルートからのパス情報を管理する．
- 辞書テーブル
  ノードテーブルの外部キー tagid から参照されるテーブル群．

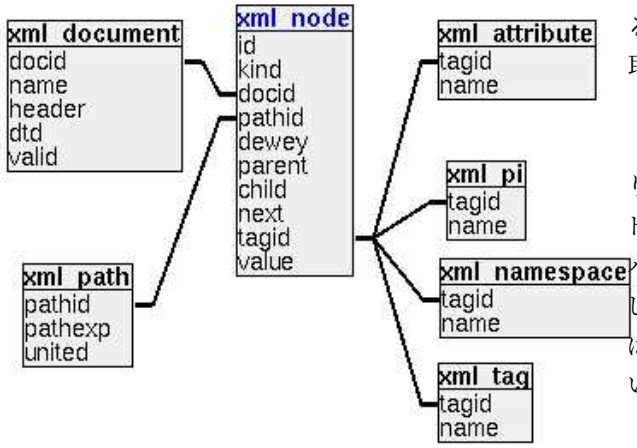

図 7: XpSQL のデータベーススキーマ

- xml_tag テーブル
  エレメント名を一意に管理する.
- xml_att テーブル
  属性名を一意に管理する.
- xml_namespace テーブル
  名前空間名を一意に管理する.
- xml_pi テーブル
  Processing Instruction を一意に管理する.

XpSQL において，XML データはデータベースへのロード時に図 9のようにモデル化され，格納されます.

スペースの都合もあり，細かくひとつひとつ説明していくことはできませんが，図 9のモデルと実際の関係表の表現 (図 10) とを照らし合わせて戴くと，その仕組みを掴んで戴けるものと思います.

ここでは，１つ例を挙げて，サンプル XML 文書中の article ノードについて見ていきましょう. article ノードは図 9において要素ノードの 4 番に当ります.

このノード情報の xml_node テーブルへのエントリの意味は，以下のようなものになります.

id	4	ノードのユニーク識別子
kind	1	ノードの種別が要素ノードである
docid	1	sample.xml に所属するノードである
pathid	4	ノードが表すパス情報のパス表における識別子
dewey	0.1.1.1	Dewey Order による順序情報を表現する
parent	3	親ノードのノード ID を表す
child	6	第一子 (FirstChild) のノード ID を表す
next	NULL	弟ノードは存在しない
tagid	2	要素名はタグ表の tagid が 2 に対応する値
value	NULL	要素ノードでは不必要 (常に NULL)

XMLPGSQL と違うのは，pathid 列と，dewey 列を有する事です. その他にも細かい点で，ノードの識別子など取扱いを改良しました.

XMLPGSQL では各ノードの識別子は，SQL の CHAR(16)，つまり 16 文字の文字列を使って表現されていました. その為，ノードの識別子に PostgreSQL における CHAR(16) 分，16+4(20) バイト空間を消費していました. これでは，格納サイズの増加しディスクリードが増えて検索にも影響が出る為，XpSQL ではノードの識別子は INTEGER 型，つまり 4 バイトの整数を用いて管理しています.

```
<?xml version="1.0"?>
<book>
    <article name="Introduction of XpSQL.">
        <author>
            <name>Makoto Yui</name>
            <email>yuin@bb.din.or.jp</email>
        </author>
        <author>anonymous-chan</author>
        <date>2003.03.12</date>
    </article>
</book>
```

図 8: XML 文書のサンプル

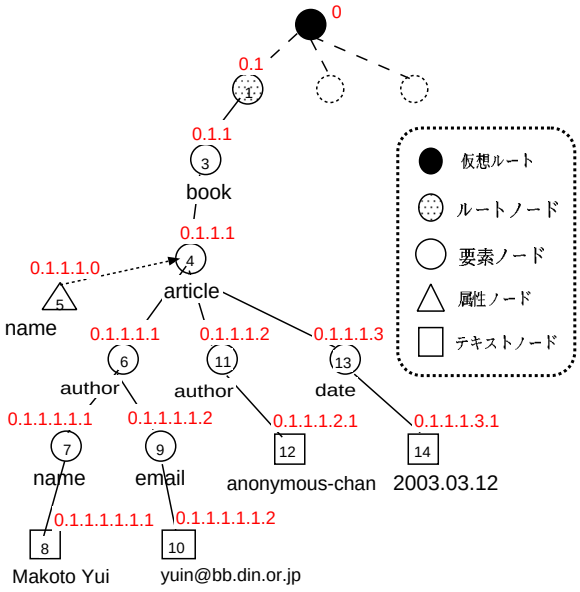

図 9: 図 8の木構造を Dewey Order を用いてモデリングしたもの

xml_node テーブル

id	kind	docid	pathid	dewey	parent	child	next	tagid	value
1	0	1	0	{0,1}		3			
3	1	1	3	{0,1,1}	1	4		1	
4	1	1	4	{0,1,1,1}	3	6		2	
5	5	1	5	{0,1,1,1,0}	4			1	Introduction of XpSQL.
6	1	1	6	{0,1,1,1,1}	4	7	11	3	
7	1	1	7	{0,1,1,1,1,1}	6	8	9	4	
8	2	1	8	{0,1,1,1,1,1,1}	7				Makoto Yui
9	1	1	9	{0,1,1,1,1,2}	6	10		5	
10	2	1	10	{0,1,1,1,1,2,1}	9				yuin@bb.din.or.jp
11	1	1	11	{0,1,1,1,2}	4	12	13	3	
12	2	1	12	{0,1,1,1,2,1}	11				anonymous-chan
13	1	1	13	{0,1,1,1,3}	4	14		6	
14	2	1	14	{0,1,1,1,3,1}	13				2003.03.12

xml_path テーブル

pathid	pathexp
0	
3	#/book
4	#/book#/article
5	#/book#/article#/@name
6	#/book#/article#/author
7	#/book#/article#/author#/name
8	#/book#/article#/author#/name#/text()
9	#/book#/article#/author#/email
10	#/book#/article#/author#/email#/text()
11	#/book#/article#/author
12	#/book#/article#/author#/text()
13	#/book#/article#/date
14	#/book#/article#/date#/text()

xml_document テーブル

docid	name
1	sample.xml

xml_node の tagid フィールドに対応する実体を保持する辞書テーブル.

xml_tag テーブル

tagid	name
1	book
2	article
3	author
4	name
5	email
6	date

xml_attribute テーブル

tagid	name
1	name

図 10: 図 8の XML 文書を関係表に格納したもの

3.3 XpSQL を構成するモジュール群

XpSQL は，図 11のシステム構成図から見て取れるように，基本的に 3 つのモジュール (ユーザ定義関数) からなります.

ここでいう 3 つのモジュールの役目は次の通りです.

- DOM **操作関数** DOM Level2 Core に準拠した API を提供する.
- XPath **処理関数** XPath 1.0 による問い合わせ機能を提供する.
- **更新処理用関数** 木構造を表現する木ラベルの更新を行なうトリガ.

更新処理用関数は，データの更新時にトリガによって呼ばれるものです．木ラベルの更新をトリガによって行なうことにした理由としては，専用の拡張 API だけではなく標準 SQL からの更新に対応すること，初期設計時に XMLPGSQL の DOM 操作との互換性を維持しようと考えたことに起因しています[10].

ユーザが主に扱うのは，DOM 操作関数と，XPath 処理関数となります．DOM 操作関数，XPath 操作関数については，この後詳しく説明していきます.

3.4 XPath 式から SQL への変換

XPath 式から SQL への変換は XPath 処理関数の中で行なわれます．変換のアルゴリズムは，ここに示すにはスペースも足りませんので，いくつか簡単な例を挙げてイメージを掴んで戴くこととします.

より詳しいアルゴリズムを知りたい方は，大学時代の指導教官と共同執筆したものが論文誌に載っていますので，そち

[10] トリガによる更新は少からず更新時性能に影響します．現在開発中の製品では，部分更新は DOM 関数等の専用 API による操作のみと割り切っています.

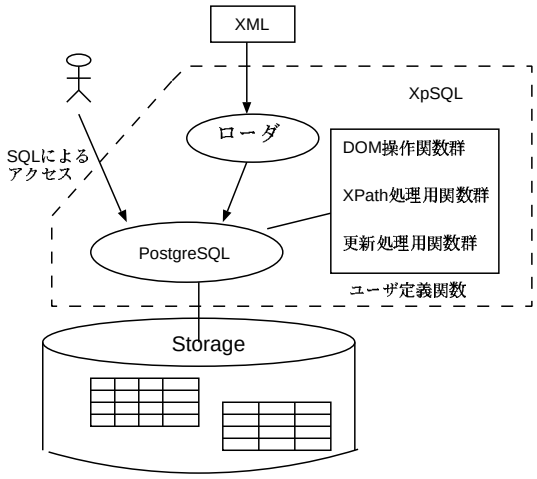

図 11: XpSQL のアーキテクチャ

らを参考にしてください[11].

例 1 XPath 式/book//author から変換された SQL 問合せ

```
SELECT DISTINCT xn3.docid,xn3.id
FROM xml_path xp3,xml_node xn3
WHERE
xp3.pathexp LIKE '#/book#%/author'(a)
AND xn3.pathid = xp3.pathid
```

挿入

このようなシンプルな XPath 式では，パス情報を用いた選択が行なわれます．パス情報によるフィルタリングはルートからのパスマッチにのみ使います． このように前方一致探索しか試みない理由としては，(b-tree) インデックスによるインデックスアクセスを行なう場合の制限に由来します．

生成された SQL 式を見てみますと，パスによる選択 (a) が行なわれることで結合演算の数が抑えられていることが分ります．パスを用いたフィルタリングはパスの階層が深い問い合わせに得に有効です．[12]

例 2 XPath 式

```
/book/article/following-sibling::node()
```

から変換された SQL 問合せ

```
SELECT DISTINCT xn4.docid,xn4.id
FROM xml_path xp3,xml_node xn3,xml_node xn4
WHERE
xp3.pathexp = '#/book#/article#/author'(a)
AND xn3.pathid = xp3.pathid
AND xn4.parent = xn3.parent
AND xn4.dewey > xn3.dewey(b) AND xn4.kind < 5
```

この問い合わせは，図 8の XML 文書から，図 9の 11 番と 13 番地の要素，つまり

- <author>anonymous-chan</author>
- <date>2003.03.12</date>

を選択する，XPath 問い合わせです．following-sibling は，同じ親を持つ兄弟ノードで自分より文書順で後にある要素を特定する XPath Axis（軸）です．

このようにパスだけで選択演算が行なえない場合には，Dewey Order による選択が行なわれます．この例では，/book/article/author まではパスによる選択 (a) が行なわれ，その後，順序情報を用いた選択 (b) が追加されます[13].

4 XpSQL を利用してみよう

さて，概要を理解したところで，実際に XpSQL を動かしてみましょう．まずは，セットアップです．

4.1 インストール

ここでは，XpSQL のインストール手順を説明します．既に説明したように XpSQL はストレージバックエンドとして，PostgreSQL を利用しますので予め PostgreSQL をインストールしておいてください．PostgreSQL のインストール手順は Web 上にもリソースはあることですし，ここでは，省略します．尚，サーバサイドの開発ライブラリを用いることもあり，PostgreSQL はソースからインストールをすることを前提にして話を進めます．

XpSQL に必要な環境，ライブラリは以下になります[14].

[11] IPSJ TOD Vol.44 No.SIG12 です．

[12] 深い階層への問い合わせは他の XML-RDB 製品では，弱点となることがあります．

[13] 実際は,DBMS のオプティマイザによって，統計情報やアクセスコストを考慮した最適なアクセスパスが選択されます．ここで挙げた評価順は説明する都合上のものです．

[14] この記事を執筆時は，Redhat Linux 9, PostgreSQL 7.4.2(及び 7.3.6) で動作確認を取りました．

PostgreSQL	Version 7.4 以上 (7.3 も可能)
GLib2	2.2.1 で動作確認済み
pkgconfig	Version 0.14 にて動作確認済み
Expat	Version 1.95 で動作確認済み
GCC	Version 3.2 以上を推奨

まず，はじめに PostgreSQL のヘッダーファイルをインストールします．PostgreSQL の通常のインストール作業 (make install) では，インストールされませんので必ず実行する必要があります．

ヘッダファイルのインストールには，PostgreSQL のソースディレクトリで，make install-all-headers を実行して下さい．

次に，XpSQL の最新のパッケージを Gborg のプロジェクトサイト[15] から取得してください．現在の最新バージョンは 0.9.2b で，xpsql-0.9.2b.tar.gz というファイルです．ここからの作業は，話を簡潔にする為に root ユーザとして作業をすることを前提とします．

1. データベース操作ユーザの追加.
   useradd コマンドで postgres という DB アクセスユーザを追加してください.
   $ useradd postgres

2. ファイルを解凍.
   $ tar xzvf xpsql-0.9.2b.tar.gz
   解凍が，終ったら展開されてできたディレクトリ XpSQL に移動してください．[16]

3. コアライブラリのコンパイルとインストール.
   $ make clean; make [17]
   $ make install

4. XpSQL 用のデータベース領域の作成.
   $ su - postgres -c
   "/usr/local/pgsql/bin/createdb
   --echo yourdbname" [18]

5. intarray モジュールのインストール.
   XpSQL は，PostgreSQL の contrib の intarray というモジュールを改造して用います．導入手順はやや複雑ですが，以下のステップとなります.

   - intarray ディレクトリへ移動.
     ここには，以下の 2 つのディレクトリが在ります.
     $ ls
     7.3 7.4
     ここで，自分の環境にあった PostgreSQL のバージョンに移動してください.
   - intarray モジュールのインストール
     ディレクトリを移動したら，そこで make.sh スクリプトを叩くことで，インストール作業が開始されます.
     $ ./make.sh

     スクリプトが起動されると，まず PostgreSQL のソースディレクトリへのパスを入力するよう求められますので，環境に合わせたパスを絶対パスで入力してください.

     ----- Input your postgres src dirname -----
     please input > /usr/local/src/postgresql-7.4.2

     次に，データベース名が尋ねられますので，先程作成したデータベース名を入力してください.

     ----- input database name in roman -----
     please input > yourdbname

6. データベース領域の初期化
   まず，$XPSQL_HOME/sql ディレクトリに移動してください．ここで，シェルスクリプト init.sh を実行します.
   $ ./init.sh

   スクリプトが起動されると，まずデータベース名の入力が求められますので先程と入力したデータベース名と同じものを入力してください.

   input database name in roman
   yourdbname

7. XPath 機能のセットアップ
   xpath ディレクトリに移動し，XPath 関数のセットアップを行ないます．[19]

   $ cd $XPSQL_HOME/xpath
   $ make clean; make

[15] http://gborg.postgresql.org/project/xpsql/
[16] 以下の記事中では，このディレクトリを$XPSQL_HOME として表します.
[17] ここで，Makefile は/usr/local/pgsql に PostgreSQL をインストールした前提となっています．もし，バイナリからインストールした場合などインストール先が異なる場合は，環境に合わせて編集してください.
[18] 実際には一行です
[19] この作業は XPath 機能を用いない場合は不用です.

```
$ make up
$ ./create.sh
```

ここでも，input database name in roman という表示が出ますので，データベース名を入力してください．

8．バルクローダのセットアップ

XpSQL は libpq クライアントとして pgxload というバルクローダ（XML をパースし，データベースにデータを投入するプログラム）を提供しています．$XPSQL_HOME/pgxload ディレクトリがバルクローダの配置場所です．

バルクローダのセットアップには．pgxload ディレクトリに移動し，make コマンドを実行します．

```
$ cd $XPSQL_HOME/pgxload
$ make
```

make を実行するとまず，PostgreSQL のヘッダーファイルがあるディレクトリのパスが尋ねられますので，環境に合った適切なパスを入力してください．ソースからインストールした場合，通常は /usr/local/pgsql/include が該当ディレクトリになります．

```
Please input postgres header directory
default setting is../usr/local/pgsql/include
```

次に，PostgreSQL のライブラリのパスの入力が求められます．

```
Please input postgres library directory
default setting is../usr/local/pgsql/lib
```

4.2 Bulkload

XpSQL において，XML データをデータベースに投入する方法には，DOM API を用いて XML 文書を構築する方法と，バルクローダを用いる方法の 2 種類があります．ここでは，XML 文書の一括ロードを行なうバルクローダの使い方について説明します．

XpSQL のバルクローダのセットアップ方法は前節で説明した通りです．準備ができましたら，実際に使ってみましょう．[20]

[20]バルクロード時は，可能ならば PostgreSQL の Fsync オプションはオフにして下さい．

pgxload ディレクトリに，xloader.sh というシェルスクリプトが付属していますのでそれを用います．対話式のスクリプトになっていますので，指示に従うことでデータが投入されます．尚，ロードする XML ファイルが日本語を含む文書の場合，UTF-8 でエンコーディングされている必要があります．SAX 準拠のパーザライブラリ Expat が，Shift_JIS 及び EUC-JP を受けつけない為です．

Unix には，iconv という便利な文字コード変換ツールがありますので，これを使ってエンコーディング変換しておくとよいでしょう．iconv の基本的な使いかたは，以下の通りです．詳しくは iconv のマニュアルを参照してください．

```
iconv -f EUC-JP -t UTF-8 -o output.xml source.xml
```

バルクロードの流れは次の通りです．

1．ローダを起動する．

```
$ ./xloader.sh
```

2．ロードする XML ファイルのあるディレクトリか，ファイル名を指定する．

```
input directory name or file name
../data
~~~~~~~
(ここにディレクトリかファイル名を指定します．)

 list of loading files
-----------------------
4.0K axis.xml 4.0K format.xml 4.0K quote.xml
```

指定したファイルが見つかった場合，ロードするファイルのリストと，そのサイズが表示されます．

3．データを投入するデータベース名を指定する．

```
input database name in roman
yourdbname（ここでデータベース名を入力する）
~~~~~~~~~~
```

4．データベースのユーザ名を入力する．

```
input database username
postgres
~~~~~~~~
```

5．データベースユーザのパスワードを入力する．

```
input the user password
yourpassword
~~~~~~~~~~~~
```

6．空白や改行を無視するか?
次にこのようなプロンプトが出て選択をうながされます.

```
enables strip_space ?
1) true
2) false
```

XSLT などの開発に携わったことがある方はピンと来るかもしれませんが，この strip_space は次のことを意味します.

例えば，このような XML データを投入すると仮定しましょう.

```
<aaa>
    <bbb/>
    <ccc> zzz</ccc>
</aaa>
```

スペースを認識するモードの場合，この XML データは Expat によってパースされ，次のような SAX イベントになります.

```
element "aaa"
textnode "\n\t"
element "bbb"
textnode "\n\t"
element "ccc"
textnode " zzz"
textnode "\n "
```

strip_space をオンにした場合は次のようになります.

```
element "aaa"
element "bbb"
element "ccc"
textnode " zzz"
```

```
     aaa
    /  |
 bbb   ccc
       |
      _zzz
```

つまり，スペースや改行を無視して，このような XML データとして，データが格納されます.

```
<aaa><bbb/><ccc> zzz</ccc></aaa>
```

strip_space をオンにすることで，余分なノードの生成を抑えることができますので，処理効率を高めることがあります．ただし，戻される XML データは厳密には異なりますので，スペースや改行も忠実に再現したい場合には，このオプションはオフにしてください.

7．DB のエンコーディングを指定する.
まずスクリプト側で DB の encoding が自動的にチェックされます．そこで取得されたエンコード方式が iconv のリストにない場合，入力が求められます.

```
checking your db encoding ?
EUC_JP

Please input your DB encoding in more normal name.
Ex). not EUC_JP postgres but EUC-JP general.
EUC-JP
~~~~~~
```

ここでは，EUC-JP の文字コードの識別子が PostgreSQL 側（EUC_JP）と iconv(EUC-JP）で違うので，入力が求められています．iconv -l で，変換リストを確認し，iconv の変換コードを入力してください.

このあと，バルクロードが始まります．私のマシン（Thinkpad X23）ですと，1MB の文書で 4 分程度ロードに時間がかかりましたので，目安としてください．数キロバイト程度なら数十秒です.

4.3 XPath による検索

XpSQL は PostgreSQL の拡張機能であるユーザ定義関数として XPath 処理機能を提供します．XPath 処理関数には，次の 4 つがあります.

1．xpath_eval(VARHCAR)
引数に与えられた XPath 式を評価し，XPath 式によって選択されたノードの docid とノード id をリレーションとして返します.

```
SELECT docid, id
    FROM xpath_eval('/article//author[3]')
```

```
 docid | id
-------+-----
     1 | 282
```

検索対象とする XML 文書を絞って問い合わせる (例えば，sd.xml のみを対象とする) には，検索式に次のような構文を用いてください．

```
document("sd.xml")/article//author[3]
```

尚，文書の絞り込みには，SQL の SIMILER TO 構文と同等のフィルタリング指定が利用できます．

2．xpath2sql(VARCHAR)
引数に与えられた XPath 式を SQL に変換します．xpath_eval 関数などは xpath2sql によって変換された SQL を内部で実行します．

XPath 式がどのような SQL 文に変換するか確かめる時，クエリが静的な場合など手作業で SQL を最適化したい場合はこの関数を用いるとよいでしょう．

XPath2SQL によって生成された SQL 文を解析し，カスタマイズすることで，既存の外部テーブルのデータを参照し XPath による問い合わせ結果と組み合わせるといった使い方も面白いでしょう．また SQRT() や ATAN() といった高度な算術処理をユーザ定義関数を用いて定義し，それを用いてノード値をフィルタリングすることも可能です．

3．toXML(INTEGER)
この関数はノード Id(xml_node テーブルの id 列の値) を引数と取り，XML の再構成を行なう関数です．

```
SELECT toXML(id)
   FROM xml_eval('/article//author[3]')
```

といった具合に利用します．

4．xpath_evalx(VARCHAR)
与えられた XPath 式を評価し，XML 形式で結果を返します．正確には docid と再構成された XML 文字列の二項リレーションを返します．

XPath による実行結果を SELECT 句の toXML 関数によって計算する (3) よりも，効率よく，XML の再構成を行なうことができます．

```
SELECT *
   FROM xml_evalx('/article//author[3]')
```

```
 docid |                result
-------+----------------------------------------
     1 | <author><name>tokumeikibou</name></author>
```

4.3.1 XPath の拡張構文

XpSQL では，XPath の仕様を拡張した問い合わせをサポートしています．ここでは，その主要な 2 種類の拡張について説明します．

- String-Value を用いないノードのフィルタリング

XPath の仕様[21]で，要素ノード，ルートノードに関して文字列値 (String-Value) は，
子孫テキストノードの文字列値を文書順に連結したもの
と定義されています．

例えば，下記の XML 文書における sec 要素の文字列値は"aaabbbccc"です．これは，XPath 問い合わせ /top[sec="aaabbbccc"]/sec でマッチする sec 要素が存在するということになります．

```
<top>
    <sec>aaa<third>bbb</third>ccc</sec>
</top>
```

今回は簡単な例なのでまだよいのですが，このように子孫テキストノードの文字列値を文書順に連結するコストは，一般的に高くなります[22]．

そこで，XpSQL では XPath Predicate 部 (条件を表す) における文字列値の比較 ([sec="aaabbbccc"] の部分) に対して，変わりとなる構文を用意し，代替手法として推奨しています．

この代替手法のオペレータは，==です．否定は!==です．

```
/top[sec=="aaa"]
```

この例では，子要素 sec が更にその子ノードとして aaa というテキストノードを有する，/top エレメントを選択します．/top[sec=="ccc"] についても同様にヒットすることとなります．

テキストノード値の指定部に%(パーセント) または，*(アスタリスク) が存在する場合，
例えば，Predicate 部の表現が [sec=="%ccc"] といったケースでは，.. LIKE '%ccc' といった具合に LIKE 式を用いたパターンマッチが行なわれます．

[21] http://www.w3.org/TR/xpath#dt-string-value

[22] XML データベースの中には，文字列値を厳格に扱っていないものもあります．オープンソース XML データベースの eXist でこの問い合わせを評価したところ，no match という結果になってしまいました．

このように，XpSQLでは，文字列値ではなく子テキストノード値によるフィルタリングを行なうXPath拡張構文を提供しています[23].

- **正規表現を用いたフィルタリングのサポート**

  XpSQL では，XPath Predicate 部において，POSIX 正規表現を用いた高度なノードのフィルタリングを行なうことが可能です．利用には，/top[sec=~"^a*"] のように，オペレータとして =~（イコール-チルダ）を用います．

  このオペレータを用いた問い合わせでは，(/top/sec) 要素の文字列値に対して正規表現によるフィルタリングが行なわれます．
  文字列値の計算にはコストがかかりますので，選択数が大きい場合はこの関数は使わずに，先に挙げた構文（/top[sec=="a*"] など）を用いてください．

4.4 DOM APIの利用

XML の更新用の言語としては，XML:DB Initiative の XUpdate(http://www.xmldb.org/xupdate/) があります．しかし，有力ベンダの参加はなくデファクトとしての地位は築けていません．また，XQuery には更新用の機能は現時点で存在しません．

XpSQLでは，部分更新用途に標準的な XML 操作 API としてユーザにも馴染みの深い DOM を使います．

DOM 準拠の API は XMLPGSQL にもありましたが，標準とはやや異なった実装となっていました．DOM API はオブジェクト指向言語でない，手続型の言語や SQL で表現するには向かない部分もありますが，標準に沿うことは DOM に慣れた技術者に対するアピールといった面でも重要です．

そこで XpSQL では，DOM L2 Core にそった API を新規に作成しました．XpSQL は基本的に XPath 1.0 データモデルに沿った XPath データベースですので，全ての DOM の機能を提供できるわけではありませんが，主要な API はカバーしています．

DOM API は将来のポータビリティを考え，コアとなる部分を除き PL/PGSQL で実装しました．まだこの機能はまだベータ版という位置付けですが，現在 DOM L2 Core の 6 割程度と基本的な木の Traverse 操作をカバーしています．詳細は$XPSQL_HOME/sql/dom_l2.sql をご覧戴くとして，ここでは主要な関数を説明します．

XpSQL が提供する主要な DOM 操作関数

- createDocumentFragment(Document INTGER)::INTEGER
  空の Document Fragment ノードを生成し，文書ルードの識別子を返す．

- createElement(Document INTEGER, tagName VARCHAR)::INTEGER
  エレメントを生成し，そのノード識別子を返す．

- createAttribute(Document INTEGER, attname VARCHAR)::INTEGER
  属性を生成し，そのノード識別子を返す．

- createComment(Document INTEGER, Data VARCHAR)::INTEGER
  コメントノードを作成し，そのノード識別子を返す．

- createProcessingInstruction(Document INTEGER, Target VARCHAR, Data VARCHAR)::INTEGER
  処理命令ノードを作成し，そのノード識別子を返す．

- createTextNode(Document INTEGER, Data VARCHAR)::INTEGER
  テキストノードを作成し，そのノード識別子を返す．

- hasAttributes(Node INTEGER)::BOOLEAN
  指定されたノードが属性を持つかを真偽値で返す．

- SetAttribute(Node INTEGER,Name VARCHAR,Value VARCHAR)::INTEGER
  Node 要素に属性をセットし，属性ノードの識別子を返す．属性名は Name，属性値は Value に指定する．

- appendChild(Parent INTEGER, Node INTEGER)::BOOLEAN
  親ノード（Parent）下にノード（Node）を子ノードとして付ける．

[23] 文字列値を想定した XPath 問い合わせより，ユーザにとってはこのような子テキストノード値によってフィルタリングを行なう方が直感的ではないでしょうか．

- getElementsByTagName(Element INTEGER, tagName VARCHAR)
  Element 要素の下の子孫の要素で，タグ名が tagName であるものを抽出する．返されるリストの順序は文書順となる．

- appendData(CharacterDataNode INTEGER, value VARCHAR)::BOOLEAN
  CharacterDataNode がテキストノードならば末尾に value を連結し，成功かどうか真偽値を返す．

- getFirstChild(Node INTEGER)::INTEGER
  指定された要素の第一子を返す．

- getNextSibling(Node INTEGER)::INTEGER
  このノードの次の兄弟ノードの識別子を返す．

DOM 関数の利用イメージを以下に示します．

```
# XML 文書の断片を作成する．ルートノードのノード識別子が帰る．
postgres=# select createDocumentFragment('hello.xml');
 createdocumentfragment
------------------------
                  69329

# Document コンテキスト内に，エレメントを作成する．
postgres=# select createElement(69329, 'first');
 createelement
---------------
         69331

# Document コンテキスト内に，テキストノードを作成する．
postgres=# select createTextNode(69329, 'textvalue');
 createtextnode
----------------
          69332

# ルートノードに，既に作成した first エレメントを付ける．

postgres=# select appendChild(69329, 69331);
 appendchild
-------------
 t

# first エレメントに，既に作成したテキストノードを付ける．
postgres=# select appendChild(69331, 69332);
 appendchild
-------------
 t

# 生成した DOM ツリーを文書ルートから XML にシリアライズする．
postgres=# select toxml(69329);
          toxml
--------------------------
 <first>textvalue</first>
```

DOM 操作関数は，要素を追加するにも基本的に CreateElement した後に，appendChild をして DOM 木に足すという手順を踏む必要があります．そこで XpSQL では，利便性向上の為に libxml2 の Tree 操作操作関数に見られるような直接，木にノードを足すような関数を拡張機能として提供しています．[24]

XpSQL が提供する拡張 Tree 操作関数で，主要なものを以下に挙げます．

XpSQL が提供する拡張 Tree 操作関数

- addChildElement(Node INTEGER,tagName VARCHAR)::INTEGER
  指定したノードに子要素を追加し，その識別子を返す．tagName が要素名となる．

- addChildPI(Node INTEGER,Target VARCHAR,Data VARCHAR)::INTEGER
  指定したノードの子要素として処理命令ノードを作成し，そのノード識別子を返す．

- addChildText(Node INTEGER,Data VARCHAR)::INTEGER
  指定したノードの子要素として，テキストノードを作成し，そのノード識別子を返す．

- addChildComment(Node INTEGER,Data VARCHAR)::INTEGER
  コメントノードを作成し，そのノード識別子を返す．

使い方の一例を，以下に示します．ここでは，DOM 操作関数を用いた場合と同じ DOM ツリーを作ることとします．先に挙げた DOM 関数の場合と比較してみると違いが見えてくるでしょう．

```
# ルートノード（識別子 69333 と仮定）に子要素を追加する．

postgres=# select addChildElement(69333, 'first');
 addchildelement
-----------------
           69335

# 作成されたノードにテキストノードを追加する．
postgres=# select addChildText(69335,'textvalue');
 addchildtext
--------------
         69336
```

[24] XMLPGSQL で DOM 準拠関数と言われていたものに相当します．

4.5 XpSQL の特徴と応用

4.5.1 問い合わせ結果の取扱い

XpSQL の特徴の一つに，既存のデータベースアクセス API を用いてアプリケーションの実装が行なえるということがあります．

XML-RDB である XpSQL では，XPath による問い合わせ結果は ResultSet で返されます．JDBC などの既存の API を用いてアプリケーション側でデータを取扱うことができます．

```
<family>
    <name>イチロウ</name><name>ジロウ</name><name>
サブロウ</name>
</family>
```

という，XML 文書に対して/family/name/text() という XPath 検索で結果を受け取る例を見てみましょう．

XPath データベースにこのような問い合わせをすると，結果の返り方はまちまちです．イチロウジロウサブロウというテキストが返されることもありますし，結果が

```
<row>イ チ ロ ウ<row><row>ジ ロ ウ</row><row>サ ブ ロ
ウ</row>
```

といった具合にタグ付けられて返るものもあります．これでは，アプリケーション側でも結果をパースして加工することになってしまうので二度手間です．[25]

次世代 XML 問い合わせ言語の標準である XQuery の場合ではどうするかというと，次のような手順を踏むこととなります．

XQuery の FLWR(For Let While Return) 式の実行例

- 問い合わせの例

```
<resut>
{
  for $name in
    document("data/family.xml")/family/name
  let $content := $name/text()
  return
    <name> { $content} </name>
}
</result>
```

- FLWR 式の実行結果

```
<result>
   <name>イチロウ</name>
   <name>ジロウ</name>
   <name>サブロウ</name>
</result>
```

[25]XPath は XML 文書のアドレッシングを行なう為の仕様ですから，問い合わせ結果の出力方法まで明確に規定されているわけではありません．

XQuery による問い合わせ結果は，やはり XML データで返されます．この結果 XML データをパースして，目的の要素を抽出するわけですから，XML データベースにおける問い合わせ結果の処理にはこの部分のコストも考える必要があります．

XpSQL では，内部では SQL への変換が行なわれ SQL を実行しますので，クエリの実行結果は当然リレーションで返ります．結果データサイズが大きいようでしたら，カーソルを使うこともできます．

```
postgres=# select toxml(id) from xpath_eval(
'document("family.xml")/family/name/text()');

  toxml
----------
 イチロウ
 サブロウ
 ジロウ
```

4.5.2 XPath 問い合わせ結果を用いた部分更新

応用例として，XPath による検索結果を用いて，トランザクションブロックの中で該当データの部分更新を行なうことも可能でしょう．

```
BEGIN;
UPDATE xml_node
SET value = '2004,4,4'
FROM xpath_eval('/PLAY/date/text()[1]') as Q
WGERE xml_node.id = Q.id;
COMMIT;
```

4.5.3 豊富なクライアント インタフェース

XpSQL は基本的に SQL によるアクセスとなりますので，PostgreSQL の豊富なクライアントインタフェースをフルに活かすことができます．

XpSQL では配布パッケージの webapp ディレクトリ下に PHP によるサンプルアプリケーションを用意しています．図 12が，ブラウザから起動したインタフェースです．

この WEB アプリケーションは，データベースに格納された XML データに XPath 検索を行なって結果を取得するだけのシンプルなものです．webapp ディレクトリ下に導入方法を説明した INSTALL ファイルがありますので，使ってみたい方はそちらをご参照下さい．

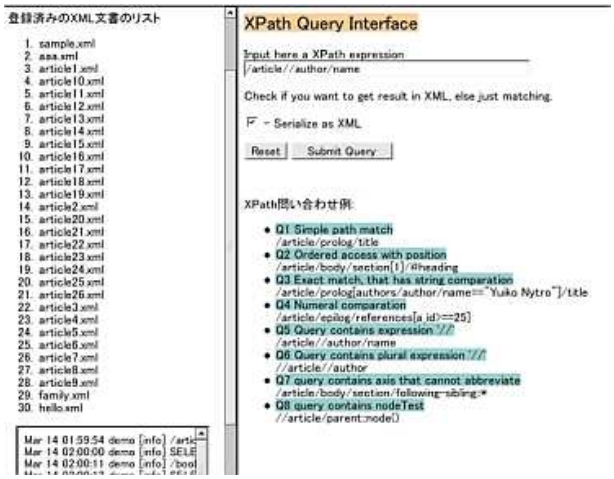

図 12: サンプルアプリケーション

4.6 データベースのチューニング

XpSQL では，$XPSQL_HOME/tools ディレクトリ下にデータベースをチューニングする為の 2 種類のスクリプトを用意しています．

- **ガベージコレクタ** gc.sh
  XpSQL では，データ更新時の負荷を下げる為に，xml_path テーブルへのパス情報は重複のチェックをすることなく，ノード毎にエントリの追加を行なっています．一般的に同じパス情報は，ドキュメント内で複数回登場することが多いです．そこでデータ量の削減，及び検索時の効率を高める為に，定期的に重複の除去 (ガベージコレクション) を行います．ガベージコレクションでは，xml_path テーブルに pathexp フィールドの重複値が削除されます．その際，削除される行の pathid を参照しているものが xml_node の pathid フィールドにあれば，それを新しい値に更新します．

  ただし，ガベージコレクタを走らせた後は，ひとつの pathid が同一のパス情報を持つ複数のノードの間で共有されることとなります．その為，ガベージコレクションを行なった後にパス情報が変わるようなデータの更新を行なうことはサポートしていませんので注意してください．

- **チューニングスクリプト** tuning.sh
  PostgreSQL は追記型のアーキテクチャを採用していますので，検索時の性能向上の為には定期的に Vacuum コマンドを走らせる必要があります[26]．

  このスクリプトでは，データのクラスタリング，インデックスの再構成，Vacuum による不用領域の除去，オプティマイザが用いる統計情報の更新といったことをデータベースに対して指示します[27]．

- **部分インデックス**
  XpSQL では，XPath アクセスに頻繁に用いるカラムにインデックスを貼っています．

  ノードテーブルに関しては，デフォルトで id 列，dewey 列，pathid 列，parent 列，tagid 列に B*-Tree インデックスを貼っています (図 9参照)．

  ノードテーブルの value テーブルなどはインデックスは貼っていませんので，ユーザが必要に応じて CREATE INDEX コマンドを発行することとなります．

  一般的に，XML データベースには自動的インデックスを貼るタイプ (eXist，NeoCore XMS など) と手作業で必要に応じてユーザがインデックスを貼るタイプがあります．自動インデックスは便利ですが，データベースのサイズが大きくなったときにインデックスファイルのサイズが大きくなってしまうという問題があります．インデックスファイルが大きくなってしまえば，インデックスの元も子もありません．

  XpSQL は，PostgreSQL をベースとしていますので，RDBMS が提供する部分インデックスの仕組みを用いる事ができます．部分インデックスを用いることで，頻出値を除外してインデックスを貼るといったカスタマイズが可能となります．

  用途によっては，頻繁に利用するパスや，クエリに特化した形でこうした部分インデックスや複数列インデックスを貼るのも一つの手段です．

5 評価, 比較

皆さんとしては，XML データベースの性能はやはり気になるところと思います．少し前にはなりますが，XML データベースのベンチマークツールである XBench [28]を用いて性能の評価を行いました．

[26] Version 7.4 からは autovacuum ツールも用意がされています．
[27] 単に CLUSTER,REINDEX,VACUUM,ANALYZE コマンドの発行をまとめたモノです．
[28] http://db.uwaterloo.ca/ ddbms/projects/xbench/

XBench の特徴としては，データ指向 (Data-Centric) か文書指向 (Document-Centric) な XML データを生成するかを選べる所，そして用途に応じた複数の XML 文書が生成できることがあります．[29].

比較した対象は，代表的なオープンソース NXDB である Xindice 1.0 と eXist 0.9.2 です[30].

5.1 計測結果

計測に用いたデータは，XBench で生成した以下のものです．詳しいテスト条件はリリース物件の data フォルダ下に入っていますのでご参照ください．

- ドキュメント数 - 26 個
- 合計サイズ - 11MB
- 最小文書サイズ - 120KB
- 最大文書サイズ - 1.2MB

計測に用いた問い合わせは，XBench のドキュメントより抜き出した以下の 8 つです．ベンチマーク結果は図 13に示します[31].

```
Q1 Simple path match
/article/prolog/title

Q2 Ordered access with position
/article/body/section[1]/@heading

Q3 Exact match, that has string comparation
/article/prolog[authors/author/name="Yuiko Nytro"]/title

Q4 Numeral comparation
/article/epilog/references[a_id>=25]

Q5 Query contains expression '//'
/article//author/name

Q6 Query contains plural expression '//'
//article//author

Q7 query contains axis that cannot abbreviate
/article/body/section/following-sibling::*

Q8 query contains nodeTest
//article/parent::node()
```

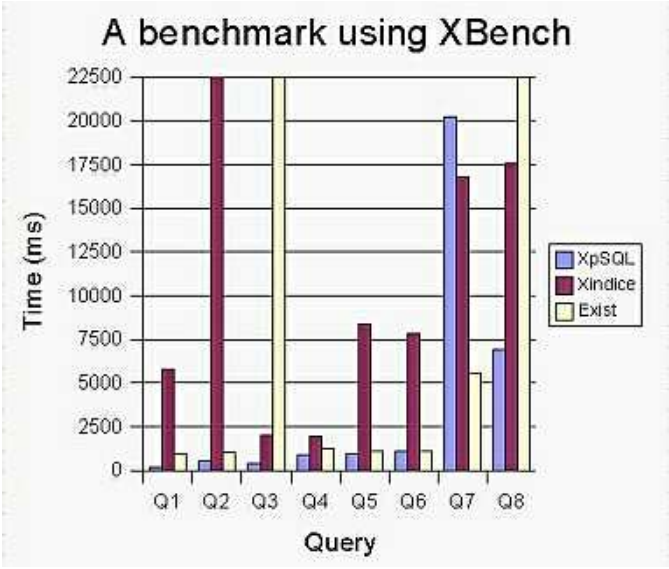

図 13: ベンチマーク結果

XpSQL には Dewey 列のデータ型，リンク情報の削除などまだ改良のよちは多々あるので，eXist にはかなわないものなのかと危惧していたのですが，健闘した結果と言えると思うのですが，どうでしょうか．

唯一，他の XMLDB エンジンと比較しても Q7 の問い合わせで時間がかかってしまっています．この理由としては，XPath による検索結果で 168 個の該当ノードがみつかったのですが，この際のシリアライズ処理で，結果 XML 文書のサイズが合計 8.7M にも上っていたからです．シリアライズまで含めない選択までの実行速度は，0.35 秒ほどでした．

Q8 についてもヒットしたノード数は 26 と小さいものでしたが，結果セットのサイズが 10.7MB 程度となったことが，性能に影響しています．

XpSQL では，結果サイズが DB エンジンのバッファを大きく超えるような場合で，性能が落ちると言えます．eXist などは，サーバ側にも JVM のヒープサイズとして最大 128MB が指定してありました．一方で XpSQL は，計測時．PostgreSQL の Shared Buffer は 30MB 程度に設定していました．

この値は，闇雲に増やせばよいというものでもありませんので，このようなケースではカーソルを利用も考慮に入れてください．

[29] これら Data-Centric や Document-Centric といったキーフレーズは Ronald Bourret 氏による XML and Databases という記事が詳しいです．
http://www.rpbourret.com/xml/XMLAndDatabases.htm
この分野においてマスターピースとも言えるような大変よい記事ですので，XML データベースに興味がある方は一読することをお薦めします．

[30] 当時，最新のリリースでした

[31] 棒が天辺 (22500ms) まで伸びているものは，サポートされていないのか計測不能だった項目です

XMLデータベースでは，問い合わせ結果をXMLとして返す都合，結果サイズが大きくなりがちです．PostgreSQLの初期設定では，バッファ領域のサイズ設定，ソート用に割り当てるメモリのサイズは比較的少めですので，XMLデータベースとして本格的に利用する場合はチューニングを行なうとよいでしょう[32]．

また，同様にしてクライアント側のバッファサイズといったことも，XMLデータベースを用いる上では重要になってきます．

6 応用事例の紹介

XpSQLはまだドキュメントが不充分で，知名度はあまりないものと思いますが，既にいくつかの企業では，どこからかその存在を知って戴き，実際に採用されています．

いくつかのユーザ企業の方にアンケートに応えて戴きましたので，そのいくつかを紹介しようと思います．調査では，採用分野/用途，XpSQLと比較した対象と選定理由という事で質問に答えて戴きました．

- 採用分野/用途
  - リアルタイムで流れてくるXML形式のニュースデータを管理．
  - XMLによる生産管理情報の蓄積，参照．
- XpSQLと比較した対象
  Xindide，eXist，XMLPGSQL，Oracle．
- 選定理由
  - 信頼性面からフリーのNative XML-DBではなく，PostgreSQLをバックエンドとしているXpSQLを選択した．
  - RDBの技術ノウハウ，SQLを活かせる．
  - XPathあるいはXQueryを利用できることが条件であった．
  - オープンソースソフトであるため．

この調査からも，PostgreSQLの知名度，信頼性により選んで戴いているケースが多いようです．オープンソースのNXDBなどでは，テストが充分に行なわれていない事が多く，CVSの最新バージョンのBugFixを常に追いかけなければならないと言うこともあります．クエリ周りの不具合ならば対処も容易ですが，ストレージ周りでデータに欠損/破損などが生じますと致命的な問題にもなりかねません．

その点，XpSQLは基本的にPostgreSQLをストレージバックエンドとしていますので，データの欠落やロック/トランザクション周りの信頼性は高いものと言えるでしょう．

7 次期バージョンに向けて

XpSQLの開発は，未踏ユースの期間中に何とかアイデアを形にした所で期限が来てしまいました．名前空間の扱いを始め，バルクロードの高速化，冗長であるリンク情報の除去，Deweyカラムの表現方法の効率化（現在はIntarrayで表現しているので，格納領域が大きい），問い合わせ結果／プランのキャッシュ，XPath問い合わせ処理のチューニングといった課題が残っています．

現在，XpSQLの開発で得られた知見を活かして，次期XMLデータベースシステムの設計，プロトタイプシステム（XBiRDと呼んでいます）の試作を行なっています．最大の目標はなんと言っても，スケーラビリティの向上です．Clustered-JDBC(http://c-jdbc.objectweb.org/)を用いた仮想データベース層を提供する事でデータの分散を行ない，GBサイズのデータ量で合理的な性能を出すことを目標としています．

まだ，やっとローダとクエリパーザができた所ではあるのですが，次期システムは，Java言語による実装，ストレージバックエンドとするRDBMSを限定しない（MySQL5，Oracleにも対応する）XML Relational Bridgeシステムとして装い新たに登場させる予定です．

8 さいごに

XMLの柔軟性と拡張性は，システム開発に柔軟性が求められる今日のビジネス環境を考慮すると魅力的なものです．一方でこのXMLの性質は，現在のストレージ・アーキテクチャにとっては困難な存在です．

XMLの普及に伴って，これから企業，そして技術者は、生成するXMLの量と当該データの特性に基づいて，ストレージ・アーキテクチャを決定しなければならない時代に入ったと言えるでしょう．

近年では，GISデータ（GXML），遺伝子データ，ニュースデータ（NewsML）といったような標準フォーマットとしてのXMLの採用が進んでいます．このような大規模

[32] Bruce Momjian氏によるPostgreSQL Hardware Performance Tuningという記事が参考になります．
http://www.postgresql.org/docs/aw_pgsql_book/hw_performance/

XML データを扱うケースで，XML データベースの適用を考えている方は多いと思います．

商用の XML データベース製品は，初期の導入コストはもちろんですが，その独自性から開発・運用面でベンダのサポートが必要になるなど，見えないコストが浮上してくることがあります．実際に適用してみたはよいが，導入した XML データベースの運用ノウハウは持っている企業は，XML データベース製品の開発ベンダしかないというような状況も出ているのではないでしょうか．

こうした環境下の限られた予算の中で，開発を行なう必要がある研究開発用途やミドルレンジの業務利用にオープンソースの XML データベース環境である XpSQL がお役に立てることができたのならば幸いです．現在は仕事として XpSQL の開発に専念できる環境にはありませんが，できる限りの事ならば力になるつもりです．

この連載を機に，ぜひ一度手軽に導入することができる XpSQL に触れてみてはいかがでしょうか．