Vol.2010-DBS-151 No.30
2010/11/13

エンティティ中心アプローチによるXMLDB設計手法

榎本俊文[†1]　谷口展郎[†1,*1]　鈴木源吾[†1]
小林伸幸[†1]　山室雅司[†1]

XML は, Web 系サービス, 金融分野, 流通分野等で, その適用範囲は益々拡大している. その一方で, XML データベース (XMLDB) の採用事例は, リレーショナル・データベース (RDB) に比べると非常に少ない. その要因の一つは, XMLDB を活用するための環境 (設計手法, ツール, 教育等) が, RDB に比べて不十分なことにある. このため XMLDB は, RDB に比べて多様なデータ構造を柔軟に扱えるなどの利点を持つにも関わらず, 普及がなかなか進んでこなかった. そこで本論文では, XMLDB を用いたシステム開発におけるデータベース設計手法を提案する. 本手法による設計例を示すことにより, 本手法のシステム構築における優位性を示す.

XMLDB Design Method based on ER Diagrams

Toshifumi Enomoto,[†1] Noburou Taniguchi,[†1,*1]
Gengo Suzuki,[†1] Nobuyuki Kobayashi[†1]
and Masashi Yamamuro[†1]

The XML is used as a data exchange format with many systems in various fields. But, there are not so many systems using XML Database (XMLDB) that are using the XML as a data storage format. One of the reasons is that the environment (design approach, tools and education, etc.) to use XMLDB is insufficient compared with the one to use Relational Database (RDB). Therefore, we propose a database design method for XMLDB and show examples using our method.

†1 NTT サイバースペース研究所
NTT Cyber Space Laboratories
*1 現在, NTT ソフトウェア株式会社
Presently with NTT Softoware Corporation

1. はじめに

XML(eXtensible Markup Language) は, Web 系サービスにおけるデータ交換フォーマットとして盛んに利用されている. さらに近年, 金融分野, 流通分野等の法人系システムのデータ交換フォーマットとして, XML 標準規格の採用が進展するなど, その適用範囲は益々拡大している.

その一方で, データ格納フォーマットとしての XML, 即ち XML データベース (XMLDB) の採用事例は, リレーショナル・データベース (RDB) に比べると非常に少ない. その要因の一つは, XMLDB を活用するための環境 (設計手法, ツール, 教育等) が, RDB に比べて不十分なことにある. このため XMLDB は, RDB に比べて多様なデータ構造を柔軟に扱えるなどの利点を持つにも関わらず, 普及がなかなか進んでこなかったと考えられる.

そこで本論文では, XMLDB を用いたシステム開発におけるデータベース設計手法を提案する. これにより, XMLDB を活用した, 構築が容易で性能的にも有利なシステム開発を実現できる.

2. 研究の背景

2.1 RDB における設計手法

RDB は, データを表形式でモデリングするシンプルな方針を採用したことより, データベース設計に理論的な裏付けがされ, データの重複なく, 更新を矛盾なくするための正規化の考え方が利用されるようになった.

また, データベースの実際の構造を決定する前段階として, 実世界とデータベースを結びつける概念的な設計として, 概念設計のプロセスが明確化された. 概念設計で使われるデータモデルとしては, 「実体・関連モデル (エンティティ・リレーションシップモデル, ER モデル)」が最も一般的になった. これらの設計手法について, 多くの書籍が出版され[1)2)3)], 情報処理の資格試験にも出題され, データベースを含むシステムを設計するエンジニアにとって必須な知識となっている.

2.2 XML データベースとその設計手法

RDB は, 定型的なビジネス領域において大きな成功を収めた. しかし, その一方で, 非定型的で, データ構造が複雑で, 変更の多い応用分野も, 情報処理の進歩・インターネットの普及等に従って増えてきた. そのような領域に適した様々なデータベースが提案されたが, 現在その中で最も有望な選択肢は XML データベース (XMLDB) と言えるであろう (実装例：

NeoCoreXMS[4], TX1[5], pgBoscage[6]). XMLDB の特徴を以下にあげる.

- 多様なデータ構造 (スキーマレス含む) の XML を蓄積し, 高速に検索できる
  (データ構造の標準 W3C XML Schema[7] は自由度の高い仕様である)
- スキーマ・データ構造の変更に柔軟に対応できる
- データ交換で流通する XML データをそのまま蓄積・検索できる

しかし, XMLDB は当初期待されたほどには, 普及していない. その理由には, XMLDB でなければならないというキラーアプリケーションが見出されていないという課題もあるが, これまで RDB による構築手法に馴染んでいるエンジニアが, XMLDB の長所を知りつつも, 慣れた RDB による実装を選択しているという実態がある. スキーマが多様で自由であることはメリットだが, その自由が逆に, データベースエンジニアに戸惑いを与えている. そのような現状を克服するための, XMLDB の設計手法の検討は必ずしも多くはない[8]. しかし, XMLDB を実用的なビジネスに適用していくにあたって, その設計方法論が強く求められている, と我々は考えている.

3. XMLDB 設計手法の提案

そこで本論文では, RDB エンジニアが慣れ親しんだ ER モデルや設計プロセスを基礎とした XMLDB 設計手法を提案する. 本手法によるデータベースの設計工程の全体像を**図 1** に示す. 左が従来の RDB 向けの手法であり, 右が本論文で示す XMLDB 向けの手法である.

本論文で提示する XMLDB 向けデータベース設計手法の特徴は, 以下のとおりである.

- XML(木構造) データモデルの表現力の高さから, 論理設計が非常に平易になる.
- 性能向上とデータ冗長性回避を両立したデータ設計 (物理設計) ができる. 具体的には, 複数のエンティティを 1 つの XML にまとめて表現することが可能である.
- エンティティ内のデータ項目を適切にグループ化し, 階層化することにより, クエリ設計や業務アプリケーションの開発効率化に寄与できる.

上の記述でもわかるように, この手法の特徴は, エンティティを強く意識したデータ設計・クエリ設計を行う点にある. そこで以後, 本論文で提示する XMLDB 向けデータベース設計手法全般を, 「エンティティ中心アプローチ」と総称する. 以降の章で, その理由とそれぞれの作業の詳細について説明していく.

4. 概念設計

まず, 概念設計はデータベース非依存であるため, 従来の手法をそのまま用いる. エンティティ(事象/人/物などの管理対象) と, リレーションシップ (エンティティ間の関連), さらに属性 (エンティティのデータ項目) を洗い出し, 情報の意味的な側面を ER 図に記述する. 概念データモデルの ER 図例を**図 2** に示す. IDEF1X 表記法[9] を用いている.

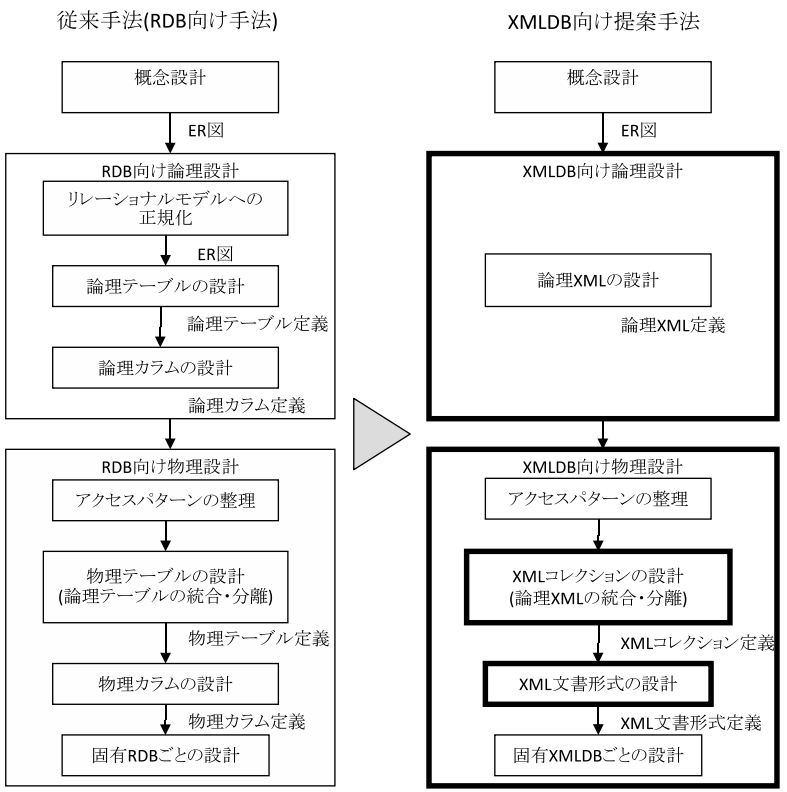

図 1 データベース設計工程

IDEF1X 表記法では, エンティティ(実体) を四角形で表し, その上にエンティティ名を記述する. 角の丸い四角形は, 他のエンティティに依存しないと存在できないエンティティ=依存エンティティである. 四角形の間に引かれた線は, エンティティ間のリレーションシップ (関連) を表す. この例では, クラス, 生徒, 部活動, 試験成績, の 4 つのエンティティとその間のリレーションシップが示されている. 片側に黒丸があるリレーションシップは, 1:N

Vol.2010-DBS-151 No.30
2010/11/13

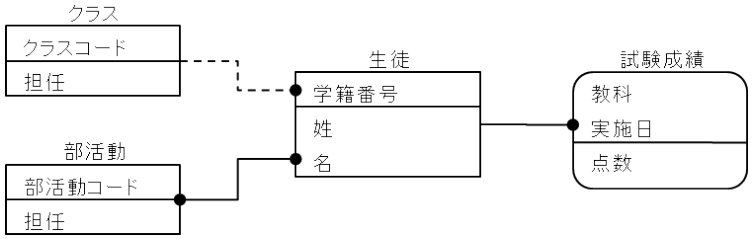

図 2 概念データモデルの ER 図例

(1 対多) の数的関係を表しており, 黒丸のある側が N 側である. したがって, クラスと生徒の関係は 1:N であり, 生徒と試験成績の関係も 1:N である. リレーションシップの両端に黒丸がある部活動と生徒の関係は M:N (多対多) の数的関係を表している. 四角形内に書かれているものはエンティティの属性で, 区切り線の上に主キーとなる属性 (群) を, 下に それ以外の属性を記述する.

以下, この例に沿って設計工程を論じる.

5. 論理 XML の設計

RDB 向けの論理設計では, リレーショナル・モデルの制限により, ER 図の変形 (リレーショナル・モデルの正規化) が必要であった. しかし XMLDB の場合は, 木構造モデルの表現力の高さにより, 変形は不要である. したがって, 図 2 の例においても, これがそのまま論理データモデルとなる. XMLDB 向けの論理設計で行うのは, ER 図と対応した「論理 XML」の定義である. RDB の場合, ER 図との対応付けとしては, エンティティが論理テーブル, 属性が論理カラムに対応し, リレーションシップは外部キー属性として論理カラムに割り当てられる. 一方, 本論文の論理 XML 設計において は, 以下のように対応付け/定義を行う.

- エンティティ: 1 つの論理 XML に対応
  エンティティ中心アプローチでは, XML のルート・エレメントとして表現する. エレメント名に はエンティティ名をそのまま用いる.
- 属性: 論理 XML 内のノードに対応
  エンティティ中心アプローチでは, XML のルート・エレメント直下のエレメントとして表現する. エレメント名には属性名を用い, 値をテキストノードに保持する.

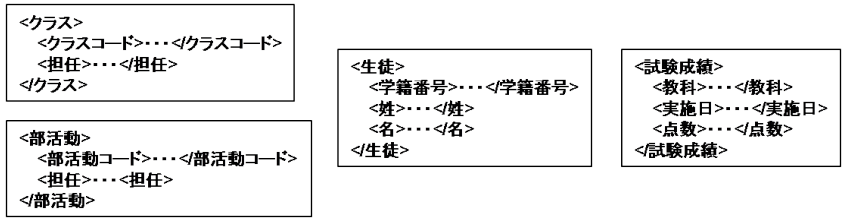

図 3 論理 XML の例 (XML インスタンスのイメージ)

- リレーションシップ: 自由形式で保持
  リレーションシップの具体的なデータ設計は物理設計で行うため, ここでは情報保持ができればよい. したがって, ここでは, 形式を特に規定しない.

上記の定義に基づいて定義した, 図 2 の ER 図の例に対応する論理 XML を**図 3** に示す (リレーションシップは省略).

クラス, 生徒, 部活動, 試験成績, の 4 つのエンティティを, それぞれのエンティティをルート・エレメントとする 4 つの論理 XML で表し, それぞれの属性は, 論理 XML のルート・エレメント直下の エレメントとする.

6. 物 理 設 計

物理設計は, 主として性能向上を目的に, アクセス・パターンの整理, XML コレクションの設計, XML 文書形式の設計, 各 XMLDB 実装固有の設計という手順で行う. ここで,「XML コレクション」とは, エンティティに対応する論理 XML をアクセスパターンの観点で整理した XML の集まりとする. また,「XML 文書形式」は最終的に設計された XML 文書を指す. ただし本論文では, 基本的に実装依存の情報については取り扱い対象としないため,「各 XMLDB 実装固有の設計」については記述しない.

6.1 アクセスパターンの整理

従来の RDB 向け設計でも行われている, 以下の作業を行う. [⋆1]

(1) 必要なアプリケーション (以下 AP) を洗い出す.
(2) 各 AP がアクセスするエンティティ(=論理 XML) とその関係 (参照/更新やアクセス頻度など) を洗い出す.

⋆1 業務アプリケーション (AP) の設計とも関連した作業である.

Vol.2010-DBS-151 No.30
2010/11/13

AP	頻度	エンティティ			
		クラス	部活動	生徒	試験成績
クラスの一覧取得	低	R		R	
部活動の生徒一覧取得	低		R	R	
生徒の姓・名と試験結果の取得	高			R	R
生徒の情報変更	中			U	
試験結果の登録	低				C

図 4 AP とエンティティの関係表

AP	頻度	生徒 エンティティ				
		学籍番号	姓	名	クラスコード (FK)	部活動コード (FK)
クラスの一覧取得	低		R	R		
部活動の生徒一覧取得	低		R	R		
生徒の姓・名と試験結果の取得	高		R	R		
生徒の情報変更	中				U	CUD

図 5 AP と項目の関係表

(3) 各エンティティ(=論理 XML) ごとに, 各 AP がアクセスする属性 (項目) とその関係 (参照/更新や アクセス頻度, 検索キー項目など) を洗い出す.

上記を CRUD 図[*1] を用いて整理した例を**図 4・図 5** に示す. また, 上記以外にも, ピーク時のアクセス頻度や, アクセスごとのデータの平均取得件数, エンティティごとの想定されるデータ量, など性能に関連する情報について, 必要に応じ整理しておく.

⋆1 データがどこで作成 (Create)・参照 (Read)・更新 (Update)・削除 (Delete) される をマトリクス形式で表現した図[10].

基本形式

```
<生徒>
  <学籍番号>・・・</学籍番号>
  <姓>・・・</姓>
  <名>・・・</名>
</生徒>
```

```
<試験成績>
  <教科>・・・</教科>
  <実施日>・・・</実施日>
  <学籍番号>・・・</学籍番号>
  <点数>・・・</点数>
</試験成績>
```

統合形式

```
<生徒>
  <学籍番号>・・・</学籍番号>
  <姓>・・・</姓>
  <名>・・・</名>
  <試験成績リスト>
    <試験成績>
      <教科>・・・</教科>
      <実施日>・・・</実施日>
      <学籍番号>・・・</学籍番号>
      <点数>・・・</点数>
    </試験成績>
    <試験成績>
      <教科>・・・</教科>
      <実施日>・・・</実施日>
      <学籍番号>・・・</学籍番号>
      <点数>・・・</点数>
    </試験成績>
    ・・・
  </試験成績リスト>
</生徒>
```

図 6 リレーションシップの表現例

6.2 XML コレクションの設計

6.2.1 予備知識・背景理論

まず, XML コレクションの設計手順を理解するための知識および方法論について述べる.

●基本形式と統合形式

リレーションシップを XML 構造として表現する場合, 2 つの方法から選ぶことができる.

- 主キー・外部キー属性の追加 (RDB と同様の方法)
- XML ドキュメント内の木構造による表現
  (XML 表現として, 1 エンティティ=1 種類の XML ドキュメントである必要はない. 複数エンティティを 1 種類の XML ドキュメントで表現することもできる.)

以下, 前者 (外部キー表現) を「基本形式」, 後者 (木構造表現) を「統合形式」と呼ぶこととする.

例を**図 6** に示す. 左側が, 従来の外部キーによる表現で, 2 種類の XML ドキュメントに分かれており, 試験成績に「学籍番号」という外部キーが追加されている. 右側が, XML の木構造による表現の一例で, 1 種類の XML ドキュメントに統合され, 試験成績は繰り返し項目として表現されている. これは, 生徒と試験成績が 1:N 関係にあるためである.

つまり, 2 つのエンティティに対し, 基本形式では 2 つの XML コレクションとなり, 統合

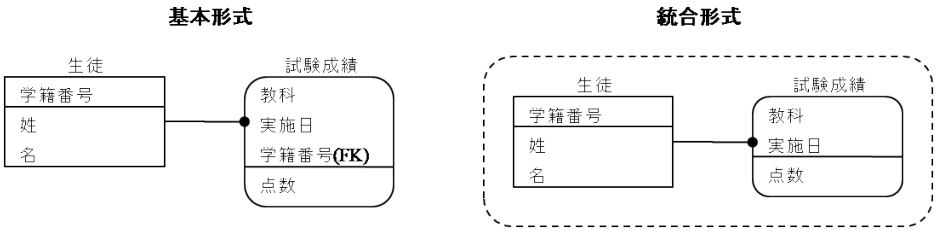

図 7 ER 図の表記の拡張 (統合形式の表記)

形式では 1 つの XML コレクションとなる.

また, 以降の説明のために, ER 図の表記を拡張し, **図 7** に示すような点線で囲んだエンティティ群は統合形式の適用を表すこととする.

統合形式では, 最も代表的な 1:N のリレーションシップを例にとると, 1 側のエンティティを親ノードとし, N 側をその木構造上の子の繰り返しノードとして表現する. [*1]

以後, 統合形式において, 統合後に木構造上の親ノードとなるエンティティを「統合エンティティ」または「上位エンティティ」, 子ノードとなるエンティティを「被統合エンティティ」または「下位エンティティ」と呼ぶことにする. 被統合エンティティが, 自身を 1 側とする 1:N リレーションシップを別に持っている場合, 入れ子状に統合形式を適用することもできる. このように, RDB 的な意味での正規形が成立しない場合でも, データが冗長にならず整合性を保つことができる点が, ここで示した統合形式の重要な特徴である.

ただし統合形式には以下のような制約もある.

- ある子 (N 側) エンティティが複数の親 (1 側) エンティティを持つ (=あるエンティティが, 自身を N 側とする複数の 1:N リレーションシップを持つ) 場合, 1 つの親に対してのみ統合形式で表現可能であり, 残りのリレーションシップは基本形式 (外部キー) で表現することになる.
- M:N 関係のリレーションシップは表現できない (基本形式で表現する).

●リレーションシップごとの統合手法

ここでは, 1:N, 0..1:N, 1:1, M:N, という 4 種のリレーションシップのそれぞれについて, 統合形式の木構造表現を記述する.

⋆1 この文における「親」「子」は, 木構造上の親子である.

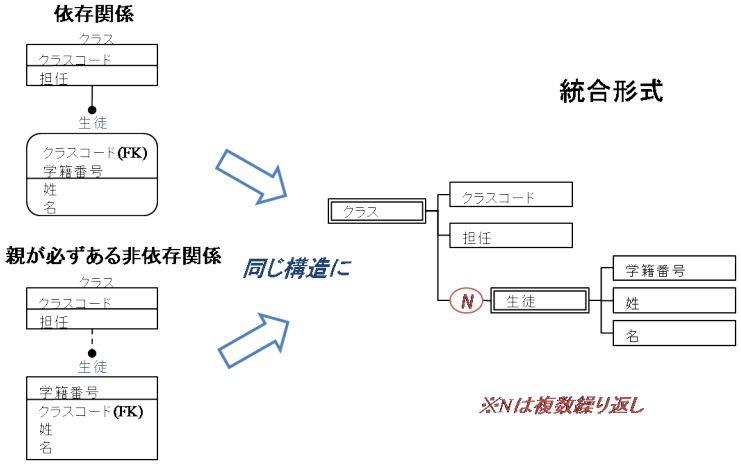

図 8 統合ルール (1 対 N 関係)

○ 1:N 関係の統合

1:N 関係の場合, 依存関係/非依存関係のどちらも 1 側のエンティティに統合可能である. **図 8** の例では, N 側エンティティを 1 側エンティティの子[*2] の繰り返しノードとして表現している.

○ 0..1:N 関係の統合

0..1:N 関係の場合も, 0..1 側エンティティに統合可能である. [*3] ただし, N 側エンティティに対応する 0..1 側エンティティが存在しない場合を表現できるような構造が必要である (NULL 値可能等).

○ 1:1 関係の統合

1:1 関係は, どちらのエンティティへも統合可能である. 1:1 関係であっても繰り返しノードとして表現できるが, 実情に応じてこの繰り返し階層を持たないような設計にすることも考えられる.

⋆2 この場合の「子」は木構造上の親子関係における子である.
⋆3 0..1:N 関係は,1:N 関係の特殊ケースとみなすことができ, 一般に 0..1 側を「親 (エンティティ)」,N 側を「子 (エンティティ)」と呼ぶ. しかし, 前述した通り, 本論文ではこの表現を用いない.

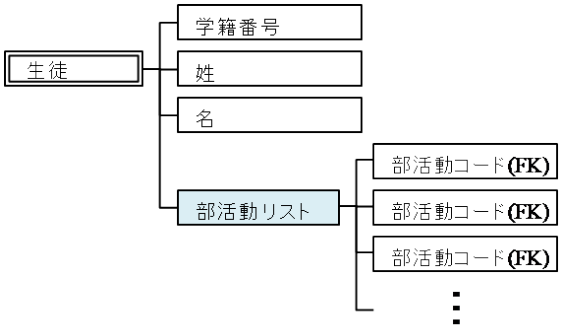

図 9 木構造モデルによる M:N 関係の表現

	参照系			更新系								
	親のみ検索	子のみ検索	親子結合	親のみ挿入	子のみ挿入	親子同時挿入	親の値変更	子の値変更	(子)関連先の変更	親のみ削除	子のみ削除	親子同時削除
基本形式					◎				◎	◎		
統合形式			◎			◎						◎

図 10 基本形式と統合形式の性能比較

○ M:N 関係の表現

M:N 関係は, 基本形式で表現する. 例えば, **図 9** に示すような, 木構造モデルの繰り返し項目表現を利用したデータ構造でキーを保持し, 表すことができる.

●有利な形式の判断基準

基本形式と統合形式では, 期待できる性能の特徴が異なる. **図 10** に, 基本操作に対する両形式の性能面での比較表を示す. “◎” が付いている操作が有利であることを示している. ⋆1

この表をまとめると, 一般的には以下のことが言える.

- 基本形式が性能有利な場合:
  エンティティごとに個別に更新系のアクセスがなされる場合. 特に, 親エンティティの削除, 子エンティティの挿入・外部キーの更新時が有利である.
- 統合形式が性能有利な場合:
  双方のエンティティをまとめてアクセスされる場合. 結合検索 (参照), 一括挿入・一括削除時が有利である.

統合形式を選択することで, RDB でコストが高い結合処理が不要となり, 高速化につながることが期待できる. その一方で, 統合形式では 1 つのデータサイズが大きくなり, 構造も深く複雑になってしまうことによる低速化が懸念されるが, これに関しては XMLDB 特有の高速化手法 (要約情報の利用等[12]) によって解消することができる.

6.2.2 XML コレクションの設計手順

6.2.1 節で述べたとおり, XML コレクションの構造には基本形式と統合形式があり, 統合形式の場合の構造は対象となる 2 つのエンティティ間のリレーションシップに依存する.

XML コレクションは RDB のテーブルに相当するものであるため, 与えられたエンティティ群を, どのような XML コレクションにマッピングするかは, 6.2.1 節の最後で述べたように性能の観点から考える必要がある.

ここでは, 6.1 節で抽出したアクセス・パターンに基づいて, 各 XML コレクションの構造を決定する手順について説明する. 手順としては,

(1) 属性 (項目) ごとの AP のアクセス・パターンから, エンティティの分割を検討する.
(2) エンティティごとの AP のアクセス・パターンから, どのエンティティを統合するかを検討する.

の順で行う.

●属性へのアクセス・パターンに基づくエンティティの分割

従来の RDB の場合と同様, アクセス・パターン, 特に更新パターンが大きく異なる属性 (項目) 群がある場合, エンティティの分割を検討する.

●エンティティへのアクセス・パターンに基づくエンティティの統合

2 つのエンティティを統合形式にするかどうかは, 以下の手順で判断する.

(1) まずは全てのリレーションシップに対し, 基本形式を選択する.
(2) エンティティへのアクセス・パターンから, 同時にアクセス (代表的なものとして, 図 10 の統合形式に “◎” が付いているもの) されているエンティティの組で, M:N 以外のリレーションシップを持つものを抽出する.
(同時にアクセスしていても, 関連のないデータ同士の場合は, 除外すること.)

⋆1 有利であるかは, クエリおよび固有の XMLDB に依存するが, 本論文では XQuery[11] による標準的なクエリと XMLDB の一般的な挙動を前提に仮定している.

Vol.2010-DBS-151 No.30
2010/11/13

(3) 抽出されたエンティティの組に対し,
- 図 10 の基本形式に “◎” がついているものに代表される, 統合形式で不利となるアクセスを行う他の AP が存在しない場合, 統合形式を選択する
- 上記アクセスを行う AP(群) が存在する場合, アクセス頻度等から総合的に判断し, 有利な形式を選択する. ⋆1

(4) すべてのリレーションシップについて上記を行った後, 統合形式と判断された複数のリレーションシップが, 同一の親エンティティを持っている場合, 総合的に判断し, 最も効果が大きいリレーションシップを 1 つ選択し, それ以外のリレーションシップは基本形式に変更する. ⋆2

例えば, 図 2 の ER 図に対し, 図 4 のアクセス・パターンに基づいて, 上記の手順で設計を行うと, 以下のような判断になる.

- クラス (親) と生徒 (子): 基本形式
  - クラスの一覧取得 AP で, まとめて参照している
  - 生徒の情報変更 AP で生徒 (子) のみを更新している
  - 生徒の情報変更 AP のアクセス頻度が高いため, 基本形式を選択
- 部活動 (親) と生徒 (子): 基本形式
  - M:N 関係のため
- 生徒 (親) と試験成績 (子): 統合形式
  - 生徒の姓・名と試験結果の取得 AP で, まとめて参照している
  - 生徒の情報変更 AP で, 生徒 (親) のみを更新している
    試験結果の登録 AP で, 試験成績 (子) のみを作成している
  - 生徒の姓・名と試験結果の取得 AP のアクセス頻度が高いため, 統合形式を選択

この例では, 競合時の判断を, AP のアクセス頻度のみに基づいて行っている.

これにより, XML コレクションを考慮した ER 図は**図 11** のようになる.

［生徒］と［試験成績］は統合形式を示す点線で囲まれる. 一方, ［生徒］と［クラス］, ［部活動］のリレーションシップには基本形式表現が選択され, ［生徒］にはそれぞれに対応する外部キーが追加される.

⋆1 アクセス頻度が重要な指標であることは確かだが, 他にもデータ量や取得件数なども考慮するべきである.
⋆2 従来同様, 正規形を崩しても性能向上を求める場合, この限りではない.

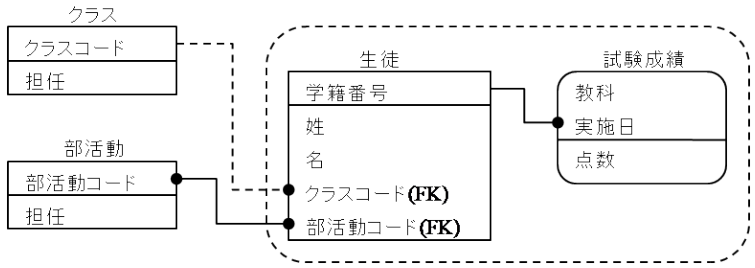

図 11 XML コレクション設計後の ER 図の例

6.3 XML 文書形式の設計

最後に, これまでの検討に基いて最終的な XML 表現を定義し, 物理 XML データを決定する. ここでいう「物理 XML データ」は, 木構造全般ではなく XML に特化している⋆3 が, スキーマ記述言語からは独立な, XML データセット表現である.

本論文では, 以下の手順で物理 XML データを設計することを推奨する.

- 1 つの XML コレクションを 1 つの XML ドキュメントとする.
- XML コレクション内で最上位のエンティティをルート・エレメントとする.
- 統合形式の被統合エンティティは, リレーションシップが 1:N, 0..1:N, 1:1 ⋆4 の場合, 統合エンティティのエレメント直下に, 「該被統合エンティティ名+“_list”」という名前の中間ノードを作成し, その下の繰り返しエレメントとする.
- 属性は, グループ名をエレメント名とした中間ノードの下位に, エレメントとして表現し, その直下のテキストノードに値を格納する.
- M:N 関係の外部キー属性は, 「属性名+“_list”」という名前の中間ノードを作成し, 外部キー属性を繰り返しエレメントとして表現する.
- アトリビュートは使わず, 全てエレメントとして定義する
- “mixed” エレメントは使わない

図 11 に対応する XML データ (3 つの XML コレクション) の例を**図 12** に示す.

⋆3 これまでの議論は, 基本的には木構造全般に通じる議論であったが, ここからは木構造の一つの実現形態である XML に特化した議論になるという意味.
⋆4 6.2.1 節「1:1 関係の統合」の項でも説明した通り, 本論文では 1:1 関係を 1:N 関係と共通に取り扱うアプローチを採る. しかし別のものとして扱ううことを認めないわけではない. また 1:1 関係についてはこの他に, 概念/論理設計の段階で一つのエンティティに統合してしまうやり方もある.

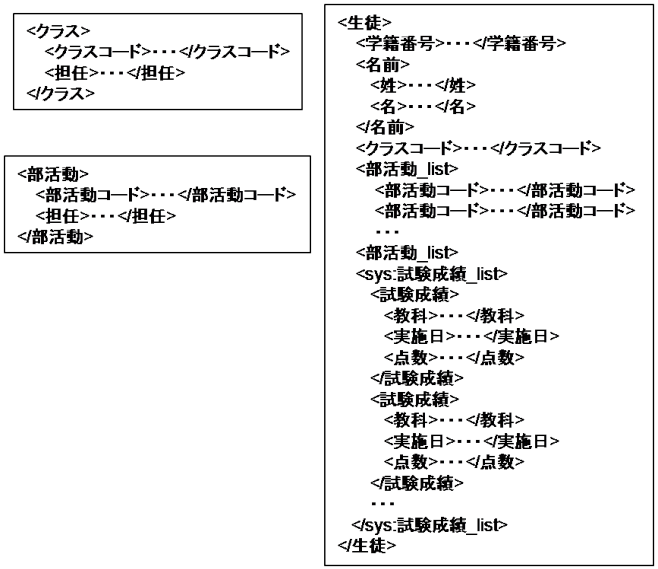

図 12 XML データの例 (XML コレクション)

この例では, 全てのエンティティが同一の名前空間に属すものとし (表記は省略), 統合形式のエンティティをつなぐ中間ノードは “sys” プレフィクスで示される, システム管理用名前空間に属する設計になっているが, 名前空間については, データ操作や保守性, 将来的な変更を考慮して, 以下の方針での設計を推奨する.

- エンティティ単位で名前空間を使用する.
  - 1 つのエンティティ内に項目ごとに異なるといった複数の名前空間は使用しない.
  - エンティティごとに全て異なる名前空間としてもよいし, 複数のエンティティが同一の名前空間に属していてもよい.
- 統合形式でのエンティティ間をつなぐ中間ノードは, エンティティとは異なる名前空間を設定する.

7. 従来手法との比較と課題

本手法は従来手法[8] に比較して, 以下の特徴がある.

- ER モデルを利用した設計プロセスにより, RDB のエンジニアにも導入しやすい
- 物理設計を詳細化・具体化している
- XMLDB に特有な性能を考慮している

ただし, XMLDB の性能特性は, RDB と比較して製品毎にかなり異なっている. 本手法では一般的な性能特性を考慮しているが, 製品毎のチューニングは課題である.

XMLDB の適用事例は少なく, 我々の手法も完全に検証されているとは言えない. 特に, XMLDB の特徴であるデータ構造の変更に強いという点は重要であり, 本手法の有効性を検証する必要がある. 今後, 実際の XMLDB での適用例を増やし, フィードバックしていくことが課題である.

8. おわりに

本論文では, XMLDB を用いたシステム開発において, シンプルかつ効率的な設計・処理を可能にする「エンティティ中心アプローチ」を提案した.

XMLDB を用いたシステム開発は, 経験の蓄積がまだほとんどないために多くの開発者が取り組みをためらっているのが実情である. 本論文が, そうした開発者の XMLDB 活用の最初の一歩を踏み出す一助になれば幸いである.

参 考 文 献

1) 池田哲夫, 川下満, 坂田哲夫, 関根純, 中川優: データベース概念設計, 阿部出版 (1993).
2) 大久保成隆, 関根純, 中川優, 町原広毅: データベース論理設計, 阿部出版 (1993)
3) C. Batini, S. Ceri, S. B. Navathe: *Conceptual Database Design: An Entity-Relationship Approach*, Addison Wesley (1991)
4) http://www.cybertech.co.jp/xml/xmldb/neocorexms/
5) http://www.toshiba-sol.co.jp/pro/xml/
6) 兵藤正樹, 江田毅晴, 榎本俊文, 山室雅司: pgBoscage : PostgreSQL を用いた XMLDB の実装, 電子情報通信学会総合大会 (2008)
7) http://www.w3.org/XML/Schema
8) 日本アイ・ビー・エム株式会社: プロジェクトの流れで理解する XMLDB デザイン解説, 日経 BP 社 (2008)
9) T. A.Bruce:*Designing Quality Databases with IDEF1X Information Models*, Dorest House(1992)
10) 真野正: 実践的データモデリング入門, 翔泳社 (2003)
11) http://www.w3.org/TR/xquery/
12) 江田毅晴, 鬼塚真, 山室雅司: XML データの要約情報を用いた高速な XPath 処理方法, 電子情報通信学会論文誌 (2006)