データ・ベースシステム 71－8
（１９８９．５．１８）

# 再帰的 UNION

芝野 耕司　林田 定一　佐藤 明子
東京国際大学　㈱アスキー　三菱電機㈱

ISO（国際標準化機構）で開発が行われているデータベース言語 SQL3 における再帰的 UNION の設計について述べる。再帰的 UNION の開発は、従来リレーショナルデータベース言語が不得手としてきた部品展開問題をより一般的にサポートすることを目指したものである。これによって NDL などの言語が受け持ってきた適用業務領域を SQL 言語でより効果的に扱うことが可能になり、SQL 言語だけでデータベース適用業務をすべてカバーすることができるようになる。この論文では、構文形式の検討経緯を中心に述べ、併せて、SQL3 での再帰的 UNION の動作モデル、再帰的な問合せのアルゴリズム及び実装上の考慮について述べる。

# Recursive UNION

Kohji Shibano　Sadaichi Hayashida　Akiko Sato
Tokyo International University　ASCII Corporation　Mitsubishi Electric Corporation

In this paper, we describe the design rational of Recursiove UNION in the Database Language SQL3, which is being developed by ISO (International Organization for Standardization). The objective of the development is to support the parts explosion problem, which is one of the most cumbersome task in programming by conventional relational database languages including the SQL. Recursive UNION is especially effective in an application field which has been supported by non relational languages such as NDL. We describe the development of the syntax of Recursive UNION, the processing model of it in the SQL3, recursive query algorithms, and then we consider an implementation of it.

## 1. はじめに

SQL3 [1] の開発意図は、SQL1 [2] が関係データベース言語の基本となる規格を開発することを目指し、SQL2 が SQL1 を改善し、より完全な関係データベース言語の開発を目指したのに対して、従来の関係データベース言語族が苦手としていた適用業務を含め、より広い課題に挑戦することである。

本稿では、SQL3 の大きな新機能の一つである再帰的な問合せについて、これまでの研究及び ISO での提案を含め、仕様について検討する。この機能は、関係データベース言語が苦手としている木構造又はネットワーク構造のデータを関係データベース言語の特徴を生かして、簡単でかつ柔軟に処理することを目指している。

2. では、再帰的 UNION の典型的な適用業務である部品展開問題と機構造のサポートについて述べ、3. では、再帰的 UNION の構文形式の検討経緯、4. では、SQL3 における動作モデルについて述べ、最後に、再帰的な問合せについて知られているアルゴリズム及び実装上の考慮について述べる。

## 2. 部品展開問題と“木構造”のサポート

データベース管理システムの起源は、NASA での宇宙開発における部品管理のためである、といわれるほど、部品展開問題は、データベース管理システムにとって、典型的な問題である。

また、実世界のデータを自然に写像するモデルとして、古くから階層構造やネットワーク構造、“木構造”が用いられ、これに基づいたデータベース管理システムが作られてきた。こうしたデータモデルに基づく典型的なデータベース言語は、JIS/ISO データベース言語 NDL [3] である。

現在の NDL 規格には、NDL プログラムの例がついている。 1985 年に SQL1 の DIS (ISO 規格原案) 化の審議を行ったときに、日本からこの NDL の例に対応した例を SQL1 に追加することを提案した [4]。しかし、この NDL の例は、 SQL1 の例として適切ではないこと及び SQL1 の例を入れることは、多くの書物など例があることを理由に必要がないと、否決された。この例は、いわゆる部品展開問題を扱ったものであり、関係データベースでは、扱うのが難しい問題でもあった。このことが NDLの例を SQL 1 に入れることに反対された一つの根拠となったと思う。

部品展開問題とは、一つの部品が別の複数の部品で構成されているとき、ある部品を作るのに必要な部品をすべて求める問題である。この種の問題は、“木構造”で表わすことが多い。そのため、SQL でも、当初、“木構造”での検索サポートとしてこの問題をとらえてきた。

簡単な部品展開問題の例を次に示そう。

部品表

| 親部品 | 子部品 |
|---|---|
| P1 | P2 |
| P1 | P3 |
| P1 | P4 |
| P2 | P3 |
| P2 | P5 |
| P3 | P5 |
| P3 | P6 |
| P4 | P3 |
| P4 | P6 |
| P7 | P8 |
| P7 | P9 |

ここで、部品表は、二つの列、親部品及び子部品で構成されているとする。

部品展開問題は、例えば、組織表の展開問題などの適用業務でしばしばお目にかかる問題であり、NDL のようなネットワーク構造や“木構造”をもつデータベースでは簡単に表現することができる。

一方、SQL1 でもこの問題を扱うことは可能である。この問題を解くための SQL1 でのプログラミングの一つの方法は、PL/I などのホスト言語で再帰的なプログラムを書くことである。ホスト言語での再帰的なプログラミングを回避するためには、1985 年に NDL の例を SQL1 で書いて日本から ISO に

提案したときのように、作業用の一時的な表を作成し、この表に対する複数の挿入、取出し及び削除を繰り返す必要がある。

このように、この問題は、SQL 言語又は関係データベース言語が最も不得意とする問題の一つであり、この問題を SQL の特徴を生かした形で扱えれば、NDL などの言語がカバーしてきた適用業務領域も SQL でもっと簡単に扱うことが可能になるであろう。

また最近は、再帰的問合せについて、演繹データベースや帰納データベースにおいても、幾つかの問合せ処理法式が提案されている。特に、AI の研究開発の進展とともに、知識ベースの核として、数多くの研究がなされている [10]。

## 3. 構文形式の検討

### 3.1 SQL 言語での部品展開問題についての従来の扱い

ISO での検討が始る以前に SQL 言語を拡張して、“木構造”やネットワーク構造のデータを扱うことについて、幾つかの試みがなされている。実際に、代表的な SQL データベース管理システムの一つである Oracle [5] では、この部品展開の問題を扱うために SQL 言語の拡張を行っている。Oracle SQL で、前章の問題に対する問合せを書くと、次のようになる。

```
SELECT  親部品, 子部品
FROM 部品表
CONNECT BY PRIOR 親部品 TO 子部品
START WITH 親部品='P2'
```

この例では、部品 P2 を構成するすべての部品の列挙を行なうことができる。しかし、この形式の SQL では、構文があまりに制約されている。また、“木構造”サポートにともなう特殊な情報、例えば、“木構造”の深さなどの情報については、特別にシステムが用意した列が結果表に加えられる。

### 3.2 ISO での検討経緯

ANSI での最初の提案は、Oracle SQL の構文をもとにした数年前の提案である。1987 年の後半に、将来の SQL がサポートすべき機能として、NBS (National Beaureau of Standard) の Joan Sullivan [6] によって、再度、Oracle SQL をもとにした提案がなされた。同時に、IBMの Robert Engels と Phil Shaw による別々の提案が Phil Shaw によりなされた [7]。

Joan Sullivan の提案は、前に述べた Oracle SQL と基本的には同じ内容の提案であり、新しい句を導入するとともに、システムが設定する特別な列によって“木構造”に特有な値のやりとりを実現している。

Robert Engels の提案は、Joan Sullivan の提案を一歩進めてより洗練された形式での“木構造”サポートと考えられる。彼の提案も、本質的には、Joan Sullivan や Oracle SQL と同様である。Engels の提案する構文形式で書くと、次のようになる。

```
SELECT * FROM 部品表
STRUCTURE BY 親部品 TO 子部品
FOR 親部品 = 'P2'
ORDER BY TREE, NODE;
```

これらの提案は、基本的には、Oracle SQL での提案と基を一にするものであり、木構造のサポートを特別な場合と考え、特別な構文を導入することによって、SQL 中でサポートすることを目指している。

Phil Shaw の提案 [8] は、より SQL に合った形式で、特別な変数や一つの適用業務に固有な形でのセマンティックスをサポートするのではなく、現在の SQL 言語仕様の拡張として、特に、再帰的な問合せとして“木構造”サポートをとらえているという意味で、新しいアプローチとして高く評価できる。Shaw の提案では、次のように書ける。

```
RECURSIVE P (親部品, 子部品)
INITIAL    (SELECT *
            FROM 部品表
            WHERE 親部品 ='P2')
ITERATION (SELECT *
```

FROM 部品表
WHERE 部品表.子部品 = P.親部品)

### 3.3 再帰的な問合せの実現は、どうあるべきか？

これまでの議論の中で、すでに幾つかの評価点について、暗に触れてきた。しかし、もう一度どのような形での SQL 言語の拡張が望ましいのか、について考えてみよう。

一つの観点は、ある拡張が従来からある言語機能に対して、直交的な機能として実現され、他の機能と組み合わせて用いることができることが望ましい。また、特殊な前提条件を想定せずに、より一般的な方法で実現されることが望ましい。

Oracle SQL, Joan Sullivan 及び Robert Engels の提案では、特別な形式の表に対する特別な操作として“木構造”サポートが扱われている。一般化されているのは、親部品及び子部品のコードを含む列名のみであり、“木構造”の展開には、親部品のコードと子部品のコードが等しい場合にのみ親子関係が存在するとしている。こうして想定される親子関係に対して、“木構造”をシステムが規定した順序で展開したものが結果表に書き込まれる。

Phil Shaw のアプローチの優れている点は、親子関係を一般的な探索条件で指定できるようにしたこと及び操作上の細かい配慮である。

すなわち、親子関係については、親部品のコードと子部品のコードが等しい場合のみでなく、親部品と子部品のコードの関係を任意の探索条件として指定できるように拡張されている。

また、彼の提案では、探査句での指定によって、木構造を走査する順番として、深さ方向優先と広がり優先を選択することができ、適用業務での要求もカバーされている。循環句及び打ち切り句も導入され、循環的な再帰的問合せに対する対処などの問題も考慮されている。この意味では、以前の提案に対して、より良い提案になっている。

しかし、彼のアプローチでは、再帰的な問合せは、従来の SQL 言語要素とは全く別の機能として捉えられており、RECURSIVE 句として定式化されている。この意味で Phil Shaw 提案は、基本的な機能仕様としては、Oracle SQL と同等のものである。

これらの従来の提案の問題点を考慮に入れ、新しい形式の再帰的な問合せをで提案した。

### 3.4 再帰的関係演算としての再帰的な問合せの実現 [9]

Phil Shaw の提案をもう一歩一般化することにより、より整合的な形で、より一般的な再帰的な問合せのサポートを SQL のなかで実現できないのかが、このアプローチの背後の動機である。

彼の提案をもう一度検討してみよう。彼の提案では、新しく幾つかの句が導入され、従来の関係演算子とは別の新しい機能として再帰的な問い合わせが提案されている。しかし、この提案を従来の UNION の直接的な拡張としてとらえることは、可能であり、また、望ましい。実際、再帰的な問合せは、SQL で従来からサポートしている UNION を再帰的に行うことに他ならない。1985 年の日本からの提案も一時的に定義した表に対する挿入及び削除を行っているが、この操作は、UNION を再帰的に行っていることを表現したものである。

“木構造”あるいは再帰的な問合せを再帰的 UNION として捉えることにより、より一般的で SQL 言語と整合的な仕様を与えることができる。

以下に、再帰的 UNION 提案の構文を示そう。

```
select ...
from (select ...        -- INITIAL
      from PP
      where ...
    RECURSIVE UNION
      select ...        -- ITERATION
      from PP
      where PP.COL2=PX.COL1
    LIMIT ..)
    PX (COL1,COL2)
where ...
```

この例では、再帰的 UNION は、従来の UNION にオプショナルなキーワードを追加することにより指

定できる。また、Phil Shaw 提案のもっている機能をすべて実現できる。同時に新たに導入される言語構成要素は、基本的には、オプショナルなキーワードのみであり、構文形式の変更を最小限に抑えることができる。

この例からは、もう一つの可能性が示唆される。すなわち、UNION 以外の関係演算にも同様の再帰的な取扱を許すことである。次に、再帰的な JOIN の実行例の枠組みを示す。

```
select
from
(select col1, col2
 from PP
 where ..
RECURSIVE JOIN
 select PP.col1, Px.col2
 from PP
 where ...
ON PP.col2 = PX.col1)
PX(col1,col2)
```

このように、より一般性を目指し、新しい構文を検討することは、言語設計の上で重要なことと思う。

ISO では、この提案を採用し、再帰的 UNION として、SQL3 の作業文書に入れた。

この提案での構文形式は、Phil Shaw の提案をより一般化するのみならず、再帰性を直交的な概念として捉えることを意味する。直交的に捉えることにより、以下で示すように効率的な実装が可能になる。

## 4. ISO SQL3 での再帰的 UNION の構文と動作モデル

### 4.1 SQL3 での再帰的 UNION の構文

SQL3 での再帰的問合せは、再帰的 UNION を導入することによって実現した。構文形式は、次のとおりである。

```
<再帰的 UNION>::=
        (<初期式>
        RECURSIVE UNION
        <相関名>
        [(<再帰列リスト>)]
        <反復式>
        [<探索句>]
        [<循環句>]
        [<打ち切り句>])
```

この構文を使って以下のような部品表の部品展開の問題を扱ってみる。

部品表

| 親部品 | 子部品 |
|---|---|
| P1 | P2 |
| P1 | P3 |
| P1 | P4 |
| P2 | P3 |
| P2 | P5 |
| P3 | P5 |
| P3 | P6 |
| P4 | P3 |
| P4 | P6 |
| P7 | P8 |
| P7 | P9 |

P1 を構成する部品をすべて列挙する再帰的 UNION の構文例は以下のようになる。

```
(SELECT  親部品, 子部品          --a
         FROM   部品表
         WHERE  親部品 = 'P1'
RECURSIVE UNION                  --b
P (親部品, 子部品)               --c
SELECT   親部品, 子部品          --d
         FROM   部品表
         WHERE  親部品 = P.子部品)
```

aが <初期式>、cの P が <相関名> で (親部品、子部品) が<再帰列リスト>、dが <反復式>である。<探索句>は、部品展開のための探索順序を深さ優先

とするか又は広がり優先とするかの指定を行う。〈循環句〉は、循環的な参照があった場合に、その循環的な参照がどこであったかを結果表から分るように結果表の列の指定されたマークを書込むことを指示する。〈打ち切り句〉は、部品展開を指定されたレベルで打ち切ることを指示し、打ち切られた場合の動作を指定する。

**4.2 SQL3 での動作モデル**

この SQL 文がさきの部品表に対して、どう処理されるかを、SQL3 の現在の作業文書の一般規則に沿って説明する。

ステップ1. 〈初期式〉 a を評価する。結果をIT とする。
ステップ2. IT をスタック S に積む。
ステップ3. スタックが空なら終り。そうでないならスタック S から1行取り出してくる。この行を R とする。
ステップ4. R を結果の集合 RT に加える。
ステップ5. R の列 子部品 の値を相関名の P に束縛して〈反復式〉 d を評価する。
ステップ6. もしステップ５の結果が空でなければ、これをスタックに積む。
ステップ7. ステップ３からステップ６をスタックが空になるまで繰り返す。

結果は RT に求められる。
図示するとステップ１で

```
          --------------
IT        親部品   子部品
          --------------
          P1       P2
          P1       P3
          P1       P4
```

となる。
ステップ６までを１回終ったところで

```
          -------------
RT        親部品   子部品
          -------------
          P1       P2 <---- R
```

スタック

```
          -------------
S         親部品   子部品
          -------------
          P2       P3     1 回目のステップ5の
                          結果
          P2       P5
          P1       P3     IT の残り
          P1       P4
```

ステップ６までを２回終ったところで

```
          -------------
RT        親部品   子部品
          -------------
          P1       P2
          P2       P3 <---- R
```

スタック

```
          -------------
S         親部品   子部品
          -------------
          P3       P5     ２回目のステップ５の
                          結果
          P3       P6
          P2       P5     １回目のステップ５の
                          残り
          P1       P3     IT の残り
          P1       P4
```

最後にスタックSが空になって、RTが求める結果となる。

```
          -------------
RT        親部品   子部品
          -------------
          P1       P2
          P2       P3
          P3       P5
          P3       P6
          P2       P5
          P1       P3
          P3       P5
```

P3　　P6
P1　　P4
P4　　P3
P3　　P5
P3　　P6
P4　　P6

スタック

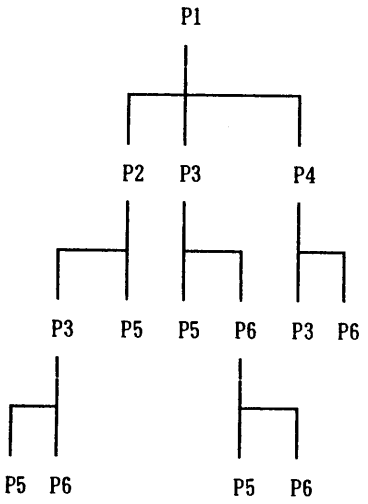
---

S　　親部品　子部品

---

先の部品表は、以下のような木構造に表すことができる。このような木構造を検索する場合、各ノードを訪れる順序に、深さ優先と同一レベル優先が一般に考えられる。

〈探索句〉は、この各ノードを訪れる順序を指定するものである。




〈探索句〉に DEPTH を指定した場合、ステップ２と６でスタックに昇順で積み、ステップ３でスタックの一番上から取り出す。例えば、最初 P2, P3, P4 をこの順でスタックに積み、P4 を取り出して、P3 と P6 をスタックに積む。これを繰り返すと深さ優先で各ノードを訪問する。上の例では、P1,P4,P6,P3,P6,P5,P3,P6,P5,P2,P5,P3,P6,P5となる。BREADTH を指定した場合、スタックに降順で積み、スタックの一番下から取り出す。これは同一レベル優先で検索する。上の例では、P1,P4,P3,P2,P6,P3,P6,P5,P5,P3,P6,P5,P6,P5となる。PREORDERを指定した場合や何も指定しない場合は、システムに依存した順となる。

〈循環句〉は、検索の無限のループを防ぐために指定する。これを行うためには、ステップ３で取り出した R の被参照列が２回以上現れていないかどうかをチェックすればよい。

〈循環句〉で指定された〈循環列リスト〉(省略された場合は、〈反復句〉内で〈相関名〉で修飾された、外への参照をしている列) を被参照列と呼ぶ (上の例では 子部品)。

R の被参照列の値と同じ値を持つ行が結果の集合 RT 中に存在するかどうかチェックする。もし存在すれば、ステップ６で〈反復式〉の評価の結果が空でなくてもスタックに積まない。これにより無限に問合せが展開されるのを防ぐ。

〈打ち切り句〉は、展開の最大回数を指定する。〈打ち切り句〉で指定された値がＬだとすれば、ステップ３からステップ６の繰り返しを最大Ｌ回までしか行わない。これにより、展開される結果のサイズを制限できる。

## 5. 再帰的問合せのアルゴリズムと SQL3 での再帰的 UNION の動作モデルのの実装上の考慮

[10] に再帰的な問い合わせを評価する種種のアリゴリズムが解説されている。単純なボトムアップ評価とその高速版、主なトップダウン評価として QSQR とHenschen-Naqvi のアリゴリズムが説明してある。

ISO SQL3 での再帰的 UNION の扱いは Henschen-Naqvi のアリゴリズムと同等であり、潜在的に「結合演算の前に選択演算と射影演算を実行する」という性質を備えている。

[10] で紹介されているアルゴリズムは、ファイルアクセスの視点ではなく、手続の観点からアルゴリズムを検討している。SQL3 における動作モデルも基本的には、再帰的 UNION の結果がどの様に期待されるかを規定するために手続的な動作モデルとして規定されている。しかし、再帰的 UNION のデ

ータベース的な扱いには、ファイルアクセスを考慮に入れた物理アクセス法を含めた検討が必要である。この様なアルゴリズムの提案には、[11] がある。

[11] では、物理的な配置を考慮に入れ、directed acyclic graphのクラスタリングとして、部品展開問題の扱いをエンジニアリングデータベースでの検討から述べている。

しかし、SQL 言語の視点からは、これらの検討とは別の検討が可能である。すなわち、SQL 言語と従来のデータベース言語との最も大きな差異は、SQL 言語が関係データベース言語であることではなく、コンパイラ言語であり、問合せの最適化を行うことに最大の特徴がある。そして、SQL 言語では、データベース言語としてのコンパイレーションと最適化を可能にするために、関係データモデルを採用したとみることが可能である。

この視点に立って、再帰的 UNION の動作モデルを検討すると、再帰的 UNION の最適化技術としては、入れ子型の問合せの最適化のための手法 [12] が適用可能であることが再帰的 UNION の前述の動作モデルから分る。

このように再帰的 UNION は、従来の SQL 言語の実装技術の延長線上で、効率的に実装することが可能であり、一方、これによる適用業務分野の拡大及び適用業務プログラムの開発の簡単化に大きく貢献すると考えられる。

また、データベース実装技術上に再帰的 UNION として、再帰的問合せを実現することにより、演繹データベースや大規模な知識ベースの構築にあたっても、データベース技術の支援を得られるようになると考えられる。

## 6. 終りに

この論文では、SQL 言語に再帰的な問合せを取入れるためにどの様な検討を行い、どの様な視点で、検討を行い、提案を行ったかを述べた。

再帰的問合せを再帰的 UNION として捉え、再帰性を直交的な概念として扱うことによって、より汎用的で、柔軟な取扱いが初めて可能になる。すなわち、従来 navigational な言語でのみサポートされてきた機能を集合的に扱うことによって、適用業務のプログラミングは、大幅に簡略化される。また、従来の副問合せの最適化の問題として、再帰的問合せを扱うことができ、効率的な実装が可能になる。

JIS/ISO でのデータベース言語規格の開発は、データベース理論研究及びデータベース実装技術研究の評価を行うとともに、独自の視点で、新しく実装までを考慮に入れた技術の開発が必要であり、この意味で、極めて挑戦的な課題である。

## 参考文献


[1] 日本規格協会, JIS X3005-1987 データベース言語 SQL

[2] ISO/IEC JTC1/SC21/WG3 DBL CAN-3a, Database Language SQL2 and SQL3, April 1989

[3] 日本規格協会, JIS X3004-1987 データベース言語 NDL

[4] ISO/TC97/SC21/WG3, Japanese Comments on SQL DP, 1985

[5] Oracle, SQL*Plus User's Guide, ORACLE Part No 3201

[6] J. Sullivan, Tree structured retrieval, ANSI-X3H2 87 306, 1987

[7] R. W. Engels, Structured tables, ANSI-X3H2 87 331

[8] P. Shaw, CLOSURE expressions, ANSI-X3H2 87 330, 1987

[9] 芝野耕司, Recursive UNION and JOIN, ISO/IEC JTC1/SC21/WG3 DBL CPH, 1987

[10] 西尾章次郎、楠見雄規, 演繹データベースにおける再帰的な問い合わせの評価法, 情報処理 Vol. 29 No. 3, March 1988

[11] J. Banerjee, W. Kim, S. Kim, J. F.Garza, Clustering a DAG for CAD Databases, IEEE Transaction fo Software Engineering, Vol. 14 No. 11, Nov. 1988

[12] M. M. Astraha et al, Access path selection in relational database management system, Proceedings of SIGMOD Conference, 1979