集合指向言語 SOL のデータベースへの応用†

重松保弘†† 與那覇誠††† 吉田将††††

関係データベース言語 SQL をホスト言語埋込み形式で使用する場合，現在の JIS (ANSI, ISO) 規格で標準化されているホスト言語では SQL との不整合性が問題とされている．これは，本質的にはホスト言語が集合や関係を言語仕様に含まないことに起因する．そこで，この不整合性の改善，とくに集合変数を用いた SQL インタフェースの改善をはかる目的で，著者らが開発した集合指向言語 SOL を SQL のホスト言語として応用し，SOL/SQL システムを開発した．具体的には SQL 文のうち SELECT 文，INSERT 文および UPDATE 文についてスカラ変数列，集合変数および写像列が指定できるよう埋込み構文を拡張するとともに，カーソル処理によるデータアクセス機構などを SOL/SQL 言語処理系によって隠ぺいすることにした．その結果，ホスト言語 SOL と SQL 間で集合単位および関係単位のデータの受渡しが自然な形式で記述できるようになり，ホスト言語と SQL 言語の整合性を改善することができた．また，応用プログラムの記述量も PLI/SQL の場合と比較して大幅に縮小できるようになった．本稿では，SQL 文の拡張埋込み構文の仕様と IBM 4381 上で開発した SOL/SQL 言語処理系について述べるとともに，SOL/SQL と PLI/SQL で比較記述した応用プログラム例を示す．

1. まえがき

関係データベースモデル[1]におけるデータベースプログラミング言語(DBPL)の研究は，既存のプログラミング言語の拡張に基礎を置くものとデータベース問い合わせ言語に基づくものに大別できる[7]．前者の代表的なものは PASCAL/R[2] であり，後者の代表的なものはホスト言語埋込み SQL[6] である．PASCAL/R は，関係構成子，データベース構成子，関係演算子，1階述語論理式などを導入することにより PASCAL を DBPL に発展させたものである．関係データベースモデルの属性，タプル，リレーションは，PASCAL/R ではフィールド，レコード，リレーション (SQL では列，行，表) で表現される．PASCAL/R は，主としてデータベースの教育・研究用に使用されており，プロトタイプシステムの稼働が報告されている[7]．SQL (Structured Query Language) は，1987年に ISO により国際規格化され，同年，JIS 規格化された代表的な関係データベース言語であり，商用データベースシステムの言語として広く普及している．

SQL の特徴は，対話形式構文とホスト言語埋込み形式構文を持つことである．対話形式構文は，非手続き的であり自然な英文に近いという利点があるが，計算能力に限界がある．そのため，多くのアプリケーションはホスト言語埋込み形式で開発されている．ホスト言語埋込み形式を使用する場合，問題となるのは，現在の JIS (ANSI, ISO) 規格のホスト言語 (COBOL, FORTRAN, PASCAL, PL/I) が SQL と不整合を引き起こす点である．Date は，この問題は本質的には，SQL が集合レベルのデータを扱うのに対しホスト言語がレコードレベルのデータしか扱えないことに起因すると指摘している[10]．これは，具体的には，ホスト言語と SQL の間で直接受け渡すことができるデータがスカラ値に限られ，集合データの受渡しを行うためには複雑なカーソル処理を陽に記述しなければならないことを意味している．しかし，こうした問題を解決するための高度な言語間インタフェースを構築する試みは，これまで全く行われていない[7]．

筆者らは，これまで，アルゴリズムの自然なプログラム化を実現するために集合指向言語 SOL (Set Oriented Language) を開発してきた[15]．SOL は PASCAL の集合型について機能の強化・拡張を行うとともに，写像と述語論理の記法を導入した手続き型言語である．したがって，これを SQL のホスト言語として使用することにより，従来のホスト言語の使用と比較して言語間インタフェースの改善が期待できると考えられる．

そこで筆者らは，SOL と SQL の言語仕様および SOL 処理系を拡張することにより，SQL 埋込み SOL

† On the Application of a Set Oriented Language SOL to Database Systems by YASUHIRO SHIGEMATSU (Department of Electrical, Electronic and Computer Engineering, Faculty of Engineering, Kyushu Institute of Technology), MAKOTO YONAHA (Computer Science Department, Hardware Engineering Laboratory, Sumitomo Metal Industries Ltd.) and SHO YOSHIDA (Department of Artificial Intelligence, Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology).
†† 九州工業大学工学部電気工学科
††† 住友金属工業・システムエンジニアリング事業本部・システム研究開発部
†††† 九州工業大学情報工学部知能情報工学科

システム（SOL/SQL システム）のプロトタイプを開発した[16)~19)]．このシステムの目標は，SQL と SOL 間で言語表現上，直接的な集合の受渡し，および，SQL の表と SOL の写像間で形式上の相互変換を可能にすることである．この受渡しと変換により，SQL の列の値として現れるマルチ集合（multi set）は SOL では集合変数の値としての（非マルチ）集合で表現され，SQL の関係としての表は SOL では関係のグラフ（処理系内部ではリンク構造）として表現されることになる．関係のグラフ化は消費記憶領域の増加を招く反面，データの検索効率の改善が期待できるという利点も持っている．この特徴は，情報構造を陽に扱うことの多い応用分野では有用であると考えられる[14)]．

本稿では，SQL 文の SOL 埋込み仕様と SOL/SQL 言語処理系について述べるとともに，PL/I埋込みSQL システム（PLI/SQL システム[12)]）と SOL/SQL で比較記述したプログラム例を示す．なお，以下の例題プログラムでは，付録1のデータベース[9)]を使用する．また，構文（特に，非端記号）の説明には JIS 規格[6)]で使用されている名称を用いることにする．

2. SOL 埋込み SQL 文の仕様

2.1 埋込み可能な SQL 文と埋込み形式

SOL プログラムに埋め込むことができるのは IBM 社の SQL/DS（VM/CMS 版）の SQL 文（一部の例外を除いて JIS 規格の SQL 文と同じ）である[19)]．これらの SQL 文は SOL プログラムの実行文として任意の場所に埋め込むことができる．SQL 文は JIS 規格と同様，SQL 先頭子（EXEC SQL）と SQL 終端子（;）で囲んで埋め込む．

2.2 手続きの省略と制限の緩和

従来のホスト言語の場合，①SQL 文の終了状態をホスト言語に受け渡す領域 SQLCA (SQL Communication Area)[13)] を宣言する，②SQL 文中で使用するホスト変数はすべて埋込み SQL 宣言節中で宣言する，③表から行の値を取り出すためにカーソル宣言とカーソル処理手順を記述する，④検索する列が NULL 値を許す時には NULL 標識を指定する，などの宣言や手続きが必要である．また，使用できるホスト言語の対象はスカラ変数だけという制限もある．

SOL/SQL では，これらの手続きの省略と制限の緩和を行った．すなわち，①SQLCA 領域を SOL/SQL 処理系内部で用意し，その宣言を不要にした．②SQL 文中のホスト変数名と列名の区別は構文上明確なので SOL/SQL 処理系で識別することにした．その結果，埋込み SQL 宣言節は本質的に不要になった．③SOL の集合変数や写像と SQL の表の値を受け渡す処理を SOL/SQL 処理系内部に実装したので，カーソル宣言とカーソル処理も本質的に不要になった（ただし，利用者がカーソルを陽に記述して使用することもできる）．④NULL 値を値として許すことにしたので NULL 標識は不要となった．これらの改良によりプログラムの記述量が減少した．

また，埋込み SQL 文中で使用できる SOL の対象として変数，関数，および写像を許すことにした．使用できる変数はスカラ変数，配列要素および組要素である．また，配列の添字，関数の引き数，写像の原像指定部には SOL の式をおいてよい．また，写像の使用を許すことにより，後述するように，SELECT 文などの〈選択相手リスト〉中に SOL の写像を置いて SQL の表との間で相互変換が可能になった．

SOL の対象を置いてよい場所は，現在のところ，SELECT 文の WHERE 句の〈探索条件〉，探索 UPDATE 文の SET に続く〈設定句：探索〉，INSERT 文の VALUES に続く〈挿入値リスト〉，SELECT 文の INTO に続く〈選択相手リスト〉，CONNECT 文である．これらの場所に SOL の対象を記述する場合は JIS 規格と同様，SQL の対象（列名など）と区別するために SOL の対象の先頭にコロン（:）をつける．ただし，〈選択相手リスト〉のように，明らかに SOL の対象とわかる場合にはコロンを省略できることにした．

2.3 代入と参照に関する拡張機能

従来のホスト言語の場合，SELECT-INTO 形式の検索命令では，検索結果はスカラ変数にしか代入することができなかった．これに対し，SOL/SQL では検索結果をスカラ変数に加えて組変数，集合変数および写像に代入できるよう機能拡張を行った．さらに，検索結果を集合変数と写像に追加する機能を持つ SELECT-APPEND 形式を新たに用意した．以下に拡張機能を例で説明する．

次の例は，単一行の検索結果をスカラ変数（int 1, str 1）と組変数（tup 1）に代入するものである．

```
var int1 : int;
    str1 : str;
    tup1 : tupleof [a, b : str];
    …
EXEC SQL SELECT SNO, SNAME,
```

```
                STATUS, CITY
        INTO    tup 1, int 1, str 1
        FROM    S
        WHERE SNO='S 1';
```

この例は，表Sから WHERE 句の条件を満足する列 SNO, SNAME, STATUS, CITY の値を検索し，SNO と SNAME の値の組を tup 1 に，STATUS と CITY の値を各々 int 1 と str 1 に代入するものである．この結果，各変数の値は次のようになる．

tup 1=[S 1, Smith]
int 1=20
str 1=London

複数行から成る検索結果は集合変数，または集合変数と写像に代入することができる．前者の例を次に示す．

```
var tupset : setof tupleof
            [a, b : str; c : int; d : str];
...
EXEC SQL SELECT SNO, SNAME,
                STATUS, CITY
        INTO    tupset
        FROM    S
        WHERE STATUS>15;
```

この例は，WHERE 句を満足する列 SNO, SNAME, STATUS, CITY の値を組とし，その組の集合を構成し変数 tupset に代入するものである．

後者の場合は，各列を集合と考え，その集合間に写像を定義する．検索結果をどのような形式で代入するかを INTO 節において記述する．例1では，写像 f の定義域と値域にそれぞれ SNO と SNAME の値を加え，この間に写像関係を定義する．同様のことを写像 g についても行う．例1の実行結果を図1に示す．

[例1]

```
var sno, sname, city : setof str;
map f : sno→sname;
```

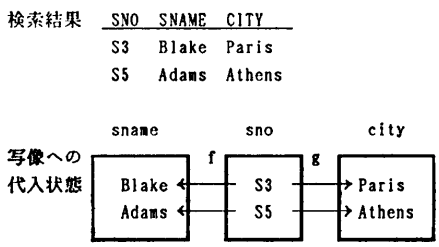

図 1 例1の検索結果と写像への代入
Fig. 1 Search result of example-1 and its assignment to mapping relations.

```
        g : sno→city;
...
EXEC SQL SELECT SNO, SNAME, CITY
        INTO    〈f(SNO)=SNAME,
                    g(SNO)=CITY〉
        FROM    S
        WHERE STATUS>20;
```

値域を集合族型とする写像を INTO 節に指定することもできる．これを例2に示す．

[例2]

```
var sno      : setof str;
    pno_set  : setof setof str;
map f        : sno→pno_set;
...
EXEC SQL SELECT SNO, PNO
        INTO    〈f(SNO)= {PNO}〉
        FROM    SP
        WHERE QTY>200;
```

例2の SELECT 文は，SNO の値に対応する PNO の値を集合化したうえで写像関係を定義する．この SELECT 文の実行結果を図2に示す．図2の検索結果(a)はネスト (nest) 操作[14]によって(b)のように非正規化され，さらに(c)に示すように集合と写像を定義する．以上，拡張 SELECT 文の例を示した（構文は付録2を参照）．

また，INSERT 文の VALUES において，集合変数を参照することにより，その全要素を一括して表に挿入できる．たとえば，次に示す INSERT 文の実行

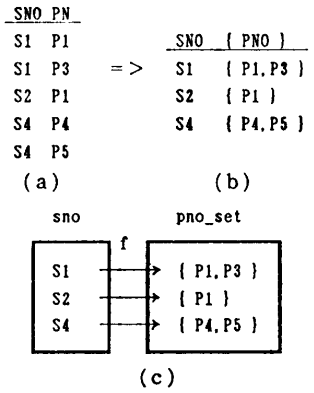

図 2 非正規化変換を伴う集合と写像への代入
(a) 検索結果，(b) 非正規化された検索結果，(c) 代入状態
Fig. 2 Assignment to sets and mapping relations with unnormalized transformation.
(a) Search result, (b) Unnormalized search result, (c) Mapping relation.

により，図3（a）の集合変数の全要素が（b）のように挿入される．

```
EXEC SQL INSERT
        INTO S (SNO, SNAME, CITY)
        VALUES : set 1;
```

また，INSERT 文と UPDATE 文において，SOL の集合と写像によって貯えられている関係単位のデータを参照することができる（各文の構文は付録2を参照）．

最後に，正規化変換を伴う INSERT 文の例を示す．図4（a）の写像データが存在するとき次の文が実行されると，f の値域の値はアンネスト（unnest）操作[14]により正規化されて（b）に示すように SP 表に挿入される．この例は，VALUES に続いて〈写像〉を指定するものである．

```
EXEC SQL INSERT INTO SP
        VALUES 〈f(SNO)= {[PNO,
                          QTY]}〉;
```

2.4 集合に関する直交性

SQL 文を SOL 埋込み形式に拡張するにあたっては，集合に関する直交性を保つことに重点を置いた．すなわち，SQL 文中で，集合値が定義される構文位置に SOL の集合変数が指定でき，集合値が参照される構文位置に SOL の集合式が指定できることである．具体的には，前者は，SELECT 文の INTO および APPEND に続く〈選択相手リスト〉の位置であり，後者は，WHERE 句の IN 述語の位置，および，INSERT 文の VALUES に続く位置である（ただし，現在の SOL/SQL 処理系では，型検査を簡単化するために VALUES に続く位置には集合変数しか許していない）．

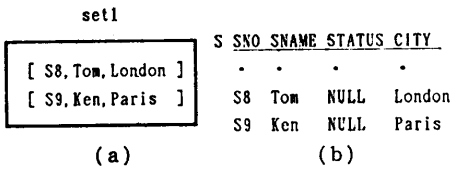

図3 集合データの挿入
（a）挿入集合，（b）挿入結果
Fig. 3 Result of insertion of set data.
(a) Insertion sets, (b) Insertion result.

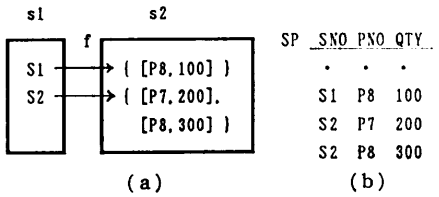

図4 正規化変換を伴う挿入
（a）写像データ，（b）挿入結果
Fig. 4 Result of insertion with normalized transformation.
(a) Mapping data, (b) Insertion result.

3. 処 理 系

SOL の処理系は，これまでコンパイラ・インタプリタ方式として PASCAL 言語で開発されてきた[15]．SOL/SQL 処理系は，この処理系をすべて PLI/SQL 言語で記述し直したうえで，その中間コードを拡張することにより実現した[18]．PLI/SQL を使用した理由は，IBM 4381 の CMS 上では，SQL のホスト言語として PL/I が最も制限が少ないからである．SOL 埋込み SQL 文は，動的実行が可能な SQL 文と不可能な SQL 文で処理が異なる．以下，これらの処理方法について述べる．

3.1 動的実行可能な SQL 文の処理

SELECT 文以外の SQL 文は，EXECUTE IMMEDIATE 文により動的に実行させることができる．次の例は，変数 STRING に代入した DELETE 文を動的実行するものである．

```
STRING='DELETE FROM S WHERE
                    STATUS<10';
EXEC SQL EXECUTE IMMEDIATE :
                    STRING;
```

SELECT 文は，SQLDA (SQL Descriptor Area) 領域を介したカーソル処理によって動的に実行する．SQLDA の構造を図5に，カーソル処理の手順を図6に示す．

SQLDA 領域のうちで，処理系が主として使用するのは SQLN, SQLD, SQLVAR である．SQLN には，検索予定の列数をセットする．SQLVAR には列の型 (SQLTYPE)，列のサイズ (SQLLEN)，データ領域のアドレス (SQLDATA)，NULL 値の検査データを格納するアドレス (SQLIND)，列名 (SQLNAME) が格納される．

図6において，文字列変数 STRING に動的実行を行う SELECT 文が代入されているとする．すると，カーソル処理は次のように実行される．①PREPARE 命令により，ホストの文字列変数 STRING に入っている SQL 文に対するモジュール OBJ を生成する．②SQLDA の SQLN 欄の値を定めた後，SQLDA 領域を動的に確保する．③DESCRIBE 文により，検索

される列の数(SQLD), データ型(SQLTYPE)およびデータ長(SQLLEN)を SQLDA に得る. ④もし, 検索される列の数(SQLD)が予想した数(SQLN)より多ければ, SQLN の値を SQLD の値として, 再度 DESCRIBE 文を実行する. ⑤SQLTYPE と SQLLEN の値に基づいて処理系内部にデータ受取のためのバッファ領域を確保し, その先頭番地を SQLDATA に, NULL 評価値を格納する番地を SQLIND に格納する. これを SQLVAR(1)～SQLVAR(*n*) に対して行う(*n* は検索結果の列数). ⑥カーソルを用いて, 動的に確保した領域に検索結果を取り出しながら処理を進める.

3.2 動的実行不可能な SQL 文の処理

動的実行機能で実行できないものには COMMIT 文, ROLLBACK 文, CONNECT 文, WHENEVER 文, カーソル文(DECLARE-CURSOR, OPEN, FETCH, CLOSE)がある.

COMMIT 文, ROLLBACK 文はフォーマットが限定されているため, 静的に用意することができる. ユーザが埋め込んだ COMMIT 文, ROLLBACK 文の代わりに, 静的に用意した SQL 文に実行を代行させる.

CONNECT 文はユーザ ID とパスワード部に変数を置くことにより, 静的に用意することができる. ユーザが埋め込んだ CONNECT 文のユーザ ID とパスワード部を取り出し, 静的に用意した CONNECT 文の各変数に代入した後, 実行を行う.

WHENEVER 文は, SQL/DS の例外処理機能を模倣する. SQL 文の実行後に SQLCODE 変数の値を調べ, その値にしたがって例外処理を行う.

カーソル文は, SOL/SQL 処理系内部に静的に用意したカーソル文に実行を代行させる. SOL/SQL プログラム中で一度に複数個のカーソルを定義できるよう複数個のカーソル文を用意している.

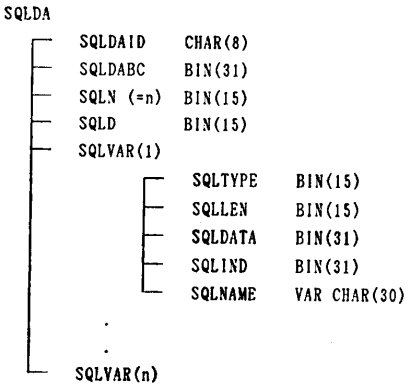

図 5 SQLDA の構造
Fig. 5 Structure of SQLDA (SQL Descriptor Area).

```
① EXEC SQL PREPARE OBJ FROM :STRING;
② SQLN=n;    ALLOCATE SQLDA;
③ EXEC SQL DESCRIBE OBJ INTO SQLDA;
④ IF (SQLN<SQLD) THEN DO;
    SQLN=SQLD;    ALLOCATE SQLDA;
    EXEC SQL DESCRIBE OBJ INTO SQLDA;
  END;
⑤ DO I=1 TO SQLD;
    SQLDATA= ... ;  SQLIND= ... ;
  END;
⑥ EXEC SQL DECLARE C1 CURSOR FOR OBJ;
  EXEC SQL OPEN C1;
  EXEC SQL FETCH C1 USING DESCRIPTOR SQLDA;
  DO WHILE(SQLCODE=0);
    ...;.....
    EXEC SQL FETCH C1 USING DESCRIPTOR SQLDA;
  END;
  EXEC SQL CLOSE C1;
```

図 6 拡張 SELECT 文の動的実行手順
Fig. 6 Execution sequence of extended SELECT statement.

3.3 集合と写像の参照に対する処理

集合と写像の参照を伴う INSERT (UPDATE) 文の実行は, SOL/SQL 処理系内部では INSERT (UPDATE) 文の繰返し実行で実現される. したがって, INSERT 文で集合を参照する場合は, その各要素の格納番地を SQLDA 領域に格納しておき USING DESCRIPTOR SQLDA を指定した形の INSERT 文を動的に繰返し実行する. 次に例を示す.

```
EXEC SQL INSERT
        INTO SP (SNO, PNO)
        VALUES : set 1;
```

この例で, set 1 を 2 つ組の集合とすると, SOL/SQL 処理系は次の文字列変数 S1 を定義したうえで, SQL/DS に PREPARE 処理を実行させ, 実行モジュール(OBJ)を作る. ここで, set 1 の 2 つ組の要素は各々パラメータ(?)に対応している.

```
S 1='INSERT INTO SP (SNO, PNO)
                    VALUES (?, ?)'
```

ついで, SOL/SQL 処理系は set 1 の各要素の格納番地を SQLDA 領域に格納したうえで, 次の EXECUTE 文を実行させることにより INSERT 処理を実現する.

```
EXECUTE OBJ USING DESCRIPTOR
                    SQLDA ;
```

INSERT 文で写像を参照する場合は（必要ならアンネスト処理をした後），定義域と値域の組に対して，上述と同様の処理を行う．また，UPDATE 文で写像を参照する場合は（必要ならアンネスト処理をした後），定義域の値を WHERE 句の〈探索条件〉とし，値域の値を SET の〈設定句：位置づけ〉とする形で UPDATE 文を動的に繰返し実行する．この処理も INSERT 文の場合とほぼ同じである．

3.4 型適合性検査

動的実行機能を利用して実行する埋込み SQL 文は，SOL/SQL 処理系が構文解析時に，動的実行可能な形式の SQL 文に変換した後，SQL/DS の PREPARE 処理を用いて構文検査と型適合検査を行う．具体的には，INSERT 文と UPDATE 文に対し，埋込み SQL 文中のホスト変数を同じ型の定数に置換したうえで PREPARE 文を実行することにより上記の検査を実現する．SELECT 文については，上記と同様の処理を行うとともに，DESCRIBE 文を実行して検索すべき列の型情報を（SQLDA 領域に）取り出し，ホスト変数の型情報と比較検査する．これらの検査の結果，型不適合が検出された場合は，原プログラムのリスト中にエラーが表示される．

以上の検査では検出できない型不適合も起こる．たとえば，SOL の整数型の値を SQL の短精度整数型（SMALLINT 型）の列に渡す場合，構文解析時には共に整数型なので SOL/SQL 処理系は型不適合エラーの表示をしない．しかし，SOL の整数型は 32 ビット長，SQL の短精度整数型は 16 ビット長で各々内部表現されるので，実行時の評価値によってオーバフローエラーが発生することがある．同様の不適合は文字列型についても発生する．現在の処理系は，こうした内部表現形式の相違による実行時の型不適合エラー処理を行っていないので，利用者は SQLCA 領域の SQLCODE 変数の値を調べてエラーの発生を検出しなければならない．

4. 例題プログラム

ここでは SOL/SQL のプログラム例を示す．一部の例では PLI/SQL との比較を示す．

4.1 集合データの受渡し

図 7 の SOL/SQL プログラムは，「入力された都市（複数個）にある供給者名を出力する」ものである．このプログラムに入力として集合値 {"London", "Athens"} を与えると，出力として集合 {Smith, Clark, Adams} が得られる．これと同じ処理を行う PLI/SQL プログラムを図 8 に示す．ただし，PLI/SQL の場合，集合入力は不可能であり都市名は 1 つずつ入力しなくてはならない．また，出力結果も単一化できないので，図 7 と図 8 は全く同じプログラムというわけではない．

図 8 からわかるように，PLI/SQL では検索結果をスカラ値単位で SQL から受け取る．そこでカーソル処理（カーソル宣言，OPEN，FETCH，CLOSE などの処理）が必要となる．また，SQL との通信領域（SQLCA）の宣言，SQLCA 領域のシステム変数 SQLCODE による FETCH 処理の評価，および変数 NULLIND を使った NULL 値処理など，かなり複雑な処理が必要になる．SOL/SQL では，こうした集合データのアクセスメカニズムを陽に記述する必要がないので，プログラムの記述量が減少するとともに自然な形で SQL とのインタフェースが実現されていることがわかる．

```
proc supplier;
  var cityset, supplset: setof str;
  begin
    read(cityset);
    EXEC SQL SELECT SNAME
             INTO   supplset
             FROM   S
             WHERE  CITY IN cityset;
    writeln(supplset);
  end.
```

図 7 SOL/SQL で記述した例題プログラム "supplier"
Fig. 7 Example program "supplier" written in SOL/SQL.

```
SUPPLIER: PROC OPTIONS(MAIN);
  EXEC SQL INCLUDE SQLCA;
  EXEC SQL BEGIN DECLARE SECTION;
    DCL (CITYNAME, SUPPL) CHAR(20);
    DCL NULLIND FIXED BIN;
  EXEC SQL END   DECLARE SECTION;
  EXEC SQL DECLARE C1 CURSOR FOR
           SELECT SNAME
           FROM   S
           WHERE  CITY = :CITYNAME;
  GET LIST(CITYNAME);
  DO WHILE (CITYNAME ¬= '');
    EXEC SQL OPEN C1;
    EXEC SQL FETCH C1 INTO :SUPPL:NULLIND;
    DO WHILE (SQLCODE=0);
      IF NULLIND>=0 THEN
        PUT SKIP EDIT(SUPPL)(A);
      EXEC SQL FETCH C1 INTO :SUPPL:NULLIND;
    END;
    EXEC SQL CLOSE C1;
    CITYNAME='';
    GET LIST(CITYNAME);
  END;
END SUPPLIER;
```

図 8 PLI/SQL で記述した例題プログラム "SUPPLIER"
Fig. 8 Example program "SUPPLIER" written in PLI/SQL.

4.2 関係単位の代入

図9(a)の SOL/SQL プログラムは「都市別に，その都市にある供給者の番号とその都市に保管されている部品の番号を求める」ものである．このプログラムでは，関係単位の SQL データを SOL の集合 (city, supplset, partset) と写像 (suppl, parts) に代入している．実行結果は図9(b)のように非正規化表現された形で出力される．PLI/SQL で同じ処理を行うプログラムを作ろうとすると，カーソル処理を含む複雑なプ

```
proc exam1;
  var c: str;
      city : setof str;
      supplset, partset: setof setof str;
  map suppl: city → supplset;
      parts: city → partset;
  begin
    EXEC SQL SELECT CITY, SNO
             INTO   < suppl(CITY)={SNO} >
             FROM   S ;
    partset ← Φ ;
    EXEC SQL SELECT CITY, PNO
             APPEND < parts(CITY)={PNO} >
             FROM   P ;
    forall c ∈ city do;
      writeln(c:10, suppl(c):12, parts(c));
    od;
  end.
```

(a) プログラム
(a) Program.

```
ATHENS    { S5 }      NULL
LONDON    { S1,S4 }   { P1,P4,P6 }
PARIS     { S2,S3 }   { P2,P5 }
ROME      NULL        { P3 }
```

(b) 実行結果
(b) Execution results.

図 9 例題プログラム "exam 1" とその実行結果
Fig. 9 Example program "exam 1" and its execution results.

```
proc exam2;
  var parts: str;
      inset, workset, supplset: setof str;
  begin
    read(inset);
    parts ← getel(inset);
    EXEC SQL SELECT SNO
             INTO   supplset
             FROM   SP
             WHERE  PNO=:parts;
    forall parts ∈ inset do
      EXEC SQL SELECT SNO
               INTO   workset
               FROM   SP
               WHERE  PNO=:parts;
      supplset ← supplset ∩ workset;
    od;
    writeln(supplset);
  end.
```

図 10 例題プログラム "exam 2"
Fig. 10 Example program "exam 2."

```
proc exam3;
  var s: str;
      inset, outset, supplno: setof str;
      partset: setof setof str;
  map parts: supplno → partset;
  begin
    EXEC SQL SELECT SNO, PNO
             INTO   <parts(SNO)={PNO}>
             FROM   SP;
    read(inset);
    outset ← { s | s ∈ supplno, inset ⊂ parts(s)};
    writeln(outset);
  end.
```

図 11 exam 2 を拡張内包式を用いて書き直したプログラム
Fig. 11 A remake from exam 2 using extended comprehensional form.

```
proc exam4;
  var parts: str;
      ansset, workset: setof str;
  begin
    ansset ← Φ ;
    read(parts);
    workset ← {parts};
    while( workset ≠ Φ ) do;
      EXEC SQL SELECT MINOR_PNO
               INTO   workset
               FROM   COMPONENT
               WHERE  MAJOR_PNO IN workset;
      ansset ← ansset ∪ workset;
    od;
    writeln("   ", parts:6, ansset);
  end.
```

図 12 例題プログラム "exam 4"
Fig. 12 Example program "exam 4."

```
EXAM4: PROC OPTIONS(MAIN);
  EXEC SQL INCLUDE SQLCA;
  DCL IN_PNO CHAR(6);
  GET EDIT(IN_PNO)(A(6));
  CALL RECURSION(IN_PNO);

  RECURSION: PROC(UPPER_PNO) RECURSIVE;
    EXEC SQL BEGIN DECLARE SECTION;
      DCL UPPER_PNO CHAR(6);
      DCL MAX_PNO BIN FIXED;
      DCL FETCHED_PNO CHAR(6);
    EXEC SQL END DECLARE SECTION;
    DCL CNT FIXED,
        LOWER_PNO(MAX_PNO) CHAR(6) CONTROLLED;
    EXEC SQL DECLARE C CURSOR FOR
      SELECT MINOR_PNO
             FROM COMPONENT
             WHERE MAJOR_PNO = :UPPER_PNO
             ORDER BY MINOR_PNO;

    PUT SKIP EDIT(UPPER_PNO)(A);
    EXEC SQL SELECT COUNT(DISTINCT MINOR_PNO)
                    INTO :MAX_PNO
                    FROM COMPONENT
                    WHERE MAJOR_PNO=:UPPER_PNO;
    IF MAX_PNO=0 THEN RETURN;

    ALLOCATE LOWER_PNO;
    EXEC SQL OPEN C;
    LOOP1: DO CNT=1 TO MAX_PNO;
      EXEC SQL FETCH C INTO :FETCHED_PNO;
      LOWER_PNO(CNT)=FETCHED_PNO;
    END LOOP1;
    EXEC SQL CLOSE C;

    LOOP2: DO CNT=1 TO MAX_PNO;
      CALL RECURSION(LOWER_PNO(CNT));
    END LOOP2;
    FREE LOWER_PNO;
  END RECURSION;
END EXAM4;
```

図 13 PLI/SQL による "exam 4" のプログラム例
Fig. 13 Example program of "exam 4" written in PLI/SQL.

ログラムとなる．

図 10 の SOL/SQL プログラムは「入力された部品番号（複数個）のすべてを供給している供給者番号を出力する」ものである．プログラム中の関数 getel は，引き数となる集合変数から要素を1つ取り出すとともに，それを集合変数から取り除く．SP 表のサイズが小さいときは，**図 11** のプログラムでも同じ処理が実現できる．図 11 のプログラムでは，まず SP 表のすべてを集合と写像に代入し，問い合わせの結果は ZF 式 (Zermelo-Frankel expression) に類似した形式に拡張した内包集合式を利用して求めている．

4.3 部品展開問題

SQL 文のみでは解決できない問題として部品展開問題がある．これは「入力された部品のすべてのレベルにわたるすべての構成部品を並べる」ものである．SOL/SQL プログラムを**図 12** に示す．同じ処理を PLI/SQL で記述したものが**図 13** のプログラムである．ここでは，再帰と繰返しにより問い合わせの結果を求めている．まず，入力された部品番号 (IN_PNO) をもとに下位の部品番号数 (FETCHED_PNO) を求め，カーソル処理の繰返しにより部品番号を配列 LOWER_PNO に読み込む．ついで，LOWER_PNO の配列要素ごとに再帰的に下位の部品番号を求める．図8の例と同様，出力結果は単一化できない．

4.4 関係の結合

SELECT-APPEND 形式を使用することによって関係の結合を実現できる．**図 14**（a）は「個々の部品について，それを保管してある都市に所在する，その部品の供給者を求める」SOL/SQL プログラムの例である．ここで，stored_in は部品番号から保管都市への写像，seat_of は都市名からその都市に所在する供給者集合への写像，supplied_by は部品番号から供給者と部品量の組集合への写像である（この問題に限定すれば，supplied_by はたんに部品番号から供給者集合への写像と宣言して使用してもよい）．

プログラムでは，まず，SELECT 文によって**図 15** に示す関係のグラフを構成する．ついで，forall 文中

```
proc same_city;
    var pno: str;
        pno_set,city_set,ans_set: setof str;
        sno_set: setof setof str;
        sno_qty: setof setof tupleof[sno:str; qty:int];
        sq_tup: tupleof[sno:str; qty:int];
    map stored_in: pno_set  →  city_set;
        seat_of: city_set  →  sno_set;
        supplied_by: pno_set  →  sno_qty;

    begin
        EXEC SQL SELECT PNO,CITY
                 INTO   <stored_in(PNO)=CITY>
                 FROM   P;
        sno_set ←Φ ;
        EXEC SQL SELECT SNO,CITY
                 APPEND <seat_of(CITY)={SNO}>
                 FROM   S;
        sno_qty ←Φ ;
        EXEC SQL SELECT PNO,SNO,QTY
                 APPEND <supplied_by(PNO)={[SNO,QTY]}>
                 FROM   SP;

        writeln("PNO  CITY      SNO");
        forall pno  ∈  pno_set do
            ans_set ← { sq_tup.sno | sq_tup ∈ supplied_by(pno) };
            ans_set ← ans_set ∩ seat_of(stored_in(pno));
            writeln(pno:5, stored_in(pno):10, ans_set);
        od;
    end.
```

（a） プログラム
（a） Program.

```
PNO  CITY      SNO
P1   LONDON    { S1 }
P2   PARIS     { S2,S3 }
P3   ROME      {  }
P4   LONDON    { S1,S4 }
P5   PARIS     {  }
P6   LONDON    { S1 }
```

（b） 実行結果
（b） Execution results.

図 14 関係の結合の例
Fig. 14 Example on the join of relations.

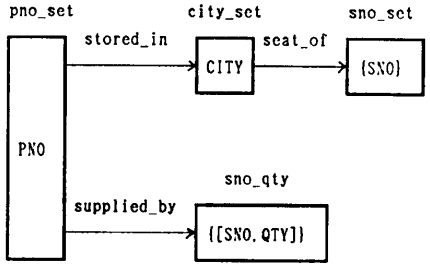

図 15 プログラム same_city によって構成される関係のグラフ
Fig. 15 A graph representation of relations, constructed by the program same_city.

で部品 (pno) の供給者集合 (ans_set) を求め，部品の保管都市に所在する供給者集合 (seat_of (stored_in (pno))) との共通集合を計算する．図 14（b）は，プログラムの実行結果である．

SELECT 文による写像 stored_in と seat_of の定義は，関係 P と S に選択 (selection) 演算を行い，その結果の関係に対して CITY を結合属性とする自然

結合 (natural join) 演算を施すことに等しい．また，写像 supplied_by の定義は，上記の演算結果の関係と関係 SP に対して PNO を結合属性とする自然結合演算を施すことに等しい．したがって，SELECT-APPEND 形式の SQL 文によって関係を結合することは，データベースの1つのグラフ表現されたビュー (view) を作ることに類似している（図15の関係のグラフは，仮想的な関係であるビューと異なり SOL/SQL 処理系内部に実際に作られる）．

5. あとがき

本稿では，例題を中心に SOL と SQL の整合性の良さを示すとともに，整合性を保つための言語処理系の実現方法について述べた．ホスト言語に SOL を使用し，SQL 埋込み構文の拡張とカーソルによるデータアクセス機構の隠ぺいなどを行うことによって，本研究の目的であったホスト言語と SQL 間のインタフェースの改善がかなりの程度達成された．これは，PLI/SQL との比較でもわかるようにドキュメントとしてのプログラムの解読性の向上をもたらし，ひいてはデータベースアプリケーションプログラムの生産性と保守性の向上をもたらすものと考えられる．SOL では写像型を導入したが，この結果，データ検索を関数引用の形式で簡潔に記述できるようになった（図9，図14参照）．また，写像は処理系内部ではリンク構造で実現されているため，一度写像化すれば，SELECT 文による表の検索に比べてデータ検索効率の向上が期待される．

現在の SOL/SQL システムには，いくつかの問題点が残っている．①SELECT 文の GROUP-GY と HAVING 句については拡張を行っていないが，直交性を保つためには HAVING 句の IN 述語に集合式を記述できるようにする必要がある．②逆関数や関数合成を記述することができない．これについては，現在，アポロ版 SOL[20] で写像，逆写像，対応，逆対応の言語仕様を導入した言語処理系が動作しているので，この仕様を導入することを検討している．③合成写像と写像の足し合わせ機能の導入については今後の課題である．④集合，写像の内部データ構造の表現形式をより効率的なものに改善していくことも必要である．

現在，SOL/SQL 処理系は，IBM 4381 の VM/CMS (Release 5) 上で動作している．処理系は PLI/SQL 言語で記述してあり約7,000行である．また，端末は IBM 5540 日本語端末を使用している．図7から図14までは，いずれも IBM 5540 の出力結果である．

謝辞 システムの作成および本論文の整理に協力していただいたオムロン直方(株)の野原庄太氏と九州工業大学工学部電気工学科の岡出明紀君に感謝します．また，本論文に対する貴重なコメントを頂いた査読者に感謝します．

参考文献

1) Codd, E. F.: A Relational Model of Data for Large Shared Data Banks, *Comm. ACM*, Vol. 13, No. 6, pp. 377-387 (1970).
2) Schmidt, J. W.: Some High Level Language Constructs for Data of Type Relation, *ACM Trans. Database Syst.*, Vol. 2, No. 3, pp. 247-261 (1977).
3) 植村：データベースの基礎，p. 237，オーム社，東京 (1979).
4) Date, C. J. (藤原 (訳))：データベース・システム概論，p. 598，丸善，東京 (1984).
5) Copeland, G. and Maier, D.: Making Smalltalk a Database System, *SIGMOD Record*, Vol. 14, No. 2, pp. 316-324 (1984).
6) データベース言語 SQL, JIS X 3005-1987 (1987).
7) Atkinson, M. P. and Buneman, O. P.: Types and Persistence in Database Programming Languages, *ACM Computing Surveys*, Vol. 19. No. 2, pp. 165-190 (1987).
8) Lecluse, C. and Richard, P.: The O 2 Database Programming Language, *Proc. of the Fifteenth International Conference on Very Large Data Bases*, pp. 411-422 (1989).
9) Date, C. J. (芝野 (監訳))：標準 SQL (改訂第2版)，p. 269，トッパン，東京 (1990).
10) Date, C. J.: *An Introduction to Database Systems*, Vol. 1 (5th ed.), p. 854, Addison-Wesley, New York (1990).
11) 増永ほか：次世代データベースシステム宣言に向けて，情報処理学会研究会報告，DBS-79-8 (1990).
12) 増永：リレーショナルデータベース入門，p. 213，サイエンス社，東京 (1991).
13) IBM SQL/Data System Application Programming for VM/System Product (Release 3), Program Number 5748-xxj (1984).
14) 勝野ほか：データ工学（新技術動向)，情報処理ハンドブック，pp. 718-733，オーム社，東京 (1989).
15) 重松，吉見，吉田：集合指向言語 SOL とその言語処理系の開発，情報処理学会論文誌，Vol. 30, No. 3, pp. 357-365 (1989).
16) 與那覇，重松，吉田：集合指向言語 SOL のデータベースへの応用，第39回情報処理学会全国大

会論文集（II），pp. 1084-1085 (1989).
17) 與那覇，重松，吉田：集合指向言語 SOL のデータベースへの応用，第4回情報処理学会九州支部講演会資料，pp. 51-60 (1990).
18) 與那覇：集合指向言語のデータベースへの応用に関する研究，平成2年度九州工業大学工学部電気工学科修士論文，p. 66 (1990).
19) 重松，與那覇，吉田：集合指向言語のデータベースへの応用，第 78 回情報処理学会データベースシステム研究会資料，pp. 53-62 (1990).
20) 松浦，重松，吉田：集合指向言語 SOL の拡張とプログラムフローグラフへの応用，第4回情報処理学会九州支部研究会資料，pp. 41-50 (1990).

付録 1 データベース例

S表は供給者表であり，供給者番号 (SNO)，供給者名 (SNAME)，状態値 (STATUS)，所在場所 (CITY) を含む．P表は部品表であり，部品番号 (PNO)，部品名 (PNAME)，色 (COLOR)，重量 (WEIGHT)，保管場所 (CITY) を含む．SP 表は納入表であり，供給者 (SNO) が部品 (PNO) をどれだけ (QTY) 納入しているかを示している．COMPONENT 表は，部品の構成関係を表しており，親部品 (MAJOR_PNO) が子部品 (MINOR_PNO) をどれだけ (QUANTITY) 含んでいるかを示している．

S

SNO	SNAME	STATUS	CITY
S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

P

PNO	PNAME	COLOR	WEIGHT	CITY
P1	Nut	Red	12	London
P2	Bolt	Green	17	Paris
P3	Screw	Blue	17	Rome
P4	Screw	Red	14	London
P5	Cam	Blue	12	Paris
P6	Cog	Red	19	London

SP

SNO	PNO	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	200
S4	P2	200
S4	P4	300
S4	P5	400

COMPONENT

MAJOR_PNO	MINOR_PNO	QUANTITY
P1	P2	2
P1	P4	4
P5	P3	1
P3	P6	3
P6	P1	9
P5	P6	8
P2	P4	3

付録 2 SELECT 文，INSERT 文および UPDATE 文の拡張構文

SELECT 文の INTO および APPEND に続く〈選択相手リスト〉は次の構文で定義される．

〈選択相手リスト〉::=
　スカラ変数と組変数のコンマリスト|
　集合変数|'〈'〈写像〉のコンマリスト'〉'
〈写像〉::=
　写像名'('[列名|'['列名コンマリスト']']')'
　　'='[列名|
　　'['列名コンマリスト']'|
　　'{'[列名|'['列名コンマリスト']']'}']

WHERE 句の〈探索条件〉における〈値式〉には SOL のスカラ式を記述できる．また，IN 述語においては，次に示すように SOL の集合式を参照することができる．

　スカラ式 [NOT] IN SOL 集合式

INSERT 文の VALUES に続く〈挿入値リスト〉は次の構文で定義される．

　SOL 集合式|'〈'〈写像〉のコンマリスト'〉'

ここで，〈写像〉は SELECT 文の〈選択相手リスト〉の場合と同じ構文である．また，UPDATE 文の SET に続く〈設定句：探索〉は次の構文で定義される．

'〈'写像名'('[列名|'['列名コンマリスト']']')'
　　'='[列名|'['列名コンマリスト']']'〉'

(平成2年8月 2 日受付)
(平成4年6月12日採録)

重松 保弘（正会員）

昭和22年生．昭和45年九州工業大学工学部電子工学科卒業．昭和47年九州工業大学工学部情報工学科助手．昭和61年同講師．現在，同電気工学科助教授．工学博士．プログラミング言語，計算機ネットワークなどの研究に従事．著書「基礎 BASIC プログラミング」，「ネットワークアーキテクチャの基礎」など．訳書「ネットワークと分散処理」．電子情報通信学会会員．

與那覇 誠（正会員）

昭和40年生．昭和62年九州工業大学工学部情報工学科卒業．平成元年同大学大学院修士課程修了．同年住友金属工業(株)入社．同社情報・通信研究開発部において画像処理の研究開発に従事．現在，パターン認識やパターン検査に関する研究開発業務に従事中．特に官能検査に興味を持つ．

吉田 将（正会員）

昭和8年生．昭和33年九州工業大学電気工学科卒業．昭和35年九州大学大学院工学研究科修士課程修了．工学博士．昭和37年九州大学工学部講師，その後，九州工業大学教授，九州大学工学部教授を経て，昭和61年10月九州工業大学情報工学部長．この間，九州工業大学および九州大学情報処理教育センタ長，九州大学大型計算機センタ長を歴任．自然言語処理の研究に従事．人工知能学会，日本認知科学会，米国 ACL などの会員．