データベース・システム 78－6
（１９９０．７．１９）

集合指向言語のデータベースへの応用

重松 保弘　　與那覇 誠　　吉田 将

（九州工業大学工学部）　　（九州工業大学情報工学部）

関係データベース言語ＳＱＬをホスト言語に埋込んで使用する場合、現在のＪＩＳ規格で標準化されているホスト言語では、ＳＱＬとのデータ受渡しに対しての制限が強いという問題点があった。著者らは、ＳＱＬとの言語間インタフェースの改善をはかる目的で、集合指向言語ＳＯＬをＳＱＬのホスト言語として応用することにし、ＩＢＭ４３８１上でＳＯＬ言語処理系の拡張を行った。その結果、ＳＱＬとＳＯＬ間で、集合単位および関係単位のデータの受渡しが容易に実現できるようになった。本稿では、ＳＱＬ文の拡張埋込み仕様とＩＢＭ版ＳＯＬ言語処理系について述べるとともに、ＰＬ／ＩとＳＯＬで比較記述したプログラム例を示す。

ON THE APPLICATION OF A SET ORIENTED LANGUAGE TO DATABASE SYSTEMS

Yasuhiro Shigematsu*　　Makoto Yonaha*　　Sho Yoshida**

* Department of Electrical, Electronic and Computer Engineering
Faculty of Engineering, Kyushu Institute of Technology
Tobata, Kitakyushu 804, Japan

** Department of Artificial Intelligence
Faculty of Computer Science and Systems Engineering
Kyushu Institute of Technology
Kawazu, Iizuka 820, Japan

In Japanese Industrial Standard, four languages are used as host languages for embedding SQL host programs. However, it is not easy for host programs to exchange data with SQL statements. This is mainly because set type is not included in the specification of host languages. Thus, we decided to use SOL, which is a set oriented language we have been developed, as a host language of SQL, extended the language specification of SOL and SQL, and developed their language processors. In this paper, the extended specification of SOL and SQL, their language processors and example programs are shown.

1．まえがき

1970 年、E.F.Codd によりデータベースモデルの1つである関係データベースモデルが提案された[1]。関係モデルは、表という明確な対象の採用と数学的な厳密さにより多くの支持を得た。それ以来、関係モデルはデータベースモデルの中心的存在となってきた。関係データベースでは、データがすべて表の形で見える。この表に関する様々な処理を実現するための言語がＳＱＬ（Structured Query Language）[6]である。ＳＱＬは、1986年に米国国家規格協会（ANSI）、1987年には国際標準化機構（ISO）、さらに同年、日本工業規格（JIS）[5] により標準化が行われている。

ＳＱＬの利用形態としては、対話形式と埋込み形式がある。対話形式とは、端末から、対話環境を与えるアプリケーションを通して、データベース管理システムに処理を依頼する形式であり、埋込み形式とは、ＳＱＬ文をホスト言語に埋込み、そこからデータベース管理システムに処理を依頼する形式である。対話形式では問題はないが、埋込み形式では両言語の結合の弱さが指摘されている[6] 。別々に開発された２つの言語を同時に使うのであるから、両言語間の相性が問題となる。特に、両言語間でのデータの受渡しに関しての相性が重要である。この点において、現在のＪＩＳ規格で標準化されているホスト言語（Cobol, Fortran, Pascal, PL/I）はＳＱＬと相性がよいとは言えない。それは、ＳＱＬが集合単位の操作を行うのに対し、従来のホスト言語が、集合とその操作手順を言語仕様に含まないことに起因すると考えられる。すなわち、直接受渡すことができるデータは atom 単位に限られ、集合単位のデータの受渡しを行うためには、ホスト言語のレベルで繰り返しを含む特別な処理が必要となり、その結果プログラムが複雑になる。

この問題はホスト言語に著者らの開発した集合指向言語ＳＯＬ（Set Oriented Language）[8]を用いることにより解決できると考えられる。ＳＯＬは PASCAL の集合型について強化・拡張を行い、さらに集合間に写像の概念を導入することによって、自然なアルゴリズムの記述を可能とした言語であり、これをＳＱＬのホスト言語として使用することにより、集合単位のデータの受渡しもスムーズに行うことができるようになる。さらに、ＳＯＬの集合と写像を用いることにより、関係単位のデータの受渡しも可能となる。また同時に、正規・非正規形の変換機能を持たせることにより、ＳＱＬでは扱えない非正規形のデータが扱えるようになる。こうしたことから、ユーザ・ＳＱＬインタフェースの改善を目的として、ＳＯＬを関係データベース言語ＳＱＬのホスト言語として使用できるように、ＳＯＬ言語処理系の拡張を行うことにした。また、ＳＯＬの言語仕様を拡張し、内包集合式の記法としてＫＲＣのＺＦ式[4]に類似した記法を許すことにした。

本稿では、ＳＯＬをホスト言語としたＳＱＬ文の拡張埋込み仕様とＩＢＭ版ＳＯＬ言語処理系について述べるとともに、ＰＬ／ＩとＳＯＬ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[6]を使用する。

2．ＳＱＬ文の埋込み仕様

基本的には、従来のホスト言語の場合と同様な埋込みが可能である。従来のホスト言語に比べ、余分な手続きが省略され、制限が緩められている。また、機能拡張も行われている。

2.1 基本事項

ＳＱＬ文はＳＯＬプログラムの実行部の任意の場所に埋込むことができる。ＳＱＬ文は、ＪＩＳ規格と同様、ＳＱＬ先頭子（EXEC SQL）とセミコロン（;）で囲んで埋込む。埋込むことができる SQL/DS（VM/CMS 版）のＳＱＬ文を以下に挙げる。このうち、＊印の付いているものはＪＩＳ仕様に含まれるものである。

ACQUIRE DBSPACE	DROP SYNONYM
ALTER DBSPACE	〃 TABLE
〃 TABLE	〃 VIEW
CLOSE*	EXPLAIN
COMMENT	FETCH*
COMMIT*	GRANT*
CONNECT	INSERT*
CREATE INDEX	LABEL
〃 SYNONYM	LOCK
〃 TABLE*	OPEN*
〃 VIEW*	REVOKE
DECLARE-CURSOR*	ROLLBACK*
DELETE*	SELECT*
DROP DBSPACE	UPDATE*
〃 INDEX	WHENEVER*
〃 PROGRAM	

ＳＱＬとＳＯＬとの間では、以下のデータ型変換規則に従ったデータの受渡しが可能である。

ＳＱＬ		ＳＯＬ
SMALLINT, INTEGER	<=>	INT
DECIMAL(m,n), FLOAT	<=>	REAL
CHAR(n), VARCHAR(n), LONG VARCHAR	<=>	STR

2.2 手続きの省略

① ＳＱＬＣＡ(SQL Communication Area)はシステム領域としてサポートしているので宣言する必要はない。

② 従来のホスト言語の場合、ＳＱＬ文内で参照される変数はすべて埋込みＳＱＬ宣言節中で、他の変数と区別して宣言する必要がある。ＳＯＬでは、ＳＱＬ文内で参照される変数を特別扱いする必要はない。したがって、埋込みＳＱＬ宣言節は不要である。

③ 従来のホスト言語の場合、検索する列がＮＵＬＬ値を許す時には、ＮＵＬＬ標識が必要である。ＳＯＬでは、ＮＵＬＬ値を値として許すので、ＮＵＬＬ標識は不要である。

2.3 制限の緩和

従来のホスト言語の場合、参照できるホストの対象は変数だけである。さらに、変数は修飾されたり、添え字を付けられていてはならないという制限もある。

ＳＯＬの場合、参照可能な対象としては変数、関数、および写像がある。 参照する変数は単純な変数（X）に加え、配列要素（A[3]）や組要素（T.a）であってもよい。また、配列要素番号指定、関数の引数、写像の原像指定部にＳＯＬの式をおいてもよい。

ＳＯＬの対象が現れる場所としては、探索条件における WHERE 節と HAVING 節、UPDATE-SET 節、INSERT-VALUE 節、SELECT-INTO 節、CONNECT 文がある。これらの場所に、さきに挙げたＳＯＬの対象を明記する場合には、ＪＩＳ規格と同様に、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（：）をつける必要がある。ただし、INTO 節のように、明らかにＳＯＬの対象と分かる場合には省略できる。

2.4 例外処理指定に関する拡張機能

ＳＱＬ文実行時の例外処理は WHENEVER 文で指定する。例外処理には、次の３つの文が選択できる。

① ＣＯＮＴＩＮＵＥ	（特に何もしない）
② ＳＴＯＰ	（実行の停止）
③ ｄｏ ・・・ ｏｄ文	（ｄｏ文の実行）

従来のホスト言語の場合、例外発生時にこのような処理をさせるためには、GOTO 文を用い、ジャンプ先に処理を書かなければならない。さらに、例外処理を実行した後で元の処理に戻るようにするためには特別な処理が必要となる。

2.5 検索結果の代入に関する拡張機能

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 atom 変数にしか代入することができない。ＳＯＬの場合さらに、検索結果を組変数、集合変数、および集合と写像に代入することができる。また、検索結果を集合変数や集合と写像に追加するために SELECT-APPEND 形式を用意した。集合変数に代入する場合には、自動的に DISTINCT オプションが付加される。

次の例は、単一行の検索結果を atom、tuple 変数に代入するものである。代入時には、atom、tuple 変数を同時に使ってよい。

```
var int1 : int;
    str1 : str;
    tup1 : tupleof( a : str; b : int );
  . . . . .
EXEC SQL  SELECT SNO,SNAME,STATUS,CITY
          INTO   :tup1, :int1, :str1
          FROM   S
          WHERE  SNO='S1';
```

この例の結果は次のようになる。

```
tup1 = [ S1,Smith ]
int1 = 20
str1 = London
```

複数行の検索結果を集合変数に代入し、追加する例を次に示す。

```
var set1 : setof str;
  . . . . .
EXEC SQL SELECT SNO
         INTO   :set1
         FROM   S
         WHERE  CITY='London';
EXEC SQL SELECT SNO
         APPEND :set1
         FROM   S
         WHERE  CITY='Paris';
```

最初のＳＱＬ文で set1 の値は {S1, S4} となり、２番目のＳＱＬ文で set1 の値は {S1, S2, S3, S4} となる。

複数列から成る検索結果は集合と写像に代入することができる。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 節において明記する。例１は、写像ｆの SNO に対する像が SNAME、写像ｇの SNO に対する像が CITY となるように代入することを意味する。また、列名の代わりに列番号を指定してもよい。例えば、例１の INTO 節は INTO <f(1)=2, g(1)=3> と同等である。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;
    g : sno → city;
  . . . . .
EXEC SQL SELECT SNO, SNAME, CITY
         INTO   <f(SNO)=SNAME, g(SNO)=CITY>
         FROM   S
         WHERE  STATUS>20 ;
```

検索結果

SNO	SNAME	CITY
S3	Blake	Paris
S5	Adams	Athens

写像への代入状態

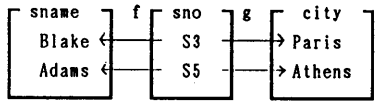


図1　例1の検索結果と写像への代入

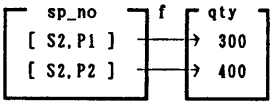
主キーが複数列の場合を例2に示す。また、実行結果を図2に示す。検索結果の追加は APPEND を使用する。

[例2]

```
var sp_no : setof tuplcof( sno, pno : str);
    qty : setof int;
map f : sp_no → qty;
  . . . .
EXEC SQL SELECT SNO, PNO, QTY
         INTO   < f([SNO, PNO])=QTY >
         FROM   SP
         WHERE  SNO='S2' ;
```

SNO	PNO	QTY
S2	P1	300
S2	P2	400

検索結果



写像への代入状態

図2　例2の検索結果と写像への代入

例3は、検索結果を非正規化して代入するものである。

[例3]

```
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  ;
```

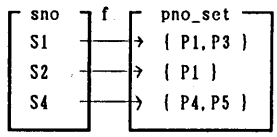
例3の実行結果を図3に示す。図3の検索結果(a)を(b)のように非正規化し、それを(c)のように集合と写像に代入する。ＳＱＬでは非正規形の表を取り扱うことはできないが、ＳＯＬを通して検索結果を見ることにより、ユーザは自然な形で検索結果を見ることができるようになる。

SNO	PN
S1	P1
S1	P3
S2	P1
S4	P4
S4	P5

＝＞

SNO	{ PNO }
S1	{ P1, P3 }
S2	{ P1 }
S4	{ P4, P5 }

（a）検索結果　（b）非正規化された検索結果



（c）代入状態

図3　非正規化変換を伴う集合と写像への代入

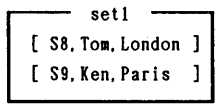
2.6 ホスト参照に関する拡張機能

WHERE 節内のＩＮ述語においては、次に示すようにＳＯＬの集合式(inp_set)の参照が可能である。

```
EXEC SQL SELECT SNAME
         INTO   :ans_set
         FROM   S
         WHERE  CITY IN :inp_set ;
```

また、INSERT 文の VALUE 節において、集合変数を参照することにより、その全要素を一括してテーブルに挿入することができる。例えば、次の INSERT 文の実行により、図4(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 :set1 ;
```



（a） 挿入集合

S

SNO	SNAME	STATUS	CITY
・	・	・	・
S8	Tom	NULL	London
S9	Ken	NULL	Paris

（b） 挿入結果

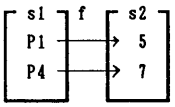
図4　集合データの挿入

UPDATE-SET 節、INSERT-INTO 節において、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。参照時の書式は SELECT 文と同様である。

次のＳＱＬ文は UPDATE-SET 節における集合と写像の参照例である。

```
EXEC SQL UPDATE P SET <f(PNO)=WEIGHT>;
```

この結果、図5(a)のデータをもとにして、(b)のように一括変換を行うことができる。



（a） 写像データ

P

PNO	WEIGHT
P1	12
P2	17
P3	17
P4	14
P5	12
P6	19

＝＞

P

PNO	WEIGHT
P1	5
P2	17
P3	17
P4	7
P5	12
P6	19

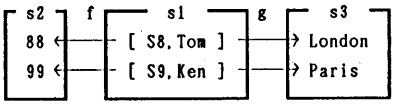
（b） 更新結果

図5　UPDATE-SET節による更新

次のＳＱＬ文は、INSERT-SET 節における集合と写像の参照例である。

```
EXEC SQL INSERT INTO S
        VALUES < f([SNO,SNAME])=STATUS,
                 g([SNO,SNAME])=CITY > ;
```

この結果、図6(a)の写像データを(b)のように一括して挿入できる。



（a） 写像データ

S

SNO	SNAME	STATUS	CITY
・	・	・	・
S8	Tom	88	London
S9	Ken	99	Paris

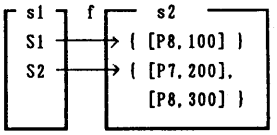
（b） 挿入結果

図6　INSERT-INTO 節による挿入

次のＳＱＬ文は、正規化変換を伴う一括挿入である。

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

この結果、図7(a)の写像データが正規化されて(b)のように挿入される。



（a）写像データ

SP

SNO	PNO	QTY
・	・	・
S1	P8	100
S2	P7	200
S2	P8	300

（b） 挿入結果

図7　正規化変換を伴う挿入

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[8]。本処理系は、この処理系をすべて PL/I 言語で記述し直したうえで、その中間コード（Ｓコード）を拡張することにより実現した[11]（拡張Ｓコードを付録２に示す）。PL/I 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埋め込み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８に、カーソル処理の手順を図９に示す。

```
SQLDA
  ├─ SQLDAID    CHAR(8)
  ├─ SQLDABC    BIN(31)
  ├─ SQLN (=n)  BIN(15)
  ├─ SQLD       BIN(15)
  ├─ SQLVAR(1)
  │     ├─ SQLTYPE    BIN(15)
  │     ├─ SQLLEN     BIN(15)
  │     ├─ SQLDATA    BIN(31)
  │     ├─ SQLIND     BIN(31)
  │     └─ SQLNAME    VAR CHAR(30)
  │
  └─ SQLVAR(n)
```

図８　ＳＱＬＤＡの構造

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。

```
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９　SELECT文の動的実行手順

図９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。① PREPARE 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。② SQLDA の SQLN 欄の値を定めた後、SQLDA 領域を動的に確保する。③ DESCRIBE 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。⑤ SQLTYPE と SQLLEN の値をもとにして、SQLVAR(1)～SQLVAR(SQLD) の SQLDATA と SQLIND を定める。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。

図１０は、次のＳＱＬ文に対する中間コードと、その実行に伴うＳＱＬ文字列の生成状態である。

```
EXEC SQL SELECT SNAME
         INTO   :ans
         FROM   S
         WHERE  STATUS IN set1 ∪ set2;
```

図１０のようにして生成されたＳＱＬ文(STRING) は、図９の手順で動的に実行される。

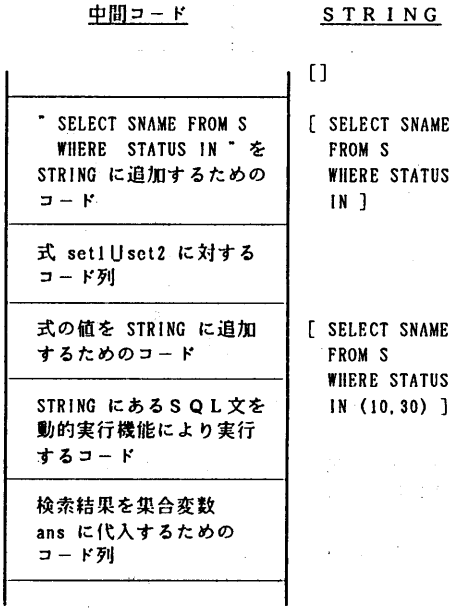


図１０　中間コードとＳＱＬ文の生成過程

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、静的に用意された動的実行機能のカーソル文に実行を代行させる。一度に複数個のカーソルを定義することを可能にするため、複数の動的実行機能のカーソル文が必要である。

3.3 ホスト参照に関する拡張機能の処理方式

集合や関係単位データの参照を伴うUPDATE（INSERT）文の実行は、実際には UPDATE（INSERT）文の繰り返し実行である。このため、動的実行するＳＱＬ文にパラメタを置き、そのパラメタ（?）に渡す値を替えながら実行を繰り返す方法を取る。埋込まれたＳＱＬ文は、次のように、動的実行可能な形に変形される。

[関係データ参照を伴う UPDATE 文の変形]

```
EXEC SQL                      UPDATE SP
  UPDATE SP             =>    SET    QTY=?
  SET <f([SNO,PNO])=QTY>;     WHERE  SNO=?
                              AND    PNO=?
```

[集合データ参照を伴う INSERT 文の変形]

```
EXEC SQL INSERT               INSERT
  INTO SP(SNO,PNO)      =>    INTO SP(SNO,PNO)
  VALUES :set1 ;              VALUES (?,?)
```

[関係データ参照を伴う INSERT 文の変形]

```
EXEC SQL INSERT               INSERT
  INTO S                =>    INTO S(SNO,CITY)
  VALUES <f(SNO)=CITY>;       VALUES (?,?)
```

4. 例題プログラム

ここでは、ＳＱＬをホスト言語としたときのプログラム例を示す。一部の例では、PL/I との比較を示す。

4.1 集合データの受渡し

次のＳＯＬ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。

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

このプログラムに入力値として{"London","Athens"}を与えると、出力として{Smith,Clark,Adams}が得られる。

これと同じ処理を行う PL/I プログラムを以下に示す。ただし、PL/I では集合型がないため都市名は一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。

```
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 :ANS:NULLIND;
      END;
      EXEC SQL CLOSE C1;
      CITYNAME='';
      GET LIST(CITYNAME);
   END;
END SUPPLIER;
```

PL/I では、検索結果を atom 単位でしかホスト変数に渡すことができない。そのためカーソル処理が必要となる。すなわち、カーソル宣言、OPEN、FETCH、CLOSEなどの処理である。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 値処理などが必要になる。

4.2 関係単位の代入

次のＳＯＬ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。

```
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 <patrs(CITY)={PNO}>
                FROM   P ;
       forall c ∈ city do
          writeln(c:10, suppl(c):12, parts(c))
       od;
   end.
```

このプログラムでは、関係単位のＳＱＬデータをＳＯＬの集合（CITY, SNO, PNO）と写像（suppl, parts）に代入している。実行結果は、次のようになる。

Athens	{ S5 }	NULL
London	{ S1,S4 }	{ P1,P4,P6 }
Paris	{ S2,S3 }	{ P2,P5 }
Rome	NULL	{ P3 }

PL/I 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ログラムとなる。

次のＳＯＬ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。

```
proc exam2;
  var parts : str;
      inset, 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.
```

SP 表のサイズが小さいときは、次のプログラムでも同じ処理が実現できる。

```
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.
```

exam3 のプログラムは、まず SP 表を集合と写像に代入し、問合せの結果は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によるプログラムを次に示す。

```
proc exam4;
  var parts : str;
      partset, 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.
```

同じ処理を PL/I で記述したものが次のプログラムである。ここでは、再帰と繰り返しにより問合せの結果を求めている。

```
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(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;
```

5．あとがき

ＳＱＬのホスト言語にＳＯＬを用いることにより、従来のホスト言語の場合と同等の埋込み機能に加え、ＳＯＬの特徴を生かした埋込み機能を実現することができた。その主なものは、① ＳＱＬ文内でＳＯＬの集合が参照できる。② 検索結果を集合変数に代入することができる。③ 関係単位のデータを参照することができる。④ 関係単位の検索結果を集合と写像に代入することができる。⑤ 埋込み方式利用時のみに現れる余分な手続き（カーソル操作、ＮＵＬＬ値処理、ホスト変数の宣言、ＳＱＬＣＡ領域の宣言）が不用、などである。この結果、ＳＯＬを用いれば、ＳＱＬのユーザインタフェースを改善できることがわかった。

ＳＯＬは、写像と述語論理を導入したという点で、Shipman によって開発された関数データモデル ＦＤＭの質問言語 Daplex[4] に類似している。したがって、ＳＯＬにＳＱＬを組み込むことは、関係モデルで実現されたデータベースを、関数表現を用いてアクセスすることを可能にするものといえる。現在のＩＢＭ版ＳＯＬでは、Daplex のように多値関数は記述できない（ただし、集合族への写像という形で記述することはできる）。しかし、アポロ版ＳＯＬでは、写像、逆写像、対応、逆対応の言語仕様が導入されており、この仕様に基づく言語処理系が Apollo DN4000, DN3000 ワークステーション上で動作している[12]。この仕様をＩＢＭ版ＳＯＬにも導入することにより、検索プログラムの記述能力の向上が期待される。

今後の課題としては、合成写像と写像の足し合わせの導入があげられる。特に、合成写像の導入は検索効率の向上に直接結び付くものと考えられる。また、集合、写像の内部データ構造の表現の改善も必要と思われる。

現在、ＩＢＭ版ＳＯＬの処理系は、 IBM 4381 の VM/CMS (Release 5) 上で動作している。処理系は PL/I 言語で記述してあり約 7000 行である。また、端末は IBM 5540 日本語端末を使用している。

謝　辞

システム作成に協力していただいた九州工業大学工学部情報工学科の野原庄太君に感謝します。

参考文献

[1] Codd, E.F. : A Relational Model of Data for Large Shared Data Banks, Comm.Acm, 13, 6, pp.377-387, (1970).

[2] 植村：データベースの基礎，p.237，オーム社，東京（1979）．
[3] 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）．
[4] P.M.D.Gray，田中 他訳：論理・代数・データベース，p.306，産業図書，東京（1990）．
[5] データベース言語ＳＱＬ，JIS X3005-1987．
[6] Date,C.J.，芝野 監訳：標準ＳＱＬ，P.247，トッパン，東京（1988）．
[7] IBM SQL/Data System Application Programming for VM/System Product (Release 3), Program Number 5748-xxj.
[8] 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)．
[9] 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，情報処理学会 第３９回全国大会講演論文集（Ⅱ），PP.1084-1085，(1989)．
[10] 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)．
[11] 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）．
[12] 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．データベースの例

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

２．　Ｓコードの拡張部

コード番号	コードの意味
１００	スタック上の値の文字列形式を生成し、動的実行するＳＱＬ文に追加する。
１０１	SELECT以外のＳＱＬ文を動的実行する。
１０２	単一行 SELECT 文を動的実行し、結果を中間バッファに得る。
１０３	中間バッファの検索結果から、パラメタ数の結果を取り出し、スタックに積む。
１０４	複数行 SELECT 文を動的実行し、結果を中間バッファに得る。
１０５	集合への代入、および追加。
１０６	集合と写像への代入、および追加。
１０７	OPEN 文の実行。
１０８	FETCH 文の実行。
１０９	CLOSE 文の実行。
１１０	COMMIT WORK 文の実行。
１１１	ROLLBACK WORK 文の実行。
１１２	例外処理指定の変更。
１１３	SQLCODE の値により例外処理を行う。
１１４	例外処理が do 文実行の時のリタンコード。
１１５	CONNECT 文の実行。
１１６	関係単位のデータ参照を伴う UPDATE 文の実行。
１１７	集合単位のデータ参照を伴う INSERT 文の実行。
１１８	関係単位のデータ参照を伴う INSERT 文の実行。