データベースのバッチ変換システム TRTS/DB について
～ MPDLコンパイラによるデータベース変換プログラムの生成 ～

米田 茂　　塚野 博史
（(株)日立製作所システム開発研究所）　（日立マイクロコンピュータエンジニアリング(株)）

1. はじめに

データベース（以下DB）は，ともすれば既存の計算機資源に，随時，アドホックに追加された資源となっている。そしてこれらのDBは体系立って整備されているとは言い難く，一つの計算機環境に幾種類ものデータベース・マネジメント・システム（以下DBMS）が存在することも決して稀ではない。このような環境はDB本来の目的であるデータの共用を阻害するばかりでなく，新しいハードウェア・ソフトウェア資源の導入に大きな障害となっている。

このような問題に対する一つの解決策が「異種データベースの統合システム」であり，我々は次の諸点を中心に研究を進めてきた。

(1) 統合用DBMSが持つべき機能の明確化
(2) 既存のDBを新たな環境に移行・統合する際の技術的問題点の解明と対応策の明確化

上記(1)については，これらの機能を，データ特性記述言語DCDL（Data Characteristic Description Language），変換手続き記述言語MPDL（Mapping Procedure Description Language）として具現化している（付録A）。この言語仕様を実現し上記(2)の技術課題を明確にするため，DBのバッチ変換システムTRTS／DBの試作を進めて来た。

本発表では，TRTS／DBの中核部でプログラム生成を担うMPDLコンパイラ，およびそれにより生成されるRestructurerについて各々の処理方式を述べる。

2. TRTS／DBの概要とMPDLコンパイラの位置付け

TRTS／DBの全体構成，および処理の流れを図2．1に示す。TRTS／DBはその構成要素として，機能別に以下の4つに大別することができる。

(1) DCDLコンパイラ
　データ構造を物理的，論理的に定義できるDCDLで記述された原始データの構造定義をテーブル形式（以下DCDLカタログ）に変換してディスク上に展開する。またこれと同様の操作を目的データに対しても行う。
(2) MPDLコンパイラ
　MPDLコンパイラは2つの部分から構成されている。すなわち，ユーザの定義した変換手続きを中間形式（以下マッピング・カタログ）に変換するモジュール群と，このマッピング・カタログとDCDLカタログから実際にDB変換を行うPL／Iプログラム，すなわちRestructurerを出力するモジュール群とである。このRestructurerをコンパイル，リンクし，実行することにより実際のDB変換が行われる。
　本報告では後者のモジュール群をMPDLコンパイラとして扱う。
(3) 内部形式変換モジュール
　TRTS／DBでは処理対象となる原始データ（DBもしくはファイル）をRestructurerが取り扱えるように内部形式にフォーマットを統一し，

それに対して変換操作を加えている。

本モジュールは，この原始データから内部形式へのフォーマット変換，およびこの逆操作である内部形式から目的データへのフォーマット変換を行う。

(4) 入出力モジュール

TRTS／DBでは全ての入出力モジュールは他のモジュールとは独立に存在する。MPDLコンパイラはこの入出力モジュールを介してDCDLカタログやマッピング・カタログとの入出力操作を行い，またRestructurerはInternal Reader／Writerを介して内部形式原始データを読み込み，変換し，内部形式目的データとして出力する。

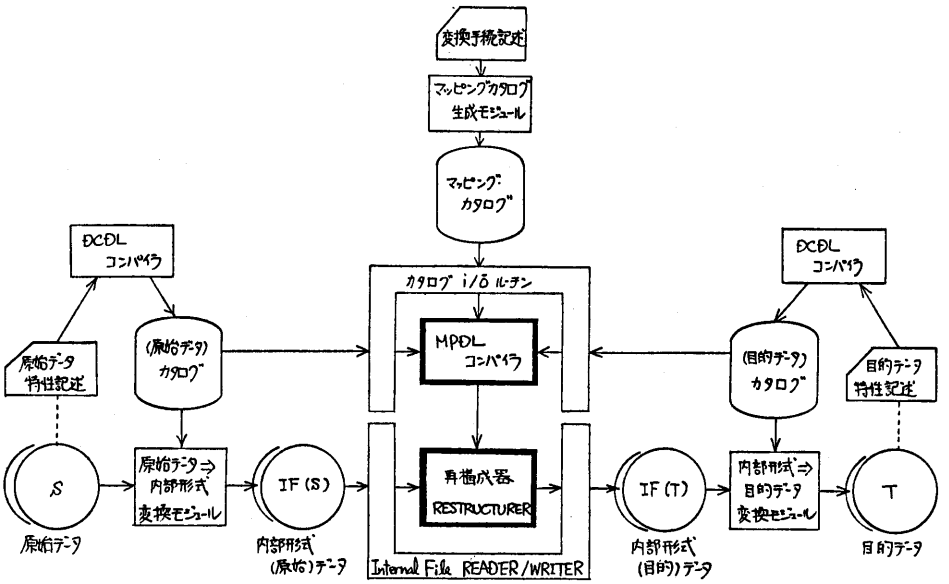

図2.1 TRTS/DBにおけるMPDLコンパイラの位置付け

3. 開発の基本方針

データベース変換システムの処理方式としては解釈実行方式とプログラム生成方式の2つが提唱されている。MPDLコンパイラの開発に当ってはこの両方式を検討し，次の理由からプログラム生成方式を採用した。

(1) 実行速度が速い

解釈実行方式では個々の処理ごとにDCDLカタログを参照し，また種々のチェックを行ないながらデータ変換を行わねばならない。これに対してプログラム生成方式ではそれらのオーバ・ヘッドは全てコンパイル処理段階にかかるため，変換処理速度のみを比較した場合，プログラム生成方式の方が速いと考えられる。

(2) プログラム生成段階の分離が可能

解釈実行方式ではユーザの記述した変換手続き定義の解釈とデータ変換の実行とが不可分であるのに対し，プログラム生成方式ではこれらの分離が可能であり，柔軟な運用が期待できる。

(3) オブジェクト言語として高級言語PL/Iを採用
高級言語を用いることにより，ユーザは独自に作成したモジュールをRestructurer中に容易に組み込むことができる。

4. MPDLコンパイラの実行方式

原始データ定義，目的データ定義を記述しDCDLコンパイラによりDCDLカタログを作成したユーザは，両者の変換手続きを記述する。MPDLコンパイラはこの手続きがマッピング・カタログとしてディスク上に展開された後，そのカタログの先頭アドレスを引数として呼び出されることにより起動する。

4.1 マッピング・カタログの読み込み

ユーザによって記述された変換手続きは，図4.1のごとく，1つのHeaderと複数のTrailerから成る一連のマッピング・カタログに展開される。マッピング・カタログはそれ自体はほとんど情報を持たず，DCDLカタログをポインタで指し示すことにより変換手続き情報を蓄えている。

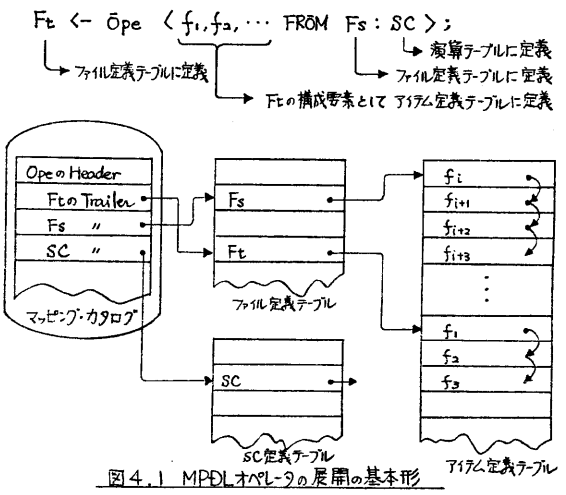

図4.1 MPDLオペレータの展開の基本形

MPDLコンパイラはHeaderを読み込むことによりそれがどのオペレーションであるかを認識し，対応するオペレーションの処理モジュールを呼び出す。各処理モジュール中で，そのHeaderに続いて存在する目的ファイルのTrailerを読み込み，この目的ファイルの種類により繰り返しグループ形式の処理モジュール，もしくはセグメント形式の処理モジュールを呼び出し，それらのモジュール内で残りのTrailer情報からユーザの定義した変換手続きを復元していく。
（繰り返しグループ，セグメント形式については後述する。）

4.2 ファイル情報の主記憶への展開

TRVS/DBではデータ定義情報をカタログとしてディスク上に持つためこれに対する入出力動作に要する時間がMPDLコンパイル処理時，最大のオーバヘッドになるものと推察された。これを回避するため，MPDLコンパイラは1つのオペレーション単位に必要となるファイルの全定義情報を主記憶中に読み込み，これに対してアクセスを行い，そのオペレーションに対するRestructurerの生成が終了した時点でそのエリアを解放する方式を採用した。

4.3 階層パスとキーの処理

TRVS/DBでは原則として階層パスを単位として処理を行う。この階層パス上のあるアイテムに対する処理の必要が生じた場合，その処理に関係するアイテムはそのアイテムの全祖先および，これらが繰り返し参照されることとな

る。またデータの抽出，すなわち内部形式の原始データから目的データへの移動は，フィールド（もしくはセグメント）を単位とすることが非常に多くしたがってこの情報を効率よく参照する必要がある。

このためＭＰＤＬコンパイラは，処理対象となっているパス情報をファイル情報とは別に，図４．２のごとくフィールド名称とキー情報から成る階層パス・イメージを主記憶中に作り出し，参照処理の効率化を図っている。

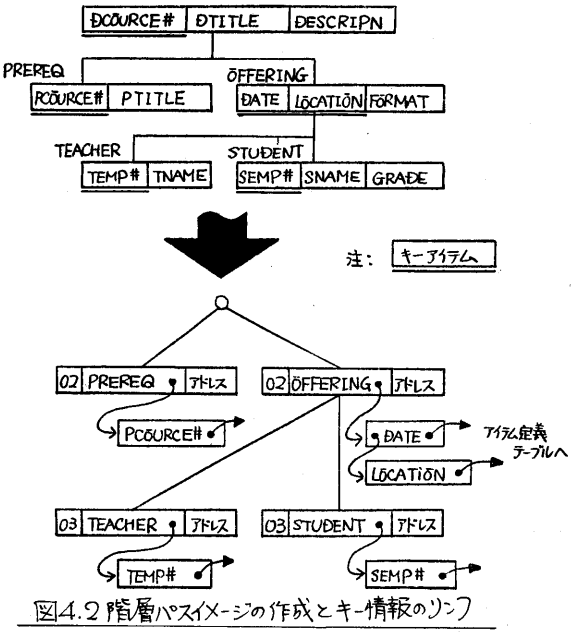

図4.2 階層パスイメージの作成とキー情報のリンク

4.4 ＭＰＤＬコンパイラの出力

Trailerを全てコア中に取り込み，変換手続きの解析・復元が完了するとＭＰＤＬコンパイラはRestructurer，すなわち変換処理を行うプログラムを出力する。

プログラムは処理単位毎，大きくはＭＰＤＬオペレータごとに，小さくはファイルの宣言，オープン・クローズ，入出力処理などにかなりの部分が予め定型化され，最適なプログラムが設計されている。ＭＰＤＬコンパイラはプログラム生成時，これらを組合せることによりRestructurerを形成していく。

データを実際に移動させるためのプログラム生成は，各オペレータにより，またファイルの形式により異るが，繰り返しグループ形式の場合を例にとると次のように処理される。

(1) データの移動を必要とするアイテムがどの階層パス上のどのレベルにあるかを探す。

(2) 階層パス・イメージを根方向にたどることにより，各レベルのフィールド名称，キー・アイテム名称，繰り返し数を得る。

(3) (2)によりDO文を階層の深さに応じてネスト形式で出力する。

(4) 対象となっているアイテムに対して，もしくはそのアイテムが存在する階層パス上のより上位のアイテムに対して，選択条件式（以下ＳＣ：Selection Criteria）が適用されているか否かを決定する。ＳＣが適用されている場合，ＩＦ文が適用され

```
GET001:                                 /*-------------------------------------*/ 00
  SYS07.FILE='SRCFILE';                 /*      ソース ファイル RECORD ヲ GET      */ 00
  CALL TRVIFM(SYS07.SRCFILE);           /*-------------------------------------*/ 00
  IF SYS07.STATUS='****'                                                        00
    THEN DO;                                                                    00
      TFILE12.DCOURCE#=SRCFILE.DCOURCE#; /* MOVE ROOT ITEM    */               00
      NO1=0;                             /* 2 カイソウ ノ COUNTER */           00
      DO I=1 TO 10;                                                             00
        IF SRCFILE.DATE(IO1)=820701 &    /* SELECTION CONDITIONS */           00
           SRCFILE.DATE(IO1)=820931                                             00
          THEN DO;                                                              00
            NO1=NO1+1;                                                          00
            TFILE12.DATE(NO1)=SRCFILE.DATE(IO1); /* MOVE 2 LEVEL ITEM */    00
            TFILE12.LOCATION(NO1)=SRCFILE.LOCATION(IO1); /*     "        */    00
            NO2=0;                       /* 3 カイソウ ノ COUNTER */          00
            DO IO2=1 TO 50;                                                     00
              NO2=NO2+1;                                                        00
              TFILE12.TEMP#(NO1,NO2)=SRCFILE.TEMP#(IO1,IO2);                    00
              TFILE12.TNAME(NO1,NO2)=SRCFILE.TNAME(IO1,IO2);                    00
              END;                                                              00
            TFILE12.TCNT(NO1)=NO2;                                              00
            END;                                                                00
          ELSE GO TO GET001;                                                    00
        END;                                                                    00
      TFILE12.OCNT=NO1;                  /* SET THE NUMBER OF 3RD ITEMS */    00
      SYS08.FILE='TFILE12';              /*-------------------------------------*/ 00
      CALL TRVIFM (SYS08.TFILE12);/*   ターゲット ファイル RECORD ヲ PUT      */ 00
      IF SYS08.STATUS='****'             /*-------------------------------------*/ 00
        THEN GO TO GET001;                                                      00
        ELSE DO;                                                                00
          CALL #TREERP(SYS08.STATUS,'0012'); /* ERROR PROCEDURE */            00
          END;                                                                  00
    END;                                                                        00
  ELSE IF SYS07.STATUS='0001'                /* END OF SOURCE FILE */         00
        THEN                                                                    00
```

図4.3 SELECTにおけるDOループの例

る条件式>文を出力した後，データ移動のためのプログラムを出力する。この場合，目的データとして構造体が定義されていればそれら全体を移動させるプログラムを生成する。個々の実現値が指定された場合は要素の値を1つ1つ移すプログラムを生成する。

(5) 上記(3)，(4)を処理対象となっている最も深いレベルのアイテムに至るまで繰り返す。

(6) このネストしたDO文を抜け出し，出力処理を行うためのGOTO文を生成する。

(7) DO文に対応したENDO文を出力する。

以上を，SELECTオペレーションを例にとり，その出力結果を図4．3に示す。

5. Restructurerのモジュール構成

5.1 Restructurerの入出力モジュール

Restructurerは，ユーザが変換手続き記述として要求したデータ変換を行う，1つの主プロシジャTRVMPDLと複数の外部プロシジャから成るPL／Iプログラムである。このプログラム中で内部形式ファイルに対するオープン・クローズ，入出力操作などは，アセンブラで記述されたRestructurer入出力専用モジュールTRVIFMを介して行う。Restructurerは所定の引数を設定してTRVIFMを呼び出すことによりファイル操作を行うため，内部形式ファイルの媒体，制御情報を全く意識する必要がない。TRVIFMはライブラリとして用意されており，リンク時にRestructurerに組み込まれる。

```
START PICKUP1
  TFILE11 <- SELECT <DCOURCE#,PCOURCE# FROM SRCFILE: DCOURCE# GT 100>:(1)
  TFILE12 <- SELECT <DCOURCE#,DATE,LOCATION,TEMP#,TNAME
                     FROM SRCFILE:
                     DATE GT 820701 AND DATE LE 820931>:(2)
  END
START PICKUP2
  TFILE21 <- SELECT <DCOURCE#,OFFERING,SEMP#,SNAME FROM SRCFILE: (3)
                     DCOURCE# GT 100 AND GRADE EQ 'A'>:
  END
START BUILDUP
  TARGET <- BIND <TFILE11 | TFILE12 | TFILE21 :
                  (TFILE11.DCOURCE# EQ TFILE12.DCOURCE#) AND
                  (TFILE11.DCOURCE# EQ TFILE21.DCOURCE#)>: (4)
  BACKUP <- TARGET: (5)
  END
```

```
TRVMPDL: PROCEDURE OPTIONS (MAIN);
  DCL PICKUP1 ENTRY;
  DCL PICKUP2 ENTRY;
  DCL BUILDUP ENTRY;
  CALL PICKUP1;
  CALL PICKUP2;
  CALL BUILTUP;
  END TRVMPDL;

PICKUP1: PROCEDURE;
  DCL SELECO1 ENTRY;
  DCL SELECO2 ENTRY;
  CALL SELECO1;
  CALL SELECO2;
  RETURN;
  END PICKUP1;

PICKUP2: PROCEDURE;
  DCL SELECO3 ENTRY;
  CALL SELECO3;
  RETURN;
  END PICKUP2;

BUILDUP: PROCEDURE;
  DCL BIND04  ENTRY;
  DCL ASSIG05 ENTRY;
  CALL BIND04;
  CALL ASSIG05;
  RETURN;
  END BUILDUP;
```

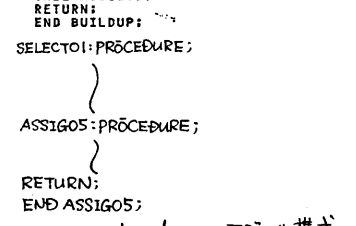

図5.1 Restructurerのモジュール構成

5.2 Restructurerの階層化

RestructurerはTRVMPDLを最上位とする3階の階層構造を成す。これはMPDLがユーザの記述上の便を考慮して，START～ENDで囲まれた複数のMPDLオペレータを1つの処理単位とし，この処理単位の集合をもって1つの変換手続き記述としていることによる。

TRVMPDLはそれに含まれる全ての処理単位に対するCALL文から成り，さらにその処理単位はそれに属するMPDLオペレータに対するCALL文から構成される。各MPDLオペレータは各々1つのPL／Iプロシジャを成す。これらのプロシジャ間には実行順序以外の依存関係は全く無く，完全に独立している。この階層関係を図5．1に示す。

このようなプロシジャの階層化を行うことにより，ユーザは，MPDLを用いて記述した変換手続き定義と，それに対応するRestructurerとの関係を明確に把握することが可能になり，独自のモジュールの追加，Restructurerの拡張などを容易に行い得る。

6. 異る階層構造データへの対応

MPDLでは異るデータ形式，すなわち繰り返しグループ，セグメントの双方を同一形式で取り扱うことが可能である。したがって全く同一のMPDL記述であっても原始データの違いにより，繰り返しグループ型データに対する変換プログラム，セグメント型データに対する変換プログラムの異ったプログラムが生成され得る。

以下にこれらがRestructurerでどのように扱われるか一般的な形式を述べる。

6.1 繰り返しグループ

繰り返しグループ型の階層構造とは図6.1に示すごとく，1つのレコード内で階層構造を実現している形式で，例えばCOBOL，PL/Iなどにおける構造体配列レコードがこれに相当する。

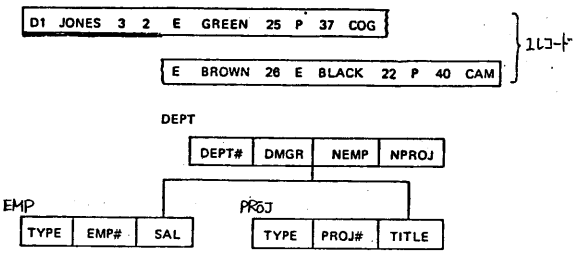

図6.1 繰り返しグループ型データの例

この形式では物理レコードの入力で主記憶中に階層構造の全実現値を取り込むため，レコード内の閉じた範囲でこれを操作し目的データを生成し得る。

繰り返し数が決定している場合，この形式の処理は一般にDOループのネスト形式で実現される（4.4参照）。

ルート・フィールドのみから成るデータ，いわゆるフラットなデータは繰り返しグループの特殊な場合と考えられる。フラットなデータの場合繰り返し部分が存在しないためDOループを用いる必要はなく，入力～処理～出力を入力終るまで繰り返す。

6.2 セグメント

繰り返しグループ型レコードが1つの物理レコードで1つのデータベース・レコード，すなわちルート・フィールドに対する階層構造全体を表現しているのに対し，セグメントは1つの物理レコードで1つのグループ値を表現するに過ぎない。

したがってTRVIFMを介してのセグメントの入力は，入力時点までそれがどのセグメントであるか不明である。このためRestructurerでは，原始ファイル中で最大のセグメント長に相当する入出力エリアを用意してそれに対してセグメントの入力を行い，このアドレス値を基底付変数として宣言したセグメント定義のポインタ変数に割り当てることにより両者の対応を図っている。

TRVIFMによる原始ファイルの入力は全て順次検索処理を前提としているため，図6.2(a)のごとく，元の階層構造を保持したままの変換には，入力レ

たセグメントの要・不要の選択を行い，条件にかなったセグメントのみを目的ファイルに出力する方式で実現できる。これに対して図6．2(b)のごとく，構造体を変えてしまう処理の場合，全階層の実現値を主記憶内に取り込むことが記憶容量の制約上不可能であるため一時ファイルの使用が不可欠である。

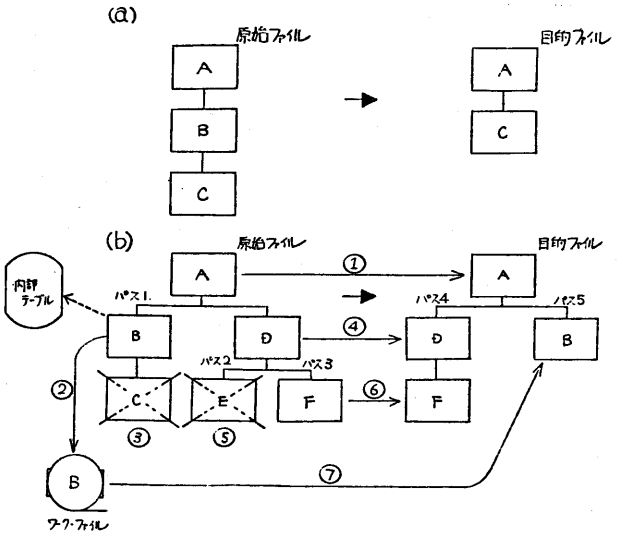

図6.2 セグメント形式のデータ処理例

図6．2(b)の処理を例にとると，次の手順により原始ファイルから目的ファイルへの変換を行う。

① ルート・セグメントを目的ファイルへ出力する。

② 一時ファイルを用いるためファイル名称を割り当て，このファイルに関する情報をDCDLカタログに登録する。パス1について，セグメントBを一時ファイルへ出力する。

③ セグメントCは目的ファイルに不要であるため除去する。

④ セグメントDは目的ファイルの階層パス上Aの次に存在するため，これを目的ファイルへ出力する。

⑤ セグメントEは目的ファイルに不要であるため除去する。

⑥ セグメントFは目的ファイルの階層パス上Dの次に存在するため，これを目的ファイルへ出力する。

⑦ ①から⑥までの操作で目的ファイルのパス4が完成する。パス5についてはセグメントBしか存在しないため，これを一時ファイルから目的ファイルへ移し，1つのDBレコードが完成する。

⑧ 一時ファイルを削除する。

以上の処理を原始ファイルの終了まで繰り返す。

7. TRVS／DBの期待効果

TRVS／DBで設定した言語機能を以上述べてきたような処理方式を持つMPDLコンパイラでPL／Iプログラムに翻訳した場合，どの程度開発工数の削減に貢献するかを，TRVS／DBとPL／Iとの比較により検討する。

表1 MPDLのPL/Iの展開 TRVSによる工数削減効果をみるため，変換規則記述言語に対応して，PL/Iステートメントが何ステート必要かを表わしている。

項番	MPDL	PL/I 文		
		DCL文	手続き文	計
1	←	11	19	30
2	SELECT	25	57	82
3	FLAT	22	45	67
4	BIND	32	85	117
5	CONSTRUCT	22	75	97
6	SORT	30	110	140
7	MERGE	32	85	117
8	HNORMAL	25	68	93
9	DROP	1	14	15

表1は1つのMPDLオペレータがMPDLコンパイラにより何ステップのPL／I文に展開されるかの対応を示したものである。これらはMPDL記述の複雑さにより大きく変化するが，数十倍の記述効率の向上が期待できることがわかる。

8. おわりに

MPDLコンパイラではその処理方式としてプログラム生成方式を採用したが，解釈実行方式の提案も幾つかなされている。

今後，本システムの性能評価を行うとともに，解釈実行方式との比較評価を行っていきたい。

9. 参考文献

1) C.J. Date : An Introduction to Database Systems : Addison-Wesley Pub. Comp. (1981)

2) James P.Fry : Conversion Technology,an Assignment : DATABASE & SIGMOD RECORD (1982.1)

3) N.Adam Rin, Maxine Brown : An Overview of a System for Automatic Generation of File Conversion Programs : SOFTWARE-PRACTICE AND EXPERIENCE Vol.5 (1975)

4) N.C.Shu, B.C.Housel他 : EXPRESS ; A Data Extraction, Processing and Restructuring System : ACM Trans (1977.6)

5) Edward W.Briss, James P.Fry : Generalized Software for Translating Data : Proc. NCC(1976)

6) John F.Burger : Semantic Database Mapping in Eufid : Proc. ACM-SIGMOD Conf. (1980)

7) 特集 第4世代コンピュータ : 日経エレクトロニクス 10.4 (1982)

8) 半田,吉田 : 複数データベースの統合・利用技術 : 日立評論 第64巻第5号 (1981)

9) N.C. Shu ,B.C. Housel他 : CONVERT : A High Level Translation Definition Language for Data Conversion : IBM Research Report (1975.2)

付録 A

表2 MPDLオペレータ一覧

項番	MPDLオペレータ	機能
1.	<——	ファイル名称,データ項目名称の割り当て
2.	SELECT	条件に合致したデータの抽出
3.	FLAT	階層構造をフラット構造へ変換
4.	BIND	複数ファイルの結合
5.	CONSTRUCT	フラット構造の階層化
6.	SORT	データのソート
7.	MERGE	複数ファイルのマージ
8.	HNORMAL	階層パス上の重複排除
9.	DROP	一時ファイルの削除
10.	組み込み関数	SUM/MAX/MIN/AVG/COUNT