

解説

階層モデル, ネットワークモデルとその実現技術†



真名垣 昌夫††

1. はじめに

1960 年頃からデータベースの概念が展開され, 特に CODASYL (Conference on Data System Language) より情報代数¹⁾が提案されて以来, データベース及びデータモデルの研究・開発が盛んに行われ, 20 年余が経過した. この間に階層モデル, ネットワークモデル, リレーショナルモデル等のデータモデルの研究²⁾が進み, これと並行して, これらの各モデルを中心としたデータベース管理システムが開発された. 現在ではデータベース管理システムのアーキテクチャは固まり, またユーザが任意に商用データベース管理システムを選択, 利用できる環境にある. 特にビジネス分野においては, 国内でも数百に及ぶデータベースシステムが稼動しており, 今後の EDPS はデータベースを抜きにして考えられない. そして, 今や図形やイメージ, 知識処理等の新分野へのデータベースの適用が試行され始め¹⁰⁾, 新しい時代に突入している. この状況において現在のデータベースの源泉であり, 普及に貢献した階層モデル及びネットワークモデルのデータベース及びその実現技術を再確認することは, 今後のデータベースの方向をさぐる上で有益かと考える.

2. 階層モデルとネットワークモデル

実世界における情報構造の表現として, エンティティを構成する属性間の関係をレコードタイプで表現し, エンティティ間の関係をレコードタイプ間の親子集合で表現する発想は 1950 年半ばから考えられていた. これは従来ファイルのフラット構造によるデータ管理上生じる冗長性の除去を実現するという考えに基づくデータモデルである. このとき, レコードタイプ間の関係が木構造の制限内で表現する場合に階層モデ

ル, ネットワーク構造まで可能とする場合にネットワークモデルと一般に呼ばれる.

2.1 階層モデル

階層モデルを中心概念とするデータ処理の歴史は古く, 1950 年代にまで逆のぼり, その基本的な考え方は『実世界を形成するエンティティ間の関係の大半は木構造で表現でき, かつ木構造が人間にとって最も理解容易である』という点に立っている. この代表的なシステムが IBM の IMS³⁾である. 以下では IMS を中心に述べる.

IMS IMS は GUAM (Generalized Update Access Method) と RATS (Remote Access Terminal System) に端を発し, 結合され作成された DB/DC (Data Base Data Communication) 機能をもつソフトウェアシステムである(図-1).

IMS においてデータ構造のクラスは, (1) アプリケーションプログラマが見るデータ構造; AP データ構造, 及び (2) データ管理者が見るデータ構造; DA データ構造及び DA 記憶構造の 2 クラスに分かれる. これらは ANSI/X3/SPARC⁴⁾の 3 層スキーマと次のように対応する.

外部スキーマ: AP データ構造

概念スキーマ: DA データ構造

内部スキーマ: DA 記憶構造

DA データ構造におけるデータ構造のタイプは, 図-2 に示すごとく関係づけられ, 物理データベースの集

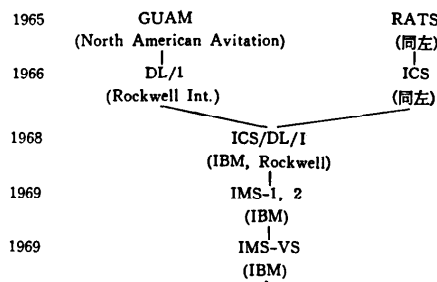


図-1 IMS 系列

† Hierarchical and Network Model and Their Implementation by Masao MANAGAKI (Application System Res. Lab. C&C Systems Res. Lab., Nippon Electric Co., Ltd.).

†† 日本電気(株) C&C システム研究所応用システム研究部

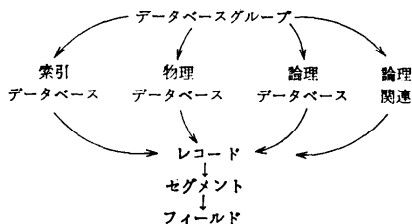


図-2 IMS データベースデータ構造

りでデータベースが形成される。データ構造の基本タイプはセグメントである。セグメントタイプの属性はフィールド・名前、○長さタイプ(固定長/可変長)、○長さまたは最小限、最大長等で定義される。セグメント間の従属関係を親子関係で表現したものがレコードタイプである。レコードタイプは1つのルートセグメントをもち、それに従属するセグメントタイプで階層的に構成される。さらにレコードタイプの枠を超えてセグメント間に論理的関係をシンボリックな値で関連づけることにより疑似的にネットワーク構造を表現することもできる。論理データベースは物理データベースに対し一部セグメント削除等の変換をほどこし論理データ構造を設定して、2次記憶に存在しない仮想データベースとして定義し利用される。以上のようなDAデータ構造の定義は次に示す形で DBDGEN と呼ばれるバッチユーティリティを用いて行われる。

```

DBD NAME=データベース名
(DATA SET, SEGM, LCHILD, FIELD,
XDFLD ステートメント)
DBGEN
FINISH
END
  
```

AP データ構造は木構造に限定され、物理データベースとは異なり仮想データベースである。この構造は以下に示す形式で各プログラム内で定義される。このとき感度 (Sensitivity) 指定によりデータ構造の操作に制限を加えることができる。

```

PCB TYPE=DB,
DBDNAME={物理データベース名
          論理データベース名}
PROCSEQ=2次インデックス名
SENSEG...
:
  
```

データベース操作において、応用プログラムは論理データベースを構成するセグメントを、ルートセグメントに始まる部分木を左から右に進むように見せられ

る。このルールに従い、IMS では次の4種のデータベース操作を提供している。

1. GET Call; 検索
 - ・GET UNIQUE (略: GU)
 - ・GET NEXT (略: GN)
 - ・GET NEXT WITHIN PARENT (略: GNP)
 - ・GET HOLD (略: GH)

2. INSERT; インスタンスの挿入
3. DELETE; インスタンスの削除
4. REPLACE; 値の変更

以上のごときデータ構造定義、操作の言語は DL/1 プロセッサの下で定義、処理される。図-3 に DL/1 コールの例を示す。図に示すように各操作言語は DL/1 Call のパラメータとして現れる。

2.2 ネットワークモデル

ネットワーク構造をもつデータモデルは GE の IDS, GM の APL のアイディアに基盤をおき、CODASYL 仕様に代表される。CODASYL はデータベースを扱う COBOL を検討するためタスクグループを発足させ、1969 年及び '71 年に CODASYL-DBTG 案を提案し、特に '71 年レポートにより今日のデータベースの基礎を築いたといえる。CODASYL 方式のデータモデルは、(1)データベースの共通言語体系を提案し、(2)'70年代のコンピュータ技術で実現可能なアーキテクチャであり、(3)COBOL, FORTRAN 等を親言語としている等の特徴をもつ。そのため、'70年代において最も盛んにその実現を目指した研究、開発がなされ、商用システムも数多く開発されるなど今日のデータベースの普及に大きな貢献をなしている。この CODASYL 仕様もデータベースの進展とともに改訂されてきている(図-4)。以下では '78 年のデータベース記述言語 DDL を中心にとりあげネットワークモデルを説明する⁶⁾。

CODASYL

CODASYL 方式のモデルにおいて、データ構造のクラスは、(1)プログラマが見るデータ構造; サブスキーマ、(2)データ管理者がみる論理データ構造; スキーマ、(3)データ管理者がみる記憶構造、アクセス構造; ストレージスキーマの3クラスに分かれ、前述の ANSI 3層スキーマに対応している。

スキーマを構成するデータ構造のタイプは、図-5 に示すように関係づけられ、表-1 に示すエンティリで形成されるスキーマ DDL (Data Description Lan-

```

DLITPLI: PROCEDURE (QUERY_PCB) OPTIONS (MAIN);
  DECLARE QUERY_PCB POINTER;
  /*Communication Buffer*/
  DECLARE 1 PCB BASED (QUERY_PCB),
    2 DATA_BASE_NAME CHAR (8),
    2 SEGMENT_LEVEL CHAR (2),
    2 STATUS_CODE CHAR (2),
    2 PROCESSING_OPTIONS CHAR (4),
  /*I/O Buffers*/
  DECLARE PRES_IO_AREA CHAR (65),
    1 PRESIDENT_DEFINED PRES_IO_AREA,
    2 PRES_NUMBER CHAR (4),
    2 PRES_NAME CHAR (20),
    2 BIRTHDATE CHAR (8),
  /*Segment Search Argument*/
  DECLARE 1 PRESIDENT_SSA STATIC UNALIGNED,
    2 SEGMENT_NAME CHAR (8) INIT ('PRES'),
    2 LEFT_PARENTHESIS CHAR (1) INIT ('('),
  /*Some necessary variables*/
  DECLARE GU CHAR (4) INIT ('GU'),
    GN CHAR (4) INIT ('GN'),
    GNP CHAR (4) INIT ('GNP'),
    FOUR_FIXED_BINARY (31) INIT (4),
    SUCCESSFUL CHAR (2) INIT (' '),
    RECORD_NOT_FOUND CHAR (2) INIT ('GE');
  /*This procedure handles IMS error conditions*/
  ERROR: PROCEDURE (ERROR_CODE);
  :
  END ERROR;
  /*Main Procedure*/
  CALL PLITDLI (FOUR, GU, QUERY_PCB, PRES_IO_AREA, PRESIDENT_SSA);
  DO WHILE (PCB.STATUS_CODE=SUCCESSFUL);
    CALL PLITDLI (FOUR, GNP, QUERY_PCB, SADMIT_IO_AREA, STATE_ADMITTED_SSA);
    DO WHILE (PCB.STATUS_CODE=SUCCESSFUL);
      PUT EDIT (STATE_NAME) (A);
    CALL PLITDLI (FOUR, GNP, QUERY_PCB, SADMIT_IO_AREA, STATE_ADMITTED_SSA);
    END;
    IF PCB.STATUS_CODE=RECORD_NOT_FOUND
      THEN DO;
        CALL ERROR (PCB.STATUS_CODE);
        RETURN;
      END;
    CALL PLITDLI (FOUR, GN, QUERY_PCB, PRES_IO_AREA, PRESIDENT_SSA);
  END;
  IF PCB.STATUS_CODE=RECORD_NOT_FOUND
    THEN DO;
      CALL ERROR (PCB.STATUS_CODE);
      RETURN;
    END;
  END DLITPLI;

```

Computing Survey, Vol. 8, No. 1, March 1976

図-3 DL/1 記述例

guage) で記述される。階層モデルと同様に、エンティティはレコードに、エンティティを構成する属性がデータアイテムに対応する。レコード内のデータ構造はフラットな構造から木構造まで表現できる。レコードタイプには複数のキーが指定でき、インスタンスを識別する手段が与えられるとともに、キーの1つを順序キーと指定してレコードを順序づけることもできる。

レコード間の関係はセットと呼ばれるレコードの親

子集合で表現され、親をオーナレコード、子をメンバレコードと呼ぶ。スキーマ DDL におけるデータ構造は、○順データ構造、○木構造、○ネットワーク構造が表現できる。さらに '78 年仕様では従来制限のあった単一サイクル (図-6 レコード 'MATITM' における 'FAMILY') が多重サイクル構造 (図-6, 'ADMC-TR', 'WORK', 'RESOUC') に加えて許されている。

しかし、CODASYL 方式では図-6 に示す顧客オーダレコード 'CSTODR' と製品レコード 'MATIT-

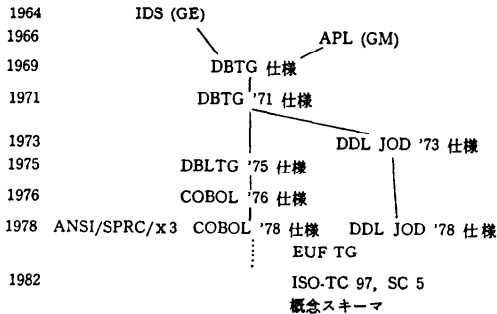


図-4 CODASYL 系列

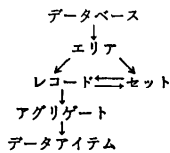


図-5 CODASYL データベースデータ構造

M' 間の $n:m$ 写像をもつレコードの関係を表現することはできず、中間にデマンドレコード 'DEMAND' のごとく関連レコードを介した疑似的表現を必要がある。現実には関連レコードに対しエンティティ間の関係が生起して始めて生じる属性が存在することが多く、問題とはならない。むしろ、CODASYL 方式のモデルの特徴は、現実的であるという以外に、リレーショナルモデル等で陽に表現できないデータのもつ意味が表現できる点にある。たとえばセット表現において、エンティティ間の抽象関係やインスタンスの存在特性を反映した AUTOMATIC/MANUAL, FIXED/MANDATORY/OPTIONAL 句等のセット関係の自動（手動）挿入／削除の指定、またデータアイテム値の生起、変更の動作を記述できるなどの意味的記述が陽になされ得る。図-7 に DDL 仕様を示す。

サブスキーマはスキーマデータ構造のサブセットであり、アプリケーションプログラム言語に依存する。データベース操作の言語として DML (Data Manip-

表-1 CODASYL DDL 構成

エントリ名	概 要
SCHEMA	・ 1 論理データベース (SCHEMA エントリから END エントリ) の名前、アクセス制御等を宣言。
AREA	・ 1 論理データベースを分割するエリアの宣言。 ・ エリア名、同時制御、アクセス制御等を宣言。
RECORD	・ レコード名、レコードとエリアの関係、アクセス制御等を宣言。 ・ レコードの構成としてサブエントリでデータアイテムを宣言。
SET	・ セット名、セットを構成するレコードの親子関係、子レコードのセット内順序関係アクセス方式、アクセス制御等を宣言。
END	・ スキーマ定義の終了宣言。

ulation Language) が与えられ、次の基本機能を提供する。

- FIND: レコードの選択
- GET: レコードアイティの検索
- MODIFY: レコードの値あるいはセットオカレンス関係の変更
- INSERT: セットへのレコードの挿入
- DELETE: レコードの削除
- STORE: レコードの生成

各プログラムは、これら DML を用いて、レコード、セット構造をたどり (Navigation)、所定のレコードをアクセスする。このために非同期データベース処理の単位である実行単位において、レコードをたどる過程の現在地を示すためにカレンシンディケータがレコードタイプ、セットタイプ、エリア、そして実行単位に1つ設けられ、この内容をもとにデータベース操作が行われる。

3. 実 現 方 式

階層モデル中心のデータベース管理システムにし、ネットワークモデルのデータベース管理システム

にし、そのシステム構造は図-8 に見るような構造をなしている。データベース管理システムとしては、各処理部のプログラム論理もしかることながら、データ構造を記憶構造にいかにか写像し実現しているかがポイントである。以下ではこのデータ構造の編成方式として、○順編成、○ポインタ方式について述べる。

(1) 順編成方式；順編成方式はデータ

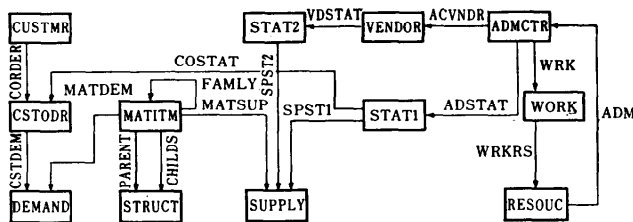


図-6 ネットワーク型データ構造例

SCHEMA NAME IS schema-name-1

[CALL procedure-name-1 [BEFORE ON ERROR DURING AFTER] [ALTER COPY DISPLAY LOCKS]]...

[ACCESS-CONTROL LOCK FOR [ALTER COPY DISPLAY LOCKS]]

IS {literal-1 lock-name-1 [PROCEDURE procedure-name-2]} OR {literal-2 lock-name-2 [PROCEDURE procedure-name-3]} ...

AREANAME IS area-name-1

[CALL procedure-name-1]
[ACCESS-CONTROL LOCK] OPEN の排他制御モードに対して.

RECORD NAME IS record-name-1

WITHIN {ANY AREA {area-name-1 ...} [AREA-ID IS parameter-name-1 [USING PROCEDURE procedure-name-1]]}
[AREA OF OWNER OF set-name-1]

[KEY key-name-1 IS [ASCENDING DESCENDING] data-identifier-1]...

DUPLICATES ARE {FIRST LAST NOT ALLOWED SYSTEM-DEFAULT}

[FREQUENCY OF [DIRECT SEQUENTIAL] RETRIEVAL IS HIGH]...

[CALL procedure-name-2]
[ACCESS-CONTROL LOCK] 各 DML に対して.

[level-number-1] data-name-1

[PICTURE IS {character-string-picture-specification-1 'numeric-picture-specification-1'} [DEPENDING ON data-identifier-1]]

[TYPE IS {BINARY DECIMAL [FIXED FLOAT] [integer-1 [integer-2]] [REAL COMPLEX] [BIT CHARACTER] [integer-3 [DEPENDING ON data-identifier-2]]}]
implementor-name

[OCCURS {integer-4 data-identifier-3} TIMES]

[CONVERSION IS NOT ALLOWED]

[CHECK IS [NONNULL PROCEDURE procedure-name-1 VALUE [NOT] [literal-1 [THRU literal-2]]] ...]

[SOURCE IS data-identifier-4 OF OWNER set-name-1 [FREQUENCY OF [RETRIEVAL UPDATE] IS HIGH]]

[RESULT OF PROCEDURE procedure-name-2 ON CHANGE TO

{ALL DATA DATA {data-identifier-5} ...} OF THIS RECORD
[ALL DATA TENANCY] OF ALL MEMBERS OF set-name-2
{ALL DATA DATA {data-identifier-6} ...} OF MEMBER record-name-1 ... OF SET set-name-3
[TENANCY]

```

[ FREQUENCY OF [ RETRIEVAL ] IS HIGH ]
[ CALL procedure-name-3
[ ACCESS-CONTROL LOCK ] GET, MODIFY, STORE に対して,
SET NAME IS set-name-1
OWNER IS { record-name-1
          SYSTEM }
ORDER IS { PERMANENT
           TEMPORARY } INSERTION IS
{ FIRST
  LAST
  NEXT
  PRIOR
  SYSTEM-DEFAULT
  SORTED
  { WITHIN RECORD-TYPE
    BY DEFINED KEYS [ RECORD-TYPE SEQUENCE IS { record-name-2 } ... ]
    { DUPPLICATES ARE { FIRST
                      LAST
                      NOT ALLOWED
                      SYSTEM-DEFAULT } } } }
[ CALL procedure-name-1
[ ACCESS-CONTROL LOCK ] FIND, INSERT, ORDER, REMOVE に対して.
MEMBER IS record-name-1
INSERTION IS { AUTOMATIC
              MANUAL } RETENTION IS { FIXED
                                       MANDATORY
                                       OPTIONAL }
[ DUPPLICATES ARE NOT ALLOWED FOR { data-identifier-1 } ... ]...
[ [ RANGE ] KEY IS { ASCENDING
                   DESCENDING } { data-identifier-2
                                RECORD-TYPE } [ , { ASCENDING
                                                    DESCENDING } { data-identifier-3
                                                                RECORD-TYPE } ] ...
  { DUPPLICATES ARE { FIRST
                    LAST
                    NOT ALLOWED
                    SYSTEM-DEFAULT } }
  NULL IS [ NOT ] ALLOWED ]
[ STRUCTURAL CONSTRAINT IS { data-identifier-4 EQUAL TO data-identifier-5 } ... ]...
SET SELECTION IS
{ THRU set-name-1 OWNER IDENTIFIED BY
  { SYSTEM
    APPLICATION
    KEY key-name-1 [ data-identifier-6 EQUAL TO { data-identifier-7
                                                  parameter-name-1 } ] ... }
  SELECTION DEFINED FOR record-name-2
  [ THEN THRU set-name-2 WHERE OWNER IDENTIFIED BY
    { data-identifier-8 [ EQUAL TO { data-identifier-9
                                   parameter-name-2 } ] } ... ] ...
  BY PROCEDURE procedure-name-1
  BY STRUCTURAL CONSTRAINT
  [ CALL procedure-name-2
  [ ACCESS-CONTROL LOCK ] FIND, INSERT, REMOVE に対して.

```

図-7 CODASYL DDL 仕様

構造の論理的順序を物理的順序を 1:1 に対応させる最も基本的な方式である。たとえば階層構造の場合、ルートセグメントから常に左下位セグメントを最優先

し順序づけを行うルールをもつことにより順編成で表現できる。この方式は記憶効率、論理的順序に従うアクセス効率はよいが、更新の激しいデータベース環

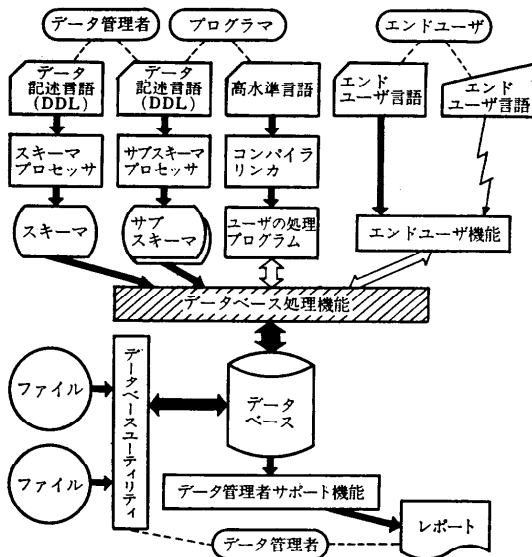


図-8 DBMS の構成

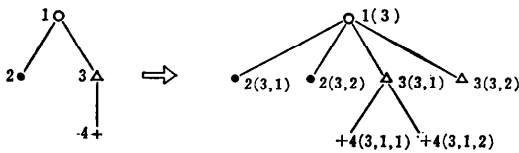


図-9 トレース方式

境、及び可変長データの取扱いにおいては、性能面での問題が生じる。この対処としては可変部にポインタを用い、固定長をベースに連結する方式がとられる。また、階層構造の場合には各レコードに対してトレースと呼ばれる識別子を与え(図-9)、これと実データ部を区別してレコード間の関係をトレーステーブル上で維持する方式もある。これにより、木構造の前後、隣接関係はトレース識別子上の上位、下位、同レベルのデジットを加減することによりアクセスできる。また実データ部においては、同一長のセグメント群で各々形成することができる。

(2) ポインタ方式; ネットワーク構造を表現するため、レコード間をシンボリックな、または論理アドレスで連結するポインタ方式が一般的に採用される。この方式では、○ポインタアレイ(前述のトレースの方式と同様)、○単方向ポインタ、○双方向ポインタ、○多重ポインタ等により親子関係が実現される。本方式の利点は次の点に集約できる。

(1) 各レコードに対し2次記憶内への配置制御が可能である。

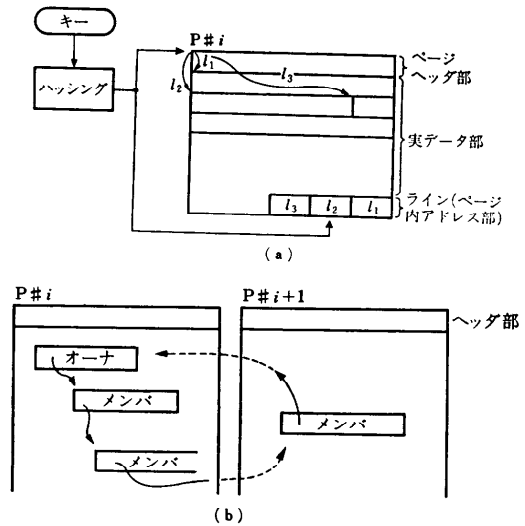


図-10 レコード格納方式

(2) 関連の挿入、削除に柔軟である。

(3) 多重インデックスの実現も容易である。

格納方式: レコードの格納方式として、2次記憶媒体と1次記憶媒体の転送単位(ページ)形式が基本となる。ページはヘッダ部、ライン部、実データ部からなる。ヘッダ部はシステムの制御情報を、ライン部はページ内レコードのアドレスを維持する。

各レコードのアドレス方式としては、(1)ハッシング(Hashing)方式(図-10(a))や(2)セットタイプごとにオーナーレコードの近傍へ配置する方式がとられる(図-10(b))。

前述の編成方式によりデータ構造を表現した各レコードは、上記アドレス方式により格納ページが決定される。これまでがデータベース固有の処理であり、

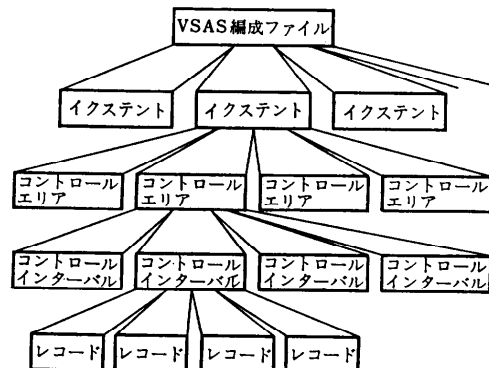


図-11 ファイル内構造

以降の記憶媒体上へのページの写像、アクセス等の管理は OS のデータ管理のもとで行われる。

データ管理は最近では B 木を拡張した B* 木の概念を導入し、データのアクセス効率、記憶効率のよいシステムが開発されている (仮想記憶媒体管理: VSAS, VSAM)⁶⁾ B* 木は次の条件を満たす B 木の葉の部分に実レコードを挿入するように拡張したものである。

B 木の条件: 節のレベルを k とする。

- (1) 根と葉以外の節の子の個数は、 $k+1$ から $k+2$ の間の個数をとる。
- (2) 根は 2 つ以上の子を持つ。
- (3) 根から葉までの経路長は全て同一である。

B* 木においては、この B 木に対し各節にキーを、葉にレコードを格納する。また、レコードの挿入、削除に伴い、動的に木の再編成を行う。

たとえば VSAS⁶⁾ では、ファイルはイクステント、コントロールエリア、コントロールインタバル、レコードからなり、次の 4 種のファイル編成に分けられる (図-11)。

- 順編成ファイル
- 相対編成ファイル
- 索引順編成ファイル
- 乱編成ファイル

図-12 に索引順及び乱編成ファイルの例を示す。VSAS における動的再編成の例を

図-13 に示す。レコードがあるコントロールインタバル (CI) に追加される場合、もし CI 内に空きスペースがなければ同一コントロールエリア中にある空き CI を探す。次に最初の CI の内容半分を空き CI 内に移し、その後追加レコードが新規 CI に格納される。もしコントロールエリア内に空き CI がなければ、コントロールイクステントに逆のばり空コントロールエリアを探す。そして最初のコントロールエリア内の

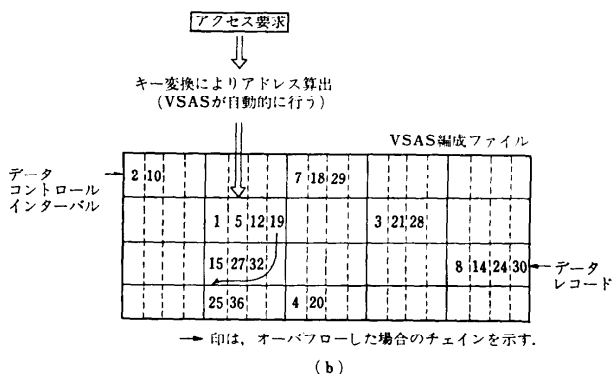
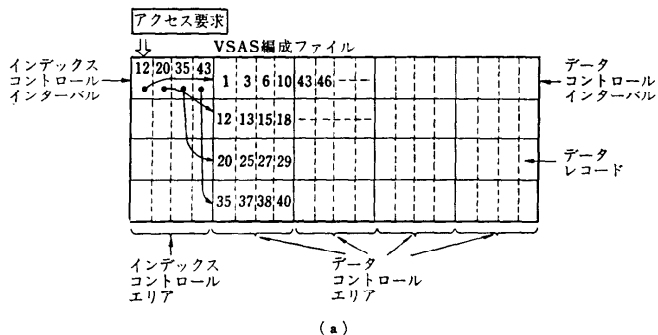


図-12 ファイル編成例

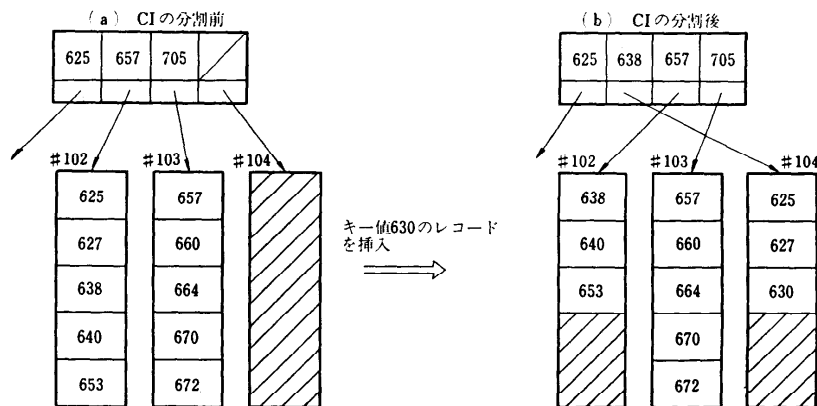


図-13 ファイル動的編成の例

半分の CI を新規コントロールエリアに移し、その後コントロールインタバルの分割を行う。このような動的再編成において、複数プロセスの同時実行制御の問題があるが、現実には頻繁に競合は生じないと思う。

4. 今後の方向

階層モデル、ネットワークモデルに基づくデータベース管理システムは、現在最もよく利用されている。両モデルはデータ構造の固さ、スキーマ設計に困難を伴うという弱点をもつが、定型業務アプリケーションに強く、各応用を考えると非常に現実的なモデルといえる。今後とも各種の分野で使用されよう。特に CODASYL 方式のモデルは、ISO SC5 TC 97 概念スキーマ (Conceptual Schema) 標準化⁹⁾の俎上にのり検討されているように、ANSI の 3 層スキーマ概念にもとづき概念スキーマの傾向を強めていき、他モデルとのインタフェースをとる方向へ進むと考えられる。また、CODASYL EUFTG⁹⁾によるエンドユーザファシリティやリレーショナルモデル等の他モデルの影響をうけると同時に C. Bachman の ROLL モデルにもとづくペア関連モデル等のごとく意味表現での充実が期待される。

参 考 文 献

- 1) CODASYL Development Committee, : An Information Algebra, Phase I Report of the Language Structure Group, Comm. ACM Vol. 5, No. 4, pp. 190-204 (1962).
- 2) ACM Computing Surveys: Special Issue, Vol. 8, No. 1 (1976).
- 3) IBM Systems Journal: Special Issue, Vol. 16, No. 2 (1977).
- 4) ANSI/X3/SPARC: The ANSI/X3/SPARC DBMS Framework, Report of the Study Group on Database Management Systems, Information Systems, Vol. 3, No. 3, pp. 173-191 (1978).
- 5) CODASYL Data Description Language Journal of Development 1978, The Secretariat of the Canadian Government EDP Standards Committee (1978).
- 6) NEC; ACOS/4 MVP データ管理マニュアル
- 7) Knuth, D. E.: The Art of Computer Programming, Vol. 3, Sorting and Searching, pp. 473-479 (1973) Addison-Wesley.
- 8) ISO/TC97/SC5/WG3 : Concepts and Terminology for the Conceptual Schema and the Information Base, March 15 (1982).
- 9) CODASYL End User Facilities Committee, Journal of Development (1980).
- 10) 植村俊亮: データベースシステムの基礎 オーム社.
- 11) 宮崎 勅: データベース管理技法, 電子科学シリーズ, 産報出版.

(昭和 57 年 6 月 21 日 受付)