講　　座

データ・ベースの実際 (2)

——保全管理システム実用化におけるチューニング法——

石　川　利　弘†

1. IMS の導入

当社は，石油製精会社であるが，他社と異なり，石油製精装置の建設は，自社で，計画から設計，現場工事，運転まで一貫して行なっている．まず LP (Linear Programming) を駆使して，基本計画を練り，基本計画を定め，この基本計画に基づいて，装置の設計が行なわれている．更に，この設計から，機械的な詳細設計が行なわれ，詳細設計から化工機メーカーに機器の発注を行ない，これとほぼ同時期に，建設現場では，基礎工事の作業に着手する．化工機メーカーに発注した機器が逐次納入され始めると，この機器の据付作業を行ない，次に配管工事に入る．そのご装置の試運転から操業に入る．

これら，一連の作業の中で，設備化される物品の発注から会計上の処理すべてを扱うシステムとして，当社では，1970 年に COMPACS-I (Construction and maintenace/Planning and Control System) を開発した．

さらに，現在では，このシステムを拡張し，Fig.20 にあるような，保全管理システム，購買管理システム，資材管理システムの実用化を行なった．

この COMPACS の開発に際し，従来のファイルの概念から一歩進めて，データ・ベースの作成を意図し，IBM 社の IMS (Information Management System) の導入に踏切った．

IMS には，Data communication feature を含んだものと，含まないものの 2 種類があるが，当社では Data communication feature のないもので，とりあえず実用化に入った．

2. IMS の機能

IMS を使った Data Base を，IBM のマニアルを読んだだけでまず設計し，実用化したのであるが，始めから，この application は，成功とはいえなかった．確かに，IMS の概念は理解して，Data Base を設計したのであるが，残念ながら IMS の文献には，効率に関することはほとんど書いていなく，処理時間のかかる application system になってしまい改良の必要が生じた．

そこで，我々は Data Base をいろいろ改良しているうちに，Data Base の access method を変えて見ると System の効率が大きく変わることに気がついた．

また IMS で logical record の構成は Hierarchical 構造しかとれないのであるが，この枝分かれしている枝の順序を変えるだけでも，ある場合には System の効率が変った．

ここで経験的なデータを含めて，簡単に IMS のデータ構造の access method 概略を記しておく．

2.1 Logical Data 構造

IMS の Data Base は Data base records の集りで，この Data base record が基本となる．Fig. 1 に SKILL に関する Data base record の logical data 構造を示しておく．

さらに，この logical data record は，いくつかの dependent segment NAME, EXPERIENCE, EDUCATION 等に細分化されている．

logical record 内にあるいくつかの segment の中で，どの segment にも従属しないもの，Fig. 1 では SKILL segment がこれに相当する．この segment を root segment という．この root segment のことをレベル 1 の segment ともいう．この root segment は，1 つの logical record 中に複数個あってはならない．

その他の segment は dependent segment という．

root segment のすぐ下位にある NAME は，SKILL と親子関係にあるといい，この NAME segment

† 東亜燃料工業株式会社

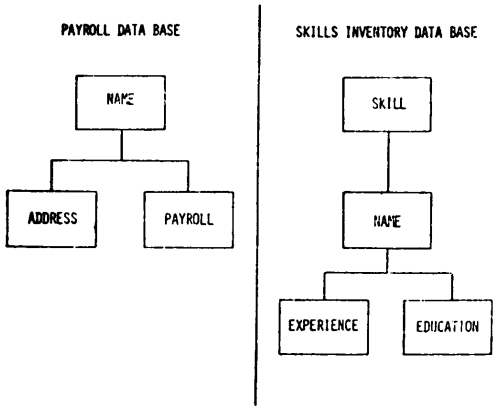


Fig. 1 Logical structures and physical data bases

をレベル2の segment という．さらに，NAME segment の子にあたる EXPERIENCE をレベル3の segment という．

Data base 設計者は，この規則を念頭に置いて，Hierarchical 構造の Data base を作成するが，構造的にリスト構造になっている logical record の data base には，IMS は不向きかも知れないが，ほとんどの data base では不便は感じないであろう．

我々の経験では，Fig. 1 の logical data base の中で，EXPERIENCE と EDUCATION の segment

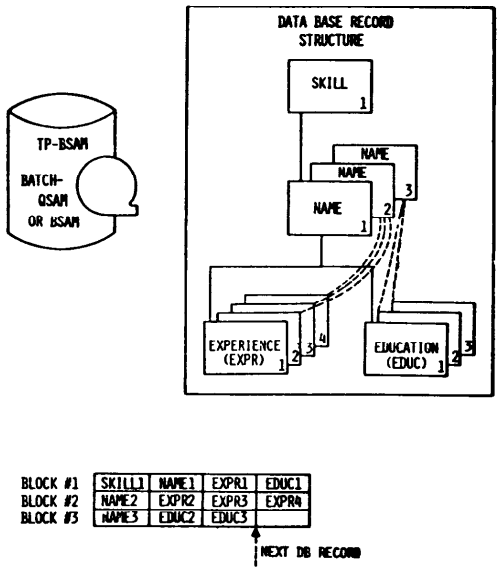


Fig. 2 HSAM physical strage of a logical structure

でどちらでも左に置き，残りを右に置くことができるのであるが，access method に依っては，data base の retrieve に同一の logical record でありながら差を生じる．もし list 構造の data base では，もっと大きな差が生じると思うので，特に我々はリスト構造にはこだわらなかった．

ほとんどのものは Hierarchical 構造の data base で表現できるであろう．

2.2 Access Method

IMS では，データ・ベースを外部記憶装置に貯える場合，次の4種類の Access method により，その記録形態は異なる．

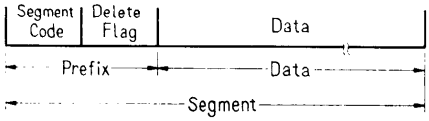
- HSAM
- HISAM
- HDAM
- HIDAM

3. Hierarchical Sequential Access Method

3.1 HSAM

前記 Fig. 1 のデータ・レコードを，HSAM を使った場合 Fig. 2 の Physical データ構造のものになる．HSAM では，階層構造の上位のセグメントから下位のセグメントへ，同位のセグメントは左から右のセグメントと順に Physical レコード上に並べられる．

HSAM の階層構造の中の1つのセグメントは，Fig. 3 のような型である．Fig. 3 のセグメントの中の prefix の Segment Code には，Root セグメントを1とし，階層順に従い連続的に上位のセグメントから下位のものへ，同位の場合には，左のセグメントから右のものへとセグメント番号をつけ，その値が入る．(他の access method の HISAM, HDAM, HIDAM でも共通のルールで，Segment Code が使われる．) Segment Code の次にある Delete Flag は，



- Prefix
  - Segment Code 1 Byte
  - Delete Flag 1 Byte
- Data

Fig. 3

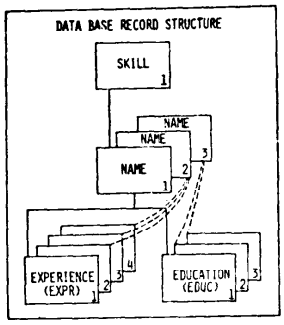
HSAM では使用していない.

この後にセグメントのデータが入る.

データ・ベースを処理する場合, 逐次処理と, ランダム処理の2通りがあるが, この HSAM のデータ・ベースは, ランダム処理は得意ではない. ランダム処理の方法は, データ・ベースの始めのレコードから全レコードを見つけるまで探索して行くか, データ・ベースの最後のレコードから逆向きに探索して行くかの2方法しかできないので, 好ましい処理方法ではない. 特にあるセグメントをいくつか更新したい場合など, ランダム処理法よりか従来の EDPS 型のファイル更新の方法がより優れている. HSAM のデータ・ベースのあるレコードの更新を施す場合, 既存の Rhysical データ・ベースでのレコードの更新ができない. また後述の logical データ・ベースの手法も使えない.

しかし, 従来のテープまたはディスク上にある Sequential データ・セットには, この HSAM のデータ・ベースに置換えるのは簡単であるので, Batch Application での用途は高いであろう. 特にファイルの設計が固まっていないで, 後からセグメントが追加されるようなデータ・ベースには有効である.

効率の面からは, HSAM データ・ベースの逐次処理は他の HISAM, HDAM, HIDAM の処理に比較すると 1/6 以下の処理時間である. その上, 記憶装置



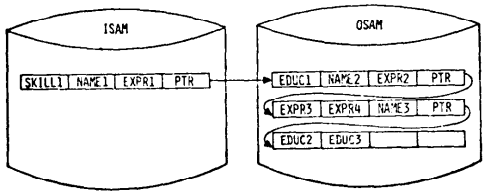
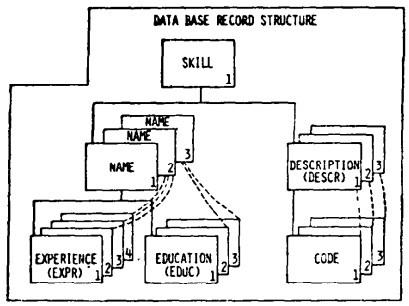


Fig. 4 HSAM physical strage of the logical data sructure-single data set group



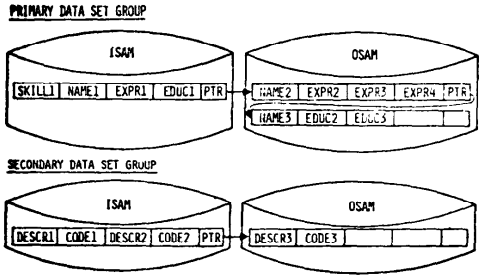
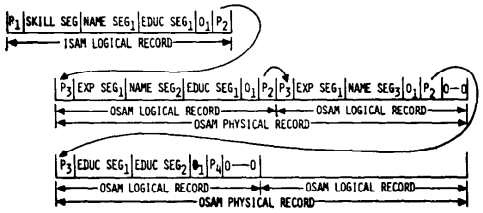


Fig. 5 HSAM physiccl strage of a logical structure-muliple data set groups



where:

P_1 = a three-byte relative block and logical record pointer to an OSAM record. The pointer in the ISAM data set points to the lowest root key inserted between this root and the prior initially loaded root. The pointer in the OSAM data set points to the next higher root in OSAM and is zero if the next root is in ISAM.

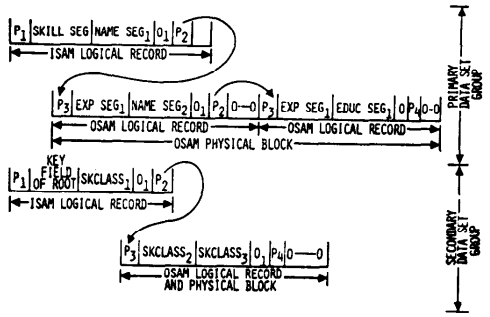
0_1 = one byte of binary zeros, indicating the end of segments within this ISAM or OSAM logical record.

P_2 = a three-byte relative block and logical record pointer to an OSAMrecord. The OSAM record contatins additional dependent segments of this data base record.

P_3 = a three-byte field of binary zeros used for record format consistency.

P_4 = a three-byte field of binary zeros indicating the absence of futher dependent segments in the data base record for this data set group.

Fig. 6 HISAM-single data set group-logical record/physical block format



P_1,…P_4 and 0_1; Same as difinition of Fig. 6

Fig. 7 HISAM-multiple data set group-logical record/physical block format

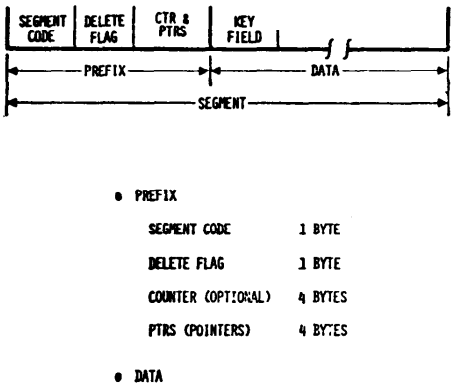


Fig. 8 HISAM segment fomat

上での Space Utilization は，HSAM が最も良い．

3.2 HISAM

Fig. 4 と Fig. 5 が HISAM を使ったときの Physical データ構造である．

Key を持った root セグメントから，階層順にセグメントが ISAM 域の logical レコード内に並べられる．(この logical レコードの大きさは，使用者がデータ・ベース作成時に定義する．) この logical レコード内から溢れたセグメントは，OSAM 域の logical レコード内に入る．さらに，この OSAM 域の logical レコードから溢れたセグメントは，次の OSAM 域の新しい logical レコード内に入る．これら logical レコード間の結合は，Fig. 6 及び Fig. 7 のように pointer で結びつけられる．

HISAM のセグメントは Fig. 8 に示してある．

Prefix の segment code は HSAM と同じである．

次の Delete flag は，このセグメントが delete されたとき Flag が立つ．但し，セグメントが delete されても DASD 上の Physical Space は，そのまま残っている．また再使用はできない．

Counter および Pointer は階層構造の logical な関係を示す direct address が入る．

データ・フィールドはセグメントの数値が入る．

HISAM のデータ・ベースの中の root セグメントには必らず1つの Key フィールドが必要で，この Key が ISAM の Key となる．dependent セグメントは Key があってもなくても良い．

HISAM のデータ・ベースは，ISAM 域と OSAM 域に分割されて記録されることは既に述べたが，いま，Retrieve したいセグメントが，何処に入っているかは，処理効率の面から眺めるとかなり大きな問題となる．ISAM 域の中のセグメントの access は早いが，(1度の I/O オペレーションで access 可能であるから) OSAM 域のセグメントで，階層構造の枝の末端にあるセグメントの access には相当 access の効率が悪くなる．したがって使用頻度の高いセグメントは階層構造の logical データを定めるとき，可能な限り root セグメントに近い所，即ち Physical レコード上では，ISAM 域に納まるよう設計するとシステム効率は大幅に良くなる．

また IMS には，データ・ベースを数個に分割する multiple データ・セットの機能が含まれている．Fig. 5 がこの例であって，特に枝分かれの多い構造のデータには I/O オペレーションを減らす有効な方法である．

HISAM は HSAM と異なり，ランダム処理は，もともと Index 部があるので適している．

しかし，この HISAM のデータ・ベースも，セグメントのそう入，削除，更新を頻繁に行なうと，pointer の付け変えや，先程述べたように，セグメントを delete しても，そのセグメントの physical space はそのまま残り再使用もできないので，使えない space が多くなる．一応 IBM 社では，これを防ぐため，時時，reorganization のための utility プログラムを準備してあるが，いずれにしても reorganization を行なわないと，space の utilization は悪くなり，retrieve の効率が悪くなる．こんな時には，HISAM より，これから述べる HDAM または HIDAM のデータ・ベースにした方が，より実際的である．

HISAM のランダム処理は HDAM, HIDAM より遅い．

HISAM の逐次処理は HDAM (特別の方法を取ら

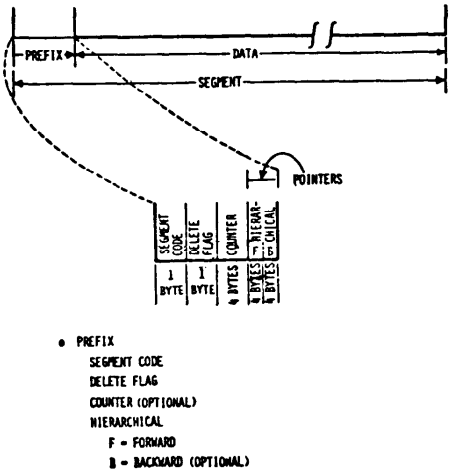


Fig. 9 HD segmet formut with hierarchie pointers

ないと逐次処理はできない), HIDAM より早い. 但し, HSAM と比較すると相当遅い.

HISAM データ・ベースを作成するとき, 即ち DASD 上に initial load しなければならないが, この initial load は HDAM より 40% 速く, HIDAM より約 50% 遅い.

4. Hierarchical Direct Access Method

4.1 Physical Pointer

HISAM および HSAM では, セグメントが並べられている順序で階層を構成していたが, HDAM, HIDAM では, いくつかのセグメントをブロックにまとめて DASD 上に記録して置くが, 各セグメント間の関係はブロックのアドレスを示す pointer でつけられる.

IMS では, 2つの Pointer 法がある.

- Hierarchical Pointer Method
- PC/PT method (Phyrical child/Phyrical Twin)

4.1.1 Hierarchical Pointer Method

HD (Hierarcical Direct) のセグメントは Fig. 9 に表わされる. この Fig. 9 の Prefix の中に counter と Hierarchial の pointer が入っているが, この Counter は, いくつ pointer がこのセグメントの中にあるか示す. Pointer には Forward と Backward の2種類ある. Fig. 10 の矢印が Hierarchical Forward Pointer の例である. Fig. 10 の中で SKILL という root セグメントには pointer が2個あり, 1つは次の SKILL セグメントを指し, 他の1つは dependent セグメントを指し, NAME は EXPERIENCE を point している.

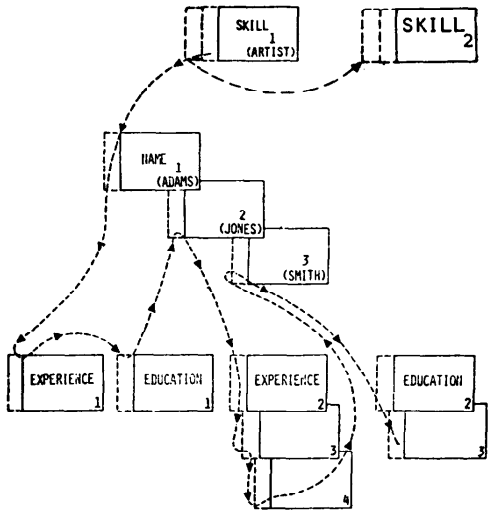


Fig. 10 Hierarchie forward pointers of a data base record

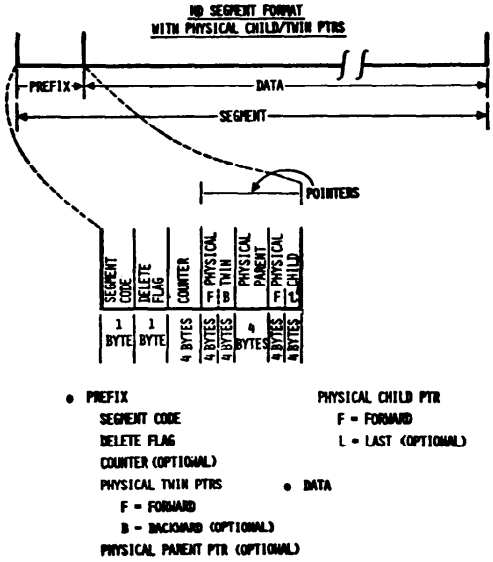


Fig. 11 HD segment format with physical child/physical twin pointers

IMS では, Hierarchical Backward Pointer (Fig. 10 の矢印を反対方向にしたもの) を指定することも可能で, この Backward Pointer により効率を上げることもできる.

4.1.2 PC/PT method

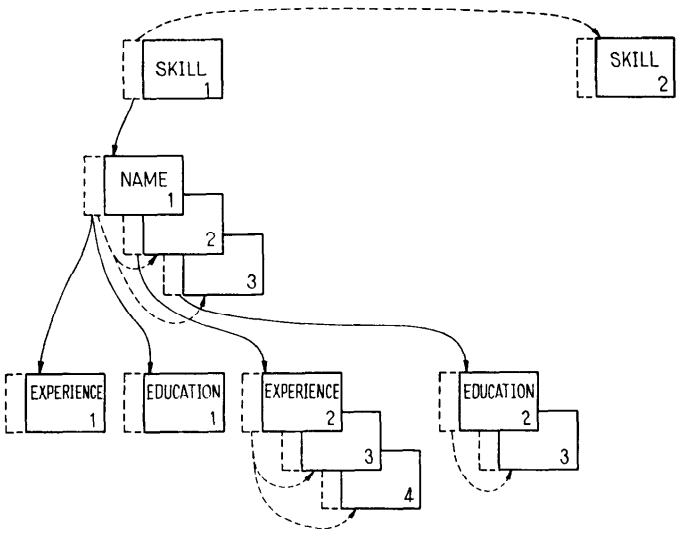


Fig. 12 PC/PT Pointer

Fig. 11 がセグメントの形である. Fig. 12 の実線の pointer が PC (Physical Child) pointer で, SKILL から NAME を point し, NAME から EXPERIENCE, EDUCATION のセグメントを指しているのが, これに概当する. 点線で示した矢印が PT (Physical Twin) pointer で, 同位のセグメントを指すもの Fig. 12 の例では NAME 間を指すものがこれにあたる.

さらにこの PC/PT には, Fig. 12 で示した最初のセグメントを指す Physical First Child Pointer と Fig. 13 に示すような Physical Last Child Pointer もある. 同様に Physical Twin Pointer にも, Forward と Backward の両方がある.

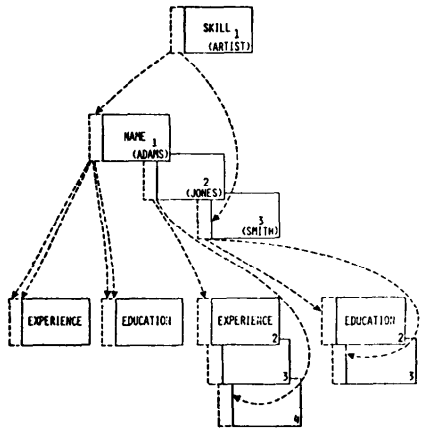


Fig. 13 First and last physical child pointers

4.2 Randomizing Module

IMS の access method のうちで, HDAM だけは特殊なプログラムをユーザーが用意しなければならない. これは Application プログラムの中で root セグメント内の Key を指定し, この Key のデータが DASD 上のどの場所に記録されているか, Key と Address を対応させる Randomizing routine を作ることである.

代表的な Randomizing Method として, IBM では下記の, 3つの代表的な方法を発表している.

- Modulo または Division Method
- Binary Having Method
- Hashing Method

4.2.1 Modulo または Division Method (Key が数字の場合)

この原理は, ある数を特定の数; x で割ったとき, その余りは, $0, 1, 2, \cdots, x-1$ であるという性質を利用している.

B…root segment area のブロック総数
A…1ブロック中の Anchor point の数
K…Key の値

とするとき,

$$L=K-\left[\frac{K}{A\times B}\right]\cdot(A\times B) \quad \text{を求め,}$$

$$\text{relative address}=\left[\frac{L}{A}\right]+1,$$

$$\text{anchor point}=L-\left[\frac{L}{A}\right]\times A+1.$$

いま, $B=50$, $A=2$ とするとき, 123, 223, 323 等は, synonyms となる. この場合, 3つの logical レコードが, 同一場所を指すことになる. この場合, anchor point で, synonyms はつなげられる. したがって anchor point の数を大きくすることは, 実質的な Blocking の増大とも見做せる.

4.2.2 Binary Having Method (Key が数の場合)

B, A, K は, 前に定めた通りとすると,

a) $N=(B\times A)/2$.

b) 次に Key の値 (K) を2進数に変換し, これを K' とする.

c) Result Register (R) を0にする.

d) K' の右端の bit を調べ，もし，この bit が1であれば，N を Result Register に加える．bit が0であれば何もしない.

e) $N/2$ を新しい N とする.

f) K' を1bit 右にシフトする.

g) N が0になるまで d)～f)を繰り返す.

h) relative address $=\left[\frac{\text{result register の値}}{A}\right]+1$

Anchor point number

$$=\frac{\text{Result register の値}}{A}\text{の余り}+1$$

4.2.3 Hashing Method (Key が文字の場合)

a) Register RN および $RN+1$ を0 にする.

b) Key の左端の 8bit の文字を RN に，ADD する.

c) RN と $RN+1$ を並べて，右に 12 Bits logical shift する.

d) RN と $RN+1$ の logical OR を取り，RN に入れる.

e) RN が奇数か，偶数かテストする．奇数ならば，RN の補数を RN に入れる．偶数のときは，そのまま.

f) $RN+1$ を0にする.

g) Key の中の次の文字を RN に logical add する.

全部の文字が処理されるまで，c)～g) を繰り返す.

h) RN の中の頭の bit を 0 にして，正の値とする.

i) この RN の中の値を Key として，modulo または Binary halving method をつかい，relative block address および anchor point を求める.

4.3 HDAM

Fig. 14 は，HDAM データ・ベースを示している．各データ・ベースは Block という Physical 単位に分けられる．Fig. 15 に block の内容を示してある．HDAM データ・ベースの始めの場所に，各々の blocks に，bit を対応させ，その bit の集りの bitmap を含んだレコードがある．この bitmap は，1つの bit に対応する Block に，データ・ベースの中で，最も大きいセグメントを記録できる Space が，block の中に，存在するときは，bit に1を立て，最大のセグメントがこの Block に入らないときは，0を立てる.(HIDAM も全く同じ bitmap の Blocks がある.)

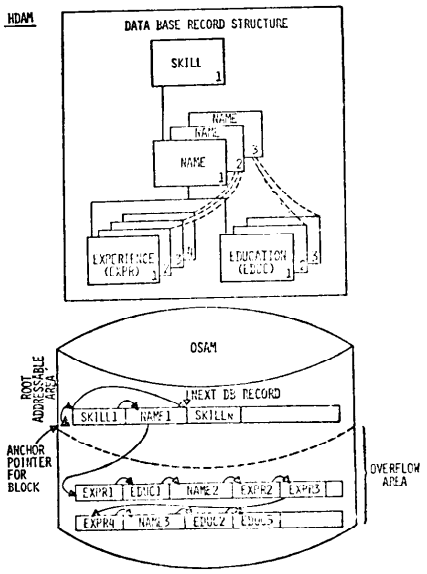


Fig. 14 Strage of hierarchical data structure in HDAM data base

もう一つのタイプの Block が，HDAM のデータ・ベースにある．この Block は次のものから成る.

- Space address (SA). Block のなかの未使用の space の relative address が入っている.
- Availale length (AL). Block の中の，使用可能

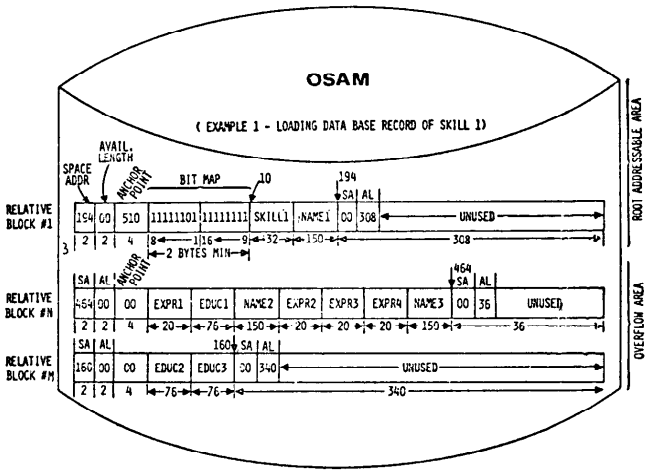


Fig. 15 Loading a data base record into HDAM data base

域の長さが入る.

• Anchor Point. Block 内にある Root セグメントの場所を示す. もし, **4.2.1** で述べたように, 複数個の Key が同一 Block および同一 anchor point を示す場合, Key 番号の最も若いものが, この anchor point のフィールドで指示される. その他の root セグメントは, セグメント内にある (Fig. 11 参照) physical twin pointer で結びつけられる.

HIDAM データ・ベースの特長としては,

• random access は, 他のどの access method より早い.

• update に関しては HISAM より優れている. 即ち, delete されたセグメントの space を再使用可能なので, HISAM と比べ, reorganization の頻度は少なくて済む.

• HISAM と同様に multiple data set group にデータ・ベースを分解し, 効率上昇が見込める.

一方, 注意すべき点としては, 次のものがある.

• HDAM データ・ベースを sequential access する場合, retrieve する root segment の順序は physical に記憶されている順であって, Key の順ではない.

• データ・ベースの総 Block 数をデータ・レコード数に比して少くすると, synonyms が大きくなり chain が増える. この場合, Block 内の anchor point の個数を増やした方が良い.

• 総 Block 数を多くすると, root セグメントを point するのは早くなるが, Space utilization が悪くなる.

• 効率を上げるために, Key の性質を利用するよう Physical 構造を考えるのも一方法である.

4.4 HIDAM

HIDAM のデータ・ベースは, Fig. 16 に示すように, INDEX 部分は ISAM 部分と OSAM 部分に分かれ, INDEX 部分に root セグメントの Key image と OSAM 部分にある root セグメントの direct address が入っている. INDEX 部分の ISAM area から溢れた index は OSAM に入る.

HIDAM の Block は HDAM で定義したものと同じで, ただ anchor point の思想はない. その他のセグメントのフィールドの定義, Pointer は HDAM と全く同じである.

HIDAM の特長としては, sequential access は HDAM より早い. また多くの Root セグメントを挿

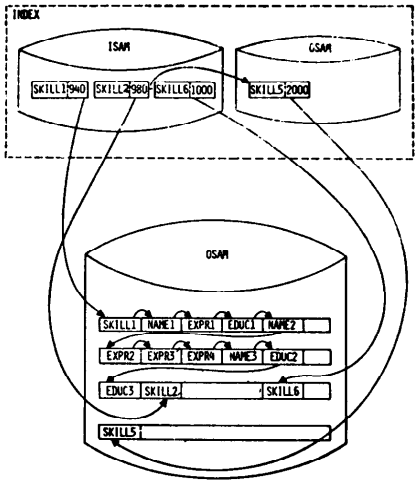


Fig. 16 Physical storage of data base records in a HIDAM data

入する場合, HDAM だと多数の synonyms が発生し, 効率が悪くなるが, HIDAM ではこれがないので HDAM より効率は良く, HISAM と比較しても update は早く, space の再使用が可能なので, reorganization の頻度は低い. 但しデータ・ベースの

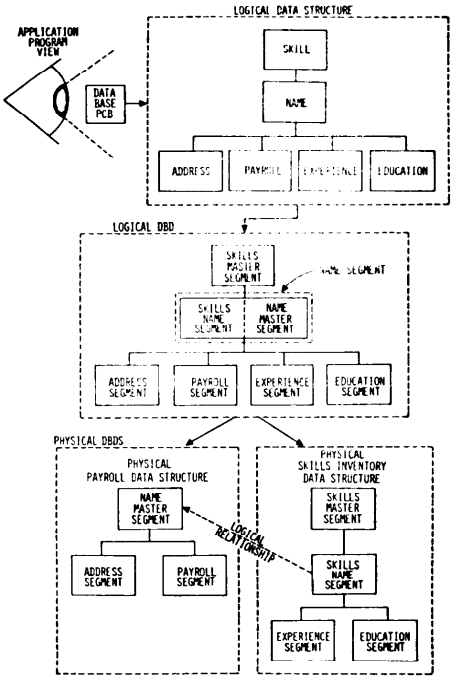


Fig. 17 Logical data structure/logical DBD/physial DBD relationships

initial load は HISAM の2倍近く掛る.

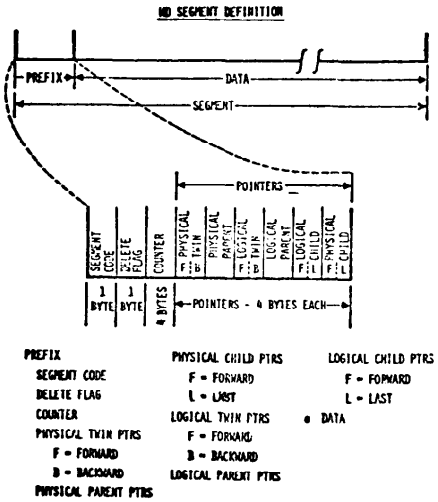


Fig. 18 HD segment format with logical pointers

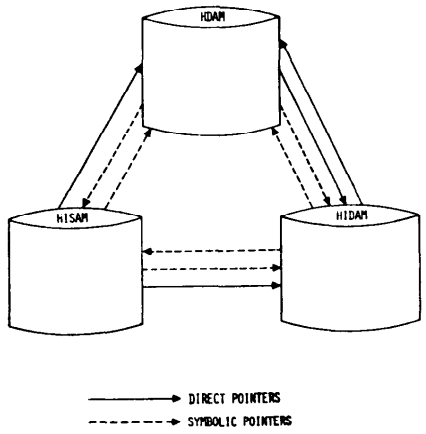


Fig. 19 Logical relationship through direct and symbolic pointers

5. Logical データ・ベース

これまでデータを階層構造に組立て，そのPhysical データ・ベースを作成するという前提であったが，あ

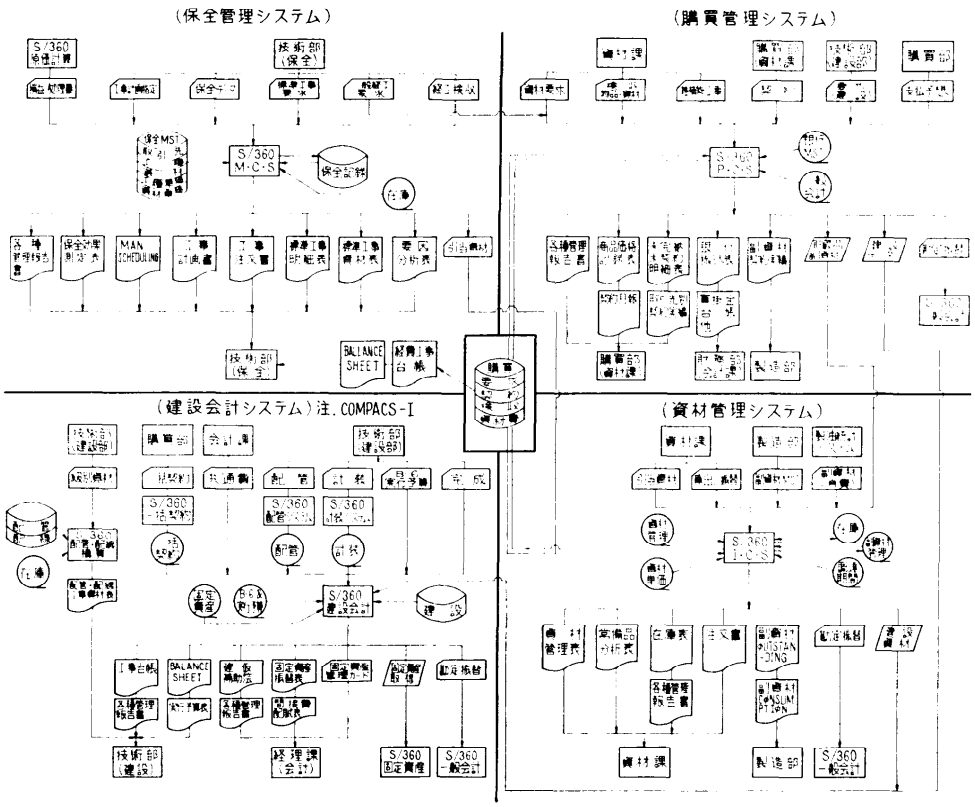


Fig. 20 COMPAS-II 概略フローチャート

る application では，Fig. 17 にあるような既存の複数個の Physical data base，即ち Payroll data base と Skills Inventory data base を1つの階層構造の data base として使いたい場合がある．いままで data base の思想の無かった時には，こんな場合最も Application に都合の良い新しい File (data base) を別に設計し，作ってしまったに違いない．その結果，同一のセグメント (data) が，複数個の physical data base の中に重複して散在するようになり，データの分散化が生じ，データの共有化が困難になってしまった．この不合理を避けるために複数個の Physieal data base から新しい logical 階層構造を設計し，各々の data base セグメントを物理的に移動しないで，あたかも新しい1つの Physical data base を定義したのと同じ効果を持たせようと IMS では Logical data base という機能が用意されている．複数個の Physical data base 中のセグメント間に階層構造を持たせるのに，Logical data base では pointer を使い logical relation をつけている．この Pointer には次の2種類がある．

{ Direct Address Pointer (Direct Pointer)
{ Symbolic Key Pointer

5.1 Direct Address Pointer

HDAM または HIDAM Physical Data base の中のセグメントを point する時に使われる．Fig. 18 の中にある Logical Twin, Logical Parent, Logical child の Field が，logical Direct Pointer である．

この Pointer の中には，HIDAM のセグメントを point するときは，前述の physical pointer と同じように relative block と relative byte 数が使われ，HDAM のセグメントを point するときは，relative block と anchor point 数が入る．

Fig. 19 の実線部分が Direct Pointer である．

5.2 Symbolic Pointer

HISAM, HDAM および HIDAM の中にあるセグメントを point するときに使われる．この pointer 中には Key が入っている．

Fig. 19 の点線部分が Symbolic Key Pointer である．

5.3 Logical data base の制約

Logical data base が，どんな形の階層構造でも作れれば良いが，かなり多くの下記のような制約があるので Logical data base を設計する際には注意を要する．制約の主なものは下記の通りである．

a) Logical data base の root セグメントは，必ず physical data base の root segment であること．

b) Physical data base のセグメント間の親子関係をある場合には逆向きにすることもできるが，この時には，必ずこのセグメント間の physical parent と physical child の両 pointer が必要である．

c) logical child には，必らず physical parent と logical parent があること．したがって root セグメントは logical child にはなれない．

d) 1つの logical child は logical parent を1つ持っていること．

e) 1つの physical data base の中のあるセグメントが，logical parent と logical child の両方を兼ねてはならない．

6. Data base の設計

IMS を利用した application として，当社では

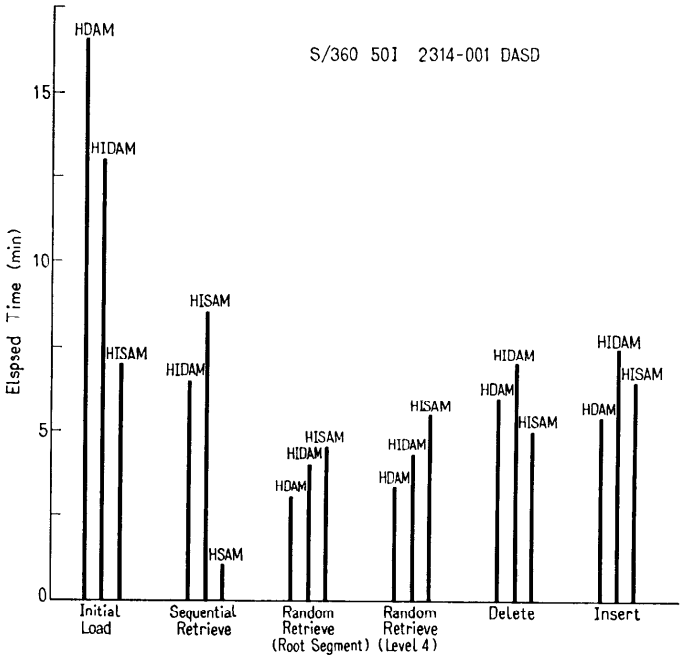


Fig. 21 IMS Performance

COMPACS (Construction and Maintenance/Planning and Control System) を開発したが，Fig. 20 に概略 Flowchart を示しておくが，当社ではぼう大なデータ・ベースを一時に完成させる考えは持っていない．この COMPACS でも，小さなデータ・ベースを設計し，これを統合する方式をとっている．しかし，プログラマーに漫然とデータ・ベースを設計させると，システムの効率は Fig. 21 にあるように access method を変えるだけで，あるものは3倍も余分に処理時間を費やすものも出てくるので，一応のデータ・ベース作成のルールとして，次のようなルールを定めている．

a）データ・ベースの階層構造は，いきなり枝分かれの多いものから始めないで，各 application に適した sub-system のデータ・ベースを作り，徐々にデータ・ベースを multi-data set groups に組み上げる．そして，効率の良いものにする．

b）access method の選択は，当社の S/360 モデル 50 以下の能力の計算機では HISAM を使ったデータ・ベースから入り，HIDAM あるものは HDAM に変えて結果を調べるのが良い．

特に access method を変えることは，application プログラムに何ら修正を施す必要がないので積極的に行なうべきである．

c）使用頻度の高いセグメントは階層構造の中で root セグメントに近い（左側）所に定義し，追加するセグメントは右側に定義した方が良い．さらに，pointer を入れてセグメントの探索時間の減少を，試みよ．但し，セグメントの追加，削除，Update に余分な時間が掛ることも念頭においておく．

d）Buffer Pool の大きさもシステム効率に，影響を与えるので，この大きさも変えて見る必要がある．

（昭和48年5月7日受付）