IMS・DBの特質と設計上のポイント

中田康雄（三菱レイヨン）

1. データ・ベース利用上の利点

(1) 当社では'75年4月よりIBMのIMS/360を全面的に活用したオンラインシステムの運用を開始した。IMS/360の機能は大きくDC機能とDB機能よりなるが，小論ではこのうちのDB機能を中心にその特質，構造，パフォーマンスおよび設計上のポイントについて述べる。活用の前提としてまずここでは，DB機能を利用することによって達成された効果の面から考えたDBの利点を，経験に基いて次の2点にあるとしてその詳細に触れておこう。

DB利用上の利点
- ダイレクト・アクセス
- 重複データ保持の回避

(2) ダイレクト・アクセス

論理レコードの集合を階層木構造のDBとして構成することによって，対象とするレコードのダイレクト・アクセスが効果的に行なえることになる。ダイレクト・アクセスの利点としては，ある1件のtransactionを対象とするDP処理を迅速に行なうことが可能になるという点で，On line real timeの不可欠の前提となることの外に，DP処理上のLogicの簡易化をもたらすと謂う利点を挙げることができる。これは例えば，「在庫マスター」が在庫場所別にソートされていて，「在庫場所〈A〉から〈B〉への転送を1件のInvoiceとしてのtransactionによって認識しなければならないとするアプリケーションを考えてみると，通常のsequential処理では，例えばこのtransactionを2件に分解し他のtransactionと共に「マスター」と同一のKeyにてSortingしたのち「在庫マスター」とMatchingすると謂う如き工夫をしなければならない。これに対して在庫マスターをDBとして保持していればこうした類のエ夫は不要となる。即ちダイレクト・アクセスはrandomに発生するtransactionを予めsortingすることなしに直接master fileとmatchingするうえで，transaction 1件あたりの処理スピードを上げるだけでなく，matchingのLogicを簡易化する意味に於てDP設計上の効率化をもたらし，場合によっては線合的な効率化を結果せしめる迄にもなりうる。この意味で，DBの利用は従来のmaster file中心のDP処理からの解放という「自由」をもたらすことになるであろう。

(3) 重複データ保持の回避

ここでの重複データは2つの意味を持つ。1つはある論理レコードの集合を考えたときその集合内での重複である。例えば販売取引という論理レコードを考えると，これは契約情報と出荷情報とのサブ集合が存在し，そこで1件の契約に対して出荷が複数に分割されるとき，1件の出荷レコードに対して常に同一内容の契約情報を結合してレコード化すると，出荷の件数だけ重複して契約情報が保持されることになる。これに対してDBを利用すれば，契

納と出荷とを階層木構造の中で位置づけることによってデータの重複を回避できる。(図-1参照)

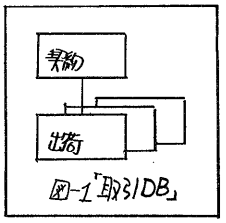


図-1「取引DB」

他方通常のSequential処理によるDP処理ではアプリケーションの目的に応じてFileを作成することによるデータの重複が避け難いことがある。例えば品名別の在庫とその受払を構成する取引をFile化するばあい，図-1の取引DBに関する情報をこの「品名別受払File」に取り込むことが必要になる。しかしもしDBを利用することによって，「取引DB」と「在庫DB」なる2つのDBを構成し，その間を何らかの手段で結合すれば，データの重複なしに新たに「品名別受払情報」を利用することが可能になる。(図-2参照)

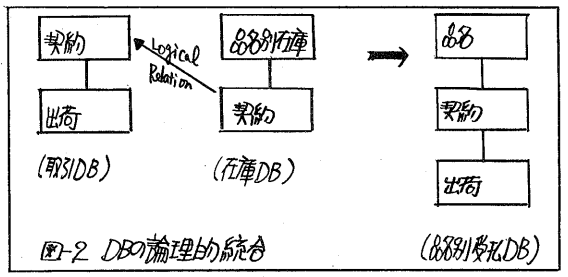


図-2 DBの論理的結合

このDBの論理的関連付けの手段は大きくIMSが用意している論理DBの機能を利用することと，更にUSERが独自にこの論理関係をアプリケーションによってmaintenanceすることの2つある。前者は論理関係を維持するDirect PointerまたはSymbolic pointerを利用することで可能となり，後者は例えば図-2の「在庫DB」の「契約セグメント」に取引DBの「契約セグメント」のKey部分だけを持ってこの関連付けをユーザーが管理することで可能となる。当然のことながらこのこともまたDBのダイレクト・アクセスの利点を利用することによってはじめて可能になる。

2. IMS・DBの特質と構造

(1) 序

DB利用上の実際的利点は特にそのダイレクト・アクセス機能に負う処が大きである。IMS・DBはそのアクセスメソッドとしてHSAM, HISAM, HIDAM, HDAMを準備しているが，ダイレクト・アクセスについてはHDAMのアクセスメソッドが優れている。ここでは，HDAMについてその特質と構造を述べることでダイレクト・アクセス機能がどのようにして可能となるかを見る。

(2) HDAM DBの特質

1. DB上の論理データ・ベース・レコードはセグメントを上から下に，左から右にポインターを用いて順次展開することで，その階層的関連づけがなされる。このポインターは次の種類がある。

Pointerの種類	Application	Space	Retrieve Speed
Hierarchical Pointer	Sequential処理に向く	節約できる	大
Direct Pointer ┌ Physical Child └ Physical Twin	Direct処理に向く	節約できない	小

Direct Pointerを利用すると，Dependent Segmentの検索がDirectに行なえるようになり，この意味で全体的に検索スピードが小さくて済むことになる。

D. Root Segment の Addressing はユーザーが準備するサブルーチン (Randomizer) によって行なわれる。Address は BLOCK NO と ANCHOR POINT NO とによって決められる。Anchor Point は各 Block 内に保持され Block NO の Sub NO 的な意味を持ち，この Anchor Point の中に，そこに Addressing された Root Segment が Store される Address が格納される。従って，Root Segment の Addressable Point は Block NO × Anchor Point NO の範囲内にある。以上の関係を図示すると図-3のようになる。

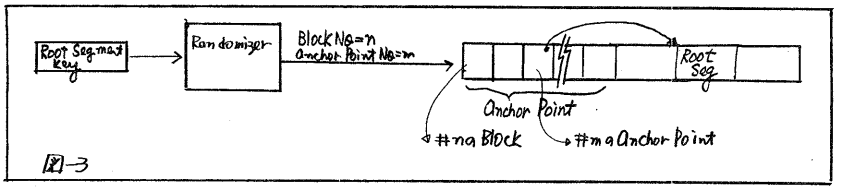


図-3

ハ. Data Set は大きく Root Segment Addressable Area (R.A.A.) と Overflow Area (OVFL) とに分けられる。R.A.A. の大きさは上に述べた Root Segment が Addressing される Block NO の最大値によって規定される。

ニ. HDAM を利用するばあい USER は DBD よび "RMNAME=" のパラメーターによって次の情報を与えねばならない。

(イ) Randomizer の Module Name
(ロ) R.A.A. の Total Size (Block数)
(ハ) Anchor Point の数
(ニ) Data Base Record のうち R.A.A. に連続して何 bytes を収容するか.

(3) HDAM の構造

イ. Bit Map

DB の第1 Block は常に Bit Map として利用される。bit map の各 bit は，それに対応する Block の利用状況を示している。bit map 内の bit の数を N とすると 1 bit map は N block の利用状況を示すことになる。data Set が N block を越えた block size を持つばあい N+1 番目の block に別の bit map が作られる。bit map の format は図-4 に見るとおりである。第1 word はこの block が bit map であることを示す flag を収容している。この次に位置するのは Anchor Point であり，Anchor Point に続く Area が bit map である。bitmap の各 bit が 1 であることは，その block にその DB の最大セグメントを収容する余地のあることを示している。従って bit map の第1 bit は bit map の block に対応するので常に ゼロ である。

D. Space Address / Available Length

Data Block の初めには 2 bytes の Space address がとられる。これによってこの block 内の初めの Space Area の始まる位置が示される。この Space Address に続く 2 bytes はその Space Area の大きさを示している。

ハ. Anchor Point

Available Length を示す Field に続く Area は n×4 bytes の Anchor Point Field である。ある block の anchor point はその block に randomizer を通して Addressing される root segment のうちの，最小の Key を持つ root segment が収容されて

いる address が保持される。即ち，複数の root segment が同一の block，同一の anchor point へ addressing されたとき（Synonym の発生），そのうちの最小の key を持つ root segment の address が anchor point へ収容される。Synonym をなす複数の root segment は，この anchor point を起点として，physical twin pointer によって，ascending に chain される。

二. Free Space

ユーザーは，Free Space として ある block の何%かを，又はいくつかの block 毎に1つの block の全体を確保することができる。これによって，DB の Load 時又は再編成時に，規則的に data Area の一部分を space Area として確保し，その後のセグメントの追加に於て，同一のDBレコードが同一の Block 内に収容される確率を高めることが可能になる。この Free Space を規定する factor は次のように呼ばれている。

fbff : free block frequency factor
fspf : free space percentage factor

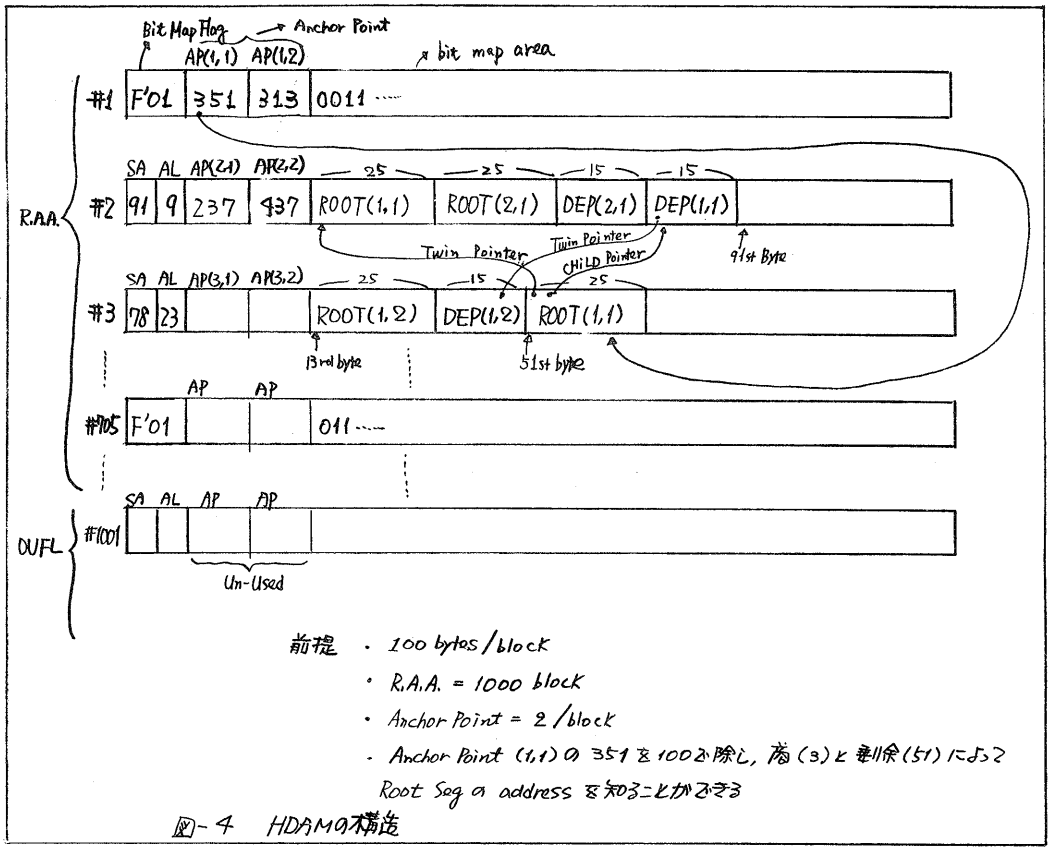


図-4 HDAM の構造

(4) Segment 追加（Insertion）のルール

新しい Segment を Insert するばあいは，その Segment に関連する既に存在する Segment に最も近い空き Area に収容されるが，その場所を選択するルールは次のようになっている。

イ. 関連Segmentの収容されている blockにSpaceがあり，しかもNew Segmentを Insertしてもその blockの bit mapが変わらないばあい（即ち Insertしたのちになおその DBの最大 Segmentを収容する余地のあるばあい），次の優先順に条件を充たす Spaceに収容する。

(イ) New Segmentに等しい大きさの Space

(ロ) New Segmentの2倍に等しい大きさの space

(ハ) New Segment＋最小 Segmentの大きさに等しい space

(ニ) New Segmentより大きい space

ロ. 関連Segmentの収容されている blockにSpaceはあるが，New Segmentを収容するとその bit mapが変わるばあい，次の優先順に条件を充たす Spaceを選択する

(イ) New Segmentの2倍に等しい大きさの Space

(ロ) New Segment＋最小 segmentの大きさに等しい space

(ハ) New Segmentより大きい space

ハ. 関連Segmentの収容されていると同一の track内のいずれかの blockで，現在 buffer pool中に存在する block

ニ. 関連 segmentの収容されていると同一の cylinder内のいずれかの blockで，現在 buffer poo中存在する block

ホ. 最大の Segment Lengthを収容している track内のいずれかの block

ヘ. 最大の Segment Lengthを収容している cylinder内のいずれかの block

ト. 関連 Segmentを収容している Cylinderを中心に±n cylinderにあるいずれかの blockで現在 buffer pool中にある block

チ. 最大の segment lengthを収容している cylinderを中心に±n cylinderにあるいずれかの block

リ. Data Setの最後の blockで buffer pool中にある block.

ヌ. Data Setの最後の blockのいずれかを bit mapを手掛りに選択

3. IMS・DBのパフォーマンス

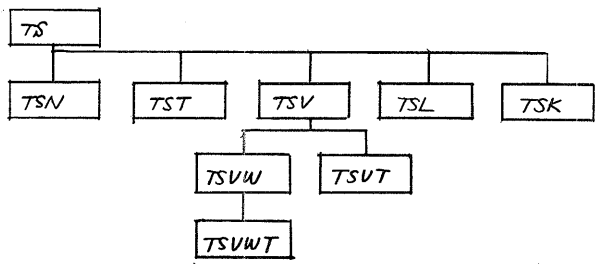
(1) IMS・DBを利用したアプリケーションを設計するうえで，DB利用上の各機能に関するパフォーマンスを知ることは不可欠の前提であり，当社ではこれを次の条件の下で測定した。

イ. Machine : IBM/370 M145 (700K bytes)

ロ. DISK UNIT: 3333-1

ハ. DB Buffer Size: 10KB

ニ. DB STRUCTURE



木. Segment Length / Occurrence / Key Length

Seg Name	S.L.	K.L.	Occurrence
TS	190	6	3000
TSN	30	6	1
TST	50	6	1
TSV	130	6	2
TSVW	170	6	2
TSVWT	50	6	1
TSVT	50	6	1
TSL	15	6	1
TSK	15	6	1

(註)
TS の Key Field は
Initial Value = 000005
Increment = 000005

1. OS : VS 2.6

(2) 測定結果は次の通り. 表-1. CALL Function別測定値

CASE			1		2		3		4		5		6		7		8	
CALL Function / Factor			GU TS		GU TS/TSV		GN TSVW		GNP TSVW		GHN TSVW		REPL TSVW		ISRT TS		ISRT TS	
			Total	AVE	Total	AVE	TOT	AVE	TOT	AVE	TOT	AVE	TOT	AVE	TOT	AVE	TOT	AVE
HIDAM	CPU TIME	OS ms	1624	16	1628	16	6	3	7	3	6	3	17	9	6960	69	7271	72
		IMS	1239	12	1442	14	16	8	16	8	16	8	18	9	6028	60	10010	100
		TOTAL	34	29	3094	31	24	12	23	12	24	12	37	18	13022	130	17316	173
	ELAPSE ms		4470		4690		30		30		20		40		34670		37280	
	CPU UTILIZATION %		64		66		80						93		36		46	
	INDEX DB SEEK 回		60		60										326		601	
	DATA DB SEEK 回		100		100										663		448	
	CHANNEL ms		813		901										4870		7211	
HDAM	CPU	OS	1553	15	1536	15	7	3	8	4	6	3	16	8	2892	29	2815	28
		IMS	1187	11	1397	14	16	8	16	8	16	8	18	9	3086	30	2958	29
		TOTAL	2775	27	35	29	23	12	25	12	24	12	35	18	6012	60	5808	58
	ELAPSE		4920		4990		30		30		30		40		12180		11870	
	CPU UTILIZATION		56		59				84		80		89		49		49	
	DATA BASE SEEK		148		148										335		336	
	CHANNEL		519		519										1455		1213	
CALL回数			100		100		2		2		2		2		100		100	

(註) CASE1 : TSセグメントを initial value = 000005, increment = 000100 で 100回 Get.
CASE2 : TS,TSVをD'commandを利用し, CASE1と同様に100回 Get
CASE3~5 : Get Unique で TS,TSV を Retrieve しその下 TSVW を unqualified SSA で Get.
CASE6 : GHN TSVW の時間を含む. 従って replace のみの Time は Case6より Case5を引いて
HIDAM CPU Total = 5 ms
HDAM CPU Total = 6 ms
CASE7 : TSの Key Field の initial value = 000006, increment value = 000005 にて ISRT.
CASE8 : TSの Key Field の initial value = 040000, increment value = 000005 にて ISRT.

表-2 DATA BASE OPEN TIME

		HIDAM	HDAM
CPU TIME	OS	791	287
	IMS	83	59
	MPP	0	0
	TOTAL	875	347
DISK SEEK COUNT	DATA BASE	14	5
	SYSRES	60	20

(3) Comment

イ. Root Segmentの Retrieve についてみたばあい，HDAM, HIDAMの間には，時間的に有意差は認められない。但しこれは，HIDAMのばあいIndex DBのOVFL chainが殆んど存在しない前提においてのみ言える。

ロ. Dependent Segmentの Retrieve にあたって，GN, GHN, GHNPのどのFunctionを用いても時間的に差はない。

ハ. REPLASEは極めて小時間で実行される。但し Replaceのばあい，synchronization pointにおいて Disk への Physical I/Oが生ずるので，この時間を考えると全体的にはそれほど小さくはない。

ニ. CASE 7, 8から窺えるとおり，Root Segment の追加に於ては，HIDAMとHDAMの差は極めて大きくなる。しかもHIDAMは Index DBの OVFL Chainの増加による パーフォーマンス低下が著しい。

ホ. DB Openの時間は極めて大であり，このオーダーの時間は real time 処理に於て致命的となる。

4. 設計上のポイント

(1) R.A.A.のSizeの決定

イ. HDAMにおける R.A.A.のサイズを決定するにあたっての一般的基準は次式によって表わされる。

$$\frac{\text{R.A.A.内のDBレコードサイズ (Bytes)} \times \text{DBレコード発生件数}}{\text{BLOCKサイズ (Bytes)}} = \text{R.A.A.サイズ (Blocks)} \quad \cdots (1)$$

(1)式によるR.A.A.のサイズはFree Spaceを考慮するといわばminimumなサイズであり，このばあいのTotal Sizeは次の式によって表わされる。

$$\text{minimum size} \times \frac{fbff}{fbff-1} \times \frac{1}{1-\frac{fspf}{100}} = \text{Total Size} \quad \cdots\cdots (2)$$

ロ. (1)式を構成する各Factorについて更に詳細にみると次のようになる。

(イ) DBレコード発生件数

アプリケーション毎にこの数値は既知を見積すことができる。

(ロ) DBレコードサイズ

このサイズは論理データベース・レコード長の最大値を最高とし，ルート・セグメントの長さを最小の値とする範囲内で種々の選択が可能となる。Disk Spaceの Utilizationを最大にする意味からは，最小の値に近い点を選択すべきであり，Hierarchical Chain を短くし，Retrieveの効率をあげるためには最大の値に近い点を選び同一データベース・レコードが同一Block内に

収容する確率を高める方がよい。DBレコードサイズの決定にあたっては、このように、DISK SPACEの物理的制限、Root Segment Keyの特質、Randomizerの特質、アプリケーションの特性等の要因の相互連関を把握することが必須であるので、何度かの試行を要するが、一般的にはルート・セグメントの長さとActiveなDependentセグメントの長さを加えたものに落着くであろう。理想的なRandomizer(変換後のAddressの分布がポアソン分布に近似する)を前提として、Disk Spaceを有効に利用することだけを制約条件とするなら、Root SegmentをKey情報のみに限定し、その他の情報は全て従属セグメントに持つ構造を考えたうえで(HIDAMにおけるINDEX DBのHDAM化と考えてもよい)DBレコード長をRootセグメントの長さに等しくするといった工夫も一考に値するだろう。

(ロ) Block Size.

DB Buffer Poolの有効利用の点からほぼ外生的に決定される。即ち、DB Buffer PoolはIMSのコントロールの下で、全てのDBが共用することになり、このBuffer PoolへのDB Blockの収容にあたって、各々のDBのBlock Sizeが異なるとBuffer内に連続したSpaceを形成するためにDataのMoveによるSpace Areaの連結(Data Compaction)が生ずる。この時間的ロスを防ぐための全てのDBのBlock Sizeは共通にすることが望ましい。当社では2KBの値を採用している。

(ハ) Anchor Point

Anchor Pointの数を増大することは、Randomizerによって得られるAddressing Pointの量を、R.A.A.の大きさ(即ちBlock数)を変えずに増やすことができ、従ってSynonymの発生を少なく抑えることが可能となる。しかし同一のSynonym chain内のRootを同一のBlockに収容してretrieveの効率をあげるには、1Block内のDBレコード数との関連に於てAnchor Pointの数を選択すべきである。

$$1\text{Anchor Point}\ \text{あたりのレコード数の期待値}\ (B) = \frac{1\text{DBのレコード数}\ (G)}{1\text{Block}\ \text{あたりのAnchor Point数}\ (E) \times \text{RAAのBLOCK数}\ (F)} \quad (3)$$

(3)より、

$$1\text{Block}\ \text{あたりのレコード数の期待値}\ (A) = \frac{(G)}{(F)} = (B) \times (E) \quad \cdots\cdots \quad (4)$$

一方、

$$(A)' = \frac{\text{Block Size}\ (C)}{\text{DB Record Size}\ (D)} \quad \cdots\cdots \quad (5)$$

によっても$(A)'$を導くことができる。いま同一のSynonym chain内のRootを同一のBlock内に収容しようと考えると$(A) = (A)'$でなければならない。更にRandomizerの特性によっては(B)はこれを平均値として幾らかの偏差を生ずるので$(A) = (A)' \times \alpha$ $(\alpha > 0)$としなければならない。こうして、$(A) = (A)' \times \alpha$を前提とし、(C)、(D)、(G)を既知とすれば(F)が決まり、更に(E)を既知とすると(B)が決まることになる。

Anchor Pointの数はDisk Spaceの有効利用の面からは少なく抑えた方がよい。なぜなら、図-4によっても知れる如くAnchor Pointは利用できない余分なSpaceとしてOVFL Areaにも設定される。従ってOVFL Areaの サイズが大であるときAnchor Pointの分だけ無駄が生ずる。

(2) Bit Map の考慮

Bit Map の利用が HDAM の複雑性を増大せしめていることは疑いえない。Randomizer が Bit Map Block の Relative Block No に Root Key を Addressing すると、Bit Map Block には DB レコードを収容することはできないから、そのレコードは別の Block に収容され、anchor point chain が Block を越えて延びることになる。従って HDAM の Load 時又は Reorganization に際して、場合によっては全ての anchor point chain が Block を越えて延びる結果を招くことがある。これを防ぐには前項の (A)=(A)'×α の α を大きめに設定しておき、Bit Map による anchor point chain の他 Block への延長がその Block の anchor point chain へ影響しないように考えておくことが必要となる。

(3) Formating.

HDAM の Bit Map の設定及び Formating は該当する領域へ Data が Store される時に始めて行なわれる。Insert の効率を高めるには R.A.A. の全領域の Formating を Load 時に行なう必要がある。このため Load 時に最高位のブロックへアドレシングする Dummy Root を作り出す工夫をしておくべきである。

(4) Loading

HDAM の Load 時の効率をあげるには、DB レコードを Randomizer によって予め Address 変換し、この Address を Key として Sort したのちに Load するとよい。

(5) DB OPEN

DB OPEN が頻発することによるパフォーマンスの低下は極めて著しい。これを防ぐには DMB Buffer を大きくすること、各 DB の DBD を小さくする(特に Field の定義を極小に抑える)こと、HIDAM より HDAM を利用して DMB の数を減らすことが望ましい。

(6) DB Structure.

Performance Up の観点より DB Structure の望ましいあり方を考えると次のようになる。

イ. Root Segment の Occurrence は大であればあるほどよい

ロ. Dependent の Occurrence は小であればあるほどよい。

ハ. 上記ロを達成するためには Dependent の同一レベルのセグメント数を増大させることは慎うべきでない。

ニ. 上記イ、ロを達成することができうるなら DB のセグメント・レベルが大になることも厭うべきでない。

(7) Free Space

Free Space の Factor には fbff と fspf の2つがあるが、このうちの fbff は DB の Load および Reorganization の際に未使用の block を作り出す意味で上記(2)の Bit Map と同様の理由によって、anchor point chain が Block を越えて延びるという結果を招くばあいがある。従って、HDAM に於ては fbff の利用を控えるほうが賢明であろう。

(8) Twin Backward Pointerの利用

Deletionが頻繁に行なわれるセグメントにはTwin Backward Pointerを付加するヒ効果的である。これはDeletionに際してその直前の兄弟セグメントのTwin Forward Pointerを付け替える時に、Backward Pointerが有効に機能するからである。

(9) Child Last Pointerの利用

Dependent Segmentのoccurrenceが大で，しかもこのSegmentがKey Fieldを持たなければあり，このDependent SegmentのParentにChild Last Pointerを持たせておくとDependentの追加にあたって極めて大きな効果をもたらす。