Vol. 27 No. 6 情報処理学会論文誌 June 1986

トランスポーズ形ファイルで蓄積した関係に対する更新操作†

佐藤隆士†† 津田孝夫†††

関係データベースは，論理的には関係と呼ばれる表形式データの集まりである．物理構造（計算機への格納構造）としては，従来，この表を行方向（タプル単位）に蓄積（水平蓄積法）し，検索効率化の手段としてインデックスファイルを用いる方法をとってきた．筆者らは先の論文で，このような物理構造をもつデータベースの問題点を指摘し，トランスポーズ形ファイル構造（表を列方向に蓄積する方法：垂直蓄積法）の有効性について述べた．本論文では，トランスポーズ形ファイル構造をもつデータベースのオンライン更新について述べる．垂直蓄積法ではタプルが複数個のトランスポーズ形ファイルに分割されて格納されるため，水平蓄積法に比べ，データの更新には非常に不利であると一般には考えられている．しかし，垂直蓄積法ではインデックスファイルの修正が不要なこと，主記憶に常駐させる更新表によるコスト改善の効果が大きいことなどにより，水平蓄積法に劣らずあるいはより高速に更新できることを示す．とくに，属性方向の変更についてはその構造上，垂直蓄積法がだんぜん有利となる．さらに，データベースのより大きな意味での更新であるデータベースの再構成について触れ，とりわけ関係スキーマの変更に対して，垂直蓄積法が柔軟に対応できることを述べる．

1. まえがき

関係データベースは，論理的には関係と呼ばれる表形式データの集まりである．関係表は行列形式の二次元構造であるが，計算機のメモリは本質的に一次元的であるため，格納のためには変換が必要である．すなわち物理構造としては，単純には行（タプル）方向に格納する方法と列（属性）方向に格納する方法が考えられる．筆者らは前者を水平蓄積法，後者を垂直蓄積法と呼んでいる．水平蓄積法は関係データベースの物理構造として従来使用されてきた構造である．データは普通レコード単位に収集され，レコードが集まってファイルとなるが，レコードをタプル，ファイルを関係表と対応づけられることからごく自然な構造であると言える．水平蓄積法ではメインファイル（関係をそのまま格納したファイル）だけでは属性方向の検索に時間がかかりすぎるため，頻繁に検索対象になると予想される属性に対しては何らかのインデックスファイルを付加するのが普通である．このような物理構造をもつ関係データベースの例としては，IBM の SQL/DS[2]，富士通の AIM/ROB[3]，日本電気の RIQS[2] などがある．これらは，更新時の効率を考慮しリンク（ポインタ）を使用しているため，必ずしも表形式データのタプルが先頭の行から順に連続して記憶されているわけではない．しかし，一つのタプルがメインファイルにおいて連続した領域に格納されているので基本的に水平蓄積法であると言える．また，初期の System R におけるアクセス法に用いられた XRM[4] および，ミニコンピュータ用関係データベース RISS[5] に用いられた物理構造ではタプルは別に格納された実際の属性値をもつファイルへのポインタとしている．これらの構造もポインタは属性値をコード化したものであるとみなせば，1タプルは属性値コードが連続領域に格納されたものになっているため，水平蓄積法に含められる．

垂直蓄積法については，データベースが出現する以前にもファイルの分割格納として同様な概念は存在していた．分割されたファイルはトランスポーズ形ファイル[6]と呼ばれ，各トランスポーズ形ファイルはファイルの各レコードのいくつかのフィールド値のみを集めたものである．とくに各ファイルが1フィールドの値のみからなるように分割したものを完全トランスポーズ形ファイルという．この意味において，本論文で言う垂直蓄積法は関係を完全トランスポーズ形ファイルで格納する方法と言うことができる．垂直蓄積法によるデータベースは統計データベースにみることができる[7]．統計データベースでは属性方向の操作が中心となり，この構造が適しているためである．関係データベースの物理構造として垂直蓄積法を用いた例はあまりないが，logic per track の考えに基づく Utah 大学のデータベースマシン RARES[8]，階層型転置ファイルを持つデータベースシステム[9] が存在する．

† Update Operations to the Relation Stored in Transposed Files by TAKASHI SATO (Department of Information Engineering, Takuma Radio Technical College) and TAKAO TSUDA (Department of Information Science, Kyoto University).

†† 詫間電波工業高等専門学校情報工学科
††† 京都大学工学部情報工学科

筆者らは先の論文[1]において，水平蓄積法ではインデックスファイルが不可欠のものとなることから生ずるさまざまな問題点を指摘した．そして，インデックスファイルを使用せずに高速な検索を行う方法としての垂直蓄積法の有効性について述べた．そこでは，関係演算列全体としてのコストの低減，ページプリフェッチなどにより検索を効率化することを提案しており，特別なハードウェアもいかなる意味のインデックスファイルも基本的には使用しないことにしている．このため，適用可能範囲は中，小規模のデータベースに限られるが，垂直蓄積法は関係データベースが本来得意とする非定形的な広範囲の検索質問に，より柔軟に対応できる点ですぐれていることを述べた．

一般に，データの更新には垂直蓄積法によるデータベースは水平蓄積法による場合に比べ，非常に不利であると考えられている．これは垂直蓄積法ではタプルが複数個のトランスポーズ形ファイルに格納されているためである．先の論文ではこの点にまでは言及していない．そこで，本論文では，垂直蓄積法でも高速な更新が可能となる方法を示すとともに，水平蓄積法との比較を行い，水平蓄積法に劣らないあるいはより高速な更新ができることを示す．

本論文では，更新操作を能率化するため，更新要求を表の形で主記憶上にコンパクトに表現する方法を用いる．この表は，追加表，変更表，削除表の3種類からなるが，これらをまとめて更新表と呼ぶ．

垂直蓄積法において更新操作が高速化できるのは，垂直蓄積法ではインデックスファイルの修正が不要なこと，主記憶中に常駐させる更新表によるコスト改善の割合が水平蓄積に比べ大きいことなどの理由による．とくに，属性方向の変更についてはその構造上，垂直蓄積法がだんぜん有利となる．さらに，本論文では，更新が多くなっても性能を下げないための工夫，二次記憶上の物理的に近いページを同時に更新し実質的な効率化を図る方法も示している．

更新表を用いて，メインファイル（二次記憶上にあるデータベース本体）へのアクセスを小さくする方法は Severance と Lohman[10] の差ファイル (differential file) の考えに似ているとも言えるが，次の点で異なっている．

1）差ファイルによる方法では，メインファイルに対する更新は一時にまとめて全体に対して行われる．一方，更新表による方法では，検索のためメインファイルにアクセスしたとき，あるいは，各更新表の記入がある一定数に達したとき，メインファイルの一部に対して更新が行われる．したがって，差ファイルによる方法ではメインファイルの更新のため，データベースの運用を止めてしまうのに対し，更新表による方法では，その必要はない．

2）差ファイルは，二次記憶上にとられるが，更新表は主記憶上にとられることを前提としている．1) でも述べたように差ファイルの方式では，メインファイルの更新はデータベース運用の停止を意味し，頻繁に行うことはできない．このため，差ファイルの容量を大きくする必要がある．これに対し，更新表による方法では，メインファイルの小部分をたえず更新していくので，更新の記憶は主記憶にはいる程度の小容量でまにあう．

以下，2章では垂直蓄積法における更新操作を，更新表の定義とともに具体的に述べる．更新表を使用する場合には，検索時に若干の付加操作が必要となるが，それについては3章でふれる．4章では，垂直蓄積法および水平蓄積法において，更新表を使用する場合と使用しない場合について，コスト* 解析をしそれらの比較を行う．そして，垂直蓄積法における更新表の有効性を示す．また，5章ではデータベースのより大きな意味での更新である再構成について述べる．6章は更新操作に関する計算機実験の結果を示したものである．

2. 更新操作

本章では，垂直蓄積法に対する効率的な更新操作の方法を具体的に述べる．ここで更新操作とは追加，変更，削除のことを言う．垂直蓄積法によるデータベースは水平蓄積法による場合に比べ，一つの属性全体に対する変更は高速に行えるが，タプル単位の更新操作はタプルが複数個のトランスポーズ形ファイル（以下 T-file）に格納されているため，このままではかなり不利であると思われる．そこで，本論文では，更新操作の効率化のために“追加表”，“変更表”，“削除表”からなる“更新表”を用いる．これらの表は主記憶中に常駐させ（仮想記憶管理によりページアウトされないようにしておき），更新操作はこれらの表の書き換えによるものとする．垂直蓄積法によるデータベース自体 (T-file) の更新は，更新表からバッチ的に行う．したがって，検索操作は T-file と更新表の両方の参照により行うことになるが，更新表は主記憶中に常駐

* 本論文での記憶環境はページ化された仮想記憶とし，コストとしては，二次記憶から主記憶へのページフェッチ数を用いる．

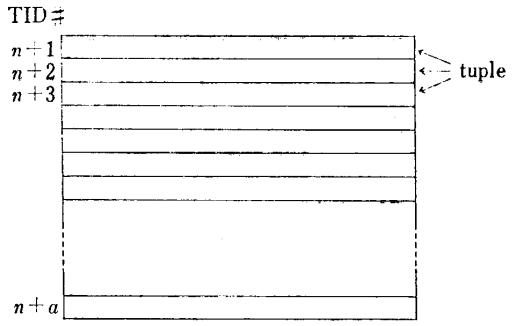

図 1 追加表
Fig. 1 Addition table.

するためページフェッチ（二次記憶へのアクセス）はT-file に対してのみである．なお，このような更新表は水平蓄積法にも使用することができるが，これについては4章で述べる．

2.1 追　　加

追加表は，主記憶中に**図1**のようにとられる．ここで，nは T-file に格納されている関係 R のタプル数である（削除されたタプルで無効データがはいっているタプルも含む）．追加はタプル単位に $n+1$ から番号順に連続した番地に割りあてられる．そして，追加表において TID＃（タプル識別番号：関係表の先頭から数えた行番号）が $n+a$ のところまで詰まると，バッチ的に T-file に書き込まれる．この際，2.3 節で述べる削除表を用いて，T-file の削除部に書き込むこともできるが，本論文では，T-file の最後に付け加えることにする．なお，T-file をディスクに格納する際，追加に備えその後ろに適当な未使用領域を用意しておくものとする．追加により未使用領域を使用し尽くした場合には，その時点における T-file と同じ大きさの領域をディスク上の別のトラック，シリンダが連続した場所に確保する．すなわち，その T-file が格納可能な場所を2倍にする．このようにすることにより，T-file の追加領域がディスクの広い範囲に散在し性能が劣化するのを防止できる．

2.2 変　　更

変更表は**図2**のように主記憶中にとられる．変更表はこのように関係の属性ごとに用意しておく．したがって，タプルの変更において二つ以上の属性値を変更するときは，それぞれの変更表について以下の操作を行わなければならない．しかし，多くの属性値に変更が生じた場合には，むしろ，削除と追加の二つの操作に置き換える方がよいであろう．

図2で“TID＃ block”は関係 R のタプル番号を

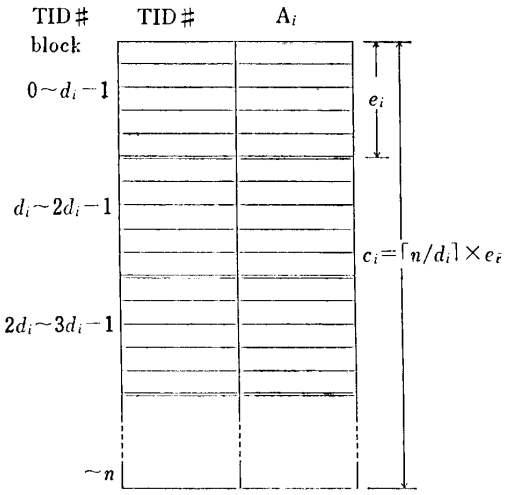

図 2 変更表
Fig. 2 Modification table.

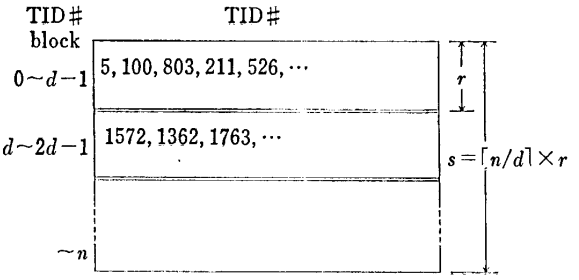

図 3 削除表
Fig. 3 Deletion table.

d_i 個ずつに区切った番号の区間である．ここで，u_i を1ページに格納可能な A_i の属性値（等長とする）の数とし，d_i は u_i の倍数となるように選ぶ．区間の概念を用いることにより，ディスク上の連続領域に生ずる変更を管理でき，T-file への効率的な変更操作が可能となる．関係Rの1タプルの属性 A_i に変更が生じたときは，そのタプルの TID＃ が含まれる区間にその TID＃ と新しいデータ（属性値）を入れておく*．ただし，既にその TID＃ が変更表に存在する場合には，その属性値を変更するだけでよく，もう一度 TID＃ を変更表に書く必要はない．各区間にはいる変更の数が一定値（たとえば図2では e_i）になると，その区間に相当する属性 A_i の T-file を修正する．

2.3 削　　除

削除表は**図3**に示すとおりであり，一つの関係表につき一つ作成する．図中，“TID＃ block”は変更表

* 変更は，変更対象タプルを見つけるため，操作の一部に検索操作を含む．ここでは，この検索操作により，変更対象タプルの TID＃ は既にわかっているものとする．次節の削除についても同様である．

と同じ意味である．表に記入されている TID# は各区間における削除されるべきタプルのうちまだ T-file では削除（タプルを構成する各属性値に削除マークを記入）されていないタプル番号である．各区間に記入可能な TID# の数 r（図3参照）までいっぱいになると，その区間の削除を T-file に対して行う．ここで言う T-file でのタプルの削除は，削除マークの記入であるから，T-file に削除マークが多くなったときは，適当に詰め合わせる．削除表に TID# を記入するときは，無矛盾性を保証するため，追加表，変更表を調べ，もし削除する TID# がその中にあれば，追加あるいは変更が無効となるようにしておく必要がある．

3. 更新表を用いた検索方法

2章で述べた更新表を用いた場合には，T-file のみを用いて検索操作を行ったのでは，最新のデータに対する正しい結果は得られない．更新表が存在する場合には，検索時に T-file とともに更新表も参照しなければならない．その方法は，各関係演算に対して次のようである．

（1） T-file と追加表よりなる関係表に対して関係演算を行う．

（2） その際，アクセス（主記憶に読込み）された T-file に対して，その属性の変更表により変更を行う．そして変更済みとなった変更は変更表より取り除く．

（3） T-file 構造を持つデータベースに対する関係演算の結果は，原関係（関係演算によりできる関係でなく，もともと，データベースに存在する関係．文献1) 参照）の TID# で出力されるが，削除表に含まれている番号は取り除く．

このように更新表が存在する場合には，検索操作時に更新表の参照が必要となるが，更新表は主記憶中に存在するので，このためのオーバヘッドは小さい．

4. 更新コストの解析

更新操作（追加，変更，削除）に要するコストを，1回の更新操作当たりのページフェッチ数と定義し，次の物理構造について比較を行う．

（1） 垂直蓄積法で更新表を用いない構造
（2） 垂直蓄積法で更新表を用いた構造
（3） 水平蓄積法で更新表を用いない構造
（4） 水平蓄積法で更新表を用いた構造

コストには F に添字を上下に付けたものを使用する．上付の添字1から4は今述べた物理構造を表す．また，下付の添字は更新の種類を示しており，a は追加，ct はタプル方向の変更，ca は属性方向の変更，r は削除を表す．

削除，変更は，検索操作をその一部に含むのが一般的だが，検索については文献1) および前章を参照することとし，本章では，それ以外に要するコストについて解析する．

4.1 垂直蓄積法で更新表を用いない場合

1) 追　　加

追加タプルの属性数を m_a とするとき，一つのタプルを追加するために m_a 個の T-file をアクセスするのでコストは，

$$F^1_a = m_a \qquad (1)$$

となる．

2) 変　　更

○一つのタプルについて，m_c 個の属性値を変更する場合

m_c 個の T-file にアクセスしなければならないので，

$$F^1_{ct} = m_c \qquad (2)$$

となる．

○関係Rの一つの属性 A_i 全体の属性値を変更する場合

コストは属性 A_i を格納する T-file のページ数となる．これを P_i とすると，

$$F^1_{ca} = P_i = \lceil n/u_i \rceil \qquad (3)$$

となる．ここで，n は関係Rのタプル数，u_i は1ページにはいる属性 A_i の属性値数である．

3) 削　　除

削除タプルの属性数を m_r とすると m_r 個の T-file にアクセスするのでコストは，

$$F^1_r = m_r \qquad (4)$$

となる．

4.2 垂直蓄積法で更新表を用いる場合

1) 追　　加

2.1 節で述べたように追加表の大きさを a タプル分（各関係表に対して）とし，追加タプルの各属性（m_a 個）の属性値を1ページに u_i $(1 \leqq i \leqq m_a)$ 個格納可能とする．$a \leqq u_i$ $(i=1,2,\cdots,m_a)$ ならば，追加表を用いない場合に比べて，1タプル追加当たり T-file へのアクセスは $1/a$ となるので，

$$F^2_a = m_a/a \qquad (5)$$

となる．

aが大きい場合には，i番目の属性を T-file に追加するのに要するページフェッチ数が$\lceil a/u_i \rceil$となるため，

$$F_a^2=\sum_{i=1}^{m_a}\lceil a/u_i\rceil/a \qquad (6)$$

となる（式(5)は式(6)に含まれる）．

2) 変　　更

○一つのタプルのm_c個の属性を変更するとき

各属性（A_i：$1\leqq i\leqq m_c$）に対して，区間がいっぱいになった時点で行う T-file へのアクセスは$\lceil d_i/u_i \rceil$であり，この要求は1回の変更当たり$1/e_i$の割合で生ずる．したがって，

$$F_{ct}^2=\sum_{i=1}^{m_c}\lceil d_i/u_i\rceil/e_i$$

$$=\sum_{i=1}^{m_c}\lceil d_i/u_i\rceil\lceil n/d_i\rceil/c_i \qquad (7)$$

となる．

○関係Rの一つの属性 A_i 全体の属性値を変更する場合

このときは，変更表を用いず，直接 T-file にアクセスすることにより，

$$F_{ca}^2=F_{ca}^1 \qquad (8)$$

となる．変更表は使用しないほうがよい．

3) 削　　除

削除表がいっぱいになった時点で行う各属性（A_1, $A_2, \cdots, A_{m_r}$）の T-file へのアクセス回数は$\lceil d_i/u_i \rceil$（$1\leqq i\leqq m_r$）で，この要求は1回の削除当たり$1/r$の割合で生ずるので，

$$F_r^2=\sum_{i=1}^{m_r}\lceil d_i/u_i\rceil/r$$

$$=\sum_{i=1}^{m_r}\lceil d_i/u_i\rceil\lceil n/d_i\rceil/s \qquad (9)$$

となる．

4.3 水平蓄積法で更新表を用いない場合

ここでは，水平蓄積法で更新表を用いない場合の更新コストについて述べる．追加，変更，削除の各操作を行う関係表について，それらの属性のそれぞれh_a, h_c, h_r（×100%）にインデックスファイルがついているものとする（$0\leqq h_a, h_c, h_r\leqq 1$）．また，一つの属性値を更新する際，インデックスファイルの修正は少なめに見積り，1回のページフェッチで可能であると仮定する．変更，削除には更新操作が含まれるが，メインファイルの修正は検索時のページフェッチで行うことができる．追加ではメインファイル1回のアクセスが必要となる．

1) 追　　加

インデックスの付けられた属性数はh_am_aなので，

$$F_a^3=h_am_a+1 \qquad (10)$$

となる．1はメインファイルへのアクセス数である．

2) 変　　更

○一つのタプルのm_c個の属性を変更する場合

インデックスの付けられた属性数はh_cm_cなので，

$$F_{ct}^3=h_cm_c \qquad (11)$$

となる．

○関係Rの一つの属性 A_i 全体の属性値を変更する場合

水平蓄積法では，A_i の属性値へアクセスすると関係Rのほかの属性値も同時にフェッチされるので，関係Rのメインファイルのページ数を$\bar{P}$とすると，メインファイル修正のため，

$$F_{ca}^3=\bar{P} \qquad (12)$$

フェッチを要する．A_i にインデックスファイルがついているときは，インデックスファイルはP_iページ以上を占めるので，修正のためにはさらにP_iフェッチ以上必要である．

3) 削　　除

インデックスの付けられた属性数はh_rm_rなので，

$$F_r^3=h_rm_r \qquad (13)$$

となる．

4.4 水平蓄積法で更新表を用いた場合

ここでは，水平蓄積法で更新表を用いた場合について述べるが，インデックスの付加割合は更新表を用いない場合と同じとする．また，更新表の形式は垂直蓄積法の更新表と同様のものと考える．

1) 追　　加

追加表の大きさをaタプル分とすると，メインファイルへのアクセスの割合は$1/a$となる．しかし，インデックスファイルの修正については垂直蓄積法の場合の T-file への追加のようにまとまった二次記憶の領域に生ずると考えることはできない．そこで，各属性のインデックスファイルの大きさをP（解析を簡単にするため，インデックスファイルの大きさは属性によらず同じ大きさの場合について述べる．本節では以下同様）とし，追加が一様に分布すると仮定すると，アクセス回数は$h_am_a\times P/a$となる．したがって，

$$F_a^4=(h_am_aP+1)/a \qquad (14)$$

となる．

2) 変　　更

○一つのタプルのm_c個の属性を変更する場合

変更表の大きさを c タプル分とすると，追加のときと同様な考え方により，

$$F^4_{ct}=h_c m_c P/c \qquad (15)$$

となる．

○関係Rの一つの属性 A_i 全体の属性値を変更する場合

垂直蓄積法の場合と同じく更新表は使えないので，4.3 節と同じである．

3) 削　　除

削除表の大きさを s タプル分とすると，

$$F^4_r=h_r m_r P/s \qquad (16)$$

となる．

4.5 比　　較

比較を簡単にするため，属性値の長さが属性によらず一定であり，1ページにちょうど u 個格納可能な場合について解析する．このとき，一つの関係Rの各属性の T-file が占めるページ数は同じであり，$P=\lceil n/u \rceil$ となる．これらの仮定より，

$$F^2_{ct}\fallingdotseq m_c P/c \qquad (7)'$$

$$F^2_r\fallingdotseq m_r P/s \qquad (9)'$$

$$F^3_{ca}\fallingdotseq mP \qquad (12)'$$

となる．ここで，c, s は c_i, s_i が i によらないために用いている．m は関係Rの属性数である．式(7)′, (9)′, (12)′を考慮しながら 4.1 から 4.4 節の場合の比較を行うと次のようになる．

$$\left.\begin{aligned}
&F^1_a : F^2_a : F^3_a : F^4_a \fallingdotseq 1 : 1/a : h_a : h_a P/a\\
&F^1_{ct} : F^2_{ct} : F^3_{ct} : F^4_{ct} \fallingdotseq 1 : P/c : h_c : h_c P/c\\
&F^1_{ca} : F^2_{ca} : F^3_{ca} : F^4_{ca} \fallingdotseq 1 : 1 : m : m\\
&F^1_r : F^2_r : F^3_r : F^4_r \fallingdotseq 1 : P/s : h_r : h_r P/s
\end{aligned}\right\} \qquad (17)$$

ただし，$a<u$ とした．また，同じ条件で，一つの関係Rに対する更新表の大きさを求めると，

$$\left.\begin{aligned}
&\text{追加表}: P_a=\lceil ma/u \rceil \text{ ページ}\\
&\text{変更表}: P_c=\lceil 2mc/u \rceil \text{ ページ}\\
&\text{削除表}: P_r=\lceil s/u \rceil \text{ ページ}
\end{aligned}\right\} \qquad (18)$$

となる．式(18)の P_c, P_r について，天井（「　」）をはずしても等号がほぼ成立するように c, s を選ぶことができるが，その場合は，

$$\left.\begin{aligned}
&1/c \fallingdotseq 2m/uP_c\\
&1/s \fallingdotseq 1/uP_r
\end{aligned}\right\} \qquad (19)$$

だから，これらを式(17)の第2，4式に代入すると，

$$\left.\begin{aligned}
&F^1_{ct} : F^2_{ct} : F^3_{ct} : F^4_{ct}\\
&\quad \fallingdotseq 1 : (2m/u)(P/P_c) : h_c : h_c(2m/u)(P/P_c)\\
&F^1_r : F^2_r : F^3_r : F^4_r\\
&\quad \fallingdotseq 1 : (1/u)(P/P_r) : h_r : h_r(1+u)(P/P_r)
\end{aligned}\right\} \qquad (20)$$

となる．

例 1

$m=10$, $u=1000$, $a=100$, $P_c=5$, $P_r=1$, $P=50$, $h_a=h_c=h_r=0.5$ とすると，

$$F^1_a : F^2_a : F^3_a : F^4_a \fallingdotseq 1 : 0.01 : 0.5 : 0.25$$

$$F^1_{ct} : F^2_{ct} : F^3_{ct} : F^4_{ct} \fallingdotseq 1 : 0.2 : 0.5 : 0.1$$

$$F^1_{ca} : F^2_{ca} : F^3_{ca} : F^4_{ca} \fallingdotseq 1 : 1 : 10 : 10$$

$$F^1_r : F^2_r : F^3_r : F^4_r \fallingdotseq 1 : 0.05 : 0.5 : 0.025$$

となる．メインファイル $\bar{P}\fallingdotseq mP=500$ ページに対して，更新表は $P_a=1$ だから，$P_a+P_c+P_r=7$ ページである．水平蓄積法のインデックスファイルは，$h_a\bar{P}=250$ ページ以上となる．□

例1のように，更新表にわずかの主記憶領域を使用することにより，更新操作をかなり高速化することができる．また更新表は主記憶上にとられるため，更新表がある場合でも，検索操作についてページフェッチ数を増加させることはないので，文献1) の議論はそのまま成立する．

更新表によるコスト改善の割合は変更，削除については水平蓄積法と垂直蓄積法で同じであるが，追加に対しては水平蓄積法は垂直蓄積法ほど改善されない．垂直蓄積法に更新表を用いた場合には更新が二次記憶の連続領域に生ずるので，文献1) の“連続するページの単純フェッチ”によるページプリフェッチが行えさらに高速化できる．

変更条件に合うタプルが多いときは属性方向の変更となるが，この場合には式(17)の第3式が示すように，更新表のあるなしにかかわらず垂直蓄積法がだんぜん有利となる．

5. データベースの再構成

データベースは最初作成された時点では最適な物理構造となるよう配慮されていたとしても，データ自体の更新により時間とともに構造が変化してくる．このためデータベースを編成しなおす必要が生じる．前章までに述べた更新方法によると削除タプルはそのまま記憶されており，無駄なメモリ使用領域が増加するので適当な時期に削除タプルを埋める必要がある．2章に述べたように，追加により T-file 格納のための未使用スペースを使用し尽くした場合には，ディスク上に別の連続領域が確保される．このように，同一 T-file はなるべくクラスタ化されるよう配慮されているが，適当な時期に同一 T-file を一つの連続領域にまとめることにより，T-file へのより効率的なアクセ

スが可能となる.

水平蓄積法においてはインデックスの付け替えが行われたり，インデックスの構造を変えたりするが，垂直蓄積法では基本的にはインデックスファイルを排除する方向なので詳細は述べない.

以上のほかに，データベースに対する要求の変化により，格納データの枠組である関係スキーマ（属性名を並べたもの）を変化する必要性が生ずる場合がある．たとえば，“社員”という関係で新たに“通勤時間”という属性を加えるといった場合である．関係スキーマの変更は，データベースの3層スキーマの物理レベルのみならず概念レベルにまで及ぶ大きな変更である．水平蓄積法では，属性の追加，削除により，関係を格納するメインファイル全体の修正が必要となる．これに対し，垂直蓄積法ではデータが関係の属性ごとに格納管理されているため，このような関係スキーマの変更が容易にできるが，この点は注目すべきである.

6. 計算機実験

本章では，計算機による更新操作時間の実測データを示し，4章の結果と比較する．使用計算機は NEC ACOS-450 であるが，本計算機はセグメンテーションによる仮想記憶計算機であるためページフェッチの概念はない．このため，実験ではページフェッチをデータファイルのブロック転送回数に置き換えている．また記載する測定値は，

I/O 時間＝経過時間－CPU 時間

である．これは文献1)に合わせたためであるが，更新操作では CPU の負荷は軽く，したがって経過時間を用いてもほぼ同じである．測定の結果を**表1**(a)～(d)に示す．データ長はすべて 4 byte で等長とし，1ブロックは 1 kbyte としたため，1 ブロックのデータ数は $p=256$ である．更新の対象となる関係は属性数 $m=8$，タプル数 $n=64p=16,384$ とした.

A．追　　加

追加タプル数を $nup=512$ タプルとし，水平・垂直蓄積法の比較を行った．測定値は表1(a)に示したとおりである．追加表はなしと，追加表の大きさ $a=16, 32, 64, 128$ タプル分についてである．ブロック数に換算すると，これらはそれぞれ $P_a=am/p=0.5, 1, 2, 4$ となる．表中の理論値は I/O 回数* であり，

* 4章ではコストとしてページフェッチ数を用いたが，本章では I/O 回数（主記憶，二次記憶間の入出力回数）を用いた．更新操作の場合，I/O 回数はページフェッチ数の2倍である.

表1 計算機実験結果（( ) 内はブロック数）
Table 1 Computer experiment results (( ) is block number).

(a) 追　加

物理構造		測定値 (ms)	理論値 (回)
垂直蓄積法	追加表なし	164,381	8,192
	$a=16$ (1/2)	10,628	512
	$a=32$ (1)	5,200	256
	$a=64$ (2)	2,519	128
	$a=128$(4)	1,284	64
水平蓄積法	追加表なし	77,908	4,096
	$a=16$ (1/2)	78,005	4,096
	$a=32$ (1)	77,820	4,096
	$a=64$ (2)	77,924	4,096
	$a=128$(4)	33,256	2,048

(b) タプル方向の変更

物理構造		測定値 (ms)	理論値 (回)
垂直蓄積法	変更表なし	73,629	4,096
	$e=8$ (8)	8,086	512
	$e=16$ (16)	4,011	256
	$e=32$ (32)	1,984	128
水平蓄積法	変更表なし	32,513	2,048
	$e=8$ (8)	5,177	256
	$e=16$ (16)	2,529	128
	$e=32$ (32)	1,296	64

(c) 属性方向の変更

物理構造		測定値 (ms)	理論値(回)
垂直蓄積法		2,498	128
水平蓄積法	インデックスなし	19,313	1,024
	インデックスあり	21,564	1,152

(d) 削　除

物理構造		測定値 (ms)	理論値 (回)
垂直蓄積法	削除表なし	141,400	8,192
	$r=16$ (1)	35,545	2,048
	$r=32$ (2)	17,725	1,024
	$r=64$ (4)	8,835	512
	$r=128$ (8)	4,385	256
水平蓄積法	削除表なし	73,568	4,096
	$r=16$ (1)	18,327	1,024
	$r=32$ (2)	9,214	512
	$r=64$ (4)	4,509	256
	$r=128$ (8)	2,376	128

垂直蓄積法の場合，追加表がないときは $nup \times m \times 2$，追加表があるときは $\lceil nup/a \rceil \times m \times 2$ から求めている．ここで2倍されているのは更新操作であるため，ブロックの書き換えが必ず生じ，しかも入出力回数が同じであるからである．水平蓄積法については属性のうちインデックスが付加されている割合を $h_a=0.5$ とした．追加表がないときは，$nup \times m \times h_a \times 2 = nup \times m$ から求めている．追加表があるときは，$nup \times m \times h_a \times 2 \times \min(1, P/a)$ から求めている．これは，追加タプルの属性値はインデックスファイルに一様に分布するとしたので $P<a$ なら更新表を用いてもその効果はないことによる．

B．タプル方向の変更

変更は4章と同じくタプル方向の変更と，属性方向の変更とに分けて扱っている．タプル方向の変更は $nup=512$ 個のタプルのそれぞれ $m_c=m/2$ 個の属性値に変更が生じた場合について測定した．変更表は，なしと $e_i=8, 16, 32$ タプル分とした（e_i については2章参照）．また区間の大きさ d_i は $4u_i$ とした（$u_i=p$）．追加表のブロック数は表1（b）中に記したとおりである．垂直蓄積法の場合，理論値は追加表がないときは，$nup \times m_c \times 2$ であり，存在するときは変更が各区間の特定ブロックに集中すると仮定すると $(nup/e) \times m_c \times 2$ となる．水平蓄積法では理論値はインデックス付加割合を $h_c=0.5$ とし，変更表がないとき $nup \times m_c$，変更表があるとき $(nup/e) \times m_c$ となっている．

C．属性方向の変更

この場合は変更表は使用しないため，変更表の大きさには無関係である．垂直蓄積法の場合，I/O 回数の理論値は変更する属性のトランスポーズ形ファイルのブロック数 $P=n/p=64$ の2倍である．水平蓄積法ではインデックスファイルのない属性の場合，I/O 回数は関係全体のブロック数 $\bar{P}=n \times m/p=512$ の2倍である．インデックスファイルのある属性についてはさらにインデックスファイルの大きさ約 $P=64$ ブロックの修正が必要となる．

D．削　　除

削除タプルの個数は $nup=512$ とした．削除表は表1（d）の場合について測定した．また，削除表の区間の大きさは $d=4p$ とした．垂直蓄積法の場合 I/O 回数の理論値は，削除表がない場合は $nup \times m \times 2$，ある場合は削除が関係全体に一様に生じるとして $(nup/r) \times (d/p) \times m \times 2$ である．水平蓄積法の場合はインデックス付加割合を $h_r=0.5$ としたため，削除表がないとき $nup \times m$，削除表があるとき $(nup/r) \times (d/p) \times m$ である．

以上，表1より I/O 回数の理論値と測定値の間には比例関係がはっきりしており，理論解析の妥当性が確認できる．

7. むすび

垂直蓄積法の更新操作について述べ，水平蓄積法との比較を行った．垂直蓄積法ではタプルが複数個のトランスポーズ形ファイルに分割されて格納されるため，水平蓄積法に比べ，データの更新には非常に不利であると一般には考えられている．しかし，垂直蓄積法ではインデックスファイルを使用しないため，その修正が不要なこと，主記憶に常駐させる更新表によるコスト改善の効果が大きいことなどにより，水平蓄積法に劣らずあるいはより高速に更新できる．とくに，属性方向の変更については更新表のあるなしにかかわらず，垂直蓄積法がだんぜん有利である．

データベースの再構成の問題に関してはデータベースそのものとそれに対する要求が時間とともに変化していくものであることを認識する必要がある．とくに関係スキーマの変更はより根本的なものであり，これに垂直蓄積法が柔軟に対応できることは注目すべきことである．

参考文献

1) 佐藤，津田：トランスポーズ形ファイルで蓄積した関係に対する関係演算，情報処理学会論文誌，Vol. 25, No. 1, pp. 150-159 (1984).
2) 上林，Kim，酒井（編）：最近のデータベース・システムとその応用，共立出版 (1984).
3) 林，鴻池：リレーショナル・データベース管理システム AIM/RDB の実現手法，日経エレクトロニクス，No. 310, pp. 171-202 (1983).
4) Chamberlin, D. D. et al.: A History and Evaluation of System R, *Comm. ACM*, Vol. 24, No. 10, pp. 632-646 (1981).
5) Mcleod, D. J. and Meldman, M. J.: RISS—A Generalized Minicomputer Relational Data Base Management System, *Proc. AFIPS National Computer Conference*, Vol. 44, pp. 397-402 (1975).
6) Wiederhold, G.: *Database Design*, pp. 167-170, McGraw-Hill, New York (1977).
7) Burnett, R. A. and Thomas, J. J.: Data Management Support for Statistical Data Editing and Subset Selection, *Proc. of the 1st LBL Workshop on Statistical Database Manage-*

ment, pp. 88-99 (1982).
8) Lin, C.S., Smith, D.C.P. and Smith, J.M.: The Design of a Rotating Associative Memory for Relational Database Applications, *ACM Trans. Database Syst.*, Vol. 1, No. 1, pp. 53-65 (1976).
9) 大久保, 津田：階層転置型ファイルに基づく関係操作アルゴリズム, 情報処理学会論文誌, Vol. 26, No. 1, pp. 130-138 (1985).
10) Severance, D.G. and Lohman, G.M.: Differential Files: Their Application to the Maintenance of Large Databases, *ACM Trans. Database Syst.*, Vol. 1, No. 3, pp. 256-267 (1976).

(昭和60年2月14日受付)
(昭和61年3月20日採録)

佐藤 隆士（正会員）

昭和28年9月生. 昭和53年岡山大学大学院修士課程(電子工学専攻)修了. 工学博士. 同年詫間電波高専助手. 現在, 同校助教授. データベース, 記憶階層の研究に従事. 電子通信学会, CAI 学会各会員.

津田 孝夫（正会員）

1932年生. 1957年京都大学工学部電気工学科卒業. 現職は京都大学工学部情報工学科教授. 工学博士. 現在の主要研究テーマは, メモリ階層間データ転送量の下界とそれによるアルゴリズムの最適化, ベクトル計算機のための自動ベクトル化と自動並列化, 実時間オペレーティングシステムなど専用 OS の構成と実現法など.