Vol. 26 No. 1 情報処理学会論文誌 Jan. 1985

階層転置型ファイルに基づく関係操作アルゴリズム†

大久保 英嗣†† 津田 孝夫††

本論文では，データベースオペレーティングシステム OPT-R におけるデータベースの物理構造と関係操作について述べる．提案するファイル構造は，転置型ファイル (transposed file) を基本としており，転置型ファイルの各サブファイルに簡単な副次索引 (secondary indexes) を付加することにより，二次記憶へのアクセス回数の低減を図っている．本ファイル構造の利点としては，以下のものがある．(1)最適インデックス選択の問題から解放される．(2)データ検索時に副次索引内のデータの分布区間を利用することにより，データの全項目サーチを避けることができる．さらに，該当データの結果の大きさの見積りが容易となる．

1. はじめに

データベースの物理構造の設計は，データベースシステムの性能に影響を与える一つの重要な因子である．逆ファイル (inverted file)，マルチリスト (multi-list)，ハッシング (hashing)，B-木 (B-tree) などの非常に多くの技法の提案がなされ，実際のシステムで採用されている[1)~3)]．しかし，転置型ファイル (transposed file) の実現方式およびアクセスの高速化に関する研究は少ない[4)~8)]．

本論文では，転置型ファイルを基本としたデータの物理構造と，それに基づく関係操作アルゴリズムについて述べる．本ファイル構造は，われわれが設計・開発しているデータベース処理専用のオペレーティングシステム OPT-R[9)~11)] において実際に採用されている．提案するファイル構造は，以下の設計方針に基づいている．

(1) すべてのファイルに副次索引を付加する．ただし，そのストレージオーバヘッドを極力少なくする．

(2) データベースへのアクセスパスの選択は，この副次索引により，関係操作単位に行う．

(3) データをエクステント単位に分割し，1エクステント内のデータを物理的に隣接させて格納する．しかし，エクステント間の配置は任意に行えるようにする．

以下，2章で階層転置型ファイルの論理構造を示し，3章でその実現方式を述べる．最後に，4章で具体的な関係操作アルゴリズムについて説明する．

† The Algorithms for Relational Operations based on the Hierarchical Transposed File by EIJI OKUBO and TAKAO TSUDA (Department of Information Science, Faculty of Engineering, Kyoto University).

†† 京都大学工学部情報工学科

2. 階層転置型ファイル (HTF) の論理構造

転置型ファイルは，サブファイルと呼ばれるファイルの集合であり，各関係のタプルは，これらすべてのファイルに分けられる．一つのサブファイルに一つの属性を割り当てるとき，それは完全転置型ファイル (fully transposed file) と呼ばれ，それ以外はクラスタ化された転置型ファイル (clustered transposed file) と呼ばれる[4)]．クラスタ化するか否かは，採用する概念モデルとの関連によるが，クラスタ化した場合のアクセスの制御が複雑になるので，OPT-R では完全転置型ファイルを採用している．さらに，転置型ファイルは副次索引をもたず，関係操作実行時には一つのサブファイルすべてを主メモリへ読み込まなければならない．したがって，タプル数に比例してアクセスコストが増加する．本論文では，転置型ファイルに簡単な副次索引を設けることにより，この問題を解決している．われわれは，本ファイル構造を階層転置型ファイル (HTF: Hierarchical Transposed File) と呼んでいる[9)]．以下，本章では，HTF の論理構造について述べる．

2.1 HTF における階層構造

図1 は関係Sの一つの属性 CITY に関する HTF を示したもので，このように HTF は以下の三つのファイルより構成される．

(1) 分布ファイル (PDF: Piecewise Distribution File)

属性の実現値の分布状態を表現するための，ビットマップ形式のファイルである．行が属性の実現値の分布区間 (実際は各区間の最大値で表現する)，列がタプル識別子 (TID: Tuple IDentifier) に対応する．

(2) 二進行列ファイル (BMF: Binary Matrix File)

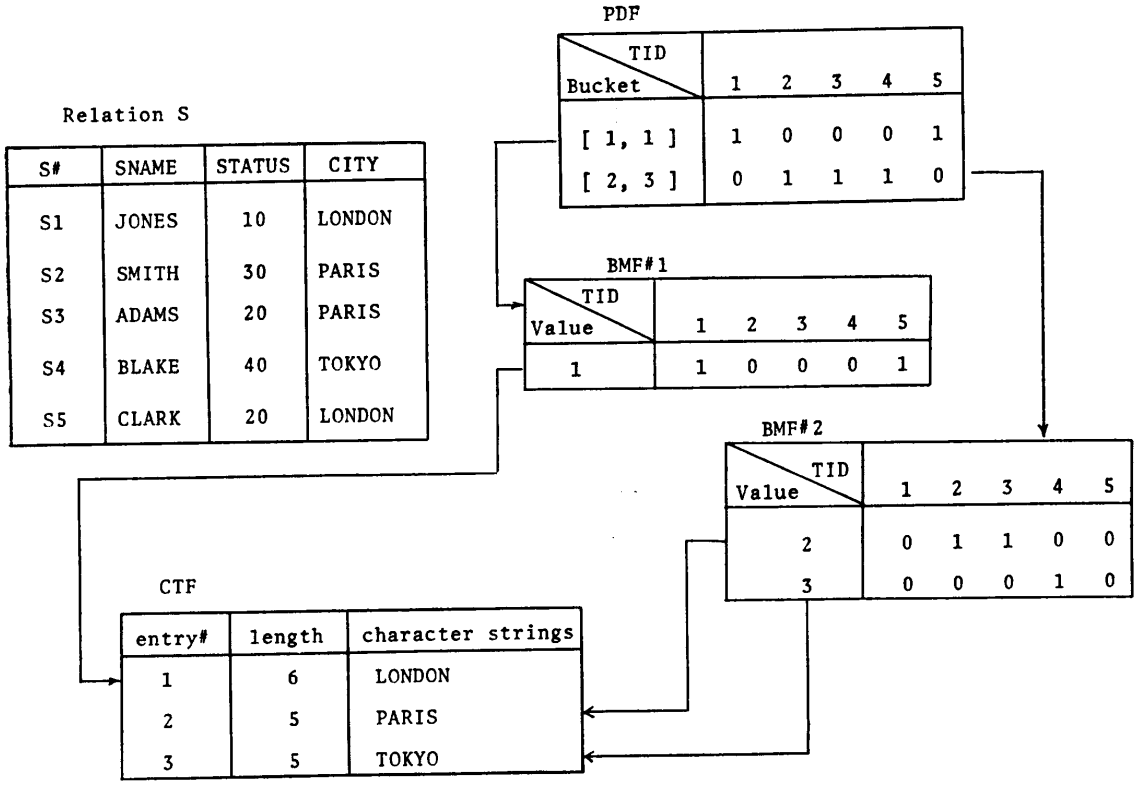

図 1 関係Sの属性 CITY に関する HTF の論理構造
Fig. 1 Logical structure of HTF for the attribute CITY in the relation S.

属性の実現値から TID あるいはその逆を求めるためのものであり，行が属性の相異なる実現値（あるいはコード化された値），列が TID に対応するビットマップ形式のファイルである．

（3） 圧縮完全転置型ファイル(CTF : Compressed fully Transposed File)

属性の相異なる実現値だけを圧縮して格納したファイルであり，コード化された値から該当のコードを検索しデコードすることによって，必要とする実現値を得るためのものである．ただし，実現値が短い場合は BMF に実現値がそのまま格納され，CTF は生成されない．

BMF は，PDF の各バケットに対応してクラスタ化される．クラスタ化とは，PDF の各バケットに対して，BMF をページ境界（1ページ=512バイト）に配置することをいう．クラスタ化することにより，二次記憶領域の断片化が生じ記憶域が増大する．しかし，このような断片化は，むしろデータベースに対する変更操作，とくに挿入操作に関して都合がよいといえる．これは，エクステントをベースとするファイル構造における分散フリースペースの一実現法と考えることができる．フリースペースがなくなった場合は，動的にエクステントを分割する（3.2 節参照）．

CTF の圧縮に関しては，種々の方法が考えられる．現在までに，差分表現，コンパクトコード方式，ハフマンコード方式等，種々の圧縮化技法が提案されている[12)-14)]．しかし，最適な圧縮法は，対象とするデータベースの属性値の特性によって変化すると考えられる．したがって，本論文では，重複データの排除だけを考慮し，具体的な圧縮法についてはデータ圧縮技術の研究分野で論じられているのでこれ以上扱わない．

さて，CTF は文字列等属性値が長い場合に必要となるものであり，PDF は BMF の副次索引である．したがって，HTF の階層構造として以下の 4 通りが考えられる．

（1） PDF—BMF—CTF
（2） PDF—BMF
（3） BMF—CTF
（4） BMF

しかし，PDF が付加されていない階層構造は，従来の転置型ファイルと同様の構造であるから，アクセス回数の低減が期待できない．したがって，以下では，

(1)あるいは(2)の階層構造だけを考えることにする.

2.2 HTF のストレージコスト

本節では，PDF—BMF の階層のストレージコストについて解析する．CTF は，本解析では考慮しない．使用する量を以下に定義する．

p: ページサイズ（単位＝バイト）

b: ブロックサイズ（単位＝ページサイズ）

n: 関係のタプル数

k: PDF のバケット数

l: 実現値の長さ（単位＝バイト）

PDF および PDF の一つのバケットに対応する BMF の総ビット数は，それぞれ $k(8l+n)$, $n(8l+n)/k$ となる（ただし，属性の実現値が相異なる場合を仮定している）．したがって，PDF および一つの BMF それぞれが1ブロックに入るためには，

$$n(8l+n)/8pb \leq k \leq 8pb/(8l+n)$$

となる必要がある．この不等式の両辺の等号が成立する場合のタプル数 n を n_s とするとき，$n \leq n_s$ なる属性に関する PDF および BMF は，バケット数 k を調整することにより，1ブロックに格納可能となる．すなわち，

$k=\lfloor 8pb/(8l+n_s) \rfloor$（ただし，$\lfloor\ \rfloor$ は floor）となる．$l=4, p=512$ としたときの，k, b および n の関係を**表1**に示す．表より，タプル数1,000の関係に対する PDF および BMF のページ数の総和は，$7+30\times7=217$ ページ以上となる．これは，ストレージコストに関して最悪の場合（データの値すべてが異なる場合）であるが，たかだか1,000タプルの関係の一つの属性の実現値を格納するストレージ量としては膨大である．

次に，PDF のサイズを無視して，BMF のストレージコストだけを考える．当該属性の相異なる実現値の数を d とすると，BMF の総ビット数は，$d(8l+n)$ となる．何ら副次索引を付加しないときの総ビット数は $8ln$ であるから

$$d(8l+n)-8ln \leq 0$$

なるとき，BMF はストレージコストに関して従来の転置型ファイルより小さくなる．上記の不等式は，

$$d \leq 8ln/(8l+n) = 8l - 64l^2/(8l+n)$$

となる．したがって，n が十分大きいとき，相異なる実現値の数が実現値の長さより小さい場合にのみ，BMF が有効となる．

以上の考察から，PDF および BMF のビットマップをそのままの形で格納すると，ストレージコストが増大することがわかる．したがって，PDF および BMF のビットマップが $O(n)$ のストレージコストになるような表現方法を考えなければならない．

表1 ブロックサイズ，タプル数およびバケット数の関係
Table 1 Relationships between b, n, and k.

b	n	k
1	235	15
2	385	19
3	511	22
4	623	25
5	727	26
6	824	28
7	914	30

b: no. of blocks, n: no. of tuples, k: no. of buckets

3. 階層転置型ファイル（HTF）の物理構造

前章に示した PDF および BMF のビット列は，最悪の場合 $O(n^2)$ ビットのストレージ量となる．本章では，この問題を解決するために OPT-R で実際に採用している HTF の物理構造（圧縮 HTF と呼ぶ）について述べる．

図2に，OPT-R における圧縮 HTF とその周辺のカタログ類を示す．圧縮 HTF は，索引部（index part）とデータ部（data part）に大きく分かれており，転置型ファイルに逆ファイル（inverted file）的な副次索引を付加したものと考えることができる．属性カタログ中には，以下の3種類のポインタに関するフィールドが設けられている．

（1） PDF へのエクステントポインタ

（2） 空値（null value）タプルファイル（NTF: Null Tuple File）へのポインタ

（3） BMF へのエクステントポインタ

さらに，関係カタログ中には，削除された TID の集合を示す DTF（Deleted Tuple File）へのポインタがある．これらは，いずれもデータベースをアクセスする際に使用される．

3.1 PDF

前章に述べた PDF のビット列は，ここで示す PDF に対応しており，バケットは属性カタログ中の BMF エクステントポインタに対応している（ただし，バケットの最大値のみを格納している）．

図2に示すように，PDF の1バイトの各エントリは，属性の実現値が含まれる BMF の番号を保持して

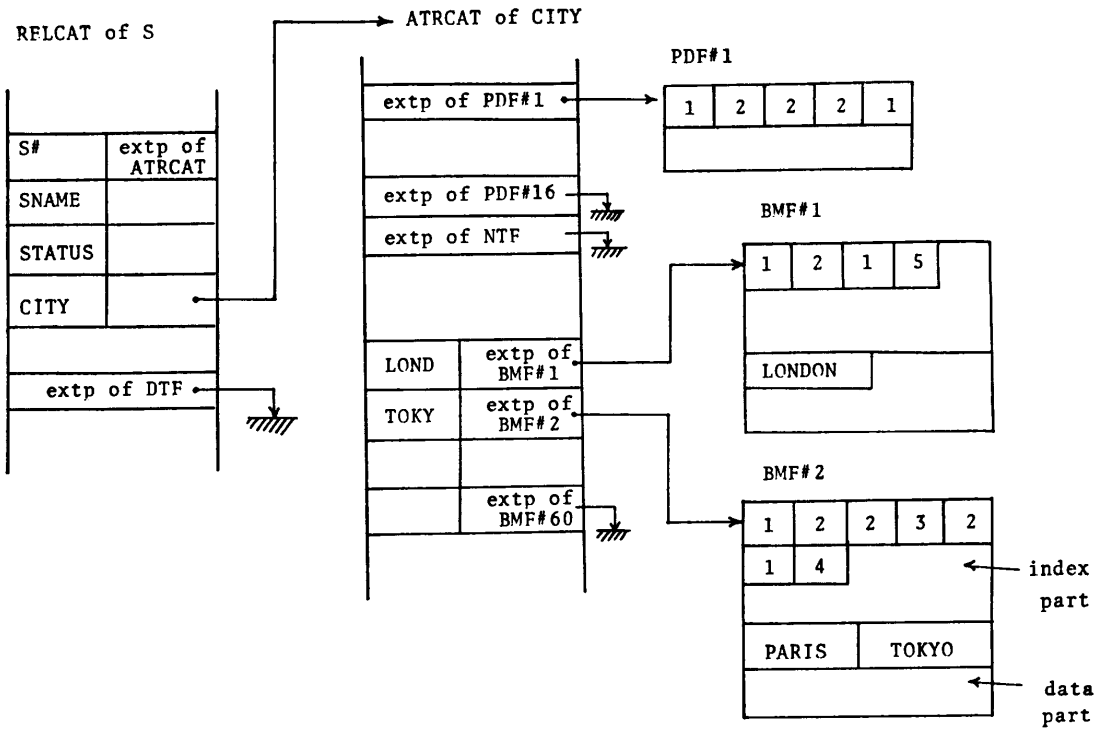

図 2 HTF の物理構造
Fig. 2 Physical structure of HTF.

いる．一つの PDF（2 ページ=1,024 バイト）で，1,024 個の情報を表現できる．すなわち，PDF #1 は，TID が 1 から 1,024 までの実現値の BMF 番号を，PDF #2 は 1,025 から 2,048 までの実現値の BMF 番号を表現している．PDF の最大数は 16 個であり，すべてを使用すると 16,384 個のタプルの情報を保持できる（ただし，これは OPT-R における制限であって，本質的な問題ではない）．

3.2 BMF

BMF は，索引部とデータ部とから構成される．データ部には，当該 BMF のバケットに入る相異なる実現値を格納している．索引部は，データ部の実現値をとるタプルの TID を並べたものであり，(データ部のエントリ番号，TID の数，TID, …, TID) なる形式をしている．すなわち，一つの BMF に入る値に関しておのおのインバートしていることになる．

索引部およびデータ部のサイズの初期値は，ともに 1 ページである．タプルの追加，更新等の変更操作に伴うデータベース容量の増大に対して，以下の方法により動的に構造の変更を行う．

（1） 分割 (splitting)

BMF の索引部あるいはデータ部があふれる場合，当該 BMF を約半分ずつの量の二つの BMF に分割する．ただし，BMF エクステントポインタに空きがある場合に限る．

（2） 索引部の拡張 (expansion of index part)

当該 BMF において同じ値をもつタプルが非常に多くなり，それまでの索引部に収容しきれなくなった場合に索引部を 1 ページ拡張する．

（3） データ部の拡張 (expansion of data part)

実現値の最大値および最小値を含む両端のバケットに対応する BMF のデータ部があふれる場合，あるいは属性カタログ中の BMF エクステントポインタがすべて使用中で当該データ部があふれる場合に，データ部を 1 ページ拡張する．

3.3 CTF

OPT-R では，32 バイトまでのデータに関する CTF は BMF のデータ部として実現しており，それを越えるものに関しては，2 章で述べた CTF を使用している．データ部として実現する場合のデータ長は，エンコード/デコードに要する時間と生データを直接操作するに要する時間の比で決定されるべきであるが，いまのところ明確な基準はない．32 バイトとしたのは，OPT-R を実現している計算機の命令体系による

ものであり，32バイトまでのデータの主メモリ内での転送が1命令で可能であることによる．

3.4 圧縮 HTF のストレージコスト

2.2節で使用した記号を再び用いると，一つの BMF の索引部のストレージコストは，

データ部のエントリ番号フィールド：$2d/k$

TID の数を表現するフィールド：$2d/k$

TID フィールド：$2n/k$

であるから，$\lceil(4d+2n)/kp\rceil$ ページとなる（ただし，$\lceil\ \rceil$ は ceiling を表す）．また，データ部のストレージコストは，$\lceil dl/p\rceil$ ページとなるから，総ストレージ量は

$$k\lceil(4d+2n)/kp\rceil+k\lceil dl/p\rceil$$

ページとなる．明らかに，これは $O(n)$ である．

ここで，すべての BMF が連続して格納されると仮定して，従来の転置型ファイルと比較すると，

$$d\leq n(l-2)/(4+kl)=\frac{n}{k}\left(1-\frac{4+2k}{4+lk}\right)$$

（ただし，$l\geq 3$）

のとき，ストレージコストは圧縮 HTF のほうが小さくなる．本不等式は，2.2節の d に関する同様の不等式と比較すると，より現実的であるといえる．

4. 関係操作アルゴリズム

OPT-R では，BEGIN コマンドから END コマンドまでを一つのトランザクションとして処理する[10]．すなわち，BEGIN コマンド投入後から END コマンド投入までに入力されたトランザクション記述用言語を，中間テキストに次々とコンパイルする．最後に，END コマンドが入力されると，それまで入力されてコンパイルされた中間テキストを一つのタスクとして実行する．本章では，前章までに述べた HTF をもとに，OPT-R でサポートしているトランザクション記述用言語における検索文（SELECT 文）およびデータベース操作文（INSERT, DELETE, UPDATE の各文）を実行する場合の処理方式を示す．

本章では，OPT-R のトランザクション記述用言語を簡単化し，以下の形式の文を扱う（ただし，大文字は予約語を表す）．

- SELECT atrname-list FROM relname-list WHERE boolean
- INSERT INTO relname (atrname-list) VALUES (value-list)
- DELETE FROM relname WHERE boolean
- UPDATE relname SET atrname=value, …, atrname=value WHERE boolean

INSERT 文の実行は，VALUES 句に指定された値，すなわちタプルを関係に追加するだけである．それ以外の各文の実行は，以下の2段階の処理から構成される．

（1） WHERE 句が指定された場合は，WHERE 句の条件式 (boolean) を評価することによって，条件を満足する TID の集合を得る．指定されない場合は，関係を構成するすべての TID が該当候補となる．

（2） (1)の TID 集合中の各タプルに対して，指定された操作（検索，削除あるいは更新）を行う．SELECT 文の場合は，SELECT 句に指定された属性の実現値を結果格納領域へ設定する．DELETE 文の場合は，タプルを構成する各属性の実現値を削除する．さらに，UPDATE 文の場合は，SET 句に指定された値によって各属性の実現値を更新する．

以下，本章では，条件式の評価に関して，関係操作のうちの選択操作 (selection) と結合操作 (join) について説明する．さらに，SELECT 文の処理に対応する結果の取り出し (answer fetch) と，変更操作について述べる．

4.1 データベースのアクセスパス

ユーザの入力した各文における条件式は，中間テキスト中の演算テーブル (OPRTAB: OPeRation TABle) に格納される．トランザクション管理は，この OPRTAB をもとに一つのトランザクション内の演算順序をスケジュールする．

条件式の評価においては，そのオペランドとして，必ず一つ以上の属性に関するデータを主メモリに読み込まなければならない．以下に，属性オペランド取出しに必要な一連の処理を示す．

（1） 関係マスタカタログのサーチにより，当該属性が含まれる関係の関係カタログを主メモリに読み込む．

（2） 関係カタログ中の属性名をサーチし，当該属性の属性カタログを主メモリへ読み込む．

（3） OPRTAB の演算コード (selection, join, and 等) と属性カタログ中の BMF エクステントポインタにより，アクセスすべき BMF を決定する．

（4） 該当する BMF を主メモリへ読み込み，OPRTAB の演算コードに対応した関係操作あるいは論理演算等を行い，結果の TID 集合を生成する．

4.2 アクセスすべき BMF の決定

関係操作実行時には，当該属性のすべての BMF を読み込む必要はなく，演算に必要な BMF を読み込むことで条件式の評価が可能となり，HTF の利点の一つになっている．本節では，3章で述べた HTF における BMF へのエクステントポインタをもとにした BMF 番号の決定法について説明する．以下で使用する記号を定義する．

$m_i(A)$: 属性 A に関する BMF $\#i$ 内の実現値の最小値（ただし，$1 \leq i \leq k_A$）

$M_i(A)$: 属性 A に関する BMF $\#i$ 内の実現値の最大値（ただし，$1 \leq i \leq k_A$）

k_A: 属性 A に関する BMF の個数

X: アクセスすべき BMF 番号の集合

θ: 比較演算子

（1） 選択操作

関係 R の属性 A に関する選択操作*

$$R[A\theta c] \triangleq \{r \mid r \in R \wedge r[A] \theta c\}$$

（ただし，c は定数）に関しては，アクセスすべき BMF 番号の集合 X は以下のようになる．

$$X = \begin{cases} \{i \mid m_i(A) \leq c \leq M_i(A)\} & (\theta: =) \\ \{i \mid M_i(A) \geq c\} & (\theta: >\text{or} \geq) \\ \{i \mid m_i(A) \leq c\} & (\theta: <\text{or} \leq) \\ \{i \mid 1 \leq i \leq k_A\} & (\theta: \neg =) \end{cases}$$

（2） 結合操作

関係 R と関係 S のそれぞれの属性 A, B に関する結合操作

$$R[A\theta B]S \triangleq \{(r, s) \mid r \in R \wedge s \in S \wedge (r[A] \theta s[B])\}$$

に関しては，アクセスすべき BMF 番号の組の集合 X は以下のようになる．

$$X = \begin{cases} \{(i, j) \mid [m_i(A), M_i(A)] \cap [m_j(B), M_j(B)] \neq \phi\} & (\theta: =) \\ \{(i, j) \mid [m_i(A), M_i(A)] \cap [m_j(B), M_j(B)] \neq \phi \text{ or } m_i(A) \geq M_j(B)\} & (\theta: >\text{or} \geq) \\ \{(i, j) \mid [m_i(A), M_i(A)] \cap [m_j(B), M_j(B)] \neq \phi \text{ or } M_i(A) \leq m_j(B)\} & (\theta: <\text{or} \leq) \\ \{(i, j) \mid 1 \leq i \leq k_A, 1 \leq j \leq k_B\} & (\theta: \neg =) \end{cases}$$

ここで，$[m_i(A), M_i(A)] \cap [m_j(B), M_j(B)] \neq \phi$ は，

$$(m_j(B) \leq m_i(A) \leq M_j(B)) \text{ or } (m_i(A) \leq m_i(B) \leq M_j(A))$$

を表しており，二つの区間に重なりがある場合を意味している．

選択操作および結合操作において比較演算子 θ が $\neg =$ の場合は，上記方法ではすべての BMF をアクセスすることになる．したがって，比較演算子が $\neg =$ の場合のみ $=$ として演算し，全体の TID 集合と $=$ を満たす TID 集合の差 (difference) によって結果を生成している．

4.3 選択操作の実行

以下の検索文を考える．

SELECT S# FROM SPJ WHERE QTY=200

WHERE 句の条件式 "QTY=200" を評価するには，選択操作を1回行えばよい．この操作の実行にあたっては，SPJ の関係カタログ（2ページ），QTY の属性カタログ（3ページ）の計2回5ページのセグメントアクセスが行われる．次に，実際の値が格納されている QTY の BMF をアクセスすることになる．この場合の選択操作における比較演算子は $=$ であるから，値 200 を含む BMF の読込みだけで条件式の評価が可能となる．

$=$ 以外の演算子に関しても，境界となるバケットの BMF のアクセスだけで条件式の評価が可能である．その他の値に関しては，結果の取出しの際に PDF をアクセスすればよい．たとえば，条件式が QTY$>$200 のときは，200 を含む BMF エクステントポインタに対応する BMF と，当該エクステントポインタ以降のエクステントポインタに対応する BMF がすべて結果となる．このとき，

min（該当する BMF の総ページ数，
　PDF の総ページ数）

を評価し，小さいほうのファイルが読み込まれることになる．ほとんどの場合，PDF が読み込まれることになるのは明らかである（たとえば，16, 384 タプルの場合，PDF は 16 ページとなる）．

4.4 結合操作の実行

関係 S と J に関する以下の検索文を考える．

SELECT S.S#, J.JNAME FROM S, J
　WHERE S.CITY=J.CITY

条件式の評価のために，S および J の関係カタログ，S.CITY および J.CITY の属性カタログの計4回10ページのセグメントアクセスが必要である．

結合操作の実行においては，両属性カタログ内の BMF エクステントポインタにおける各バケットの最大値を比較することによって，バケットが重なり合う BMF 番号の組を作る（4.2節参照）．その後，各組に現れる BMF の値を比較することにより，結果の TID 集合が得られる．

---

* 各操作の詳細は，文献15) を参照のこと．さらに記号に関しては，文献16) を参照のこと．

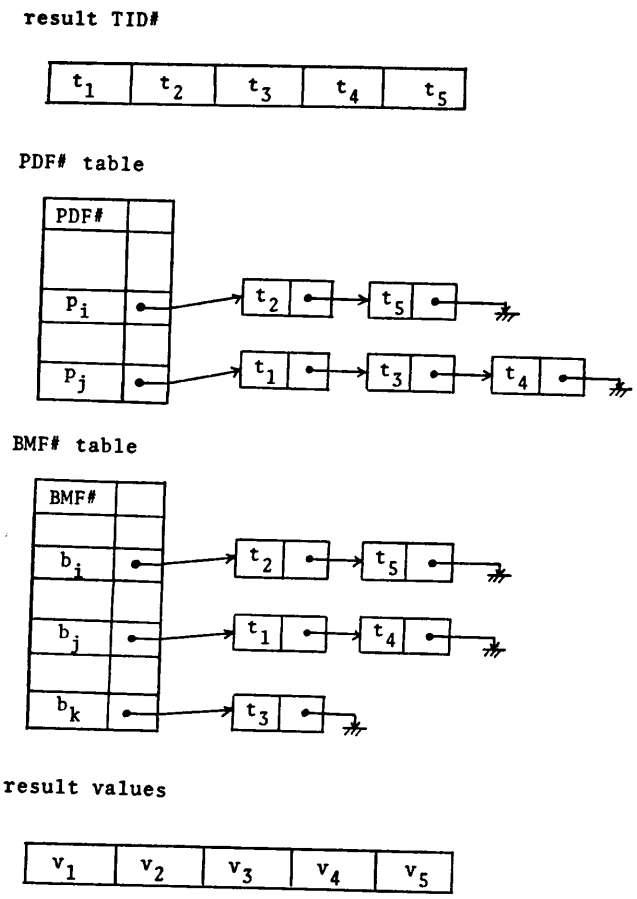

図 3 結果取出しのための PDF# テーブルと BMF# テーブル
Fig. 3 PDF# table and BMF# table for answer fetch.

BMF へのアクセス回数は，この組を構成するバケットの個数できまる．一方の属性の組に現れるバケット数を k_1，他方を k_2 とすると，最小で k_1+k_2 回，最大で $k_1 \cdot k_2$ 回のブロックアクセスが必要となる．本方式は，文献17)における nested block join と同様の方式であるが，文献17) ではこのような候補ブロックの推定は行っておらず，従来手法よりもブロックアクセス回数がかなり低減されることが期待できる．さらに，OPT-R では，ブロック単位の主メモリ内の結合操作に関しては，すでに文献18) で提案した手法を採用しており，大幅な CPU 時間の短縮を実現している．

4.5 結果の取出し (answer fetch)

本節では，SELECT 文の処理に対応する結果の取出しについて説明する．

条件式の評価によって得られた結果の TID 集合から，SELECT 句に指定されている属性の実現値を取り出し，トランザクションの結果格納域に設定していく．以下，**図3**に従ってこの処理手順を説明する．なお，この処理は，SELECT 句に指定される属性のおのおのに対して個別に行われるものである．

(1) PDF # テーブルの作成

結果の TID 集合の各 TID に関して，

PDF # = ⌈TID/1024⌉

により，当該 TID のバケット番号 (すなわち BMF の番号) を含む PDF のエクステント番号 PDF # を求める．この操作を結果の数だけ繰り返し，図3の PDF # テーブルを生成する．この PDF # テーブルは，結果の TID を含む BMF の番号を求める際の，PDF のランダムアクセスを避けるためのものである．すなわち，PDF の1エクステントに含まれる結果の BMF 番号を，一括して求めるために使用する．

(2) BMF # テーブルの作成

PDF # テーブルの各エントリに対して，その PDF # と

offset = TID − 1024(PDF # − 1)

なる PDF のエクステント内オフセットとを使用して，当該 TID を含む BMF の番号を求め，BMF # テーブルに登録する．本テーブルも上記と同様に，BMF アクセスをクラスタ化するためのものである．

(3) 結果の取出し

BMF # テーブルにおいて一つ以上の TID をもつ BMF を順次読み込んで，BMF 内の索引部のサーチにより当該 TID に対するデータ部の値 (実現値) を取り出し，結果領域に設定する．

4.6 結果の見積りと演算順序の最適化

OPT-R では，属性カタログ中の BMF へのエクステントポインタを使用して，各操作の結果の見積りを行っている．本節では，選択操作と結合操作に関してだけ説明するが，他の操作に関しても容易に適用できる．以下に使用する量を定義する．

INDEX$_i(A)$: 属性 A の BMF 番号 i の索引部のタプル数

LENGTH(A): 属性 A の実現値の長さ

p: ページサイズ，b: ブロックサイズ

選択操作の場合は，候補 BMF の番号を $i_1, i_2, \cdots, i_k$ とすると，結果のページ数は，

$$b = \left\lceil \left(\sum_{l=1}^{k} \mathrm{INDEX}_{i_l}(A) \cdot \mathrm{LENGTH}(A) \right) \Big/ p \right\rceil$$

となる．また，結合操作の場合は，候補ペアを $(i_1, j_1), \cdots, (i_k, j_k)$ とすると

$$b=\left\lceil \left(\sum_{l=1}^{k} \mathrm{INDEX}_{i_l}(A) \cdot \mathrm{INDEX}_{j_l}(B) \cdot \mathrm{LENGTH}(A) \right) \Big/ p \right\rceil$$

となる．これは，参考文献19)の手法を具体化したものであり，演算順序の最適化に関しては，上式で見積りを行った結果のページ数が最小のものから行えばよいことがただちにわかる．この方法は，multiple join のときに有効となる．

4.7 変更操作

本節では，データベースの変更を行う場合の処理方式について述べる．なお，以下の各文の処理は，属性単位に行われるものである．

（1） 挿入操作 (INSERT 文の処理)

VALUES 句で指定された値が，INTO 句に指定された属性の実現値として追加される．INTO 句に当該関係のすべての属性が指定されていないときは，未指定の属性に関して空値 (null value) が仮定される．以下に処理の流れを示す．

（a） 属性カタログを読み込み，エクステントポインタによって VALUES 句に指定された値が属す BMF の番号を求める．

（b） 当該 BMF を読み込みデータ部をサーチし，同じ値が存在するか否か調べる．同じ値が存在する場合は，索引部の既存エントリに TID を追加する．存在しない場合は，索引部およびデータ部に新しいエントリを追加する．

以上の処理において，BMF の索引部あるいはデータ部があふれる場合は，3.2節で述べた BMF の分割あるいは拡張処理が行われる．

（2） 削除操作 (DELETE 文の処理)

DELETE 文の処理の流れは，以下のようになる．

（a） WHERE 句で指定された条件を評価し，条件を満足する TID の集合を得る．

（b） DTF に削除 TID を追加する．

（c） 結果の取出しと同様に，PDF を読み込み当該 TID の属す BMF 番号を求める．

（d） BMF をアクセスし，索引部およびデータ部のエントリを削除する．

（3） 更新操作 (UPDATE 文の処理)

WHERE 句で指定された条件を満たすタプルを，SET 句で指定された値に更新する．更新操作は，削除操作と挿入操作の組合せで実現される．

以上，データベースに対する変更操作について述べたが，HTF を使用する場合，変更操作は検索操作に比較して，セグメントアクセスの回数が増大する傾向にある．とくに，変更操作がタプル単位のしかも複数タプルにまたがる場合の入出力回数の増大は避けられない．これは，属性単位の格納法である転置型ファイルを基本としていることによる．しかし，タプル単位の格納法では，条件式の評価の際に，最悪の場合，関係のすべてのデータを読み込む可能性が生ずる．変更操作あるいは結果の取出しのようなタプル単位の処理と，検索条件の評価などの属性単位の処理が，どれくらいの比率でシステムに要求されるかによってシステムの性能が左右されるといえよう．OPT-R では，頻繁な更新操作は想定していないが，大量のタプルを同時に操作するユーザ要求以外は，更新操作であっても問題はないと考えている．

5. おわりに

本論文では，データベースオペレーティングシステム OPT-R のデータベースの物理構造とそれに基づく関係操作の方法について述べた．本方式の特徴は，転置型ファイルに簡単な副次索引を付加することにより，アクセスの高速化（すなわち，前処理としてのアクセスすべき候補ブロックの推定）を可能とし，従来の転置型ファイルに比較してアクセス回数を低減していることにある．

本ファイル構造は，属性単位に逆ファイルを構成したものと見ることができる．PDF のバケット（すなわち，BMF のエクステントポインタ）は，数 KB のオーバヘッドであるから，すべての属性に関して副次索引を付加することが可能である．したがって，従来構造のファイルに見られるような，どの項目に副次索引を付加するかといったいわゆる“最適インデックス選択の問題”からは完全に解放される．今後の課題としては，時間変化に対する副次索引の変更時点の設定，タプル単位アクセスの高速化がある．なお，本ファイル構造に関する性能評価等は，OPT-R の性能評価に関する論文で発表する予定である．

参考文献

1) Senko, M. E.: Data Structure and Data Accessing in Data Base Systems Past, Present, Future, *IBM Syst. J.*, Vol. 16, No. 3, pp. 208-

257 (1977).
2) Schkolnick, M.: A Survey of Physical Database Design Methodology and Techniques, *IBM Res. Rep.*, RJ 2306, pp. 1-14 (1978).
3) Schmidt, J. W. and Brodie, M. L. ed.: Relational Database Systems Analysis and Comparison Springer-Verlag, Berlin, Heidelberg, New York (1983).
4) Batory, D. S.: On Searching Transposed Files, *ACM TODS*, Vol. 4, No. 4, pp. 531-544 (1979).
5) Svensson, P.: On Search Performance for Conjunctive Queries in Compressed, Fully Transposed Ordered Files, Proc. Very Large Data Bases, pp. 155-163 (1979).
6) Tsuda, T. and Sato, T.: Transposition of Large Tabular Data Structures with Applications to Physical Database Organization, Part 1, *Acta Inf.*, Vol. 19, No. 1, pp. 13-33 (1983).
7) Tsuda, T., Urano, A. and Sato, T.: Transposition of Large Tabular Data Structures with Applications to Physical Database Organization, Part 2, *Acta Inf.*, Vol. 19, No. 2, pp. 167-182 (1983).
8) 佐藤隆士, 津田孝夫: 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).
9) 大久保英嗣, 津田孝夫: データベースオペレーティングシステムOPT-Rの設計目標とアーキテクチャ, 情報処理学会論文誌, Vol. 25, No. 4, pp. 535-543 (1984).
10) 大久保英嗣, 津田孝夫: データベースオペレーティングシステムOPT-Rのタスク管理とトランザクションのスケジューリング技法, 情報処理学会論文誌, Vol. 25, No. 4, pp. 544-551 (1984).
11) 大久保英嗣, 津田孝夫: データベースオペレーティングシステムOPT-Rのメモリ管理方式, 情報処理学会論文誌, Vol. 25, No. 4, pp. 552-559 (1984).
12) Ragan, D. P. and Jones, A. S.: High-Level Language Implementation of Bit Map Inverted Files, *Comput. Biomed. Res.*, Vol. 11, No. 6, pp. 595-612 (1978).
13) Floyd, R. E.: Data Base Compression Through Combined Softwere Techniques, *IBM Tech. Discl. Bull.*, Vol. 21, No. 2, pp. 458-462 (1978).
14) Dishon, Y.: Data Compaction in Computer Systems, *Comput. Des.*, Vol. 16, No. 4, pp. 85-90 (1977).
15) Date, C. J.: *An Introduction to Database Systems*, 2nd ed., Addison-Wesley, Reading, Mass. (1977).
16) Codd, E. F.: Relational Completeness of Data Base Sublanguages, Courant Computer Science Symposium 6, Database Systems, ed. by Rustin, R., Prentice-Hall, Englewood Cliffs, New Jersey, pp. 65-98 (1972).
17) Kim, W.: A New Way to Compute the Product and Join of Relations, Proc. ACM SIGMOD, pp. 179-187 (1980).
18) 大久保英嗣, 津田孝夫: 順序保存ハッシュ関数による高速ジョインアルゴリズム, 情報処理学会論文誌, Vol. 25, No. 1, pp. 59-65 (1984).
19) Merrett, T. H. and Otoo, E.: Distribution Models of Relations, Proc. Very Large Data Bases, pp. 418-425 (1979).

(昭和59年4月27日受付)
(昭和59年7月19日採録)