統合化ファイルアクセス法(IFAM)の集合操作*

3W－7

古舘 丈裕　鈴鹿 豊明†
日立ソフトウェアエンジニアリング(株)‡

1 はじめに

我々はマルチメディアデータベースの研究の一環としてファイルアクセス法という観点からのアプローチを行っており、統合化ファイルアクセス法（IFAM）を試作した。これはマルチメディアデータへの対応と既存のファイルアクセス法の機能の統合化を狙った新しいアクセス法であり、長大かつ可変長のデータを効率よく扱えることと柔軟なキー検索機能を備えていることが特長である。

IFAMでは一つのタグを重複して複数のレコードに付けることができる。よって、タグをキーとして検索を行なった場合に結果として複数のレコードが該当することになる。ここで、タグによって検索した複数のレコードはひとまとめにして集合として扱えると便利である。本稿ではレコード集合のデータ構造とその操作についての実現手法について述べる。

2 IFAMのファイル構成

IFAMのファイル構成は以下のようになっている。

(1) アトム

IFAMが取り扱うデータの最小単位であり現在は1バイトで実現している。

(2) レコード

アトムの順序列。長さに制限はなく、各々のレコードの長さは一定でなくとも良い。レコードはユニークな識別子RIDで識別される。

(3) ファイル

0個以上のレコードの集まり。

(4) タグ

レコードに付けることのできる任意の値。

(5) タグクラス

タグのディスジョイントな分類を表す。

(6) タグドメイン

各タグクラスが持っているタグの型で、整数・浮動小数・バイト列がある。

レコードの長さに制限がないので、データとして数値や文字列だけでなく、画像や音声などの長大データをレコードにすることができ、レコードの長さが変化するような途中位置へのアトムの挿入や削除が可能である。また、レコードには複数のタグクラスのタグをそれぞれ複数付けることができ、タグは他のレコードに重複して付けてもよい。このような柔軟なインデックス構造により、m:nの関係を直接実現することができる。図1は複数のタグの付いたレコードの例である。

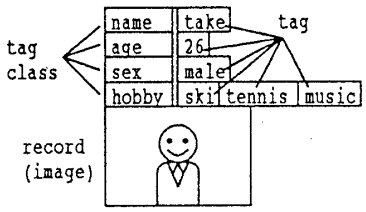

図1: IFAMレコードの例

3 集合とその操作

複数のレコードを集合として捉え、その集合あるいは複数の集合間に演算を施すことはデータベースでは頻繁に起こる処理である。この操作をアクセス法側で提供していればアプリケーション開発の負担は軽減され、論理的なプログラミングを行うことができる。そこでまず必要な集合操作の洗い出しと定義を行った。その際の方針としては汎用性を考えて、「操作の提供に当たっては単純な操作を提供し、複雑な操作はその組合せによって実現する」ことにした。

集合を生成するには、レコードを一つ一つ指定して集合を作ることと、タグによる検索結果を集合とすることが考えられる。前者は空集合を生成してそれに指定したレコードを挿入していけば良い。後者については、例えば図1のようなレコードが入っているファイルで、「性別が男」とか「20歳以上30歳未満」という検索を行うと、結果として複数のレコードが該当する。これらの検索結果をそれぞれ集合とすれば良い。検索結果が集合となれば、当然それらの集合間の演算も必要となってくる。これらより、集合操作を以下のように定義した。

(1) 集合の生成・削除

- 空集合の生成
- タグあるいはタグの範囲の指定による検索結果からの集合の生成
- 不要な集合の削除

*Set Operations of Integrated File Access Method

†Takehiro Furudate, Toyoaki Suzuka
‡Hitachi Software Engineering Co., Ltc.

(2) 集合要素に関する操作
- 指定した集合への要素の挿入
- 集合から要素数を求める
- 集合から要素を取り出す
- 指定した要素が指定した集合に含まれているかどうかを求める

(3) 集合間の操作
以下の集合を新たに生成する。
- 二つの集合の積
- 二つの集合の和
- 二つの集合の差

これらを組み合わせれば全ての複雑な演算を表現できる。ここで、否定ではなく差にしたのは、集合の補集合を求める処理はインプリメンテーション上で非常にコストがかかり、否定を何度も含む複雑な演算では性能が大きく低下してしまうからである。また、最小演算系としてNANDやNORも考えられるが同様の理由により採用しなかった。

4 集合操作の実現手法

IFAMの集合操作の実現に当たって、次の2項目を満たすことが必須である。なお、レコードの集合はレコードを一意に表すRIDの集まりとして実現する。

(1) 集合間の演算が効率良く行える。

(2) 集合の要素の取得や挿入、削除が効率良く行える。

前者については、集合の実現構造中に要素であるRIDを昇順にソートして格納することで、付き合わせ処理の高速化を図った。後者については、RID順集合の実現構造を木構造とした。

RIDを検索するときの性能を考えて、木構造にはB^+-treeを採用した。またアプリケーションが集合からRIDを取り出すインタフェースは、レコードを一つ一つ指定して入手する形とした。具体的には、集合の要素に仮に0,1,2,…と番号を付けてその番号を指定して一つずつ取り出す。このためにはB^+-treeに位置を指定して要素を取得する機能を追加する必要がある。

以上の要件をふまえたうえで、集合の実現構造を図2に示すような木構造とした。リーフノードにRIDを昇順にソートして格納する。それ以外のノードにはその子部分木に存在するRIDの数（これを重みという）を格納する。この重みを利用することで、先頭から何番目の要素を取り出すという操作が効率良く行える。集合の要素数をNとするとき、一つの要素の挿入や取り出しにかかる時間はlogNのオーダになる。

5 集合要素からの帳票作成

一般的な事務処理のアプリケーションをIFAMで実現しようと考えたとき、帳票に現れる各項目をタグクラスで表現するのが自然である。出力する帳票の内容はある検索の結果としてできた集合の要素となっている。上で

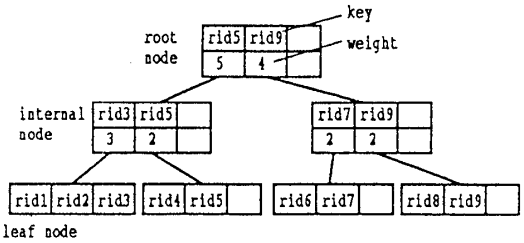

図2: 集合のデータ構造

述べたように集合中の要素はRID順で並んでいるが、帳票では項目の値によって出力する順序を決めるのが普通である。

そこで、タグ値による集合のソート操作を提供した。これは、集合に対しソートの対象となるタグクラスを指定し、新たなソート済みの集合を生成するものである。

ところで通常のソートキーとして第1キー、第2キー、というように複数のキーで優先順位を指定してソートすることもある。例えば、名前、クラス、得点がタグとして付いた個人レコードを得点順にソートする場合、同じ得点のレコードはクラス順に、クラスも同じなら名前順に並べたいとする。

実現に当たっては、今回は単純化の方針からソート操作で指定できるソートキーは一つだけとした。ただし、同じ値のキーが現れた場合、元の集合の順位を保存することにした。よって優先順位の低い順からソートするタグクラスを指定し、この操作を繰り返せば期待するソートが可能となる。先の例では、名前、クラス、得点の順でタグクラスを指定していけば良い。

6 むすび

IFAMのレコードの検索結果を集合として取り扱い、集合演算、タグ順ソートの操作を提供した。ファイルアクセス法側で集合操作を提供することにより、アプリケーション開発のコストを軽減することができる。また、レコードを集合として扱うための処理をアプリケーション側から直接行わず内部的にできるので、アクセス処理のオーバーヘッドが減少し処理速度の向上が期待できる。

参考文献

[盛屋 92] 盛屋, 鈴鹿: 統合化環境を目指したファイルアクセス法の提案, 情報処理学会第44回全国大会, 1992.

[STON84] M. Stonebraker, L. A. Rowe: Database Portals :A New Application Program Interface, Proc.10th VLDB, 1984.