データベースシステム 89−20
（1992. 7. 24）

データベース集合演算の高速化

石川 英彦　石山 政浩　有澤 博
横浜国立大学工学部

データベースにおいて、整数や文字列などの「値」の扱いは、各種のモデルやパッケージによって様々である。本稿では、「値」自体も他の表示不可能エンティティと同様に扱うべきであると考え、それを利用しやすい形で表現するための一つの方法を提案する。その方法に付随して、集合演算の高速化がデータベース全体の高速化につながることを示し、高速処理のための実現手法について考察する。

Some Techniques to Speedup Set Operations

Hidehiko ISHIKAWA　Masahiro ISHIYAMA　Hiroshi ARISAWA
Faculty of Engineering
YOKOHAMA NATIONAL UNIVERSITY

In the Database modeling, there is multifarious way to deal with VALUEs, such as INTEGERs, STRINGs, and so on. We consider that it is important to treat a VALUE as an entity related with other entities. We also present a data structure which represent a set of VALUEs as a totally ordered set, and a mechanism for high-speed calculation of set operations.

1 はじめに

いままで、データベースでは整数や文字列のような値の取り扱いについて詳細な議論はなされていなかった。ところが実際には、単位系の異なる数値を同じ「値」として扱う必要性はほとんどないように、「値」そのものを扱うだけでは検索や操作に不都合が生じる。そこで、本稿では値データの表現方法を提案し、さらにデータ間の集合演算の効率化について議論し、それらを用いることによってデータベースの検索を効率よく行うことができることを示す。

2 集合演算による検索の高速化

本節では、以降の議論の土台となる架空のデータモデルをもとにしたデータベースと、関連するいくつかの用語について定義を行い、その上での検索の高速化について議論する。

2.1 データベースの定義

ここで、想定するデータモデルの定義を行なう。定義の一部は、筆者らが提案しているAISモデル[1]と呼ばれる関数型データモデルを基にしているが、一般的な意味データモデルや関数型データモデルにも対応づけのとれるものと考えている。

ここでは、データベース化される実世界中の事象や事物として認識される全ての「もの」を**エンティティ**と呼ぶ。エンティティはデータとしては内部構造を持たず、実世界中に存在する「もの」のデータベース世界における代理物としての点にすぎない。例えば、歌謡曲のデータベースにおいて、歌手、その歌手の名前、性別、歌っている曲などは全てエンティティである。このうち、名前や性別のような、ある種のエンティティはそれ自身の値を表すだけのものである。本稿ではこのようなエンティティを**値**(value)と呼ぶ。また、同一の意味を持つエンティティの集まりをエンティティ・セットと呼ぶ。

エンティティ間の対応付けは $f(a_1) = b_1, f(a_2) = b_2 \cdots$ のようにインスタンスを列挙することにより定義される関数として表現される (a_1, b_1 などは全てエンティティである)。ここで、関数は複合値関数の場合もある (例: 一人の歌手が複数の曲を歌う場合)。

データベースを図示する際、ここではエンティティ・セットを円で表現し、エンティティ・セット間の関数を矢印つきの弧で表現することにする。矢印の向きは関数値となるエンティティ・セットを示している (例えば、ある歌手について *sing*(歌手) はその歌手の歌う曲の集合となる)。例として、歌謡曲データベースを図1に示す。

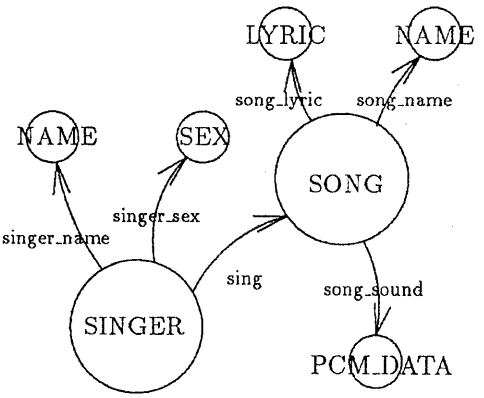

図 1: 歌謡曲データベース

ここで、ある条件を満たすエンティティの集合を得ることを、**検索**と呼ぶ。例えば、「『卒業』という名前の曲を全て挙げる」という操作は *song_name* という関数に代入すると「卒業」という値を返すエンティティの集合を得る、という検索に相当する。

2.2 集合演算を用いた検索

いま、二つ以上の条件による検索を考える。図1で表現されている「歌謡曲データベース」に対して、「『卒業』で始まる曲名の曲を歌うすべての女性歌手を知りたい」という問い合わせを考える。この検索処理の最も簡単で最も効率の悪い方法は、「歌手」の一人一人について性別と歌っている曲名を調べ、女性でない歌手と『卒業…』を歌っていない歌手を除外する、という方法である。このとき、事実上データベース中の全ての「歌手」データが検索されることになる。

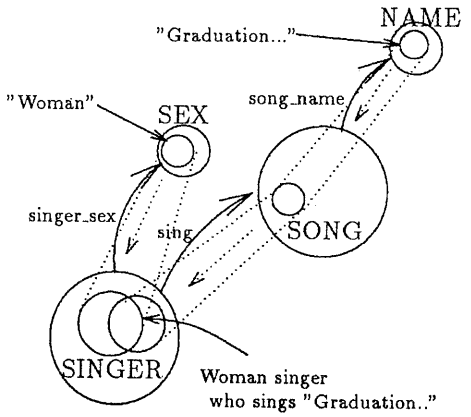

図 2: 集合演算による検索

これに対して、「女性歌手」と「『卒業…』を歌っている歌手」の集合をデータベースから検索することができれば、その積集合を計算するだけでデータの検索は完了する(図2)。これによって全ての「歌手」データを検索することなく検索処理を行なうことができる。ただし、このような処理を行なうためには、「値」から逆関数によってその値を所有しているエンティティをデータベースから検索することができ、しかも検索されたエンティティの集合に対する集合演算を効率良く実行できなければならない。

以降の節では、上記の事柄を実現するための実現手法について議論する。

3 値データの表現法

通常のデータベースでは、「値」は文字列や数字といったそのもの自身によって表現されている。しかし、値そのもので構成された値データを用いて前述の事柄を実現するのは困難である。本節では、値データの集合が持つべき特徴を考察し、それに基づく値データの表現方法を提案する。

3.1 値データに必要な条件

2.2節で述べたような検索を可能とするためには、データベースの値データはそれ自身を表す情報以外にも、以下に挙げる点を考慮した情報を持たなければならない。

(1) 何を表す「値」なのかの判別

(2) 「値」の大きさ(量)が不定

(3) 「値」の大小関係と順序との不一致

(1)は「42」というデータが体重の「42Kg」なのか温度の「42度」なのかを判別する必要がある事を示している。また、(2)は一つの値として1バイトから動画像データまでを扱う可能性を考慮しており、(3)については漢字コードとその読みの順序の違いなどを挙げることができる。

これらの条件を考慮し、筆者は値データの集合はある関数の値域ごとに独立して存在し、それぞれの値データ集合は全順序で、その順序は値自体の順序や値の格納位置とは独立に決定することができなければならないと考える。

3.2 値データ集合の表現法

本節では前述の条件を踏まえ、二つの独立した順序を持つ一つのリニアリストを利用した値データの実現方法について議論する。データベース中では、一つの値データは図3のような一つのセルで表現される。このセルは、一つの値を他の値と区別するために一意に定められる識別子(エンティティ ID)と、実際の値の格納位置の情報、そして「値」の順序と「ID」の順序とで自分の次に存在するセルの位置の情報からなっており、これをエンティティ・セルと呼ぶ。

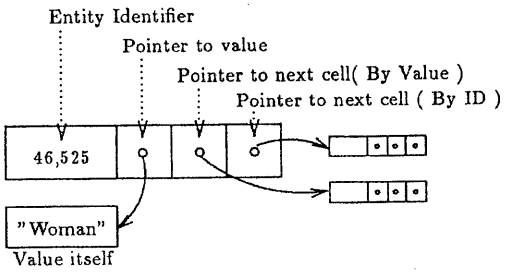

図 3: エンティティ・セルの構成

エンティティ・セルを格納位置情報によって連結することで、「値」と「ID」に関して順序付けられる値データの集合を表現することができる。このとき、値データの集合は、実際の値に関してのリストであると同時に、エンティティ ID に関してもリストとなっている。

普通、リスト構造から目的のセルを探し出す時はリストの先頭から一つづつ検索せざるを得ないが、このままでは検索の効率が悪い。そのため、検索の効率を上げるためにスキップ・セルを設定する。

スキップ・セルは図4のように、実際の値(またはエンティティ・セル)の位置を示す情報、子のセルの位置情報、子のセルの数 l、そして次の

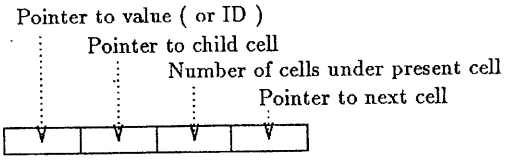

図 4: スキップ・セルの構造

スキップ・セルの位置情報からなっている。一つのスキップ・セルの次のスキップ・セルを検査することは、子のセルで l 個先のセルを検査することと同じであるため、$l-1$ 個の子セルを検査することなしにリストの先をたどることができる。また、スキップ・セルはスキップ・セルの上にも設定されるため、値データの集合は図5のように階層化されたリスト構造によって表現される。

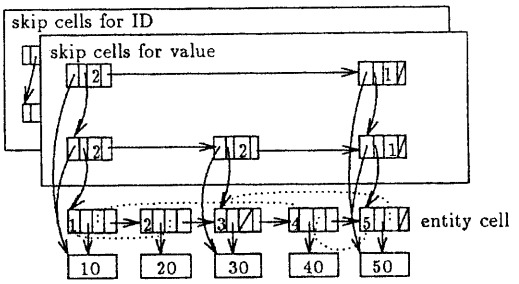

図 5: 順序つき値集合の表現

この図は値データ集合として、10, 20, 30, 40, 50 という順序を持った集合を表している。それと同時に、エンティティ ID 順に 2, 1, 4, 5, 3 という順序を持った集合も表現している。ここで、値 50 を持つエンティティを見つけたい時には、最上位のスキップ・セルを二つ検査するだけで知ることができるため、エンティティ・セ

ルのリスト構造だけを検査することに比べて目的のエンティティを速やかに検索することができる。

3.3 値データへのアクセス

本節では、上記の値データ集合に対する検索、及びエンティティの追加・削除といった操作をリスト上のどのような操作として実現するかについて述べる。

エンティティの検索

指定された値(またはID)を持つエンティティを得るためには、以下の手順を行なう。

(1) 最上位のセルを検査し、目的の値(またはID)かどうかを確認し、そうならば検索終了。

(2) 次のセルを検査し、目的かどうかを確認し、目的ならば検索終了。そうでない場合は、目的の値(またはID)と現在検査しているセルの値との前後関係から次に検査するセルを決定し、(2)を繰り返す。

(3) 次のセルが決定できない場合、検索失敗。

エンティティの追加

構造に「値」と「ID」を持ったエンティティの追加は、以下のように行なわれる。

(1) 追加するセルの位置を決定する。

(2) ポインタの付け換えによりエンティティ・セルの挿入を行なう。

(3) スキップ・セルで子セルの数がある一定数(スキップパラメータ s)より小さくなるように、スキップ・セルの挿入を行なう。

(4) (3)の必要がなくなるまで繰り返す。

エンティティの削除

指定された値またはIDを持つエンティティを削除するのは以下の手順による。

(1) 削除するエンティティ・セルの位置を決定する。

(2) エンティティ・セルを削除する。スキップ・セルによって指されているセルを削除した時は、スキップ・セルのポインタを削除したセルの隣(前または後)のセルへ付け換える。

(3) スキップ・セルを削除することが可能ならば削除する。

(4) (3)を実行されなくなるまで繰り返す。

3.4 構造へのアクセス効率

ここでは、順序つき値集合を単純なリスト構造で実現した場合に比較して、エンティティの挿入、削除及び検索に要する通過ポインタ数、ポインタの操作数そしてデータを格納するのに必要な領域を示す。

総エンティティ数を N、スキップパラメータを s、「値」の平均サイズを m_v、IDのサイズを m_{id} そしてポインタ一つのサイズを m_p とした時の結果を以下に示す。

	本稿で提案した構造	リスト構造
挿入時通過ポインタ	$s\log_s N$	N
削除時通過ポインタ	$s\log_s N$	N
検索時通過ポインタ	$(s-1)\log_s N$	N
挿入時操作ポインタ	$6\log_s N$	2
削除時操作ポインタ	$6\log_s N$	2
所要記憶容量	$N(m_v+m_{id}+3m_p)$ $+\frac{N}{s}(1+3m_p)$	$N(m_v+m_p)$

上記の結果により、本稿で述べた値データ表現方法を用いることで、妥当な記憶容量で比較的高速な操作を行なえると予測される。

4 集合演算の高速化

2節で述べたような、集合演算を用いた検索の高速化を行なうための実現手法について考察する。我々のデータベースでは、二つのデータ集合の間の集合演算(unionなど)を高速に処理するために、データの有無を表すビット列からなる作業領域を一時的に作成する。集合演算の過程で必要となる情報は、一つのエンティティについて1ビットで十分である。このとき、エンティティIDが3バイトで表されれば、作業領域の大きさは2^{24}ビット(=2Mバイト)となる。現在の計算機環境を考慮すると、この程度の大きさの作業領域をすべて主記憶上に置くことに問題はないと考える。

この作業領域を用いて先ほどの検索を実現すると、次のようになる(図6)。

- エンティティ・セットSINGERのための領域を主記憶上に確保し、初期化する。
- 与えられたエンティティ(「女性」)から展開されたSINGERのエンティティのIDに対応するビットをセットする。その後、もうひと組のエンティティ(「卒業」、「卒業写真」…)から展開されたSINGERのIDに対応するビットを検査し、論理積をとる。
- 領域のデータから積集合のエンティティIDの組を算出し、出力する。

このような手法により、集合演算の主要部分は主記憶上で行なわれるため、高速に集合演算を行なうことができ、データベース全体の効率をあげることができる。

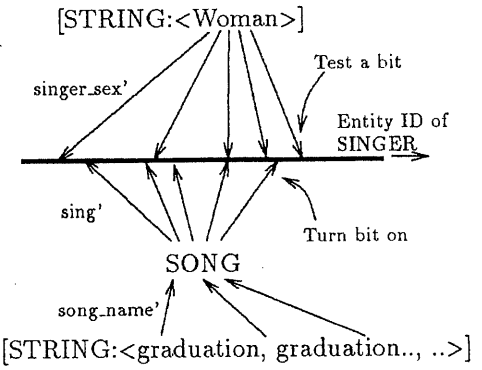

図6: 集合演算の高速化

5 おわりに

本稿において、筆者はデータベースにおける「値」の扱い方として、「値」自体をエンティティとみなし、その順序は「値」自体の大小とは独立に存在するべきであると考え、このような状態を効率的に表現するデータ型を提案した。さらに、集合演算を高速に実現するための実現手段について述べた。

これら、値データの表現方法と集合演算の高速化手法とによってデータベース検索をより効率良く実現することが可能となる。

参考文献

[1] Hiroshi ARISAWA, Hisayoshi NAGAE and Yasuko MOCHIZUKI, "Representation of Complex Objects in Semantic Data Model "AIS" and Implementation of Set Operators," IEICE TRANSACTIONS, VOL. E 74, NO. 1, (January 1991)