

(19) 日本国特許庁(JP)

(12) 公開特許公報(A)

(11) 特許出願公開番号

特開2010-198217

(P2010-198217A)

(43) 公開日 平成22年9月9日(2010.9.9)

(51) Int.Cl.	F 1	テーマコード (参考)
G 0 6 F 12/00 (2006.01)	G 0 6 F 12/00 5 1 2	5 B 0 7 5
G 0 6 F 17/30 (2006.01)	G 0 6 F 17/30 1 8 0 D	5 B 0 8 2
	G 0 6 F 17/30 1 5 0 Z	

審査請求 未請求 請求項の数 15 O L (全 36 頁)

(21) 出願番号 特願2009-41176 (P2009-41176)
 (22) 出願日 平成21年2月24日 (2009. 2. 24)

(特許庁注：以下のものは登録商標)

1. EEPROM

(71) 出願人 504145320
 国立大学法人福井大学
 福井県福井市文京3丁目9番1号
 (74) 代理人 110000338
 特許業務法人原謙三国際特許事務所
 (72) 発明者 都司 達夫
 福井県福井市文京3丁目9番1号 国立大
 学法人福井大学内
 Fターム(参考) 5B075 ND31 QT06
 5B082 GA06

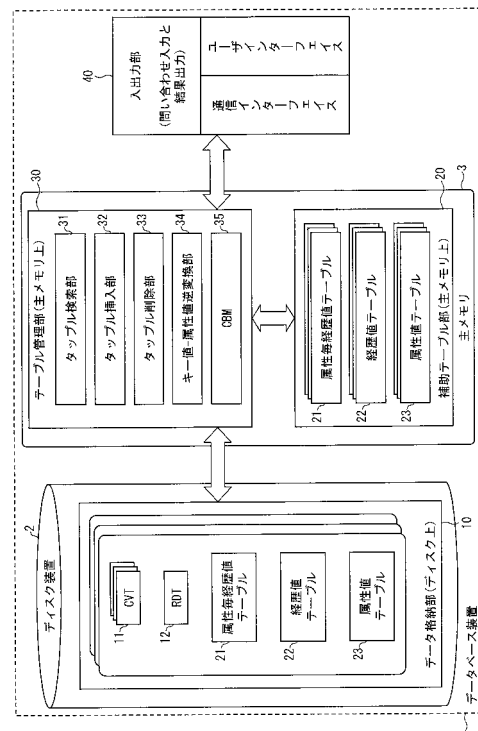
(54) 【発明の名称】 データベース装置、データベースの管理方法、データベースのデータ構造、データベースの管理プログラムおよびそれを記録したコンピュータ読み取り可能な記録媒体

(57) 【要約】

【課題】タプルの追加と高速検索が可能であり、ディスクスペースを効率良く使用する。

【解決手段】データベース装置1は、各属性値から拡張可能配列の添字に変換するためのCVT11と、タプルに対応する拡張可能配列の要素が属する部分配列の経歴値及びタプルの各属性値に対応する拡張可能配列の各添字のビットパターンを属性順に並べた座標パターンの2項組表現をキー値として登録したRDT12と、属性毎の配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブル21と、経歴値に対応する配列拡張した属性の次元、及び、当該経歴値に対応する拡張部分配列の任意の要素について、属性毎に拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブル22と、拡張可能配列の添字毎に対応する属性値及び該属性値を持つ全てのタプル数を登録した属性値テーブル23とを格納している。

【選択図】 図1



【特許請求の範囲】**【請求項 1】**

関係テーブルを用いたデータベース装置であって、

関係テーブルの各タプルに対応する拡張可能配列の要素の位置を示す位置情報をキー値として登録した要素位置データを格納したデータベース記憶部を具備するとともに、

上記位置情報が、要素が属する拡張可能配列の区画の位置を示す区画位置情報と、区画内における要素の位置を示す、当該タプルの各属性の属性値に一意に対応した値を所定の属性順に並べた座標情報と、を含む情報であることを特徴とするデータベース装置。

【請求項 2】

上記区画が拡張可能配列の部分配列であって、

上記データベース記憶部に、

関係テーブルの属性ごとに設けられ、属性の各属性値から拡張可能配列の添字に変換するための第 1 データと、

上記区画位置情報である、関係テーブルのタプルに対応する拡張可能配列の要素が属する部分配列の経歴値、および、上記座標情報である、当該タプルの各属性値に対応する上記拡張可能配列の各添字のビットパターンを所定の属性順に並べた座標パターン、の 2 項組表現をキー値として登録した、上記要素位置データである第 2 データと、

属性毎の配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、

経歴値に対応する配列拡張した属性の次元、および、当該経歴値に対応する拡張部分配列の任意の要素について、属性毎に拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、

拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、

を格納したことを特徴とする請求項 1 に記載のデータベース装置。

【請求項 3】

ある属性値を持つタプルを検索するとき、

上記属性値が属する属性の上記第 1 データを用いて、上記属性値を拡張可能配列の添字に変換し、当該次元の経歴値テーブルより経歴値を求めた後、

上記第 2 データを参照して、2 項組表現されたキー値中の座標パターンに含まれる、上記属性の属性値に対応するビットパターンが、上記添字のビットパターンと一致するタプルを抽出するタプル検索手段を具備することを特徴とする請求項 2 に記載のデータベース装置。

【請求項 4】

新たな属性値を持つタプルを挿入するとき、

配列拡張が必要である場合には、上記属性値が属する属性の部分配列を追加するとともに、経歴値をインクリメントして当該属性の上記属性毎経歴値テーブルに登録する基本データ拡張処理を行い、

配列拡張が不必要である場合、および、配列拡張が必要である場合に上記基本データ拡張処理を行った後、上記属性値を上記属性値テーブルおよび上記第 1 データに登録するとともに、当該タプルの各属性値に対応する上記座標パターンを生成して、経歴値および座標パターンの 2 項組表現をキー値として上記第 2 データへ挿入するタプル挿入手段を具備することを特徴とする請求項 2 に記載のデータベース装置。

【請求項 5】

関係テーブルを用いたデータベース装置におけるデータベースの管理方法であって、

上記データベース装置は、

データベース記憶部に、

関係テーブルの属性ごとに設けられ、属性の各属性値から拡張可能配列の添字に変換するための第 1 データと、

関係テーブルのタプルに対応する拡張可能配列の要素が属する部分配列の経歴値、および、当該タプルの各属性値に対応する上記拡張可能配列の各添字のビットパターン

10

20

30

40

50

を所定の属性順に並べた座標パターン、の 2 項組表現をキー値として登録した第 2 データと、

属性毎の配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、

経歴値に対応する配列拡張した属性の次元、および、当該経歴値に対応する拡張部分配列の任意の要素について、属性毎に拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、

拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、を格納しており、

ある属性値を持つタプルを検索するとき、

上記属性値が属する属性の上記第 1 データを用いて、上記属性値を拡張可能配列の添字に変換し、当該次元の経歴値テーブルより経歴値を求めた後、

上記第 2 データを参照して、2 項組表現されたキー値中の座標パターンに含まれる、上記属性の属性値に対応するビットパターンが、上記添字のビットパターンと一致するタプルを抽出することを特徴とするデータベースの管理方法。

【請求項 6】

関係テーブルを用いたデータベース装置におけるデータベースの管理方法であって、

上記データベース装置は、

データベース記憶部に、

関係テーブルの属性ごとに設けられ、属性の各属性値から拡張可能配列の添字に変換するための第 1 データと、

関係テーブルのタプルに対応する拡張可能配列の要素が属する部分配列の経歴値、および、当該タプルの各属性値に対応する上記拡張可能配列の各添字のビットパターンを所定の属性順に並べた座標パターン、の 2 項組表現をキー値として登録した第 2 データと、

属性毎の配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、

経歴値に対応する配列拡張した属性の次元、および、当該経歴値に対応する拡張部分配列の任意の要素について、属性毎に拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、

拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、を格納しており、

新たな属性値を持つタプルを挿入するとき、

配列拡張が必要である場合には、上記属性値が属する属性の部分配列を追加するとともに、経歴値をインクリメントして当該属性の上記属性毎経歴値テーブルに登録する基本データ拡張処理を行い、

配列拡張が不必要である場合、および、配列拡張が必要である場合に上記基本データ拡張処理を行った後、上記属性値を上記属性値テーブルおよび上記第 1 データに登録するとともに、当該タプルの各属性値に対応する上記座標パターンを生成して、経歴値および座標パターンの 2 項組表現をキー値として上記第 2 データへ挿入することを特徴とするデータベースの管理方法。

【請求項 7】

関係テーブルを用いたデータベースのデータ構造であって、

関係テーブルの属性ごとに設けられ、属性の各属性値から拡張可能配列の添字に変換するための第 1 データと、

関係テーブルのタプルに対応する拡張可能配列の要素が属する部分配列の経歴値、および、当該タプルの各属性値に対応する上記拡張可能配列の各添字のビットパターンを所定の属性順に並べた座標パターン、の 2 項組表現をキー値として登録した第 2 データと、

属性毎の配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、

経歴値に対応する配列拡張した属性の次元、および、当該経歴値に対応する拡張部分配列の任意の要素について、属性毎に拡張可能配列における対応次元の添字の表現に要する

10

20

30

40

50

ビット数を要素とする境界ベクトルを登録した経歴値テーブルと、

拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、よりなることを特徴とするデータベースのデータ構造。

【請求項 8】

上記拡張可能配列がチャンク化拡張可能配列であり、かつ、上記区画がチャンク化拡張可能配列のチャンクであって、

上記データベース記憶部に、

関係テーブルの属性ごとに設けられ、属性の各属性値からチャンク化拡張可能配列の当該チャンクの添字に変換するための第 1 データと、

10

上記区画位置情報である、関係テーブルのタプルに対応するチャンク化拡張可能配列の要素が属するチャンクのチャンク番号、および、上記座標情報である、当該タプルの各属性値に対応する上記チャンク化拡張可能配列の当該チャンクにおける各添字のビットパターンを所定の属性順に並べたチャンク内座標パターン、の 2 項組表現をキー値として登録した、上記要素位置データである第 2 データと、

属性毎のチャンク配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、

経歴値に対応するチャンク配列拡張した属性の次元、および、当該経歴値に対応する拡張チャンク部分配列中の任意のチャンクについて、属性毎にチャンク拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、

20

チャンク化拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、

を格納したことを特徴とする請求項 1 に記載のデータベース装置。

【請求項 9】

ある属性値を持つタプルを検索するとき、

上記属性値が属する属性の上記第 1 データを用いて、上記属性値をチャンク化拡張可能配列の当該次元の添字に変換し、当該次元の経歴値テーブルより経歴値を求めた後、

上記第 2 データを参照して、2 項組表現されたキー値中のチャンク内座標パターンに含まれる、上記属性の属性値に対応するビットパターンが、上記添字のビットパターンと一致するタプルを抽出するタプル検索手段を具備することを特徴とする請求項 8 に記載のデータベース装置。

30

【請求項 10】

新たな属性値を持つタプルを挿入するとき、

チャンク配列拡張が必要である場合には、上記属性値が属する属性のチャンク部分配列を追加するとともに、経歴値をインクリメントして当該属性の上記属性毎経歴値テーブルに登録する基本データ拡張処理を行い、

チャンク配列拡張が不必要である場合、および、チャンク配列拡張が必要である場合に上記基本データ拡張処理を行った後、上記属性値を上記属性値テーブルおよび上記第 1 データに登録するとともに、当該タプルの各属性値に対応する上記チャンク内座標パターンを生成して、チャンク番号およびチャンク座標パターンの 2 項組表現をキー値として上記第 2 データへ挿入するタプル挿入手段を具備することを特徴とする請求項 8 に記載のデータベース装置。

40

【請求項 11】

関係テーブルを用いたデータベース装置におけるデータベースの管理方法であって、

上記データベース装置は、

データベース記憶部に、

関係テーブルの属性ごとに設けられ、属性の各属性値からチャンク化拡張可能配列の当該チャンクの添字に変換するための第 1 データと、

関係テーブルのタプルに対応するチャンク化拡張可能配列の要素が属するチャンク

50

のチャンク番号、および、当該タプルの各属性値に対応する上記チャンク化拡張可能配列の当該チャンクにおける各添字のビットパターンを所定の属性順に並べたチャンク内座標パターン、の2項組表現をキー値として登録した第2データと、

属性毎のチャンク配列拡張の時間的順序を登録した属性毎経歴値テーブルと、

チャンク番号に対応するチャンク配列拡張した属性、および、当該経歴値に対応する拡張チャンク部分配列中の任意のチャンクについて、属性毎にチャンク拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、

チャンク化拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、を格納しており、

10

ある属性値を持つタプルを検索するとき、

上記属性値が属する属性の上記第1データを用いて、上記属性値をチャンク化拡張可能配列の添字に変換し、

上記第2データを参照して、2項組表現されたキー値中のチャンク内座標パターンに含まれる、上記属性の属性値に対応するビットパターンが、上記添字のビットパターンと一致するタプルを抽出することを特徴とするデータベースの管理方法。

【請求項12】

関係テーブルを用いたデータベース装置におけるデータベースの管理方法であって、

上記データベース装置は、

データベース記憶部に、

20

関係テーブルの属性ごとに設けられ、属性の各属性値からチャンク化拡張可能配列の当該チャンクの添字に変換するための第1データと、

関係テーブルのタプルに対応するチャンク化拡張可能配列の要素が属するチャンクのチャンク番号、および、当該タプルの各属性値に対応する上記チャンク化拡張可能配列の当該チャンクにおける各添字のビットパターンを所定の属性順に並べたチャンク内座標パターン、の2項組表現をキー値として登録した第2データと、

属性毎のチャンク配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、

チャンク番号に対応するチャンク配列拡張した属性の次元、および、当該経歴値に対応する拡張チャンク部分配列中の任意のチャンクについて、属性毎にチャンク拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、

30

チャンク化拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、を格納しており、

新たな属性値を持つタプルを挿入するとき、

チャンク配列拡張が必要である場合には、上記属性値が属する属性のチャンク部分配列を追加するとともに、経歴値をインクリメントして当該属性の上記属性毎経歴値テーブルに登録する基本データ拡張処理を行い、

チャンク配列拡張が不必要である場合、および、チャンク配列拡張が必要である場合に上記基本データ拡張処理を行った後、上記属性値を上記属性値テーブルおよび上記第1データに登録するとともに、当該タプルの各属性値に対応する上記チャンク内座標パターンを生成して、チャンク番号およびチャンク座標パターンの2項組表現をキー値として上記第2データへ挿入することを特徴とするデータベースの管理方法。

40

【請求項13】

関係テーブルを用いたデータベースのデータ構造であって、

関係テーブルの属性ごとに設けられ、属性の各属性値からチャンク化拡張可能配列の当該チャンクの添字に変換するための第1データと、

関係テーブルのタプルに対応するチャンク化拡張可能配列の要素が属するチャンクのチャンク番号、および、当該タプルの各属性値に対応する上記チャンク化拡張可能配列の当該チャンクにおける各添字のビットパターンを所定の属性順に並べたチャンク内座標

50

パターン、の 2 項組表現をキー値として登録した第 2 データと、

属性毎のチャンク配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、

経歴値に対応するチャンク配列拡張した属性の次元、および、当該経歴値に対応する拡張チャンク部分配列中の任意のチャンクについて、属性毎にチャンク拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、

チャンク化拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、よりなることを特徴とするデータベースのデータ構造。

10

【請求項 14】

請求項 3、4、9、10 のいずれか 1 項に記載のデータベース装置を動作させるデータベース管理プログラムであって、コンピュータを上記の各手段として機能させるためのデータベース管理プログラム。

【請求項 15】

請求項 14 に記載のデータベース管理プログラムを記録したコンピュータ読み取り可能な記録媒体。

【発明の詳細な説明】

【技術分野】

【0001】

20

本発明は、関係データベースを用いるデータベースに関するものであり、詳細には、データベース装置、データベースの管理方法、データベースのデータ構造、データベースの管理プログラムおよびそれを記録したコンピュータ読み取り可能な記録媒体に関するものである。

【背景技術】

【0002】

多次元データとは複数種類のデータの組（以下、タプルという）の集合であり、代表的な多次元データとして関係データベースにおける関係テーブルがあげられる。近年、情報処理分野の多様化と拡大に応じて、その重要性は急激に増大している。多次元データを計算機内部において記憶し、処理するための効率よい方式を提供することは、多次元情報処理を必要とする諸分野の発展に基本レベルで貢献し得ると考えられる。ここでは、計算機内部での記憶サイズおよび検索効率において、きわめて優れた、多次元データの処理方式を提案する。多次元データの例として、以降では、広く知られている関係データベーステーブルを取り上げて、説明する。

30

【0003】

関係データベースは、図 8 のような関係テーブルの集合であり、関係テーブルはその中のタプルの集合である。その検索は属性名や検索の条件を指定することにより、行われる。

【0004】

このような、関係テーブルは通常 2 次記憶上に置かれ、各タプルは入力された順に逐次、配置される。したがって、次のような欠点がある。

40

【0005】

(1) 例えば、年齢が 23 のタプルを検索する場合、テーブル中のすべてのタプルをメモリ上にロードして年齢属性をチェックする必要がある。したがって、検索時間が大きくなる。

【0006】

(2) 例えば、出身地が福井のタプルは数多く現れ、文字列「福井」を重複して多く格納する必要がある。したがって、ディスク使用量が大きくなる。

【0007】

このような欠点を回避するための方法として、多次元配列を使用することが考えられる

50

。ここで、配列の各次元はテーブルの属性に対応し、配列の要素は対応するタプルを表す。

【 0 0 0 8 】

図 9 は、配列による関係テーブルの表現の一例を示す説明図である。

【 0 0 0 9 】

この時、年齢が 2 3 のタプル集合は「年齢」次元の値が 2 3 の平面上に空でない配列要素として、存在する。配列要素[* , * , 23]の番地はアドレス関数により高速に求めることができる。したがって、(1)の欠点は回避される。また、各次元の値は値順にソートされて並べられ、1 度しか現れないので、(2)の欠点も回避される。

【 0 0 1 0 】

なお、拡張可能配列に関する先行技術文献としては、以下のものがある。

【先行技術文献】

【非特許文献】

【 0 0 1 1 】

【非特許文献 1】A.L.Rosenberg, "Allocating Storage for Extendible Arrays", Journal of ACM, Vol. 21, pp.652-670, 1974.

【非特許文献 2】E.J.Otoo, T.H.Merrett, "A Storage Scheme for Extendible Arrays, Computing", Vol.31, pp.1-9, 1983.

【非特許文献 3】都司達夫, 水野剛, 宝珍輝尚, 樋口健, "拡張可能配列の遅延割付け方式", 電子情報通信学会論文誌 D-1, Vol.J86-D-1, No.5, pp.351-356, 2003.

【非特許文献 4】T.Tsuji, A.Hara, K.Higuchi, "An extendible multidimensional array system for MOLAP" Proc. of ACM Applied Computing, pp.503-510, 2006.

【非特許文献 5】Masakazu Kumakiri, Li Bei, T.Tsuji, K.Higuchi, "Flexibly resizable multidimensional arrays", Proc. of the 22nd International Conference on Data Engineering Workshops, 2006.4.

【非特許文献 6】熊切正和, Li Bei, 都司達夫, 樋口健, "挿入拡張・中抜き縮小可能な多次元配列", 日本データベース学会 Letters, Vol.5, No.1, pp.61-64, 2006.

【非特許文献 7】Bei Li, T.Tsuji, K.Higuchi, "Sharing Flexibly Resizable Multidimensional Arrays in Client/Server Environment", Proc. of IEEE Int'l Workshop of Databases for Next Generation Researchers, 2007.

【非特許文献 8】E.J.Otoo, D.Rotem, "Efficient Storage Allocation of Large-Scale Extendible Multi-dimensional Scientific Datasets", Proc. of SSDBM, 179-183, 2006.

【発明の概要】

【発明が解決しようとする課題】

【 0 0 1 2 】

ここで、多次元配列によるテーブルの表現は高速にタプルを検索することができるが、次のような欠点 (a) (b) がある。

【 0 0 1 3 】

(a) 各次元のサイズは固定である。固定であるから、アドレス関数が作成できる。したがって、新たな属性値を持つタプルの追加は不可能である。

【 0 0 1 4 】

(b) 各次元の値の組み合わせがすべて存在するような密なテーブルは稀であるので、配列内の有効要素は少ない (疎配列) 。有効要素の割合は通常数 % 以下、場合によっては 0 . 1 % 以下であることが多い。このような疎配列に対しても、アドレス関数によって配列要素にアクセスできるためには空の要素 (存在しないタプル) についても記憶領域を確保する必要があり、ディスクスペースの多大な無駄になる。

【 0 0 1 5 】

これらの欠点を回避するために、新しい仕組みに基づくテーブル実装方式を提案する。この実装方式を「経歴・パターン法」(History-Pattern implementation of Multidimen

10

20

30

40

50

sional Data: H P M D) と呼ぶ。H P M D は、以下に説明する拡張可能配列の考え方に基
づいている。

【 0 0 1 6 】

拡張可能配列 (extendible array) は実行時に動的に任意の次元方向にそのサイズを拡張できる配列である。拡張可能配列では拡張部分のみが動的に割付けられ、拡張前の配列要素のデータは再配置することなくそのまま利用される。配列サイズがあらかじめ予測不能な場合や、必要サイズが実行環境の変化に応じて動的に変化し得るような種々のアプリケーション分野において使用することができる。拡張可能配列のモデルとして E. J. Ottoo 等により提案されたインデックス配列モデル (非特許文献 2) はインデックス配列用の記憶域を付加することにより高速に配列要素を参照することができ、例えばハッシュを使った他の方式 (非特許文献 1) より優れていることが示されている。本発明における着想のベースはこのインデックス配列モデルの考え方に基づいており、ここでは、このモデルについてその概略を述べる。

10

【 0 0 1 7 】

ある次元方向の配列拡張はその次元を除く $n - 1$ 次元の配列断面に相当するサイズの連続するメモリ領域 (部分配列あるいはサブ配列という) を確保し、A に追加することによって行われる。(非特許文献 2) では、拡張時に確保される部分配列はメモリ領域の 0 番地から拡張の順に、順次、連続領域に割り付けられることを前提としている。通常行われるメモリの動的割付けにおいては、必ずしもメモリの連続領域を割り付けるとは限らない。このことをはじめ、現実の使用に即したいいくつかの改良を施したモデルが (非特許文献 3) で提案されている。以下では、(非特許文献 3) のモデルについて述べる。

20

【 0 0 1 8 】

n 次元拡張可能配列 A は 1 つの経歴値カウンタと次元毎に 3 種類の補助テーブルを有している。これらの補助テーブルは経歴値テーブル、アドレステーブル、および係数テーブルと呼ばれる。経歴値テーブルは配列拡張の時間的順序を表す 1 次元配列であり、配列拡張が行われるたびに、固定配列の $n - 1$ 次元の部分配列が動的に割り付けられ、アドレス番地テーブルにその先頭番地が記録される。また、現在の経歴値カウンタが 1 インクリメントされ、その値が経歴値テーブルに順次記録される。拡張可能配列および部分配列の各次元の添え字はいずれも 0 から始まり、次元は 1 から数えるものとし、配列の 1 要素のサイズは 1 とする。

30

【 0 0 1 9 】

例えば、各次元のサイズが $[s_1, s_2, s_3, s_4]$ の通常の固定サイズの 4 次元配列要素をメモリ上に次元 1 ~ 4 の順に優先して割り付ける場合、要素 $\langle i_1, i_2, i_3, i_4 \rangle$ のアドレスはよく知られているように、

$$s_2 s_3 s_4 i_1 + s_3 s_4 i_2 + s_4 i_3 + i_4 \quad (1)$$

なる i_1, i_2, i_3, i_4 に関する 1 次関数を計算して得られる。

【 0 0 2 0 】

これに対して、例えば現在のサイズが $[s_1, s_2, s_3, s_4]$ の 4 次元拡張可能配列の場合には、次元 2 の方向に 1 つ拡張する時、サイズ $[s_2, s_3, s_4]$ の 3 次元部分配列 S が動的に確保される。アドレステーブルは各部分配列の先頭アドレスを保持する 1 次元配列である。要素 $\langle i_1, i_2, i_3, i_4 \rangle$ が格納されているアドレスは S の先頭番地に (1) 式で計算されるオフセットを加えればよい。

40

【 0 0 2 1 】

A が 3 次元以上の拡張可能配列の場合には、部分配列内の要素のオフセットを計算する 1 次関数の $n - 2$ 個の係数からなる係数ベクトルを部分配列毎に記録する係数テーブルを各次元について必要とする。例えば、上記部分配列 S の要素 $\langle i_1, i_2, i_3 \rangle$ のオフセットは (1) 式と同様、1 次関数 $s_3 s_4 i_1 + s_4 i_2 + i_3$ となる。この時、 $(s_3 s_4, s_4)$ が S の係数ベクトルである。係数ベクトルの値は拡張時の A の各次元のサイズに依存しているので、拡張時に係数ベクトルを計算し、それを拡張次元の係数テーブルのスロットに書き込む。

【 0 0 2 2 】

50

配列要素へのアクセスは次のように行われる。図10は、インデックス配列モデルの一例を示す説明図である。図10において、次元1方向および次元2方向の経歴値テーブルをそれぞれ H_1, H_2 とし、またアドレステーブルをそれぞれ A_1, A_2 とする。例えば配列要素 $\langle 3, 4 \rangle$ のアドレス計算は次のように行われる。 $H_1[3] < H_2[4]$ であるから、要素 $\langle 3, 4 \rangle$ を含む部分配列 S は経歴値 $H_2[4]=7$ の時に割付けられ、その先頭アドレスは $A_2[4]=60$ である。また、要素 $[3, 4]$ は S では要素 $\langle 3 \rangle$ であるので、求めるアドレスは 63 となる。

【0023】

なお、(非特許文献4)～(非特許文献8)はいずれも、拡張可能配列の改良や機能強化について提案している。

【0024】

本発明は、上記の問題点に鑑みてなされたものであり、その目的は、新たな属性値を持つタプルの追加とタプルの高速検索が可能であり、かつ、ディスクスペースを効率良く使用することができる、テーブル実装方式を用いたデータベース装置、データベースの管理方法、データベースのデータ構造、データベースの管理プログラムおよびそれを記録したコンピュータ読み取り可能な記録媒体を実現することにある。

【課題を解決するための手段】

【0025】

上記の課題を解決するために、本発明に係るデータベース装置は、関係テーブルを用いたデータベース装置であって、関係テーブルの各タプルに対応する拡張可能配列の要素の位置を示す位置情報をキー値として登録した要素位置データを格納したデータベース記憶部を具備するとともに、上記位置情報が、要素が属する拡張可能配列の区画の位置を示す区画位置情報と、区画内における要素の位置を示す、当該タプルの各属性の属性値に一意に対応した値を所定の属性順に並べた座標情報と、を含む情報であることを特徴としている。

【0026】

ここで、本発明では、拡張可能配列の「区画」を様々に選択可能である。そして、上記要素位置データとして、区画に応じた要素位置データを登録する。

【0027】

例えば、(1)区画を拡張可能配列の部分配列とすれば、上記要素位置データとして、上記区画位置情報を、関係テーブルのタプルに対応する拡張可能配列の要素が属する部分配列の経歴値とし、上記座標情報を、当該タプルの各属性値に対応する拡張可能配列の各添字のビットパターンを所定の属性順に並べた座標パターンとした、2項組表現をキー値として登録した2項組データを利用できる。(2)また、区画をチャンク化拡張可能配列のチャンクとすれば、上記要素位置データとして、上記区画位置情報を、関係テーブルのタプルに対応するチャンク化拡張可能配列の要素が属するチャンクのチャンク番号とし、上記座標情報を、当該タプルの各属性値に対応するチャンク化拡張可能配列の当該チャンクにおける各添字のビットパターンを所定の属性順に並べたチャンク内座標パターンとした、2項組表現をキー値として登録した2項組データを利用できる。

【0028】

具体的には、要素が属する拡張可能配列の区画の位置を示す区画位置情報と、当該タプルの各属性の属性値に一意に対応した値を所定の属性順に並べた座標情報との2項組表現は、(1)の場合、 $\langle \text{経歴値}, \text{座標パターン} \rangle$ であり、(2)の場合、 $\langle \text{チャンク番号}, \text{チャンク内座標パターン} \rangle$ となる。なお、チャンク番号は、要素の添字 $\langle i_1, i_2, \dots, i_n \rangle$ より、チャンク拡張可能配列の要素(チャンク)の位置決定機構により決定される。

【0029】

よって、上記データベース装置では、2項組データ(要素位置データ)を参照することにより、区画位置情報と座標情報との2項組表現に基づいて、拡張可能配列の要素の位置を特定することが可能となる。

【0030】

10

20

30

40

50

なお、拡張可能配列の全要素集合Eを互いに共通な要素を持たない部分集合の集合に類別(partition)したとき、その任意の部分集合Sを拡張可能配列の「区画」と定義する。そして、部分集合Sに対応する記憶表現の先頭要素の要素集合E内での位置を特定するために必要な情報を「区画位置情報」と定義する。これにより、「区画位置情報」および「座標情報」の両者により、拡張可能配列の任意の要素について、その位置が一意的に決定される。このような定義の下で、(1)では区画である部分集合Sを部分配列としており、(2)では区画である部分集合Sをチャンクとしている。さらに、(1)(2)では、「(これら2つの)2項組表現をキー値として登録した2項組データ」と記述しているが、必ずしも2項組として、キー値の記憶表現において位置的に接続している必要はない。(ただし、高速検索のためには、位置的に接続していることが望ましい。)すなわち、要素位置データには、キー値の記憶表現にこれら2つの情報(「区画位置情報」および「座標情報」)が含まれていればよい。そして、本発明に係るデータベース装置には、これらの2つの情報を使って要素に迅速にアクセスするための手段を具備していればよい。

10

20

30

40

50

【0031】

また、「所定の属性順」とは、関係テーブル毎に固定されていれば、任意に設定できる。たとえば、属性に付与された識別番号順であってもよい。

【0032】

(1)2項組データに、関係テーブルのタプルに対応する拡張可能配列の要素の経歴値と、当該タプルの各属性値に対応する拡張可能配列の各添字のビットパターンを所定の属性順に並べた座標パターンとの2項組表現をキー値として登録する場合のデータベースの構成と、タプルを検索、挿入する機能は以下のとおりである。

【0033】

本発明に係るデータベース装置は、上記区画が拡張可能配列の部分配列であって、上記データベース記憶部に、関係テーブルの属性ごとに設けられ、属性の各属性値から拡張可能配列の添字に変換するための第1データと、上記区画位置情報である、関係テーブルのタプルに対応する拡張可能配列の要素が属する部分配列の経歴値、および、上記座標情報である、当該タプルの各属性値に対応する上記拡張可能配列の各添字のビットパターンを所定の属性順に並べた座標パターン、の2項組表現をキー値として登録した、上記要素位置データである第2データと、属性毎の配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、経歴値に対応する配列拡張した属性の次元、および、当該経歴値に対応する拡張部分配列の任意の要素について、属性毎に拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、を格納したことを特徴としている。

【0034】

また、本発明に係るデータベース装置は、ある属性値を持つタプルを検索するとき、上記属性値が属する属性の上記第1データを用いて、上記属性値を拡張可能配列の当該次元の添字に変換し、当該次元の経歴値テーブルより経歴値を求めた後、上記第2データを参照して、2項組表現されたキー値中の座標パターンに含まれる、上記属性の属性値に対応するビットパターンが、上記添字のビットパターンと一致するタプルを抽出するタプル検索手段を具備することを特徴としている。

【0035】

また、本発明に係るデータベース装置は、新たな属性値を持つタプルを挿入するとき、配列拡張が必要である場合には、上記属性値が属する属性の部分配列を追加するとともに、経歴値をインクリメントして当該属性の上記属性毎経歴値テーブルに登録する基本データ拡張処理を行い、配列拡張が不必要である場合、および、配列拡張が必要である場合に上記基本データ拡張処理を行った後、上記属性値を上記属性値テーブルおよび上記第1データに登録するとともに、当該タプルの各属性値に対応する上記座標パターンを生成して、経歴値および座標パターンの2項組表現をキー値として上記第2データへ挿入するタプル挿入手段を具備することを特徴としている。

【0036】

そして、上記の構成により、本発明のデータベースは、関係テーブルを用いたデータベースのデータ構造であって、関係テーブルの属性ごとに設けられ、属性の各属性値から拡張可能配列の添字に変換するための第1データと、関係テーブルのタプルに対応する拡張可能配列の要素が属する部分配列の経歴値、および、当該タプルの各属性値に対応する上記拡張可能配列の各添字のビットパターンを所定の属性順に並べた座標パターン、の2項組表現をキー値として登録した第2データと、属性毎の配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、経歴値に対応する配列拡張した属性の次元、および、当該経歴値に対応する拡張部分配列の任意の要素について、属性毎に拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、よりなるデータ構造を有している。

10

【0037】

上記の構成によれば、関係データベースに新たな属性値を持つタプルの追加が可能であり、ディスクスペースを効率良く使用することが可能となる。

【0038】

(2) 2項組データに、関係テーブルのタプルに対応するチャンク化拡張可能配列の要素が属するチャンクのチャンク番号と、当該タプルの各属性値に対応するチャンク化拡張可能配列の当該チャンクにおける各添字のビットパターンを所定の属性順に並べたチャンク内座標パターンとの2項組表現をキー値として登録する場合のデータベースの構成と、データを検索、挿入する機能は以下のとおりである。

20

【0039】

本発明に係るデータベース装置は、上記拡張可能配列がチャンク化拡張可能配列であり、かつ、上記区画がチャンク化拡張可能配列のチャンクであって、上記データベース記憶部に、関係テーブルの属性ごとに設けられ、属性の各属性値からチャンクを要素とするチャンク化拡張可能配列の添字に変換するための第1データと、上記区画位置情報である、関係テーブルのタプルに対応するチャンク化拡張可能配列の要素が属するチャンクのチャンク番号、および、上記座標情報である、当該タプルの各属性値に対応する上記チャンク化拡張可能配列の当該チャンクにおける各添字のビットパターンを所定の属性順に並べたチャンク内座標パターン、の2項組表現をキー値として登録した、上記要素位置データである第2データと、属性毎のチャンク配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、経歴値に対応するチャンク配列拡張した属性の次元、および、当該経歴値に対応する拡張チャンク部分配列中の任意のチャンクについて、属性毎にチャンク拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、チャンク化拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、を格納したことを特徴としている。

30

【0040】

また、本発明に係るデータベース装置は、ある属性値を持つタプルを検索するとき、上記属性値が属する属性の上記第1データを用いて、上記属性値をチャンク化拡張可能配列の当該次元の添字に変換し、当該次元の経歴値テーブルより経歴値を求めた後、上記第2データを参照して、2項組表現されたキー値中のチャンク内座標パターンに含まれる、上記属性の属性値に対応するビットパターンが、上記添字のビットパターンと一致するタプルを抽出するタプル検索手段を具備することを特徴としている。

40

【0041】

また、本発明に係るデータベース装置は、新たな属性値を持つタプルを挿入するとき、チャンク配列拡張が必要である場合には、上記属性値が属する属性のチャンク部分配列を追加するとともに、経歴値をインクリメントして当該属性の上記属性毎経歴値テーブルに登録する基本データ拡張処理を行い、チャンク配列拡張が不必要である場合、および、

50

チャンク配列拡張が必要である場合に上記基本データ拡張処理を行った後、上記属性値を上記属性値テーブルおよび上記第1データに登録するとともに、当該タプルの各属性値に対応する上記チャンク内座標パターンを生成して、チャンク番号およびチャンク内座標パターンの2項組表現をキー値として上記第2データへ挿入するタプル挿入手段を具備することを特徴としている。

【0042】

そして、上記の構成により、本発明のデータベースは、関係テーブルを用いたデータベースのデータ構造であって、関係テーブルの属性ごとに設けられ、属性の各属性値からチャンク化拡張可能配列の当該チャンクの添字に変換するための第1データと、関係テーブルのタプルに対応するチャンク化拡張可能配列の要素が属するチャンクのチャンク番号、および、当該タプルの各属性値に対応する上記チャンク化拡張可能配列の当該チャンクにおける各添字のビットパターンを所定の属性順に並べたチャンク内座標パターン、の2項組表現をキー値として登録した、上記要素位置データである第2データと、属性毎のチャンク配列拡張の時間的順序を表す経歴値を登録した属性毎経歴値テーブルと、経歴値に対応するチャンク配列拡張した属性の次元、および、当該経歴値に対応する拡張チャンク部分配列中の任意のチャンクについて、属性毎にチャンク拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを登録した経歴値テーブルと、チャンク化拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を登録した属性値テーブルと、よりなるデータ構造を有している。

【0043】

上記の構成によれば、関係データベースに新たな属性値を持つタプルの追加が可能であり、ディスクスペースを効率良く使用することが可能となる。しかも、優れた検索速度が実現できる。

【0044】

なお、上記データベース装置は、コンピュータによって実現してもよく、この場合には、コンピュータを上記各手段として動作させることにより上記データベース装置をコンピュータにて実現させるデータベース管理プログラム、およびそれを記録したコンピュータ読み取り可能な記録媒体も、本発明の範疇に入る。

【発明の効果】

【0045】

以上のように、本発明に係るデータベース装置は、関係テーブルを用いたデータベース装置であって、関係テーブルの各タプルに対応する拡張可能配列の要素の位置を示す位置情報をキー値として登録した要素位置データを格納したデータベース記憶部を具備するとともに、上記位置情報が、要素が属する拡張可能配列の区画の位置を示す区画位置情報と、区画内における要素の位置を示す、当該タプルの各属性の属性値に一意に対応した値を所定の属性順に並べた座標情報と、を含む情報である構成である。

【0046】

それゆえ、関係データベースに新たな属性値を持つタプルの追加が可能であり、ディスクスペースを効率良く使用することが可能となるという効果を奏する。しかも、優れた検索速度を実現することが可能となるという効果を奏する。

【図面の簡単な説明】

【0047】

【図1】本発明の一実施の形態に係るデータベース装置の構成の概略を示す機能ブロック図である。

【図2】HPMDの基本データ構造を示す説明図である。

【図3】CVTのデータ部のデータ構造を示す説明図である。

【図4】タプルの挿入プログラムの本体int insert_rec()の流れを示すフローチャートである。

【図5】HPMDにおける属性値の検索範囲を示す説明図であり、(a)は経歴値5(次元1)の添字に対応する属性値の論理検索範囲を示し、(b)は経歴値3(次元2)の添字

10

20

30

40

50

に対応する属性値の論理検索範囲を示す。

【図6】RDTを検索して、指定された次元*i*の属性値colvalを持つキーを出力するルーチンの流れを示すフローチャートである。

【図7】C-HPMDにおける論理拡張可能配列とチャンク番号との関係を示す説明図である。

【図8】従来の技術に係る関係テーブルの一例を示す説明図である。

【図9】従来の技術に係る配列による関係テーブルの表現の一例を示す説明図である。

【図10】従来の技術に係るインデックス配列モデルの一例を示す説明図である。

【発明を実施するための形態】

【0048】

10

本発明の一実施の形態について図1から図7に基づいて説明すれば、以下のとおりである。まず、本発明に係る関係テーブルの記憶、操作方式とその実現ソフトウェアについて説明する。

【0049】

1. HPMDの基本データ構造とその処理

HPMDでは〔発明が解決しようとする課題〕で述べた、多次元データの動的な追加・拡張に対して効率よく対処できる拡張可能配列の仕組みを、その基本部分において活かしている。n個の属性からなる多次元データTはn次元HPMDにより実装される。n次元HPMDは次のデータ構造からなる。

【0050】

20

(1) n個のCVTi(1 ≤ i ≤ n) (attribute-subscript ConVersion Tree) とRDT (Real Data Tree) のn+1個のB+木。CVTiは次元*i*の属性値をキーとして、対応する拡張可能配列の添字を記録している。また、RDTのキーは以降で述べるエンコード方式による多次元データタプルのエンコード値が格納される。なお、CVTiに必要な機能は、キー指定によるランダムアクセス機能とキーの値順による順次的アクセス機能であり、RDTに必要な機能は、キー指定によるランダムアクセス機能である。よって、CVTiおよびRDTは、B+木に限らず、上記の機能をそれぞれ実現するキー編成のデータ構造であればよい。もちろん、B+木であれば、上記の機能をそれぞれ効率よく実現でき、HPMDの性能を高めることができる。

【0051】

30

(2) 各次元毎に経歴値を格納するn個の経歴値テーブルHi(1 ≤ i ≤ n)、および、経歴値を添字として、経歴値に対応する拡張次元と以降で述べる境界ベクトルを格納した経歴値テーブルH。以後、単に経歴値テーブルと言った場合、後者を指し、特に前者を指す場合には次元*i*の経歴値テーブルということとする。

【0052】

(3) 各次元の添字ごとにその添字に対応する属性値およびその属性値を持つすべてのタプル数を記録する属性値テーブルCi(1 ≤ i ≤ n)。

【0053】

図2は、HPMDの基本データ構造を示す説明図である。図2には表中の13個の2次元のタプル集合を上から順に挿入した場合の上記(1)(2)(3)のデータ構造を示す。n次元拡張可能配列Aの現在の各次元サイズを<s₁, s₂, ..., s_n>として経歴値カウンタの値をhとする。s_k-1(1 ≤ k ≤ n)の表現に要するビット数をb(s_k)として、次元kの包含サイズと呼ぶ。また、<b(s₁), b(s₂), ..., b(s_n)>を経歴値hにおけるAの境界ベクトルと呼ぶ。

40

【0054】

経歴値カウンタの値がhの拡張可能配列Aのある次元kのサイズs_kが1つ拡張して、s_kのビットパターンサイズが次元kの包含サイズを超えたときにAは次元kの方向に拡張される。拡張分の部分配列の各次元のサイズはAと同一であり、したがって、拡張後の配列の各次元サイズは<s₁, ..., s_{k-1}, 2*s_k, s_{k+1}, ..., s_n>となる(図2)。現在の経歴値カウンタの値が1インクリメントされ、その値h+1が次元kの経歴値テーブルH_kに

50

格納される。また、この新たな経歴値 $h + 1$ における A の境界ベクトルは $V = \langle b(s_1), \dots, b(s_{k-1}), b(s_k)+1, b(s_{k+1}), \dots, b(s_n) \rangle$ となり、経歴値 $h + 1$ の下に、 V が拡張次元 k とともに経歴値テーブル H に記録される。

【0055】

配列の要素位置を指定する n 次元座標 $i = \langle i_1, i_2, \dots, i_n \rangle$ は以下のように経歴値 h と座標パターン p の対にエンコードされる。境界ベクトル V はこのエンコードに使用されるとともにエンコードされた対を元の n 次元座標にデコードするのにも使用される。

【0056】

経歴値 h は i の各次元座標のビットパターンサイズ $\langle b(i_1), b(i_2), \dots, b(i_n) \rangle$ について、経歴値 $H_k(b(i_k))$ ($1 \leq k \leq n$) の最大値となる。また、座標パターン p は各次元の添え字のビットパターンを 1 long 長の上位から順に配置して得られる。このエンコード結果は RDT に格納される。逆に、エンコード結果対 (経歴値 h , 座標パターン p) からの元の n 次元座標 $i = \langle i_1, i_2, \dots, i_n \rangle$ へのデコードは、まず、経歴値テーブル H について、 $H[h]$ より要素 i が属する部分配列の次元 k を求める。 p 中の各次元の座標値 i_k ($1 \leq k \leq n$) を $H[h]$ の境界ベクトルにより、分離する。

【0057】

(タプルのエンコード、デコードの例)

たとえば、タプル (s, e) の場合、 $CVT_1(s)$ 、 $CVT_2(e)$ を検索してタプルの座標 $(5, 4)$ を引き当てる。 $5 = 101_2$ 、 $4 = 100_2$ であり、 $b(5) = 3$ 、 $b(4) = 3$ であるから、 $H_1[3] = 6$ と $H_2[3] = 4$ を比較して、 $H_1[3] > H_2[3]$ 。したがって、 $(5, 4)$ は経歴値 6 の部分配列に含まれる。 $H[6]$ の境界ベクトルは $\langle 3, 3 \rangle$ であり、 $(5, 4)$ のエンコードは $(6, 101.100_2) = (6, 44)$ となる。ここで、座標パターン 101.100_2 の '.' は各次元の座標値パターンの境界位置を表す。逆にこのエンコードより、経歴値 6 について、その部分配列の所属次元 1 と境界ベクトル $\langle 3, 3 \rangle$ を引き当てる。これより、 $44 = 101100_2$ の上位 3 ビット分が 1 次元目の添字、下位 3 ビット分が 2 次元目の添字になることがわかる。したがって、 $(6, 101.100_2)$ は $(5, 4)$ にデコードされる。 $C_1(5) = s$ 、 $C_2(4) = e$ より、元のタプル (s, e) となる。

【0058】

配列の拡張に対して柔軟に効率よく対応できる、〔発明が解決しようとする課題〕で述べた拡張可能配列の仕組みは、各次元の包含サイズを固定することなく、固定サイズ (long 長) の座標パターンの各次元のビットパターンサイズの境界を次元拡張の状況に応じて柔軟に設定していることに活かされているといえる。このような、柔軟性を確保した上で、上述のエンコード、デコードの過程では掛け算や割り算を一切使用しないでビットシフトとマスク演算のみのレジスタ命令のみで、エンコードとデコードが可能である。このことはタプルのアクセス速度および検索速度が従来手法に比べて高速であることを意味しており、非常に注目される。

【0059】

上記拡張可能配列の係数ベクトルの記憶コストは非常に大きく、 $O(n^2)$ であった。本方式の境界ベクトルはこの係数ベクトルに対応するがその記憶コストは $O(n)$ となり、大幅に削減できる。また、経歴値テーブルサイズは各次元長を $\langle s_1, s_2, \dots, s_n \rangle$ とすると、拡張可能配列の方式では、 $s_1 + s_2 + \dots + s_n$ であるのに対して、 $\log_2 s_1 + \log_2 s_2 + \dots + \log_2 s_n$ となる。以上より、経歴値テーブルの記憶コストは CVT 、 RDT 、および属性値テーブルなどの $HPMD$ の他の構成要素に比べて、ほとんど無視できる。属性値テーブルのタプル数は検索の最適化プランを立てるのに必要とされるデータである。 CVT と属性値テーブルは多次元データの属性値を扱うために必要なデータ構造であり、拡張可能配列には対応するデータ構造は存在しない。また、 RDT は多次元データソースに実際に登録されているタプルのみをエンコード値を登録しており、〔発明が解決しようとする課題〕の多次元配列の疎配列性の欠点 (b) を解消している、また、多次元配列の欠点 (a) は拡張可能配列の使用により、解消されている。ただし、多次元データソースに登録されていないタプルは除外しており、記憶域を占めないのので以後、 $HPMD$ における拡張可能配列を論理拡張可能配列と呼ぶ。この論理拡張可能配列の各次元サイズを論理サイズとい

10

20

30

40

50

い、各次元の属性値の数（カーディナリティ）で定まる各次元サイズを実サイズという。図2の論理拡張可能配列の論理サイズは [7,7] であり、各次元のカーディナリティはそれぞれ 5,5 であり、実サイズは [5,5] である（図2の論理拡張可能配列中、点線の範囲）。

【0060】

1.1 基本データ構造の実装

ここでは、1ワード64ビットの64ビットマシンを使用するとして、整数値32ビット、長整数値64ビットとする。

【0061】

1.1.1 CVTのデータ部の実装

10

図3は、CVTのデータ部のデータ構造を示す説明図である。CVTのキー部は各属性の属性値であるがデータ部は対応次元の配列添字であり、符号なし整数とする。この添字の最上位ビットの位置（0~31）を求めて、この位置でその次元の経歴値テーブルの経歴値を知る必要がある。最上位ビットの位置を求めるには、1ビット右シフト演算を繰り返しながらカウントすればよいが、これには多くの時間が必要になる。そこで、データ部の上位5ビットにこの位置をあらかじめ計算して、セットしておく。下位27ビットには添字を格納する。これにより、各次元、最大で $2^{27}=134,217,728$ までの属性値の種類（カーディナリティ）が扱える。

【0062】

次元 i のデータ部の値を d とすると経歴値テーブル H_i について、 $H_i[d \gg 27]$ で次元 i の経歴値が求まり、 $d \& 0x07FFFFFF$ で添字の値が求まる。

20

【0063】

1.1.2 境界ベクトルの実装

n 次元境界ベクトルに対して、

【0064】

【数1】

```
struct v_mask {
    unsigned char sz;    /* 次元の包含サイズ */
    unsigned char ppos; /* 次の次元のサイズパターンのビット位置 */
    unsigned int mp;    /* マスクビットパターン */
}
```

30

【0065】

として、構造体 `struct v_mask` の配列 v を境界ベクトルの実装とする。

【0066】

```
struct v_mask [ ];
```

たとえば、 $\langle 8,10,7,20,15,4 \rangle$ の6次元境界ベクトルの時、配列 v は、

【0067】

40

【数 2】

	sz	ppos	mp
0	8,	56,	00000000000000000000000011111111
1	10,	46,	0000000000000000000000001111111111
2	7,	39,	0000000000000000000000000111111111
3	20,	19,	0000000000001111111111111111111111
4	15,	4,	0000000000000000000111111111111111
5	4,	0,	0000000000000000000000000000001111

10

【0068】

となる。

【0069】

タプルエンコード後の座標パターンのサイズを long 長 (64) 以内とするためには、各境界ベクトルの包含サイズのビットパターンを上位次元のものから順に並べてできるビットパターンのサイズ、すなわち各次元の包含サイズの和、 $v[0].sz+v[1].sz+\dots+v[n-1].sz$ は、long 長 (64) 以下でなければならない。64 ビットの時には最大 10 次元の 2^{32} までの各次元のサイズが表現できる。論理拡張可能配列のすべての次元サイズが最大 2^{32} までの拡張を仮定すると各 $v[i].sz$ ($0 \leq i < n-1$) は最大 5 であり、10 次元までの論理拡張可能配列が表現可能である。 2^{32} 以下の最大次元サイズが存在するときには最大次元数は 10 を超えることができる。

20

【0070】

1.1.3 経歴値テーブル H の実装

【0071】

【数 3】

```
struct history_tbl {
    unsigned char dim;    // 次元
    struct v_mask v[];    // 境界ベクトル
} H
```

30

【0072】

なお、次元 i の経歴値テーブル H_i の実装は経歴値の配列であり、`unsigned char H_i []`

【0073】

1.1.4 属性値テーブル C_i の実装

【0074】

【数 4】

```
struct attribute_tbl {
    mkey attribute_value;    // 属性値
    unsigned int tuple_num;    // 属性値を持つタプルの数
}
```

40

【0075】

ここで、`mkey` は H P M D において許されている属性のデータ型である。

【0076】

【数 5】

```
typedef union {
    long          ik;    /* key type : long integer */
    int           ik;    /* key type : integer */
    unsigned int  uk;    /* key type : unsigned integer */
    short        sk;    /* key type : short integer */
    float        fk;    /* key type : float */
    double       dk;    /* key type : double float */
    char         *ck;    /* key type : character string */
} mkey;
```

10

【0077】

1.2 タップルのエンコーディング

タプルを $r = \langle c_1, c_2, \dots, c_n \rangle$ として、 r を \langle 経歴値 h_{\max} , 座標パターン $p \rangle$ の対にエンコードする。各次元のデータ部 $d_i = CVT_i(c_i)$ ($1 \leq i \leq n$) に対して、 h_{\max} は $H_i[d_i \gg 27]$ ($1 \leq i \leq n$) の最大値。 $d[i]$ を $d_i \& \text{MASK_B}$ の配列とすると、 p を生成するマクロは、

【0078】

【数 6】

```
#define make_pattern(n, h, d, p)  \
    { int i; for (i=0, p=0; i<n; i++) p |= d[i] << (H[h].v[i].ppos); }
```

20

【0079】

となる。

【0080】

1.3 タップルのデコーディング

\langle 経歴値 h , 座標パターン $p \rangle$ の対にエンコードされたタプルから、次元 i の添字を求める。

【0081】

【数 7】

```
#define get_subscript(h, p, i)  \
    {(p >> H[h].v[i].ppos) & (H[h].v[i].mp)}
```

30

【0082】

1.4 タップルの挿入

現在の論理拡張可能配列の各次元サイズが配列 `unsigned int c_size[]` に保持されているものとする。

【0083】

タプルの挿入 (HPMD) のプログラムの一例を示せば次の通りである。なお、〔数 8〕の外部データ (グローバルデータ) が、〔数 9〕で使われる。

【0084】

【数 8】

```

#define MASK_B 0x07FFFFFF
#define make_pattern(n, h, d, p) {int i; for (i=0, p=0; i<n; i++) p |= d[i]<<(H[h].v[i].ppos);}

MTYPE *c; // タップル(属性値の配列)
MTYPE **C; // 属性値テーブル
unsigned int *c_size; // 論理拡張可能配列の論理サイズ
unsigned int *h_size; // 論理拡張可能配列の実サイズ
// (各次元の経歴値テーブルの実サイズ)
unsigned int *coordinate; // 座標値
unsigned int h_counter; // 経歴値カウンタ
struct v_mask { // 境界ベクトル
    unsigned char sz;
    unsigned char ppos;
    unsigned long mp;
};
struct rdt_key { // RDT のキー値
    unsigned char h; // 経歴値
    unsigned long p; // 座標パターン
};

int insert_rec(unsigned char n, MTYPE c[]) { // c は属性値の配列
    int i;
    unsigned int htemp;
    unsigned int *coordinate; // 座標値
    struct rdt_key rkey; // RDT キー

    for (i=0; i<n; i++) {
        unsigned int ii;

        if (trfind(CVTi, EQUAL, &c[i], &ii)<0) { // 属性値 c[i] が未登録
            if (c_size[i]+1 > ((1<<h_size[i])-1) // 現在の拡張可能配列の次元 i のサイズを超えた
                { // 登録のためにその次元の経歴値テーブルの拡張が必要
                    h_size[i]++; // 次元 i の論理サイズをインクリメント
                    hmax = Hi[h_size[i]] = ++h_counter; // 経歴値をインクリメントして登録
                    if (construct_bvec(n, &H[h_counter].v)<0) //境界ベクトルの作成
                        {
                            printf("Address space overflows, terminates.¥n");
                            exit(-1);
                        }
                }
            C[i] += c_size[i]; // 次元 i の実サイズをインクリメントした後,
            // 次元 i の属性値テーブルに属性値を登録
            coordinate[i] = c_size[i]; // 次元 i の座標値を登録
            trins(CVTi, c[i], c_size[i] | h_size[i] << 27);
            // 属性値および座標値を含むデータ部を CVTi に格納
        }
        else { // 属性値 c[i] が登録済み
            coordinate[i] = ii & MASK_B; // 次元 i の座標値を取り出す.
        }
        if ((htemp = Hi[ii] >> 27) > hmax) hmax = htemp; //最大経歴値の更新
    }
    rkey.history = hmax; // 経歴値
    make_pattern(n, hmax, coordinate, rkey.p) ¥ // 座標パターンの生成
    trins(RDT, &rkey); // RDT に登録
}

```

【 0 0 8 5 】

【数 9】

```

// 境界ベクトルの作成
int construct_bvec(unsigned char n, struct v_mask *v)
{
    int i;
    unsigned char temp;
    unsigned char psize; // 座標パターンのパターンサイズ
    unsigned char ppos;

    for (i=0, ppos=0; i<n; i++) {
        temp = h_size[i];
        v[i].sz = temp;
        v[i].ppos=63-ppos-temp;
        ppos += temp;
        v[i].mp = (1<<temp)-1; // 次元 i のマスクパターン
    }
    if (psize=ppos+hsize[i-1] > 63) return(-1) // パターンサイズ超過
    else return psize;
}

```

10

【0086】

【数 9】は、【数 1】【数 2】で表される境界ベクトルを作成するルーチンである。このルーチンを使った、タプルの挿入プログラム（【数 8】）の本体 int insert_rec() の流れを、図 4 に示す。

20

【0087】

1.5 検索

n 次元の H P M D の論理拡張可能配列について、タプルの次元 i のある属性値 a v が指定されたとき、CVTi(av) の添字を k とする。次元 i の添字 k に対する基部分配列とは配列要素 (0,0, ..., 0,k,0, ..., 0) を含む部分配列である。このとき、次の顕著な性質がある。

【0088】

[タプルの次元 i の添字 k が指定されたとき、その基部分配列の経歴値を h とする。次元 i の添字が k であるタプルは基部分配列および、h より大きい経歴値の部分配列で次元が i でない部分配列にのみ存在する。]

30

この性質を図 2 の 2 次元 H P M D について、図 5 に図示する。図 5 は、H P M D における属性値の検索範囲を示す説明図であり、(a) は経歴値 5 (次元 1) の添字に対応する属性値の論理検索範囲を示し、(b) は経歴値 3 (次元 2) の添字に対応する属性値の論理検索範囲を示す。なお、点線は論理拡張可能配列の実サイズを表す。

【0089】

上記性質に基づいて、タプルのある属性について与えられた属性値を持つタプルを検索する手続きを示す。タプルの検索 (H P M D) のプログラムの一例を示せば次の通りである。【数 10】は R D T を検索して、指定された次元 i の属性値 colval を持つキー (エンコードされたタプル) を出力するルーチンである。このルーチンの流れを、図 6 に示す。

40

【0090】

【数 1 0】

```

#define get_subscript(h, p, i) {(p>>H[h].v[i].ppos)&(H[h].v[i].mp)}

struct rdt_key {          // RDT のキー値
    unsigned int h;      // 経歴値
    unsigned long p;     // 座標パターン
};

long *search_rec(HPMD *hp, int i, mkey colval) {
    unsigned int d;
    unsigned int sv;     // 添字の値
    unsigned char h0;    // 基部分配列の経歴値
    struct rdt_key rk;   // RDT のシーケンスセットのトラバース用

    trfind(CVTi, EQUAL, &colval, &d); // タップルの次元 i の属性値を CVTi で検索
    sv = d & MASK_B;           // 添字の値
    h0 = Hi[d >> 27];         // 経歴値取り出し

    trfind(RDT, GTEQ, <h0,0>, &rk); // 基部分配列の頭出し
    do {
        if (get_subscript(h0, rk.p, i) == sv) output(rk); // 添字が一致していれば検索結果に追加
        trfind(RDT, NEXT, &rk); // RDT のシーケンスセット上次のキーを取り出す
    } while (rk.h == h0);

    for (h=h0+1; h<=hcounter; h++) { // 基部分配列以外の部分配列の検索
        trfind(RDT, GTEQ, <h,0>, &rk); // 経歴値 h の部分配列の頭出し
        if (H[h].dim != i) { // 次元 i の部分配列はスキップ
            do {
                if ((rk.p>>H[h0].v[i].ppos)&H[h0].v[i].mp==sv) output(rk); // 検索結果に追加
                trfind(RDT, NEXT, &rk);
            } while (rk.h == h);
        }
    }
}

```

【0091】

2. チャンク化 HPMD (C-HPMD)

1.5 節で説明した HPMD における検索では検索範囲が大きく、検索対象の部分配列中のタプルは全検索を行う必要がある。また、図 6 に見るように検索対象の属性の次元や属性値が所属する部分配列の経歴値によって、検索範囲が大きく異なる。ここでは、論理拡張可能配列のランダムアクセス機能を生かして、高速な検索を行うために HPMD のチャンク化を提案する。チャンク化された HPMD を以後、C-HPMD (Chunked HPMD) と呼ぶ。HPMD のチャンク化は高速検索に資すると同時に、タプルのアドレス空間を格段に広げて、大規模な多次元データを収容することができる。

【0092】

チャンクとは 1 辺が 2^k の n 次元超立方体であり、 k をチャンクのランクという。論理拡張可能配列はチャンクを単位として拡張される。HPMD におけるタプルのエンコードは <経歴値, 座標パターン> の対であったのに対して、C-HPMD ではタプルは <チャンク番号, チャンク内座標パターン> にエンコードされる。HPMD の各配列要素をチャンクに対応させると、RDT に付属するデータ構造として CBM と呼ぶビットマップを保持する以外は C-HPMD は HPMD とほぼ同等のデータ構造である。また、HPMD に関する 1 節の説明はタプルのエンコーディング方式と検索方式以外はほぼそのまま適用できる。

【0093】

なお、C-HPMD でも HPMD と同様、CVT に必要な機能は、キー指定によるランダムアクセス機能とキーの値順による順次的アクセス機能であり、RDT に必要な機能は

、キー指定によるランダムアクセス機能である。よって、CVTおよびRDTは、B+木に限らず、上記の機能をそれぞれ実現するキー編成のデータ構造であればよい。もちろん、B+木であれば、上記の機能をそれぞれ効率よく実現でき、C-HPMDの性能を高めることができる。

【0094】

図7に図2の配列要素をチャンクにそのまま置き換えたときのC-HPMDを示す。図7は、C-HPMDにおける論理拡張可能配列とチャンク番号との関係を示す説明図である。なお、論理拡張可能配列中の数字はチャンク番号を表す。

【0095】

2.1 タプルエンコーディング

各次元の属性値のタプルを $\langle c_1, c_2, \dots, c_n \rangle$ として、このタプルを \langle チャンク番号, チャンク内座標パターン \rangle の対に変換する手順を以下に示す。

【0096】

- (a) c_j ($j=1, \dots, n$) を次元 j の CVT $_j$ により次元 j の添字 i_j に変換
- (b) タプル座標 $\langle i_1, i_2, \dots, i_n \rangle$ からチャンク座標 $dp = \langle p_1, p_2, \dots, p_n \rangle$ への変換
 $p_j = i_j \gg k$ ($j=1, \dots, n$) : i_j を k ビット右シフト
- (c) タプルの経歴値 h を求める
各次元の添字 $i_j = CVT_j(c_j)$ ($1 \leq j \leq n$) に対して、
経歴値 h は $H_j[i_j \gg 27]$ ($1 \leq j \leq n$) の最大値
- (d) 経歴値 h のチャンク部分配列の先頭チャンク番号を求める

【0097】

【数11】

$$\begin{cases} 0 & (h=0) \\ 2^{h-1} & (h>1) \end{cases}$$

【0098】

- (e) チャンク座標 dp からのチャンク番号の計算
 - ・チャンク座標から所属チャンク部分配列の経歴値 h を求める
 - ・チャンク番号 cn を求めるマクロ `get_chunk` を計算

【0099】

【数12】

```
#define get_chunk(dp, h, cn)  ¥
{ int i; dp[H[h].dim] -= 1 << H[h-1].v[H[h].dim].sz, ¥
  for (i=0, cn=0; i<n; i++) cn |= (dp[i] & MASK_B) << H[h-1].v[i].ppos; ¥
  cn += 1 << (h-1);
}
```

【0100】

〔例〕チャンク座標 (5,4) を含むチャンクのチャンク番号の計算

【0101】

10

20

30

40

【数 1 3】

所属チャンク部分配列の経歴値 $h = \max\{6, 4\} = 6$;

$H[6].dim=1, dp[1] = 5 \cdot (1 \ll H[5].v[1].sz) = 5 \cdot 1 \ll 2 = 1$

$h=5$ の境界ベクトル: $\langle 2, 3 \rangle$

$1 \& \text{MASK_B} \ll H[5].v[0].ppos = 1 \ll 3 = 8$

$4 \& \text{MASK_B} \ll H[5].v[1].ppos = 4 \ll 0 = 4$

$cn=8+4=12$;

$cn=12+1 \ll 5 = 12+32=44$

10

【0 1 0 2】

(チャンク内マスクパターン)

ランク k の n 次元チャンクの場合、マスクパターンを含むチャンク情報をグローバルに 1 つ定義する。

【0 1 0 3】

【数 1 4】

```
struct chunk_info {
    unsigned char k;    // チャンクのランク
    unsigned char mp;  // マスクパターン
    unsigned char pos[]; // 各次元のパターンの位置
} cmp;
```

20

```
set_chunk_info(unsigned char rank) { // 各情報を初期化
    int i;
    for (i=0; i<n; i++) { cmp.k=rank; cmp.mp=(1<<rank)-1; cmp.pos[i] =(n-i+1)*rank }
}
```

【0 1 0 4】

(f) タップル座標 $\langle i_1, i_2, \dots, i_n \rangle$ からチャンク内座標 $dq = \langle q_1, q_2, \dots, q_n \rangle$ への変換

【0 1 0 5】

【数 1 5】

$q_j = i_j \& \text{cmp.mp}$ ($j=1, \dots, n$) // i_j を下位 k ビット分を残してマスク

30

【0 1 0 6】

(g) タップル座標 $d = \langle i_1, i_2, \dots, i_n \rangle$ からチャンク内座標パターン cp への変換

【0 1 0 7】

【数 1 6】

```
#define make_cpatter(d, cp) \
    { int i; for(cp=0, i=0; i<n; i++) cp = (cp<<k) | (d[i] & cmp.mp) }
```

【0 1 0 8】

40

(h) タップルのエンコーディング

$\langle cn, cp \rangle$

2.2 タップルへのデコーディング

\langle チャンク番号, チャンク内座標パターン \rangle の対から元のタップルの座標への変換操作である。

【0 1 0 9】

(チャンク座標への変換)

チャンク番号 p より所属チャンク部分配列 (基チャンク部分配列) の経歴値 h を求める。これは、 p のビットパターンサイズに相当する (図 7)。 h と p からチャンク座標 $dp[]$ を求める。

50

【 0 1 1 0 】

【 数 1 7 】

```
#define get_pcoordinate(h, p, dp) ¥
    { int i; for (i=0; i<n; i++) dp[i] = (p>> H[h-1].v[i].ppos)&H[h-1].v[i].mp;
      dp[H[h].dim] += (1<<(H[h-1].v[H[h].dim].sz)); }
```

【 0 1 1 1 】

(チャンク内座標への変換)

チャンク内での座標パターンを q としてチャンク内座標 dq[] を求める。

10

【 0 1 1 2 】

【 数 1 8 】

```
#define get_qcoordinate(q, dq) ¥
    { int i; for (i=0; i<n; i++) dq[i] = (q>>cmp.pos[i])&cmp.mp; }
```

【 0 1 1 3 】

(タプルへのデコーディング)

元のタプルの次元 i の座標は $(dp[i] \ll k) | dq[i]$ となる。

【 0 1 1 4 】

20

〔例〕チャンクのランクを 4 として、図 7 における <チャンク番号 4 4, チャンク内座標パターン 5 5> のデコーディング。

4 4 のビットパターンサイズは 6。したがって、当該タプルを含むチャンク部分配列の経歴値 h は 6。その、h=5 の境界ベクトルは <2, 3>。

【 0 1 1 5 】

【 数 1 9 】

```
dp[0]=(44>>H[5].v[0].ppos)&(H[5].v[0].mp)
    =101100>>3&11=1
dp[1]=(44>>H[5].v[1].ppos)&(H[5].v[1].mp)
    =(101100 >> 0)&111=100=4
dp[0] = 1+(1<<H[5].v[0].sz)=1+1<<2=5
```

30

```
dq[0]=(55>>4)&15=11&1111=11=3
```

```
dq[1]=(55>>0)&15=110111&1111=111=7
```

タプルの座標

```
(dp[0]<<4|dq[0], dp[1]<<4|dq[1]) = (5<<4|3, 4<<4|7) = (1010011,1000111) = (83,71)
```

40

【 0 1 1 6 】

(経歴値 h のチャンク部分配列の下限座標と上限座標)

【 0 1 1 7 】

【 数 2 0 】

```
unsigned int dl[ ], du[ ];
get_pcoordinate(h, 1<<(h-1), dl);
get_pcoordinate(h, (1<<h)-1, du);
```

【 0 1 1 8 】

50

2.3 RDTの構成

タプルのエンコード結果 <チャンク番号, チャンク内座標パターン> の対はRDT (B+木) にキーとして登録される。C-HPMDが疎な場合、空のチャンクが多くを占めることを考慮して、検索のオーバヘッドを回避するために、1次元のビットマップCBMをRDTに付随させる。CBMはチャンク配列が拡張されるたびに拡張され、常に、チャンク総数のビット数を持つ。

【0119】

```
unsigned long cbm[ ];
```

先頭ビットを位置0として、チャンク番号の位置に、当該チャンクに、タプルが登録されていれば、1がセットされ、登録されていない(空)のときは0がセットされる。CBMは主メモリ上に置かれる。指定されたチャンク番号のチャンクを検索するときには、まず、CBMの当該ビットを調べて1の時にのみ実際にRDTを検索する。これにより、RDTを無駄に検索することを回避している。CBMの検索は、64ビットマシンの場合、次のマクロにより高速に行われる。

10

【0120】

【数21】

```
#define check_cbm(chunk_num, exist)  \
    {int i, j; j=cbm[i=chunk_num >> 6], exist = j&(1 << (chunk_num-64 * i))}
```

【0121】

20

2.4 C-HPMDの検索

「ある属性について、指定された属性値を持つタプルを検索する(スライス検索)」属性の次元を*i*として、属性値が所属するチャンク部分配列(基チャンク部分配列という)の経歴値を*h0*とする。当該タプルを含む可能性のあるチャンク(以下、候補チャンクという)の番号を順次計算により求めながら候補チャンク内を全検索することを繰り返す。

【0122】

基チャンク部分配列の候補チャンクを検索した後、*h0*より大きい経歴値のチャンク部分配列で所属次元が*i*でないものを順次検索する。タプルの検索(C-HPMD)のプログラムの一例を示せば次の通りである。なお、【数22】の外部データ(グローバルデータ)が、【数23】で使われる。

30

【0123】

【数 2 2】

```

#define get_pcoordinate(h, p, dp) ¥
    { int i; for (i=0; i<n; i++) dp[i] = (p>> H[h-1].v[i].ppos)&H[h-1].v[i].mp;
      dp[H[h].dim] += (1<<(H[h-1].v[H[h].dim].sz); }

#define get_chunk(dp, h, cn) ¥ // d: チャンク座標, h: 経歴値, cn: チャンク番号
    { int i; dp[H[h].dim] -= 1<<H[h-1].v[H[h].dim].sz, ¥
      for (i=0, cn=0; i<n; i++) cn |= (dp[i]&MASK_B)<<H[h-1].v[i].ppos; ¥
      cn += 1<<(h-1); }

#define check_cbm(chunk_num, exist) ¥
    { int i, j; j = cbm[i=chunk_num >> sizeof(unsigned long)*8], ¥
      exist = j << (chunk_num - i * sizeof(unsigned long)*8) }

struct rdt_key { // RDT のキー値
    unsigned int c; // チャンク番号
    unsigned long p; // 座標パターン
};
unsigned short rank; // チャンクのランク値
unsigned int dl[20]; // チャンク部分配列の下限座標
unsigned int du[20]; // チャンク部分配列の上限座標
unsigned int cc[20]; // 現在の拡張可能チャンク配列の実サイズ座標
unsigned long cbm[ ]; // RDT 検索前処理用ビットマップ

long *search_tuple(unsigned char i, mkey colval) { // i: 検索次元
    unsigned int d; // CVTi のデータ値 (添字)
    unsigned int p; // 次元 i のチャンク座標
    unsigned short q; // 次元 i のチャンク内座標
    unsigned int h0; // 基チャンク部分配列の経歴値
    struct rdt_key *rk; // RDT のシーケンスセットのトラバース用

    trfind(CVTi, EQUAL, &colval, &d); // CVTi の検索
    p = (d & MASK_B) >> rank; // チャンク座標
    q = (d & MASK_B) & ((1<<rank)-1); // チャンク内座標
    h0 = Hi[d>>27]; // 基チャンク部分配列の経歴値

    for (h=h0; H[h].dim<>i && h<=hmax; h++)
        // 経歴値, 検索次元, 検索属性値のチャンク座標およびチャンク内座標
        search_subarray(h, i, p, q);
    }

search_subarray(unsigned int h, unsigned char i, unsigned int p, unsigned short q) {
    unsigned int *wc = (unsigned int *)malloc(sizeof(unsigned int)*n); // 作業用座標
    int ii;

    for (ii=0; ii<n; ii++) wc[ii]=0;
    get_pcoordinate(h, 1<<(h-1), dl); // 当該チャンク配列の下限座標の取り込み
    get_pcoordinate(h, (1<<h)-1, du); // 当該チャンク配列の上限座標の取り込み
    for (ii=0; ii<n; ii++) if (cc[ii]<du[ii]) du[ii] = cc[ii]; // 上限座標の修正
    search_chunks(wc, h, p, q, 0);
}

```

【 0 1 2 4】

【数 2 3】

```

search_chunks(unsigned int *wc, unsigned char h, unsigned int p, unsigned short q,
unsigned int lev) {
    int i;
    unsigned long pp; // チャンク番号
    int exist;       // チャンクにタプルが存在するかどうかのフラグ

    if (lev == n-1) // 再帰のレベルがn-1に達した
        for(i=dl[lev]; i<du[lev]; i++) {
            get_chunk(wc, h, pp); // チャンク番号を計算
            check_cbm(pp, exist); // タプルの存在チェック
            if (exist) get_tuples(pp, q); // 存在するときのみRDTを検索
            wc[lev]++; // 次のチャンクの指示
        }
    else {
        for (i=dl[lev]; i<=du[lev]; i++) {
            if (lev == H[h].dim) wc[lev] = p;
            else wc[lev] = i;
            search_chunks(wc, h, p, q, lev+1);
        }
    }
}

get_tuples(unsigned int h, unsigned int pp, unsigned short q) {
    unsigned long *np;

    // RDTの当該チャンク内を順次チェック
    for (trfind(RDT, GTEQ, <pp,0>, key); key.c==pp; trfind(RDT,NEXT,key))
        if ((key.p>>cmp.pos[H[h].dim])&cmp.mp == q) output(key);
    // cmpはチャンク情報構造体(グローバル)
}

```

【0 1 2 5】

〔数 2 2〕に示すsearch_tuple()は、指定された次元 i の属性値を有するタプルを検索する。具体的には、まず、CVTiについて、指定された属性値を検索する。図 3 のフォーマットに従って、次元 i のチャンク座標 p、およびチャンク内座標 q を求める。また、次元 i の経歴値テーブル Hi を参照して、基チャンク部分配列の経歴値 h0 を求める。その後、経歴値 h0 のチャンク部分配列から始めて、次元 i 以外の次元の部分配列について、その経歴値 h、および、i, p, q を引数として、search_subarray() を呼び出して、経歴値 h のチャンク部分配列について、検索対象の候補となるチャンクを検索することを、h をインクリメントしながら繰り返す。

【0 1 2 6】

そして、search_subarray() では、受け取った引数に基づいて、現在の拡張可能チャンク配列の実サイズ座標を取得して、実際に検索が必要な各次元の上現および下限を定める。その後、再帰的に、当該チャンクを検索して、検索結果のタプルを出力する〔数 2 3〕の再帰ルーチンsearch_chunks() を呼び出す。

【0 1 2 7】

さらに、〔数 2 3〕のsearch_chunks() は、チャンク座標格納用配列 wc、検索対象チャンク部分配列の経歴値 h、チャンク座標 p、チャンク内座標 q、および再帰の深さ lev を引数として受け取る。lev の初期値は 0 であり、再帰呼び出しのたびに lev はインクリメントされる。

【0 1 2 8】

(S1)再帰の深さ lev をチェックする。再帰の深さ lev が次元数 n に対して、n - 1 に達していないときは、(S4)へ。達したときには、次の(S2)へ。

【0 1 2 9】

(S2) 検索対象チャンク部分配列はチャンクの 1 次元配列となる。この 1 次元配列中のチャンクに対して、以下の(S3)を繰り返す。

【 0 1 3 0 】

(S3) 渡されたチャンク座標 w_c からチャンク番号を計算する。さらに、このチャンク番号のチャンク内にタプルが存在するかどうかを調べるために、ビットマップ C B M をチェックする。存在する場合のみ、ルーチン `get_tuples()` を呼び出す。`get_tuple()` では、R D T 中の当該チャンクを検索した後、チャンク内を順次チェックして当該タプルを出力する。その後、 w_c を次のチャンクを指示するように更新する。

【 0 1 3 1 】

(S4) 座標値 $w_c[lev]$ を更新して、再帰の深さ lev を 1 インクリメントして、`search_chunks()` を再帰的に呼び出す。 10

【 0 1 3 2 】

2.5 チャンクのランク値の決定について

チャンクのランク値 k は記憶コストと検索時間コストに大きく影響する。すなわち、 k が大きいほど C B M のサイズが減少する。しかし、チャンク内のオフセット (タプル) は全検索する必要があるので、検索時間コストは逆に増大する。このトレードオフはチャンク内のタプルの平均密度を勘案して、決定する必要があるが、 k を変更すれば、C B M と R D T の再編成を必要とするため、実時間処理を阻害する。

【 0 1 3 3 】

3. 先行研究との比較

前掲の (非特許文献 1) ~ (非特許文献 8) は、拡張可能配列に関する研究であるが、経歴・オフセット法や経歴・パターン法のタプルエンコードの考え方は一切表れていない。下記に、経歴・オフセット法に関連する先行研究論文のリストを示す。すべて、本発明者等による研究である (文献 [9] ~ [16])。なお、これらの研究の一部はすでに、P C T 出願しており (出願番号: 05805308.1)、現在審査中である。これらの研究は本発明と同様に、拡張可能配列の概念を基盤としており、タプルのエンコード方式を経歴・オフセット法と呼んでいる。両方式ともに、 n 次元タプルの各次元の属性値は C V T により、拡張可能配列の添字の座標に変換される。変換された座標に対して、経歴・オフセット法では、それが含まれる拡張可能配列の部分配列の拡張経歴値 h と部分配列のアドレス関数により変換された、部分配列内での位置 (オフセット) を表す単一値 o の対 $\langle h, o \rangle$ に 30
エンコードされたのに対して、本方式 (経歴・パターン方式) では、経歴値 h と各次元の添字値のビットパターンを次元順に固定長の記憶域に順次並べたパターン p の対 $\langle h, p \rangle$ にエンコードされる。

【 0 1 3 4 】

両方式の長所、短所を以下に比較する。以下では、経歴・パターン法を $h p$ 法、経歴・オフセット法を $h o$ 法と表記して、タプルの次元数を n とする。

【 0 1 3 5 】

(a) 記憶領域の消費

$h p$ 法の方が消費が少ない。 $h o$ 法において必要であった、〔発明が解決しようとする課題〕で説明した係数ベクトル (サイズのオーダー: $O(n^2)$) が不要。また、 $h o$ 法では経歴値のサイズは 4 バイト程度必要であったのに対して、 $h p$ 法では 1 バイトで十分 (2^{25} の論理拡張可能配列の空間が表現可能) である。したがって、タプルのエンコード値を収容する R D T のサイズが減少し、記憶領域の消費抑制に貢献できる。 40

【 0 1 3 6 】

(b) 論理空間の消費

$h p$ 法では論理拡張可能配列の論理サイズは実サイズに比べて、最悪の場合 2^n 倍の大きさになり得る (論理拡張可能配列の論理サイズ、実サイズについては 1 節を参照のこと)。すなわち、同じタプル数に対して、最悪の場合、論理サイズのビット数は実サイズのビット数より、 n ビット余分に消費する。また、最良の場合は論理サイズと実サイズは一致し、余分な消費はない。これに対して、 $h o$ 法では論理拡張可能配列の論理サイズと 50

実サイズは常に一致しており、論理アドレス空間の無駄な消費は起こらない。

【0137】

(c) タップル挿入・検索速度

h o法ではタップル挿入時に必要なエンコーディングに $n - 2$ 回の乗算と加算が必要であり、タップルの検索時にはタップルが指定された属性値を持つかどうかのチェックに2回の割り算を必要とする。これに対して、h p法では、これらの処理はすべて、機械語命令のマスク命令とシフト命令で行うことができ、この相違がh p法での挿入・検索速度の向上に大きく貢献し得る。

【0138】

逆に、拡張可能配列の部分配列はh o法では $n - 1$ 次元であるのに対して、h p法では n 次元である。したがって、タップルの検索範囲はh p法の方が大きく、この点ではh p法は不利である。

10

【0139】

次に、2つのエンコード方式をチャンク化した場合の方式をそれぞれc - h o法、c - h p法と表記して、c - h o法、c - h p法も含めて比較する。

【0140】

(a') 記憶領域の消費

c - h o法、c - h p法の両方式ともチャンク単位に経歴値テーブル(H)および各次元の経歴値テーブル(H_i)が確保されるため、これらのテーブル領域サイズのh o法およびh p法の場合と比べて減少するが、その減少度はほぼ同一である。CVT_i、RDTおよび他のデータ構造サイズはh o法およびh p法の場合とほぼ同一である。c - h o法とc - h p法の比較では、c - h p法でも係数テーブルが不要な分c - h o法より有利である。しかし、c - h o法およびc - h p法ともにタップルのエンコード結果の対の第1項は経歴値ではなく、チャンク番号であるので、エンコード結果のサイズに変わりはない。したがって、RDTのサイズも同一である。

20

【0141】

(b') 論理空間の消費

上記(b)のh o法とh p法の比較がそのまま、c - h o法、c - h p法についても当てはまる。しかし、チャンク化の利点の一つは、エンコードされるタップルの論理アドレス空間の拡大である。すなわち、チャンクのランク k に対して、チャンク化しない場合に比べてこの論理アドレス空間は k^n 倍に拡大する。一般には論理拡張可能配列はかなり疎であることが多いので、 k の値をある程度大きくできる。c - h p法はc - h o法に比べて論理アドレス空間の使用率が低下するが、チャンク化により、論理アドレス空間を格段に拡大可能であり、大規模なタップル集合にもある程度まで対応し得る。もちろん、チャンク化による論理空間の拡大は“ただ”であり、拡大しても、実記憶量に影響はない。

30

【0142】

(c') タップル挿入・検索速度

h o法とh p法における、上記(c)での「タップルが指定された属性値を持つかどうかのチェック」の時間的コストの比較はそのまま、c - h o法、c - h p法についても当てはまる。h p法はh o法に比べて、タップルの検索範囲が大きいことが難点であったが、チャンク化によるc - h o法とc - h p法では解消される。すなわち、指定された属性値のタップルを含み得るチャンクの番号のみを計算で求めることにより、検索ターゲットのチャンク集合の範囲を大幅に絞り込むことが可能である。この範囲絞り込みの効果はc - h o法でもc - h p法でも同じであり、検索ターゲットのチャンク集合の範囲は同一である。また、h o法とh p法ともに検索速度に関して“経歴値依存性”がある。すなわち、1.5節で述べた、h p法の性質はそのまま、h o法においても当てはまり、経歴値が大きいほど、検索範囲を限定することができる。指定された属性値に対応する基部分配列の経歴値が大きいほど検索対象の部分配列集合は少ないが、この経歴値が小さいほど多くの部分配列を検索する必要がある。チャンク化を行った場合、この検索速度の“経歴値依存性”は解消される。すなわち検索対象属性が同じ場合、どのような属性値の検索でもその

40

50

検索速度はほぼ一定である。

【 0 1 4 3 】

〔 関連先行技術研究文献 〕

[9] M.Kuroda, N.Azuma, K.M.A.Hasan, T.Tsuji, K.Higuchi, " An Implementation Scheme of Relational Tables ", Proc. of International Conference of Data Engineering Workshops, 2005.

[10] K.M.A.Hasan, M.Kuroda, N.Azuma, T.Tsuji, K.Higuchi, " An Extendible Array Based Implementation of Relational Tables for Multidimensional Databases ", 7th International Conference on Data Warehousing and Knowledge Discovery, pp.233-242, 2005.

10

[11] K.M.A.Hasan, T.Tsuji, K.Higuchi, " A parallel implementation scheme of relational tables based on multidimensional extendible array ", International Journal of Data Warehousing and Mining, Vol.2, No.4, pp.66-85, 2006.

[12] K.M.A.Hasan, T.Tsuji, K.Higuchi, " An Efficient Implementation for MOLAP Basic Data Structure and Its Evaluation ", Proc. of Int'l Conference on Database Systems for Advanced Applications (DASFAA2007), pp.288-299, 2007.

[13] Tatsuo Tsuji, Masayuki Kuroda, Ken Higuchi, " History offset implementation scheme for large scale multidimensional data sets ", Proc. of the 2008 ACM Symposium on Applied Computing (SAC2008), pp.1021-1028, 2008.

[14] Dong Jin, Tatsuo Tsuji, Takayuki Tsuchida, Ken Higuchi, " An Incremental Maintenance Scheme of Data Cubes ", Proc. of Int'l Conference on Database Systems for Advanced Applications (DASFAA 2008), pp.172-187, 2008.

20

[15] Tatsuo Tsuji, Dong Jin, Ken Higuchi, " Data Compression for Incremental Data Cube Maintenance ", Proc. of Int'l Conference on Database Systems for Advanced Applications (DASFAA 2008), pp.682-685, 2008.

[16] 土田隼之, 都司達夫, 樋口健, " M O L A P 用多次元配列構築のためのバッファリング方式 ", 日本データベース学会論文誌, Vol.7, No.1, pp.19-24, 2008.

4 . データベース装置

つづいて、図 1 を参照しながら、上述した関係テーブルの記憶、操作方式を実現するデータベース装置の一構成例について説明する。図 1 は、本発明の一実施の形態に係るデータベース装置 1 の構成の概略を示す機能ブロック図である。

30

【 0 1 4 4 】

図 1 に示すように、データベース装置 1 は、データ格納部 (データベース記憶部) 1 0 、補助テーブル部 (データベース記憶部) 2 0 、テーブル管理部 3 0 、入出力部 4 0 を備えて構成されている。

【 0 1 4 5 】

データ格納部 1 0 は、ディスク装置 2 上に、C V T (第 1 データ、第 1 の B + 木データ) 1 1 、R D T (第 2 データ、第 2 の B + 木データ、要素位置データ、要素位置 B + 木データ) 1 2 、および各種補助テーブル (属性毎経歴値テーブル 2 1 、経歴値テーブル 2 2 、属性値テーブル 2 3) を格納している。

40

【 0 1 4 6 】

R D T 1 2 は、関係テーブルの各タプルに対応する拡張可能配列の要素の位置を示す位置情報をキー値として登録したデータである。具体的には、本実施の形態では、上記位置情報は、要素が属する拡張可能配列の区画の位置を示す区画位置情報と、区画内における要素の位置を示す、当該タプルの各属性の属性値に一意に対応した値を所定の属性順に並べた座標情報と、を含む情報である。

【 0 1 4 7 】

(1) H P M D 法

以下、区画が拡張可能配列の部分配列である場合について説明するが、この場合、R D T 1 2 は、関係テーブルのタプルに対応する拡張可能配列の要素が属する部分配列の経

50

歴値（区画位置情報）、および、当該タプルの各属性値に対応する上記拡張可能配列の各添字のビットパターンを所定の属性順に並べた座標パターン（座標情報）、の2項組表現をキー値として登録したB+木データである。

【0148】

CVT11は、関係テーブルの属性毎に設けられ、属性の各属性値から拡張可能配列の添字に変換するためのB+木データである。

【0149】

ここで、関係テーブルがn個の属性からなる場合、データ格納部10には、n個のCVT（key-subscript ConVersion Tree）とRDT（Real Data Tree）とからなるn+1個のB+木のデータが格納されている。なお、データベースが複数の関係テーブルから構成される場合、このn+1個のB+木のセットが複数存在することになる。

10

【0150】

なお、CVT11は、キー指定によるランダムアクセス機能とキーの値順による順次的アクセス機能を実現するキー編成のデータ構造であればよく、RDT12は、キー指定によるランダムアクセス機能を実現するキー編成のデータ構造であればよい。本実施の形態では、この2つの機能を効率よく実現し、HPMDの性能を高めるため、CVT11およびRDT12にB+木データを採用する。

【0151】

補助テーブル部20は、属性毎経歴値テーブル21、経歴値テーブル22、属性値テーブル23を、主メモリ3上に保持している。

20

【0152】

なお、CVT11、RDT12、および補助テーブル群（属性毎経歴値テーブル21、経歴値テーブル22、属性値テーブル23）は、データ格納部10に格納されている。データ格納部10は、ハードディスク等のディスク装置2上に配置されている。属性毎経歴値テーブル21、経歴値テーブル22、属性値テーブル23の補助テーブル群は、データベース装置1の処理開始時にディスク装置2から読み出されて主メモリ3上の補助テーブル部20に保持され、データベース処理中に変更が加えられた場合に、ディスク装置2上のデータ格納部10の対応部分に書き戻される。

【0153】

属性毎経歴値テーブル21は、属性毎の配列拡張の時間的順序を表す経歴値の1次元配列である。経歴値テーブル22は、経歴値に対応する配列拡張した属性の次元、および、当該経歴値に対応する拡張部分配列の任意の要素について、属性毎に拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを記録している。属性値テーブル23は、拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を記録している。

30

【0154】

テーブル管理部30は、タプル検索部（タプル検索手段）31、タプル挿入部（タプル挿入手段）32、タプル削除部（タプル削除手段）33を備えている。

【0155】

タプル検索部31は、タプルの検索の処理を行う。タプル挿入部32は、タプルの挿入の処理を行う。タプル削除部33は、タプルの削除の処理を行う。特に、タプル挿入部32、タプル削除部33は、CVT11、RDT12、属性毎経歴値テーブル21、経歴値テーブル22、属性値テーブル23に対して、必要な保守を行う。

40

【0156】

具体的には、タプル検索部31は、ある属性値を持つタプルを検索するとき、属性値が属する属性のCVT11を用いて、該属性値を拡張可能配列の添字に変換し、当該次元の経歴値テーブルより経歴値を求めた後、RDT12を参照して、2項組表現されたキー値中の座標パターンに含まれる、属性の属性値に対応するビットパターンが、上記添字のビットパターンと一致するタプルを抽出する。

【0157】

50

ここで、入出力部 40 を介してユーザから得た検索要求は、タプル検索部 31 により、< 経歴値、座標パターン > 対であるキー値の集合として検索されるが、キー値は本データベースにおけるタプルの内部表現であり、ユーザには理解できない。そこで、キー値 - 属性値逆変換部 34 は、この検索結果をユーザが理解できる表現として返すために、2 項組表現のキー値から属性値の組としてのタプルに逆変換する。具体的には、タプル検索部 31 が抽出した < 経歴値、座標パターン > の 2 項組から、各属性の属性毎経歴値テーブル 21、経歴値テーブル 22、属性値テーブル 23 を参照してデコードを行って、拡張可能配列の各属性の添字に変換し、属性毎に得た配列添字値から得た属性値を、属性順に並べてタプルを得る。

【 0 1 5 8 】

また、タプル挿入部 32 は、新たな属性値を持つタプルを挿入するとき、配列拡張が必要である場合には、属性値が属する属性の部分配列を追加するとともに、経歴値をインクリメントして当該属性の属性毎経歴値テーブル 21 に登録する基本データ拡張処理を行い、配列拡張が不必要である場合、および、配列拡張が必要である場合に上記基本データ拡張処理を行った後、属性値を属性値テーブル 23 および C V T 1 1 に登録するとともに、当該タプルの各属性値に対応する座標パターンを生成して、経歴値および座標パターンの 2 項組表現をキー値として R D T 1 2 へ挿入する。

【 0 1 5 9 】

また、タプル削除部 33 は、タプル挿入部 32 がタプルを検索する際のアルゴリズムを適用して当該タプルを同定した後、タプル削除に伴う R D T や C V T 等の H P M D データ構造に対して必要な削除とメンテナンスを行う。

【 0 1 6 0 】

なお、テーブル管理部 30 は、データベースの管理全般を行うものである。よって、タプル検索部 31 などのように個別に機能ブロックとして明記しないが、データベースの管理に付随する処理なども行うことは言うまでもない。

【 0 1 6 1 】

入出力部 40 は、データベース装置 1 を操作するためのインターフェイスである。すなわち、入出力部 40 は、データベース装置 1 に対して、ユーザが直接、処理要求を入力して、データベース装置 1 からその結果を出力するためのユーザインターフェイス、および、それらの送受信をネットワーク経由で制御するための通信インターフェイスである。

【 0 1 6 2 】

(2) C - H P M D 法

なお、2 節で説明した C - H P M D 法の場合には、上記データ記録システム 1 を以下のように変更すればよい。

【 0 1 6 3 】

区画がチャンク化拡張可能配列の部分配列である場合、R D T 1 2 は、関係テーブルのタプルに対応するチャンク化拡張可能配列の要素が属するチャンクのチャンク番号 (区画位置情報)、および、当該タプルの各属性値に対応する上記チャンク化拡張可能配列の当該チャンクにおける各添字のビットパターンを所定の属性順に並べたチャンク内座標パターン (座標情報)、の 2 項組表現をキー値として登録した B + 木データである。

【 0 1 6 4 】

C V T 1 1 は、関係テーブルの属性毎に設けられ、属性の各属性値からチャンクを要素とするチャンク化拡張可能配列の添字に変換するための B + 木データである。

【 0 1 6 5 】

なお、C - H P M D 法でも H P M D 法と同様、C V T 1 1 は、キー指定によるランダムアクセス機能とキーの値順による順次的アクセス機能を実現するキー編成のデータ構造であればよく、R D T 1 2 は、キー指定によるランダムアクセス機能を実現するキー編成のデータ構造であればよい。本実施の形態では、この 2 つの機能を効率よく実現し、C - H P M D の性能を高めるため、C V T 1 1 および R D T 1 2 に B + 木データを採用する。

【 0 1 6 6 】

10

20

30

40

50

属性毎経歴値テーブル 2 1 は、属性毎のチャンク配列拡張の時間的順序を表す経歴値の 1 次元配列である。経歴値テーブル 2 2 は、チャンク番号に対応するチャンク配列拡張した属性の次元、および、当該経歴値に対応する拡張チャンク部分配列中の任意のチャンクについて、属性毎にチャンク拡張可能配列における対応次元の添字の表現に要するビット数を要素とする境界ベクトルを記録している。属性値テーブル 2 3 は、チャンク化拡張可能配列の添字毎にその添字に対応する属性値および該属性値を持つすべてのタプル数を記録している。

【 0 1 6 7 】

ここで、C - H P M D 法の場合、主メモリ 3 上のテーブル管理部 3 0 には、C B M 3 5 が保持される。C B M 3 5 は、チャンク総数と同じビット数を持ち、チャンク番号に対応するビット位置に、当該チャンクにタプルが登録されていれば“ 1 ”、登録されていなければ“ 0 ”がセットされる 1 次元ビットマップである。

10

【 0 1 6 8 】

また、タプル検索部 3 1 は、ある属性値を持つタプルを検索するとき、属性値が属する属性の C V T 1 1 を用いて、該属性値をチャンク化拡張可能配列の添字に変換し、当該次元の経歴値テーブルより経歴値を求めた後、R D T 1 2 を参照して、2 項組表現されたキー値中のチャンク内座標パターンに含まれる、属性の属性値に対応するビットパターンが、上記添字のビットパターンと一致するタプルを抽出する。タプル検索部 3 1 は、指定されたチャンク番号のチャンクを検索する際、C B M 3 5 の当該ビットを調べ、“ 1 ”の時にのみ R D T 1 2 を検索する。

20

【 0 1 6 9 】

ここで、キー値 - 属性値逆変換部 3 4 は、H P M D 法の場合と同様に、タプル検索部 3 1 が抽出した<チャンク番号、チャンク内座標パターン>の 2 項組から、各属性の属性毎経歴値テーブル 2 1、経歴値テーブル 2 2、属性値テーブル 2 3 を参照してデコードを行って、チャンク化拡張可能配列の各属性の添字に変換し、属性毎に得た配列添字値から得た属性値を、属性順に並べてタプルを得る。

【 0 1 7 0 】

また、タプル挿入部 3 2 は、新たな属性値を持つタプルを挿入するとき、チャンク配列拡張が必要である場合には、属性値が属する属性のチャンク部分配列を追加するとともに、経歴値をインクリメントして当該属性の属性毎経歴値テーブル 2 1 に登録する基本データ拡張処理を行い、チャンク配列拡張が不必要である場合、および、チャンク配列拡張が必要である場合に上記基本データ拡張処理を行った後、属性値を属性値テーブル 2 3 および C V T 1 1 に登録するとともに、当該タプルの各属性値に対応するチャンク内座標パターンを生成して、チャンク番号およびチャンク内座標パターンの 2 項組表現をキー値として R D T 1 2 へ挿入する。

30

【 0 1 7 1 】

ここで、C - H P M D の場合、R D T 1 2 には<チャンク番号、チャンク内座標パターン>を登録するが、属性毎経歴値テーブル 2 1 には H P M D の場合と同様に経歴値を登録する。C - H P M D の場合、区画の単位はチャンクであり、区画中のタプルの同定は R D T 1 2 に格納されるキー値、すなわち、チャンクである区画の位置情報であるチャンク番号とその区画(チャンク)内の座標パターンの対で行われる。つまり、属性毎経歴値テーブル 2 1 に書き込む経歴値は、拡張時に確保されるチャンク部分配列を同定するのに使われ、このチャンク部分配列中の区画(チャンク)の同定、すなわち、チャンク番号の同定には、H P M D における区画(拡張可能配列の部分配列)中のタプル同定の方法が、タプルをチャンクと見立てることにより、大略そのまま適用される。

40

【 0 1 7 2 】

経歴値を属性毎経歴値テーブル 2 1 に書き込む操作は、H P M D でも、C - H P M D でも同様である。H P M D と C - H P M D との違いは、拡張可能配列の基本要素の単位だけの違いであり、H P M D の場合にはタプルに 1 対 1 に対応する配列要素が拡張の基本要素であり、C - H P M D の場合には配列要素の集まりであるチャンクが拡張の基本要素と

50

なる。

【0173】

最後に、データベース装置1は、ワークステーションやパーソナルコンピュータ等の汎用のコンピュータをベースに構成できる。よって、データベース装置1の各ブロック、特にテーブル管理部30は、次のようにCPUを用いてソフトウェアによって実現することができる。なお、データベース装置1は、その機能を複数の装置に分散させたシステムとして構成することもできる。

【0174】

すなわち、データベース装置1は、各機能を実現する制御プログラムの命令を実行するCPU (central processing unit)、上記プログラムおよびデータベースデータを格納した二次記憶装置 (磁気ディスク装置)、上記プログラムおよびデータベースデータを展開するRAM (random access memory)などを備えている。そして、本発明の目的は、上述した機能を実現するソフトウェアであるテーブル管理部30の制御プログラム (データベースの管理プログラム)のプログラムコード (実行形式プログラム、中間コードプログラム、ソースプログラム)をコンピュータで読み取り可能に記録した記録媒体を、上記データベース装置1に供給し、そのコンピュータ (またはCPUやMPU)が記録媒体に記録されているプログラムコードを読み出し実行することによって、達成可能である。

【0175】

上記記録媒体としては、例えば、磁気テープやカセットテープ等のテープ系、フロッピー (登録商標) ディスク / ハードディスク等の磁気ディスクやCD-ROM / MO / MD / DVD / CD-R等の光ディスクを含むディスク系、ICカード (メモリカードを含む) / 光カード等のカード系、あるいはマスクROM / EPROM / EEPROM / フラッシュROM等の半導体メモリ系などを用いることができる。

【0176】

また、データベース装置1を通信ネットワークと接続可能に構成し、上記プログラムコードを通信ネットワークを介して供給してもよい。この通信ネットワークとしては、特に限定されず、例えば、インターネット、イントラネット、エキストラネット、LAN、ISDN、VAN、CATV通信網、仮想専用網 (virtual private network)、電話回線網、移動体通信網、衛星通信網等が利用可能である。また、通信ネットワークを構成する伝送媒体としては、特に限定されず、例えば、IEEE1394、USB、電力線搬送、ケーブルTV回線、電話線、ADSL回線等の有線でも、IrDAやリモコンのような赤外線、Bluetooth (登録商標)、802.11無線、HDR、携帯電話網、衛星回線、地上波デジタル網等の無線でも利用可能である。なお、本発明は、上記プログラムコードが電子的な伝送で具現化された、搬送波に埋め込まれたコンピュータデータ信号の形態でも実現され得る。

【0177】

本実施の形態は本発明の範囲を限定するものではなく、本発明の範囲内で種々の変更が可能である。

【産業上の利用可能性】

【0178】

本発明は関係データベースに広く適用できるものであり、特に、データウェアハウジングやデータマイニング分野など、大規模関係テーブルの高速検索処理が必要な産業上の多くの分野に好適である。記憶効率も優れている。また、本発明は、オブジェクトIDの導入により、関係テーブルのみではなく、複合オブジェクトの効率よい実装方式も提供できる。さらに、大規模なXML (EXtensible Markup Language) 文書の効率よい記憶構造としての使用の他、動的に大きさが変化する多次元データ集合一般の実装にも応用できる。

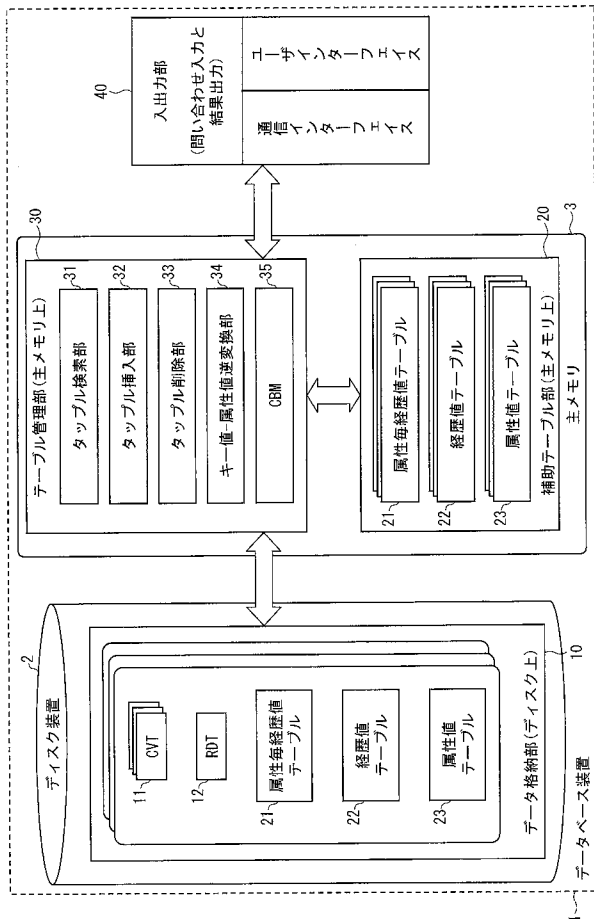
【符号の説明】

【0179】

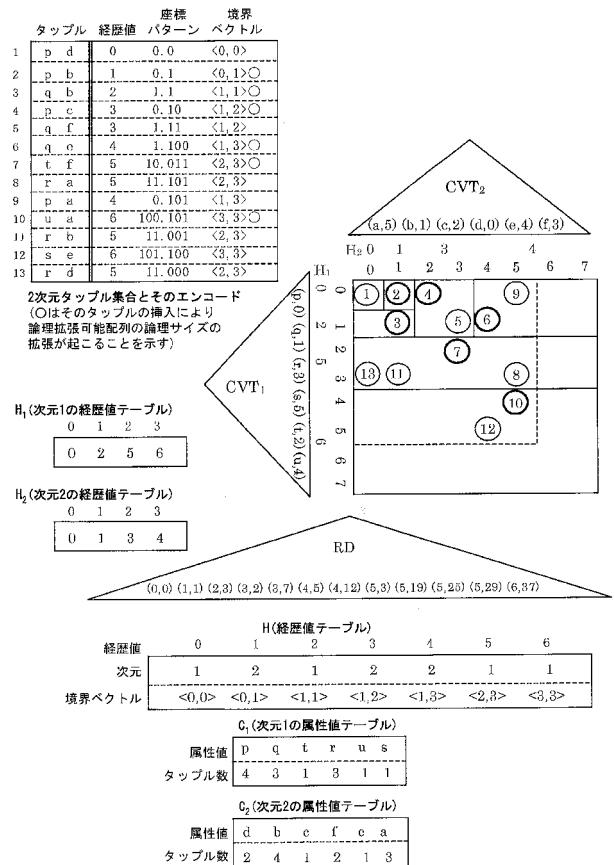
- 1 データベース装置
- 2 ディスク装置

- 3 主メモリ
- 10 データ格納部 (データベース記憶部)
- 11 C V T (第1データ、第1のB + 木データ)
- 12 R D T (第2データ、第2のB + 木データ、要素位置データ、要素位置B + 木データ)
- 20 補助テーブル部 (データベース記憶部)
- 21 属性毎経歴値テーブル
- 22 経歴値テーブル
- 23 属性値テーブル
- 30 テーブル管理部
- 31 タプル検索部 (タプル検索手段)
- 32 タプル挿入部 (タプル挿入手段)
- 33 タプル削除部 (タプル削除手段)
- 34 キー値 - 属性値逆変換部 (キー値 - 属性値逆変換手段)
- 35 C B M

【図1】



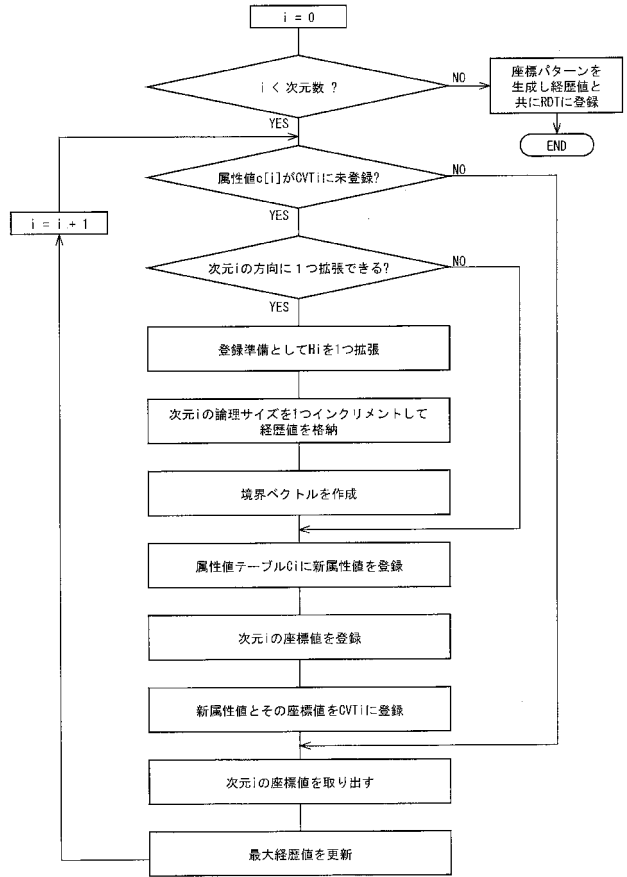
【図2】



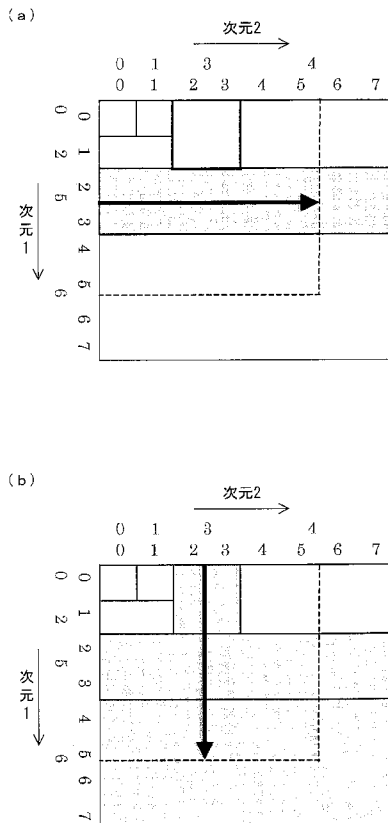
【 図 3 】



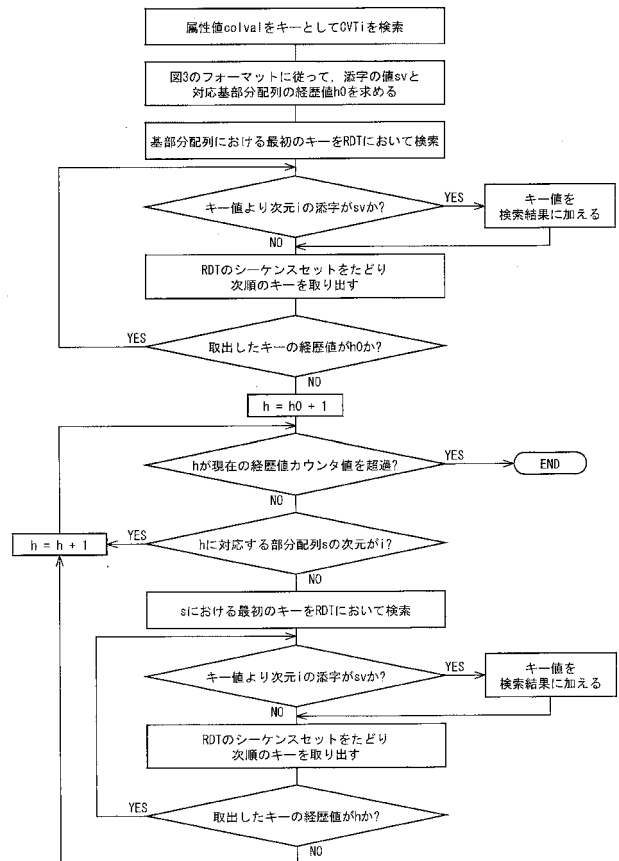
【 図 4 】



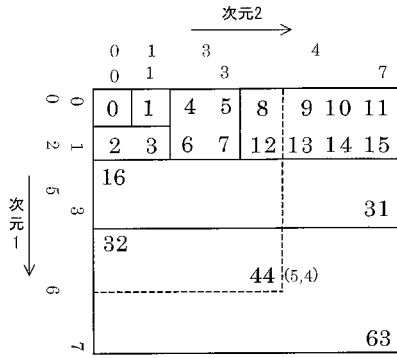
【 図 5 】



【 図 6 】



【図7】

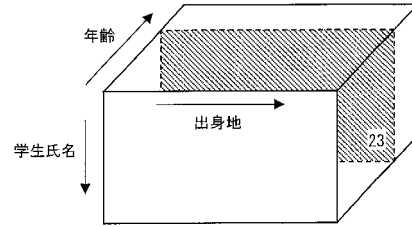


【図8】

学生氏名	学生番号	出身地	年齢	属性	
				英語	数学
堀江 武	01	福井	21	優	可
山口誠司	02	福井	23	良	優
結城宏和	03	富山	22	優	良
芳野 学	04	大阪	22	可	優

←タプル

【図9】



【図10】

