(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第5419069号
(P5419069)

(45)発行日　平成26年2月19日(2014.2.19)　(24)登録日　平成25年11月29日(2013.11.29)

(51)Int.Cl.　F I
G06F 12/00 (2006.01)　G06F 12/00 512
G06F 17/30 (2006.01)　G06F 17/30 180D
G06F 17/30 150Z

請求項の数 12　(全 38 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)
(65)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)
審査請求日　平成23年10月14日(2011.10.14)

(73)特許権者　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

審査官　加内　慎也

(56)参考文献　特開２００３－１４１１５９（ＪＰ，Ａ）
国際公開第２００６／０４６６６９（ＷＯ，Ａ１）

最終頁に続く

(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)【特許請求の範囲】
【請求項１】
関係テーブルの各タッ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、
関係テーブルには複数の属性が含まれ、当該関係テーブルに含まれるタップルは当該関係テーブルが有する上記属性のすべてを有し、
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、当該タッ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、を含む情報であるデータベース装置において、
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、属性の各属性値と拡張可能配列の添字との対応関係を規定する第１データと、
上記区画位置情報である、関係テーブルのタッ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、当該タッ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、
新たな属性値を持つタップルを挿入するとき、拡張可能配列に対して、上記属性値が属する属性の区画を追加する配列拡張において、各区画の追加が行われる時間的順序を表す上記経歴値を登録した属性毎経歴値テーブルと、

上記経歴値、各経歴値に対応する時間的順序で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ッ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ッ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ップルを抽出するタップル検索手段をさらに具備することを特徴とするデータベース装置。
【請求項２】
関係テーブルの各タッ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、
関係テーブルには複数の属性が含まれ、当該関係テーブルに含まれるタップルは当該関係テーブルが有する上記属性のすべてを有し、
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、当該タッ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、を含む情報であるデータベース装置において、
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、属性の各属性値と拡張可能配列の添字との対応関係を規定する第１データと、
上記区画位置情報である、関係テーブルのタッ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、当該タッ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、
新たな属性値を持つタップルを挿入するとき、拡張可能配列に対して、上記属性値が属する属性の区画を追加する配列拡張において、各区画の追加が行われる時間的順序を表す上記経歴値を登録した属性毎経歴値テーブルと、
上記経歴値、各経歴値に対応する時間的順序で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ッ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ッ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、当該タッ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ップル挿入手段をさらに具備することを特徴とするデータベース装置。
【請求項３】
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、
関係テーブルには複数の属性が含まれ、当該関係テーブルに含まれるタップルは当該関係テーブルが有する上記属性のすべてを有し、

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、属性の各属性値と拡張可能配列の添字との対応関係を規定する第１データと、
関係テーブルのタッ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、当該タッ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、
新たな属性値を持つタップルを挿入するとき、拡張可能配列に対して、上記属性値が属する属性の区画を追加する配列拡張において、各区画の追加が行われる時間的順序を表す上記経歴値を登録した属性毎経歴値テーブルと、
上記経歴値、各経歴値に対応する時間的順序で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ッ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ッ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ッ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４】
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、
関係テーブルには複数の属性が含まれ、当該関係テーブルに含まれるタップルは当該関係テーブルが有する上記属性のすべてを有し、
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、属性の各属性値と拡張可能配列の添字との対応関係を規定する第１データと、
関係テーブルのタッ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、当該タッ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、
新たな属性値を持つタップルを挿入するとき、拡張可能配列に対して、上記属性値が属する属性の区画を追加する配列拡張において、各区画の追加が行われる時間的順序を表す上記経歴値を登録した属性毎経歴値テーブルと、
上記経歴値、各経歴値に対応する時間的順序で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ッ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ッ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、当該タッ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５】

請求項１または２に記載のデータベース装置に用いられる、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、

上記第１データと、上記第２データと、上記属性毎経歴値テーブルと、上記経歴値テーブルと、上記属性値テーブルと、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６】

関係テーブルの各タッ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、

関係テーブルには複数の属性が含まれ、当該関係テーブルに含まれるタップルは当該関係テーブルが有する上記属性のすべてを有し、

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、当該タッ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、を含む情報であるデータベース装置において、

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、属性の各属性値とチャンク化拡張可能配列の添字との対応関係を規定する第１データと、

上記区画位置情報である、関係テーブルのタッ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、当該タッ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、

新たな属性値を持つタップルを挿入するとき、チャンク化拡張可能配列に対して、上記属性値が属する属性のチャンクを追加するチャンク配列拡張において、各チャンクを含むチャンク部分配列の追加が行われる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、

上記経歴値、各経歴値に対応する時間的順序で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、チャンク化拡張可能配列における対応次元のチャンク毎に付される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ッ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ッ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ップルを抽出するタップル検索手段をさらに具備することを特徴とするデータベース装置。

【請求項７】

関係テーブルの各タッ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、

関係テーブルには複数の属性が含まれ、当該関係テーブルに含まれるタップルは当該関係テーブルが有する上記属性のすべてを有し、

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、当該タッ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、を含む情報であるデータベース装置において、
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、属性の各属性値とチャンク化拡張可能配列の添字との対応関係を規定する第１データと、
上記区画位置情報である、関係テーブルのタッ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、当該タッ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、
新たな属性値を持つタップルを挿入するとき、チャンク化拡張可能配列に対して、上記属性値が属する属性のチャンクを追加するチャンク配列拡張において、各チャンクを含むチャンク部分配列の追加が行われる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、
上記経歴値、各経歴値に対応する時間的順序で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、チャンク化拡張可能配列における対応次元のチャンク毎に付される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ッ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ッ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、当該タッ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ップル挿入手段をさらに具備することを特徴とするデータベース装置。

【請求項８】
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、
関係テーブルには複数の属性が含まれ、当該関係テーブルに含まれるタップルは当該関係テーブルが有する上記属性のすべてを有し、
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、属性の各属性値とチャンク化拡張可能配列の添字との対応関係を規定する第１データと、
関係テーブルのタッ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、当該タッ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、
新たな属性値を持つタップルを挿入するとき、チャンク化拡張可能配列に対して、上記属性値が属する属性のチャンクを追加するチャンク配列拡張において、各チャンクを含むチャンク部分配列の追加が行われる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、
上記経歴値、各経歴値に対応する時間的順序で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、チャンク化拡張可能配列における対応次元のチャンク毎に付される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ッ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ッ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ッ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９】
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、
関係テーブルには複数の属性が含まれ、当該関係テーブルに含まれるタップルは当該関係テーブルが有する上記属性のすべてを有し、
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、属性の各属性値とチャンク化拡張可能配列の添字との対応関係を規定する第１データと、
関係テーブルのタッ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、当該タッ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、
新たな属性値を持つタップルを挿入するとき、チャンク化拡張可能配列に対して、上記属性値が属する属性のチャンクを追加するチャンク配列拡張において、各チャンクを含むチャンク部分配列の追加が行われる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、
上記経歴値、各経歴値に対応する時間的順序で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、チャンク化拡張可能配列における対応次元のチャンク毎に付される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ッ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ッ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、当該タッ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。
【請求項１０】
請求項６または７に記載のデータベース装置に用いられる、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、
上記第１データと、上記第２データと、上記属性毎経歴値テーブルと、上記経歴値テーブルと、上記属性値テーブルと、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１１】
請求項１、２、６、７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１２】

請求項１１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】

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、タップルという）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。

【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ップルの集合である。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、各タッ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。

【０００５】

（１）例えば、年齢が２３のタップルを検索する場合、テーブル中のすべてのタッ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、出身地が福井のタップルは数多く現れ、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。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ップルを表す。

【０００８】

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。

【０００９】

この時、年齢が２３のタップル集合は「年齢」次元の値が２３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。

【００１０】

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】

【００１１】

【非特許文献１】A.L.Rosenberg, “Allocating Storage for Extendible Arrays”, Jou

rnal 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-I, Vol.J86-D-I, 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】
【００１２】
ここで、多次元配列によるテーブルの表現は高速にタッ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。
【００１３】
（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ップルの追加は不可能である。
【００１４】
（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（存在しないタップル）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。
【００１５】
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 Multidimensional Data:ＨＰＭＤ）と呼ぶ。ＨＰＭＤ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。
【００１６】
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.Otoo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。
【００１７】
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。

【００１８】

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。

【００１９】

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 $<i_1,i_2,i_3,i_4>$ のアドレスはよく知られているように、

$$s_2s_3s_4i_1+s_3s_4i_2+s_4i_3+i_4 \quad (1)$$

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。

【００２０】

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 [s2,s3,s4] 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 $<i_1,i_2,i_3,i_4>$ 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。

【００２１】

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 $<i_1,i_2,i_3>$ のオフセットは(1)式と同様、１次関数 $s_3s_4i_1+s_4i_2+i_3$ となる。この時、(s_3s_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。

【００２２】

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 <3,4> のアドレス計算は次のように行われる。 H1[3] < H2[4] であるから、要素 <3,4> 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 <3> 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ップルの追加とタッ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ッ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、当該タッ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、関係テーブルのタッ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、当該タッ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、関係テーブルのタッ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、当該タッ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、当該タッ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、＜経歴値，座標パターン＞であり、（２）の場合、＜チャンク番号，チャンク内座標パターン＞となる。なお、チャンク番号は、要素の添字＜$i_1,i_2,...,i_n$＞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】

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連接している必要はない。（ただし、高速検索のためには、位置的に連接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。

【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、関係テーブルのタップルに対応する拡張可能配列の要素の経歴値と、当該タッ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、タップルを検索、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、関係テーブルのタッ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、当該タッ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ッ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ッ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ップルを抽出するタッ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ッ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、当該タッ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ッ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、関係テーブルのタッ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、当該タッ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ッ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。

【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ッ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、関係テーブルのタッ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、当該タッ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。

【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、関係テーブルのタッ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、当該タッ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ッ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。

【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ッ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ップルを抽出するタップル検索手段を具備することを特徴としている。

【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ッ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、当該タッ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ッ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、関係テーブルのタッ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、当該タッ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ッ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ッ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ッ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、当該タッ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ッ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】タッ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】

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】

（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ップルのエンコード値が格納される。なお、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、ＨＰＭＤの性能を高めることができる。

【００５１】

（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ッ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次元のタッ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 $<s_1,s_2,\cdots\cdots,s_n>$ 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、$<b(s_1),b(s_2),\cdots\cdots,b(s_n)>$ を経歴値ｈにおけるＡの境界ベクトルと呼ぶ。

【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 $<s_1,\cdots\cdots,s_{k-1},2*s_k,s_{k+1},\cdots\cdots,s_n>$ 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 $V=<b(s_1),\cdots\cdots,b(s_{k-1}),b(s_k)+1,b(s_{k+1}),\cdots\cdots,b(s_n)>$ 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 $i=<i_1,i_2,\cdots\cdots,i_n>$ 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 $<b(i_1),b(i_2),\cdots\cdots,b(i_n)>$ について、経歴値$H_k(b(i_k))$（1≦k≦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 $i=(i_1,i_2,\cdots\cdots,i_n)$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≦k≦n）をＨ[h]の境界ベクトルにより、分離する。

【００５７】
（タップルのエンコード、デコードの例）
たとえば、タップル（s,e）の場合、ＣＶＴ$_1$(s)、ＣＶＴ$_2$(e) を検索してタップルの座標（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 <3,3> 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 <3,3> 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、元のタップル(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。このことはタップルのアクセス速度および検索速度が従来の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 <s_1,s_2,……,s_n> とすると、拡張可能配列の方式では、s_1+s_2+……+s_n であるのに対して、$\log_2 s_1$+$\log_2 s_2$+……+$\log_2 s_n$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ッ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ップルのみの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ップルは除外しており、記憶域を占めないので以後、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
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Ｈｉについて、Ｈｉ[d>>27] で次元ｉの経歴値が求まり、d&0x07FFFFFF で添字の値が求まる。

【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 */
}
```

【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、<8,10,7,20,15,4> の６次元境界ベクトルの時、配列ｖは、

【００６７】

【数２】

	sz	ppos	mp
0	8,	56,	00000000000000000000000011111111
1	10,	46,	00000000000000000000001111111111
2	7,	39,	00000000000000000000000001111111
3	20,	19,	00000000000011111111111111111111
4	15,	4,	00000000000000001111111111111111
5	4,	0,	00000000000000000000000000001111

【００６８】

となる。

【００６９】

タッ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+……+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≦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。

【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
```

【００７２】

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

。

【００７３】

１．１．４　属性値テーブルＣｉの実装

【００７４】

【数４】

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

【００７５】

ここで、mkeyはＨＰＭＤ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;
```

【００７７】

１．２　タップルのエンコーディング

タップルを r=<c1,c2,……,cn> として、ｒを <経歴値hmax,座標パターンp> の対にエンコードする。各次元のデータ部 di=CVTi(ci) (1≦ｉ≦n) に対して、hmax は Hi[di>>27] (1≦ｉ≦n) の最大値。d[ ] を di&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); }
```

【００７９】

となる。

【００８０】

１．３　タップルのデコーディング

<経歴値ｈ,座標パターンｐ> の対にエンコードされたタップルから、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)}
```

【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】

タップルの挿入（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]] = c[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
}
```

【００８５】

【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;
}
```

【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、タッ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。

【００８７】

１．５　検索

ｎ次元のＨＰＭＤの論理拡張可能配列について、タップルの次元ｉのある属性値ａｖ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】

［タップルの次元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であるタッ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。］

この性質を図２の２次元ＨＰＭＤについて、図５に図示する。図５は、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、タップルのある属性について与えられた属性値を持つタップルを検索する手続きを示す。タップルの検索（ＨＰＭＤ）のプログラムの一例を示せば次の通りである。〔数１０〕はＲＤＴ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ップル）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。

【００９０】

【数１０】

```
#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ッ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、タップルのアドレス空間を格段に拡げて、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におけるタッ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ではタップルは <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節の説明はタッ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　タップルエンコーディング

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

【００９６】

(a) c_j (j=1,…,n) を次元ｊのＣＶＴｊにより次元ｊの添字 i_j に変換

(b)タップル座標 $<i_1,i_2,\cdots\cdots,i_n>$ からチャンク座標 $dp=<p_1,p_2,\cdots\cdots,p_n>$ への変換

$p_j = i_j >> k$ (j=1,…,n) ： i_j をｋビット右シフト

(c)タップルの経歴値ｈを求める

各次元の添字 $i_j = CVTj(c_j)$ (1≦j≦n)に対して、

経歴値ｈは $Hj[i_j>>27]$ (1≦j≦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】

【数１３】

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-(1<<H[5].v[1].sz) =5-1<<2=1

h=5 の境界ベクトル：<2,3>

1&MASK_B<<H[5].v[0].ppos = 1<<3=8

4&MASK_B<<H[5].v[1].ppos = 4<<0=4

cn=8+4=12;

cn=12+1<<5=12+32=44

【０１０２】

(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。

【０１０３】

【数１４】

```
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;

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}
}
```

【０１０４】

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

【０１０５】

【数１５】

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

【０１０６】

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

【０１０７】

【数１６】

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

【０１０８】

(h)タップルのエンコーディング

<cn, cp>

２．２　タップルへのデコーディング

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

【０１０９】

（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。

【０１１０】

【数１７】

```
#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); }
```

【０１１１】

（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。

【０１１２】

【数１８】

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

【０１１３】

(タップルへのデコーディング)

元のタップルの次元ｉの座標は(dp[ｉ]<<k)|dq[ｉ] となる。

【０１１４】

〔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４４,チャンク内座標パターン５５>のデコーディング。

４４のビットパターンサイズは６。したがって、当該タッ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>。

【０１１５】

【数１９】

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

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

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

タップルの座標

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

【０１１６】

（経歴値ｈのチャンク部分配列の下限座標と上限座標）

【０１１７】

【数２０】

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

【０１１８】

２．３　ＲＤＴの構成

タッ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、タッ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。

【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】

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ップルを検索する（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。当該タッ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。タップルの検索（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。

【０１２３】

【数２２】

```
#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);
}
```

【０１２４】

【数２３】

```
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;              //  チャンクにタップルか存在するかどうかのフラッグ

     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);          //  タッ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)
}
```

【０１２５】

〔数２２〕に示すsearch_tuple()は、指定された次元ｉの属性値を有するタップルを検索する。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。

【０１２６】

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、検索結果のタップルを出力する〔数２３〕の再帰ルーチンsearch_chunks()を呼び出す。

【０１２７】

さらに、〔数２３〕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。

【０１２８】

(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。

【０１２９】

(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。

【０１３０】
(S3)渡されたチャンク座標ｗｃ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ップルが存在するかどうかを調べるために、ビットマップＣＢＭ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、ＲＤＴ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ップルを出力する。その後、ｗｃを次のチャンクを指示するように更新する。
【０１３１】
(S4)座標値wc[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。
【０１３２】
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ＣＢＭのサイズが減少する。しかし、チャンク内のオフセット（タップル）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ッ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、ＣＢＭとＲＤＴの再編成を必要とするため、実時間処理を阻害する。
【０１３３】
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・パターン法のタッ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、ＰＣＴ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、タッ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次元タップルの各次元の属性値はＣＶＴ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 <h,o> に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 <h,p> にエンコードされる。
【０１３４】
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ｈｐ法、経歴・オフセット法をｈｏ法と表記して、タップルの次元数をｎとする。
【０１３５】
(a)記憶領域の消費
ｈｐ法の方が消費が少ない。ｈｏ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：$O(n^2)$）が不要。また、ｈｏ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、ｈｐ法では１バイトで十分（2^{256}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、タップルのエンコード値を収容するＲＤＴのサイズが減少し、記憶領域の消費抑制に貢献できる。
【０１３６】
(b)論理空間の消費
ｈｐ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、同じタップル数に対して、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、ｈｏ法では論理拡張可能配列の論理サイズと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

ｈｏ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、ｈｐ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ｈｐ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ｈｏ法ではｎ－１次元であるのに対して、ｈｐ法ではｎ次元である。したがって、タップルの検索範囲はｈｐ法の方が大きく、この点ではｈｐ法は不利である。

【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－ｈｏ法、ｃ－ｈｐ法と表記して、ｃ－ｈｏ法、ｃ－ｈｐ法も含めて比較する。

【０１４０】

(a')記憶領域の消費

ｃ－ｈｏ法、ｃ－ｈｐ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（Ｈｉ）が確保されるため、これらのテーブル領域サイズのｈｏ法およびｈｐ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。ＣＶＴｉ、ＲＤＴおよび他のデータ構造サイズはｈｏ法およびｈｐ法の場合とほぼ同一である。ｃ－ｈｏ法とｃ－ｈｐ法の比較では、ｃ－ｈｐ法でも係数テーブルが不要な分ｃ－ｈｏ法より有利である。しかし、ｃ－ｈｏ法およびｃ－ｈｐ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。

【０１４１】

(b')論理空間の消費

上記(b)のｈｏ法とｈｐ法の比較がそのまま、ｃ－ｈｏ法、ｃ－ｈｐ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－ｈｐ法はｃ－ｈｏ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。

【０１４２】

(c')タップル挿入・検索速度

ｈｏ法とｈｐ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－ｈｏ法、ｃ－ｈｐ法についても当てはまる。ｈｐ法はｈｏ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－ｈｏ法とｃ－ｈｐ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－ｈｏ法でもｃ－ｈｐ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、ｈｏ法とｈｐ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、ｈｐ法の性質はそのまま、ｈｏ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検索速度はほぼ一定である。

【０１４３】

〔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.
[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 Higuch, “An Incremental Maintenance Scheme of Data Cubes”, Proc. of Int'l Conference on Database Systems for Advanced Applications (DASFAA 2008), pp.172-187, 2008.
[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, “ＭＯＬＡＰ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。

【０１４４】

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）１０、補助テーブル部（データベース記憶部）２０、テーブル管理部３０、入出力部４０を備えて構成されている。

【０１４５】

データ格納部１０は、ディスク装置２上に、ＣＶＴ（第１データ、第１のＢ＋木データ）１１、ＲＤＴ（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）１２、および各種補助テーブル（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）を格納している。

【０１４６】

ＲＤＴ１２は、関係テーブルの各タッ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、当該タッ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。

【０１４７】

（１）ＨＰＭＤ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、ＲＤＴ１２は、関係テーブルのタッ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（区画位置情報）、および、当該タッ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。
【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。
【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ップル数を記録している。
【０１５４】
テーブル管理部３０は、タップル検索部（タップル検索手段）３１、タップル挿入部（タップル挿入手段）３２、タップル削除部（タップル削除手段）３３を備えている。
【０１５５】
タップル検索部３１は、タップルの検索の処理を行う。タップル挿入部３２は、タップルの挿入の処理を行う。タップル削除部３３は、タップルの削除の処理を行う。特に、タップル挿入部３２、タップル削除部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。
【０１５６】
具体的には、タップル検索部３１は、ある属性値を持つタッ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ップルを抽出する。
【０１５７】
ここで、入出力部４０を介してユーザから得た検索要求は、タップル検索部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ッ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ップルに逆変換する。具体的には、タップル検索部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、属性順に並べてタップルを得る。

【０１５８】

また、タップル挿入部３２は、新たな属性値を持つタッ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ＣＶＴ１１に登録するとともに、当該タッ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ＲＤＴ１２へ挿入する。

【０１５９】

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

【０１６０】

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、タップル検索部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。

【０１６１】

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。

【０１６２】

（２）Ｃ－ＨＰＭＤ法

なお、２節で説明したＣ－ＨＰＭＤ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。

【０１６３】

区画がチャンク化拡張可能配列の部分配列である場合、ＲＤＴ１２は、関係テーブルのタッ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、当該タッ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。

【０１６４】

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。

【０１６５】

なお、Ｃ－ＨＰＭＤ法でもＨＰＭＤ法と同様、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、Ｃ－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。

【０１６６】

属性毎経歴値テーブル２１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２２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２３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ップル数を記録している。

【０１６７】

ここで、Ｃ－ＨＰＭＤ法の場合、主メモリ３上のテーブル管理部３０には、ＣＢＭ３５が保持される。ＣＢＭ３５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、当該チャンクにタッ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。

【０１６８】

また、タップル検索部３１は、ある属性値を持つタッ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、ＲＤＴ１２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ップルを抽出する。タップル検索部３１は、指定されたチャンク番号のチャンクを検索する際、ＣＢＭ３５の当該ビットを調べ、“１”の時にのみＲＤＴ１２を検索する。

【０１６９】

ここで、キー値－属性値逆変換部３４は、ＨＰＭＤ法の場合と同様に、タップル検索部３１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、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、属性順に並べてタップルを得る。

【０１７０】

また、タップル挿入部３２は、新たな属性値を持つタッ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２１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２３およびＣＶＴ１１に登録するとともに、当該タッ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ＲＤＴ１２へ挿入する。

【０１７１】

ここで、Ｃ－ＨＰＭＤの場合、ＲＤＴ１２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２１にはＨＰＭＤの場合と同様に経歴値を登録する。Ｃ－ＨＰＭＤの場合、区画の単位はチャンクであり、区画中のタップルの同定はＲＤＴ１２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２１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、ＨＰＭＤにおける区画（拡張可能配列の部分配列）中のタップル同定の方法が、タッ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。

【０１７２】

経歴値を属性毎経歴値テーブル２１に書き込む操作は、ＨＰＭＤでも、Ｃ－ＨＰＭＤでも同様である。ＨＰＭＤとＣ－ＨＰＭＤ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、ＨＰＭＤの場合にはタップルに１対１に対応する配列要素が拡張の基本要素であり、Ｃ－ＨＰＭＤ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、データウエ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。

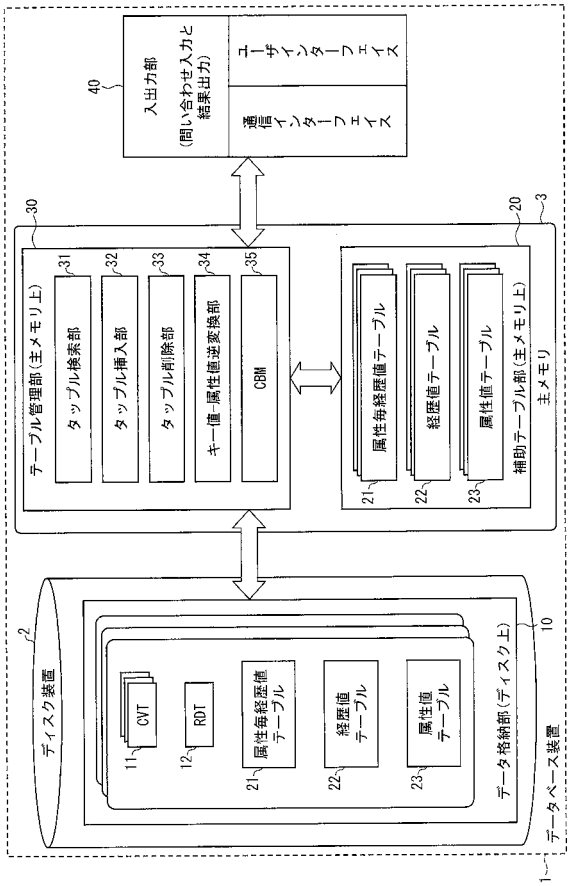
【符号の説明】

【０１７９】

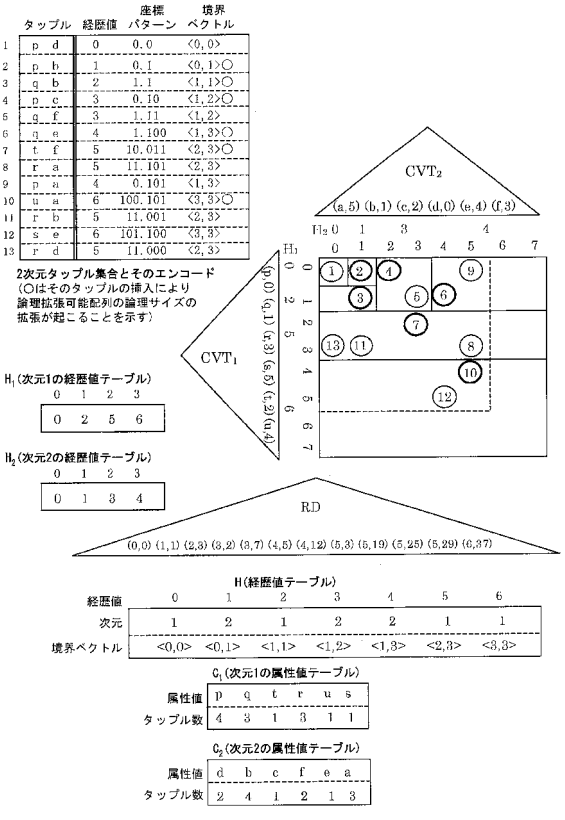
１　データベース装置
２　ディスク装置
３　主メモリ
１０　データ格納部（データベース記憶部）

１１　ＣＶＴ（第１データ、第１のＢ＋木データ）
１２　ＲＤＴ（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　タップル検索部（タップル検索手段）
３２　タップル挿入部（タップル挿入手段）
３３　タップル削除部（タップル削除手段）
３４　キー値－属性値逆変換部（キー値－属性値逆変換手段）
３５　ＣＢＭ

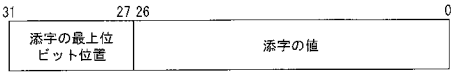
【図１】



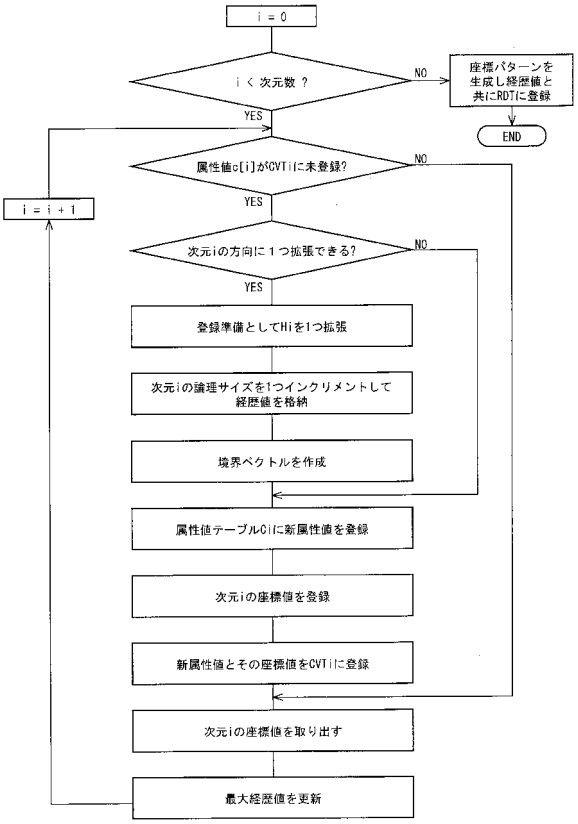
【図２】



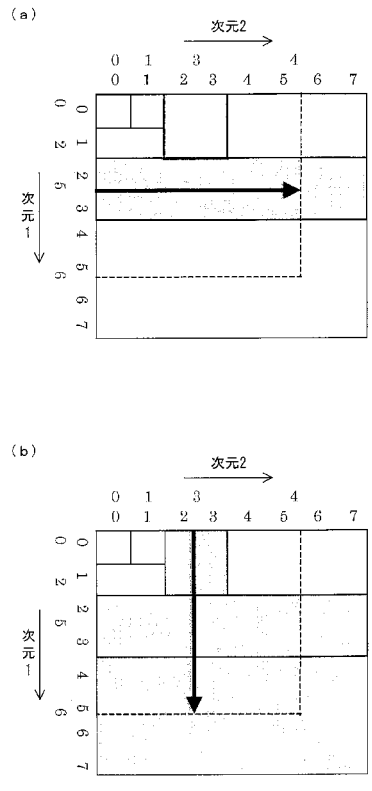
【図３】


31
27 26
0
添字の最上位ビット位置
添字の値

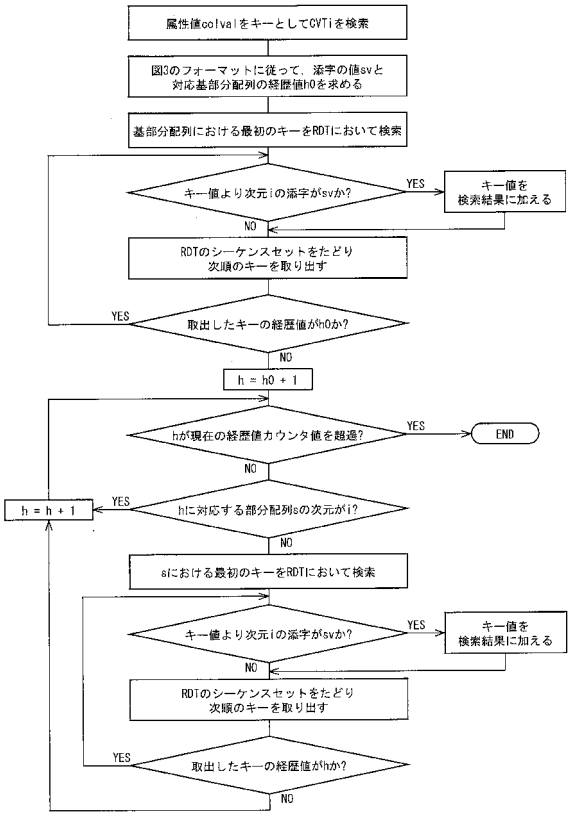
【図４】


i = 0
i < 次元数 ?
NO
座標パターンを生成し経歴値と共にRDTに登録
END
YES
属性値c[i]がCVTiに未登録?
NO
YES
i = i + 1
次元iの方向に１つ拡張できる?
NO
YES
登録準備としてHiを1つ拡張
次元iの論理サイズを1つインクリメントして経歴値を格納
境界ベクトルを作成
属性値テーブルCiに新属性値を登録
次元iの座標値を登録
新属性値とその座標値をCVTiに登録
次元iの座標値を取り出す
最大経歴値を更新

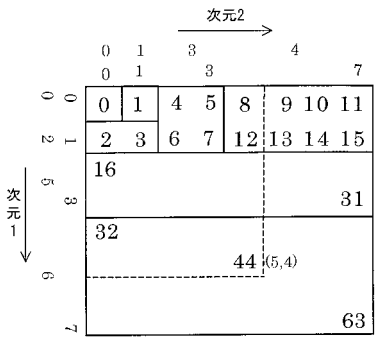
【図５】


(a)
次元2
0 1 3 4
0 1 2 3 4 5 6 7
次元1
0 2 5 6
0 1 2 3 4 5 6 7
(b)
次元2
0 1 3 4
0 1 2 3 4 5 6 7
次元1
0 2 5 6
0 1 2 3 4 5 6 7

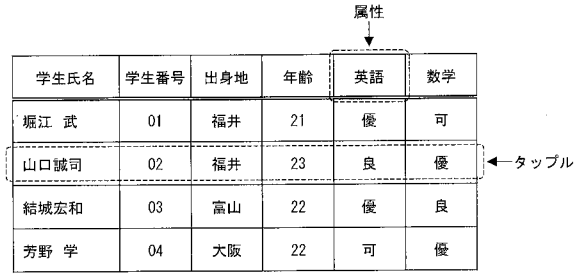
【図６】


属性値colvalをキーとしてCVTiを検索
図3のフォーマットに従って、添字の値svと対応基部分配列の経歴値h0を求める
基部分配列における最初のキーをRDTにおいて検索
キー値より次元iの添字がsvか?
YES
キー値を検索結果に加える
NO
RDTのシーケンスセットをたどり次順のキーを取り出す
YES
取出したキーの経歴値がh0か?
NO
h = h0 + 1
hが現在の経歴値カウンタ値を超過?
YES
END
NO
YES
h = h + 1
hに対応する部分配列sの次元がi?
NO
sにおける最初のキーをRDTにおいて検索
キー値より次元iの添字がsvか?
YES
キー値を検索結果に加える
NO
RDTのシーケンスセットをたどり次順のキーを取り出す
YES
取出したキーの経歴値がhか?
NO

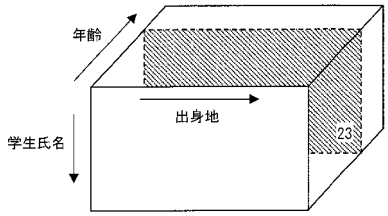
【図７】


次元2
次元1
0 1 3 4 7
0 1 3 7
0 1 4 5 8 9 10 11
2 3 6 7 12 13 14 15
16
31
32
44 (5,4)
63

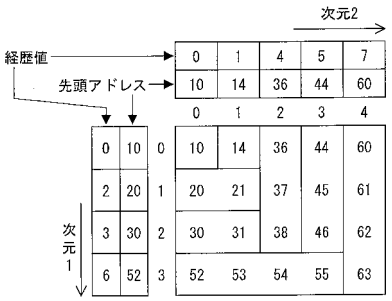
【図８】


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

【図９】


年齢
出身地
学生氏名
23

【図１０】


次元2
経歴値
先頭アドレス
次元1
0 1 4 5 7
10 14 36 44 60
0 1 2 3 4
0 10 0
2 20 1
3 30 2
6 52 3
10 14 36 44 60
20 21 37 45 61
30 31 38 46 62
52 53 54 55 63

フロントページの続き

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０６Ｆ　１２／００
Ｇ０６Ｆ　１７／３０