

(19) 日本国特許庁(JP)

(12) 特 許 公 報(B2)

(11) 特許番号

特許第4810785号

(P4810785)

(45) 発行日 平成23年11月9日(2011.11.9)

(24) 登録日 平成23年9月2日(2011.9.2)

(51) Int.Cl. F I
GO 6 F 12/00 (2006.01) GO 6 F 12/00 5 2 O A
GO 6 F 17/30 (2006.01) GO 6 F 17/30 4 1 4 Z

請求項の数 46 (全 47 頁)

(21) 出願番号	特願2002-547038 (P2002-547038)	(73) 特許権者	511193282
(86) (22) 出願日	平成13年11月28日(2001.11.28)		バーナード コンサルティング リミテッ ド
(65) 公表番号	特表2004-527813 (P2004-527813A)		イギリス領 バージン アイランド トル トラ ロード タウン ピーオー ボック ス 3 4 6 9 ウォーターフロント ドラ イブ ジュネーブ プレイス
(43) 公表日	平成16年9月9日(2004.9.9)		
(86) 国際出願番号	PCT/GB2001/005242	(74) 代理人	100100549
(87) 国際公開番号	W02002/044940		弁理士 川口 嘉之
(87) 国際公開日	平成14年6月6日(2002.6.6)	(74) 代理人	100090516
審査請求日	平成16年11月29日(2004.11.29)		弁理士 松倉 秀実
審判番号	不服2008-21738 (P2008-21738/J1)	(74) 代理人	100123098
審判請求日	平成20年8月25日(2008.8.25)		弁理士 今堀 克彦
(31) 優先権主張番号	0029238.3		
(32) 優先日	平成12年11月30日(2000.11.30)		
(33) 優先権主張国	英国 (GB)		

最終頁に続く

(54) 【発明の名称】 データベース

(57) 【特許請求の範囲】

【請求項 1】

使用時にインデックスおよびデータを備えるデータベースを編成する方法であって、前記インデックスは、複数のビットにより表現される少なくとも1つの記号を含む探索キーを用いて問合せされて、探索基準に合致するデータを見つけるようになっており、

前記インデックスは、探索中に、前記探索基準を満たし、探索キーに対応するデータ又は探索キーに対応するデータへのポイントを保持する結論セットに到達するまで探索される複数のノードの階層的構造体であり、前記インデックスは、各ノードにおける探索キーの決定群が該ノードの決定値より大きいか、又は小さいかを決定するための、各ノードにおける探索キーの決定群と該ノードの決定値との比較によって探索され、各ノードにおいて任意の値に設定可能である前記決定値は、前記探索キーのビット数より少ない複数のビットを含み、該決定群内のビット数が単一ビットと前記探索キー内の全ビットの間の任意のビット数を含むように設定可能である各ノードにおける決定群は、少なくとも1つのビットを含み、幾つかの決定群は前記探索キーのビット数より少ない複数のビットを含み、該階層的構造体は、前記記号が該階層的構造体内のノードに記憶されず、かつ各ノードが該ノードから最大2個の出口パスを有するように編成されることを特徴とする方法。

【請求項 2】

前記ノードは、該ノードに関連する探索キーの全体を記憶せず、前記決定値を有することを特徴とする、請求項1に記載の方法。

【請求項 3】

各ノードにおける決定は、問合せに使用される前記キーまたは前記キーの一部と前記ノードにおける決定値とを比較し、前記キーまたは前記キーの一部が前記決定値より大きいかまたは小さいかを決定することを含み、各ノードから最大2個の出口パスがある、請求項1又は2に記載の方法。

【請求項4】

前記データベース内のあらゆるレコードが複数の結論セットのいずれか一つに属する、請求項1ないし3のいずれか一項に記載の方法。

【請求項5】

使用時に、前記複数のノードの階層的構造体は、データプロセッサの電子メモリ内に保持される決定グラフを形成する、請求項1ないし4のいずれか一項に記載の方法。

10

【請求項6】

各結論セットが前記探索キーに対応するデータ又は探索キーに対応するデータのポイントを保持可能なサイズは最大サイズを有する、請求項4に記載の方法。

【請求項7】

結論セットの前記保持可能なサイズは、データ記憶デバイスで使用されるデータまたはアドレスブロックのサイズと同じか、またはその整数倍である、請求項6に記載の方法。

【請求項8】

結論セットに保持された前記探索キーに対応するデータ又は探索キーに対応するデータへのポイントのサイズが所定サイズに近づくとき、新しいノードが前記複数のノードの階層的構造体に挿入され、前記結論セット内にあったデータは、前記新しいノードの出口パスを介して到達される新しい結論セット内に再インデックス付けされる、請求項6に記載の方法。

20

【請求項9】

前記新しいノードは、前記所定サイズに達した結論セットを指していたノードからの出口パスに挿入される、請求項8に記載の方法。

【請求項10】

前記データベースの構造的再編成は、一度に1個または2個のノードにのみ影響する、請求項1ないし9のいずれか一項に記載の方法。

【請求項11】

前記ノードは、前記探索キーの一部に関係付けられる、請求項1ないし10のいずれか一項に記載の方法。

30

【請求項12】

前記複数のノードの階層的構造体は、キーシーケンスの意味論的順序が保存されるように構造化される、請求項1ないし11のいずれか一項に記載の方法。

【請求項13】

前記結論セット内の各エントリは、前記結論セット内の次の項目を指すリンクフィールドと、前記結論セット内のエントリに関連する探索キーの厳密な値を保持するキーフィールドと、前記探索キーに対応するデータまたは前記探索キーに対応するデータを指すポイントの一方を保持する属性フィールドとを備える、請求項4に記載の方法。

【請求項14】

40

各ノードは、使用時に探索キー内の決定群と比較される関連する決定値を有し、前記比較の結果は前記ノードからの出口パスを選択するために使用される、請求項1ないし13のいずれか一項に記載の方法。

【請求項15】

前記決定値は、

決定群のすべての可能な値の数学的メジアン、

決定群の期待値の数学的メジアン、

決定群のすべての期待値の数学的平均、

前記インデックスの作成時に選択される所定値、ならびに、

いちばん最近に使用された決定値およびそれに先行する決定値の関数として選択される

50

値

のうちの1つから選択される、請求項14に記載の方法。

【請求項16】

訪問されたノードにおける決定群の選択は、
すべてのノードについて同じである決定群、
ノードへの訪問ごとに変化する決定群、

先行するノードにおける決定値が現在の決定群の決定範囲最大値または決定範囲最小値に達するときに変化する決定群、および、

連続するノード間で前記決定値が所定しきい値未満だけ変化するときに変化する決定群のうちの1つから選択される、請求項14または15に記載の方法。

10

【請求項17】

データを追加するとき、前記インデックスは結論セットに到達するまで、または結論セットを作成する必要性が確認されるまで探索され、続いてデータが前記結論セットに追加される、請求項1ないし16のいずれか一項に記載の方法。

【請求項18】

前記ノードのうちの1つまたは複数は、部分的なキーシーケンスを含む、請求項1ないし17のいずれか一項に記載の方法。

【請求項19】

前記結論セットに記憶される探索キーは圧縮形態で記憶される、請求項13に記載の方法。

20

【請求項20】

前記圧縮は、前記探索キーを前記キーのより小さい表現にマッピングする、請求項19に記載の方法。

【請求項21】

前記マッピングは、前記マッピングが所与の探索キーに適用されるときに常に同じ結果を返すように良好に挙動する、請求項20に記載の方法。

【請求項22】

前記探索キーは、少なくとも2つの異なるアルゴリズムを用いて圧縮される、請求項19、20又は21に記載の方法。

【請求項23】

30

前記少なくとも2つの圧縮の結果は、前記結論セットに記憶される、請求項22に記載の方法。

【請求項24】

探索キーは、限定された範囲に該探索キーをマッピングする演算子によって処理される、請求項1ないし23のいずれか一項に記載の方法。

【請求項25】

前記演算子は、探索キーを前記データベース全体にわたり実質的に一様に分配する、請求項24に記載の方法。

【請求項26】

前記演算子は、モジュロ関数およびハッシュ関数のうちの1つから選択される、請求項24又は25に記載の方法。

40

【請求項27】

前記結論セット内に記憶される探索キーは、該探索キーおよび該キーデータが前記データベース内に維持されるべき持続期間を示すデータフィールドを含む、請求項1ないし26のいずれか一項に記載の方法。

【請求項28】

期限切れになった探索キーおよびデータは、新しいデータで上書きするために利用可能となる、請求項27に記載の方法。

【請求項29】

期限切れの探索キーおよびデータは、前記データベースから能動的には削除されない、

50

請求項 27 または 28 に記載の方法。

【請求項 30】

結論セットがアクセスされるとき、持続期間を有する各エントリのエイジが計算され、該計算の結果が、どのエントリが上書き可能かを判定するために記憶され、または使用されることが可能である、請求項 27、28 または 29 に記載の方法。

【請求項 31】

前記データベースは、キーインデックスおよび決定インデックスを備え、前記キーインデックス内のキーは圧縮された方式で記憶され、前記探索キーを用いて前記決定インデックスに質問する前に前記探索キーが存在するかどうかを確かめるチェックが前記キーインデックスに対して行われる、請求項 1 ないし 30 のいずれか一項に記載の方法。

10

【請求項 32】

前記探索キーが前記キーインデックス内に存在しない場合、前記決定インデックスは探索されない、請求項 31 に記載の方法。

【請求項 33】

探索キーを挿入する操作中に、前記キーインデックスのチェックが行われ、合致するエントリが見出される場合、前記探索キーは重複として拒否される、請求項 31 または 32 に記載の方法。

【請求項 34】

前記キーインデックス内の探索キーは、圧縮、符号化またはマッピングされた形式で記憶される、請求項 31 ないし 33 のいずれか一項に記載の方法。

20

【請求項 35】

前記探索キーはハッシュ関数を用いてマッピングされる、請求項 34 に記載の方法。

【請求項 36】

前記キーインデックスは要素配列を備え、該配列内の要素の数は、前記インデックス内の一意的なキー要素の数と少なくとも同数である、請求項 31 ないし 35 のいずれか一項に記載の方法。

【請求項 37】

2つの異なる関数を使用され、一方のハッシュ関数は、前記キーインデックス内の宛先要素 E (K) を計算するプロセスの一部として使用され、他方のハッシュ関数は、前記探索キーを符号化するために使用され、その結果 B (K) は前記宛先要素に記憶される、請求項 36 に記載の方法。

30

【請求項 38】

前記キーインデックスの問合せ中に、前記宛先要素が目標探索キーについて計算され、前記第 2 のハッシュ関数を用いて符号化された値が前記目標探索キーについて計算され、前記符号化された値が前記宛先要素に既存の値に合致する場合に、前記探索キーは重複であるとみなされる、請求項 37 に記載の方法。

【請求項 39】

前記インデックスは、インデックスがその作業負荷の一部を少なくとも 1 つの他のインデックスに代行させることができるように、インデックスの階層的構造体を備える、請求項 1 ないし 38 のいずれか一項に記載の方法。

40

【請求項 40】

作業を代行しているインデックスは、作業を他のインデックスに代行させることができる、請求項 39 に記載の方法。

【請求項 41】

インデックスのキー目録は、範囲によって定義され、前記範囲の外部の探索キーに対して操作するいかなる要求も前記インデックスにより拒否される、請求項 39 または 40 に記載の方法。

【請求項 42】

前記探索キーはデータベースのインデックス範囲内にあり、前記データベースは、前記探索キーがいずれかの代行インデックスの目録範囲内にある場合、前記探索キーに対する

50

操作を自分自身で実行すること、またはそのタスクを代行させることが可能である、請求項 4 1 に記載の方法。

【請求項 4 3】

各インデックスは、物理記憶デバイスに関連付けられることにより、複数の並行ディスクアクセスが起こることを可能にする、請求項 3 9 ないし 4 2 のいずれか一項に記載の方法。

【請求項 4 4】

請求項 1 ないし 4 3 のいずれか一項に記載の方法をコンピュータに実行させるコンピュータ可読命令。

【請求項 4 5】

前記命令はデータキャリア上を運ばれる、請求項 4 4 に記載のコンピュータ可読命令。

【請求項 4 6】

使用時にインデックスおよびデータを備えるデータベースであって、前記インデックスは、探索基準に合致するデータを見つけるために、複数のビットにより表現される少なくとも 1 つの記号を含む探索キーを用いて問合せされ、

前記インデックスは、探索中に前記探索基準を満たし、探索キーに対応するデータ又は探索キーに対応するデータへのポインタを保持する結論セットに到達するまで探索される複数のノードの階層的構造体であり、前記インデックスは、各ノードにおける探索キーの決定群が該ノードの決定値より大きいか、又は小さいかを決定するための、各ノードにおける探索キーの決定群と該ノードの決定値との比較によって探索され、各ノードにおいて任意の値に設定可能である前記決定値は前記探索キーのビット数より少ない複数のビットを含み、該決定群内のビット数が単一ビットと前記探索キー内の全ビットの間の任意のビット数を含むように設定可能である各ノードにおける決定群は少なくとも 1 つのビットを含み、幾つかの決定群は前記探索キーのビット数より少ない複数のビットを含み、該階層的構造体は、前記記号が前記階層的構造体内のノードに記憶されず、かつ各ノードが該ノードから最大 2 個の出口パスを有するように編成されることを特徴とするデータベース。

【発明の詳細な説明】

【0001】

本発明は、データベースをインデックス付けし、データベース内に情報を配置し、データベースに問合せを行う方法に関する。特に、本発明はデータ記憶システム内のスカラデータの効率的探索を可能にする。

【0002】

スカラデータタイプとしては、例えば、論理データ、テキストストリング、数値データ、および日時データがある。

【0003】

データプロセッサは高速であるが、指定された属性またはある範囲の属性を有する特定の項目を見つけるために、記憶システムに保持されている全データを探索するのは非効率的であり時間がかかる。また、データは大容量記憶媒体に記憶されることになるのがほとんど不可避であり、これは、CPU 速度および半導体メモリへのアクセスに比べてディスクアクセスに起因する遅延を生じる。探索キーを用いて項目を検索することができるように、インデックスを作成する方がより効率的である。

【0004】

インデックス機構は、内部インデックス構造体の作成および保守を必要とする。この構造体のナビゲーションおよび保守は、本来的に処理オーバーヘッドを被る。このオーバーヘッドは、使用されるインデックス付けメカニズムの関数として変動し得る。このオーバーヘッドはまた、データベースのサイズが変化するにつれて、または例えばデータベース内のデータが時間とともに変化するにつれて変動し得る。

【0005】

簡単な例は、誕生日のような時間順の情報を記憶するデータベースの場合である。設計者は、例えば、西暦 2000 年より前に生まれた人々に関するデータが西暦 2000 年よ

10

20

30

40

50

り後に生まれた人々に関係するデータから分離されるようにデータを構造化するかもしれない。この戦略はしばらくはうまくいくかもしれないが、理解されるように、西暦2000年より前に生まれた人の数は制約されたセットであるのに対して、この日の後に生まれた人数は無限定である。したがって、時間が経つにつれて、西暦2000年より後に生まれた人々に関係するエントリの割合が増大すると、このインデックス付けキーは冗長になる可能性がある。

【0006】

インデックス付け方法の有効性は、以下のようないくつかの属性に照らして判断され得る：

【0007】

1) インデックスのサイズ。インデックスが大きいほど、ナビゲートするのにより長い時間を必要とする。大きいインデックスはまた、ハードディスクドライブのような大容量記憶媒体、および半導体メモリのようなより高速なメモリからの転送に参与する多数のデータスワップまたは読み出し操作も必要とすることがある。

【0008】

2) インデックスの構造的制約。強く制約された構造は、最初は利点を提供するかもしれないが、データベースがより多くのデータを収容するように増大するにつれてインデックスを再構築することが要求される場合には、計算のオーバーヘッドが増大してしまう。

【0009】

3) インデックスにより課されるキーデータ制限。つまり、キーの最大サイズもしくはキーのデータタイプまたはキーにより表現される範囲は、インデックスが増大するにつれて、またはデータベースのデータ数が多くなるにつれて、インデックスのパフォーマンスに影響し始めることがある。

【0010】

4) インデックスにより課されるキー検索制限。いくつかのインデックスは厳密なキーマッチングしか許容しないのに対して、他のインデックスは範囲マッチングを許容する。

【0011】

5) インデックスの増大限界。インデックスは、最大サイズにより制約されることがある。

【0012】

6) 同時処理限界。インデックスは、キーデータの同時挿入および検索を許容することも禁止することもある。

【0013】

知られている従来技術のシステムは通常、ノードがキーデータの1つまたは複数の項目を含むようなツリーグラフを使用する。データベースに質問するためには、訪問される各ノードにあるキー値を、要求される探索キーと比較することによりツリーはナビゲートされる。このような設計においては本質的に、各ノードがそのノードの作成に関連するキーの全体を含むため、ノードの「サイズ」がデータ記憶領域に関して相当大きくなることがある。

【0014】

このようなインデックスの効率は、キーサイズと、データがインデックスに追加される順序に左右され得る。

【0015】

さらに、いくつかの方法は、それが適正に動作するためには常にツリーが「均衡化」されていることを要求する。これらの方法は、キーの挿入および削除のためにかかなりの保守オーバーヘッドを被ることがあるため、高スループットのシステムには不適當である。

【0016】

ナビゲーション

本発明の第1の態様によれば、インデックスおよびデータを有するデータベースを編成する方法が提供される。この方法において、インデックスは、探索基準に合致するデータを

10

20

30

40

50

見つけるために、複数のビットにより表現される少なくとも1つの記号を含む探索キーを用いて問合せされる。この方法は以下のことを特徴とする。すなわち、インデックスは決定ノードの階層的構造体であり、ノードの構造体は探索中に結論に到達するまでナビゲートされ、構造体は、キー記号が構造体内のノードに記憶されず、かつ各ノードが3個より少ない出口パスを有するように編成される。

【0017】

こうして、データベースの効率的な探索および更新を可能にする効率的なインデックス構造体を提供することが可能となる。ノードが当該ノードに関連するキーの全体を記憶するのではないため、インデックス構造体は従来技術のインデックスよりもずっと小さい。実際、ノードは当該ノードに関連するいかなるキーも記憶する必要がない。さらに、ノードは、ノードの質問に関係する決定規則を記憶しない。これは、規則が、ノード固有のものではなくすべてのノードにより共有されるという点で大域的であるためである。

10

【0018】

各決定ノードは二者択一を行う。すなわち、各ノードは最大2個の出口パスしか有さない。

【0019】

好ましくは、各ノードで適用される決定は単純な問合せである。これは、問合せが、キーまたはキーの一部を、ノードにおける決定基準と比較することを意味し、2進数的に言えば、テストは通常（必ずしもこれに限るわけではないが）、テストされているキーまたはその一部が、決定基準より大きいのかそれとも小さいかというテストであることを意味する。これは、インデックスの構造体自体の内部に探索キーの特徴を記憶する必要性を除去する。

20

【0020】

ノードからの各出口は、別の決定ノード、結論セットまたはヌル結果を指すことが可能である。

【0021】

ノードからの出口が別のノードを指す場合、探索は継続する。しかし、ノードからの出口が結論セットを指す場合、インデックスのナビゲーションは一般に終了する。

【0022】

どの結果も探索項目と合致しない可能性もあり、その場合、探索はヌル結果で終了する。

30

【0023】

データベース内のあらゆる「レコード」は、ある結論セットに属する。結論セット自体は、特定の探索基準を満たすデータを指すか、または場合によっては、データ自体を記憶することがある。

【0024】

決定ノードの階層的構造体は、決定グラフとみなし得る。決定グラフは、任意のサイズおよび内部構造のものでよい。しかし、決定グラフは一般に、そのサイズに関して、プロセッサのメモリ内の小さいエンティティである。

【0025】

好ましくは、決定グラフは、使用時に、データプロセッサの電子メモリ内に保持される。比較的小さい決定グラフをRAM内に保持することにより、データプロセッサは問合せの結果を素早く見つけることができる。特に、決定グラフの解析が磁気または他の物理記憶媒体へのアクセスを必要としないため、このような媒体に関連するアクセス時間はそれにより除去され、それによってパフォーマンスが向上する。

40

【0026】

好ましくは、各結論セットは、決定グラフを通る単一の経路によってのみ到達することができる。これにより、結論セット内のすべてのキーは、共有される一意的な決定のセットに適合することが保証される。

【0027】

データベースインデックスは、時間とともに発展することがあり、そしてむしろ発展しな

50

ければならない。このことは、データベースが増大するにつれてパフォーマンスの劣化につながる可能性がある。したがって、結論セット内でのデータの挿入および削除、ならびに新しい結論セットの作成を可能にする必要がある。

【0028】

好ましくは、各結論セットは最大サイズを有する。この最大サイズは、ユーザまたは設計者により制御されるパラメータである。しかし、サイズは、1つまたは複数のハードディスクを備え得る磁気記憶装置のような記憶デバイスによって使用されるデータブロックまたはアドレッシングブロックのサイズに有利には関係付けられてもよい。しかし、光記憶デバイスのような、いかなるランダムアクセスデータ記憶装置も使用され得る。

【0029】

結論セットが所定サイズに近づくか、または到達すると、新しい決定ノードが決定ツリーに挿入され、結論セット内にあったデータは、その新しい決定ノードの出口パスを介して到達される新しい結論セット内に再インデックス付けされる。新しい決定ノードは、自己の最大サイズを超えてしまった結論セットを指していたノードの出口パスに挿入される。

【0030】

こうして、結論セットは、良好に挙動するように構成される。

【0031】

決定グラフは、いずれの探索キーデータ自体を含まない。すべてのキーデータは、決定グラフの構造から暗示されるか、または演繹される。これは、決定グラフをコンパクトに保つ。またこれは、決定グラフのサイズがキーサイズとは無関係であることも意味する。

【0032】

本発明のもう1つの利点は、決定グラフの構造に変更を加えることが可能であり、これらの変更は局所的であることである。すなわち、ただ1つまたは数個の決定ノードだけがその変更により影響される。これは、構造的再編成があまり計算のオーバーヘッドとはならないことを意味する。

【0033】

好ましくは、決定ノードは、キーの部分に、より好ましくはキーの比較的小さい部分に、関係付けられる。この結果、決定グラフは、キーの構造体とは概ね無関係となる。これはまた、決定グラフが、キーの構造体に対して、あるとしても少ししか制約を課さないという利点も有する。さらに、ノードが関係付けられるキーの部分は、決定グラフがナビゲートされるのと同じ構造化された順序に従う必要はない。したがって、当該キーの部分は、グラフがナビゲートされるとともにキーに沿って単調な方式で進行する必要がない。

【0034】

有利には、決定グラフは、キーシーケンスの意味論的順序が保存されるように構造化される。したがって、例えば、決定グラフは、第1のノードまたはノードのセットが探索キー内の第1のビットまたは第1のビット群をテストし、決定グラフが結論セットへ向かってナビゲートされるにつれて、後続のノードでテストされるキーのビットがますます下位になるように構造化され得る。こうして、決定グラフは、厳密マッチングまたは範囲マッチングのいずれによっても、キーを見つけるためにナビゲートされることが可能である。範囲は、探索中に部分的にまたは完全に限定され得る。

【0035】

決定グラフの構造体、特にその構造体への変更が大域的ではなく1つ、または2つのノードにしか影響しないという点で局所的であることは、決定グラフを保守する計算タスクが概して少ないことを意味する。再編成の局所性は、インデックスを用いたデータの挿入、削除および検索を容易にする。実際、これらのイベントの2つ以上が並行して発生することもある。

【0036】

好ましくは、決定グラフのサイズに機能的限界はない。決定グラフが大量のデータを収容するように増大するにつれて、インデックスの構造体の探索時間は良好な挙動で増大する。

10

20

30

40

50

【0037】

本発明の第2の態様によれば、インデックスを有するデータベースであって、データを保持し、探索キーを用いてインデックスに質問することによりデータを見つけることができるように構成されたデータベースが提供される。キーは少なくとも1つの記号を含み、各記号は複数のビットにより表現される。このデータベースにおいて、インデックスは決定ノードの階層的構造体であり、ノードの構造体は探索中に結論に到達するまでナビゲートされ、各ノードの探索基準はインデックス内のノードの位置により符号化され、各ノードは2個以下の出口パスを有する。

【0038】

インデックスにおける局所的な構造的変更は、インデックス内での並行する挿入、削除および検索イベントを容易にする。

10

【0039】

好ましくは、データベース内のデータの各項目は、ただ1つの結論セットにのみ属し得る。

【0040】

決定グラフの分割

決定グラフは全体として半導体メモリ内に存在することが強く望まれるにもかかわらず、これは常に可能であるとは限らない。それは、インデックスが大きすぎるためかもしれないし、ホストデータプロセッサがデータベースのために「仮想マシン」環境を実施していて、ホストがマルチタスク動作し得るようになっており、オペレーティングシステムがデータベースに十分なメモリリソースを割り当てないためかもしれない。

20

【0041】

このような状況の下では、決定グラフは、コンピュータシステムの大容量記憶デバイス上のメモリのブロック内に保持され得る。ディスクコントローラは、データの不連続なブロックによってディスクを管理する。したがって、例えば、ディスクコントローラにより使用される最小ブロックサイズが64 Kバイトである場合、40 Kのインデックスと60 Kのインデックスは両方ともディスク上で同じ記憶空間、すなわち1ブロックを占有し、同様に65 Kのインデックスと130 Kのインデックスもまた同じ記憶空間、すなわち2ブロックを占有する。

【0042】

これらのシステム制約に対応しその下で作業するためには、インデックス内のキー数が増大するにつれてインデックスのサイズが増大するため、インデックスの変更は制御される必要がある。したがって、インデックスを増大させるには、より多くのメモリのブロックが必要となる。

30

【0043】

新しい決定ノードを挿入することが要求され、かつブロックがフルであるときに、インデックスを増大させる第1の方式は、新しいブロックを作成し、その新しいブロック内に新しいノードを配置することである。決定グラフは、新しいブロックが半導体メモリ内にすでに存在しているのでなければ、新しいノードを指すノードによりそのブロックが磁気記憶装置からフェッチされるように、補正される。その場合、ナビゲーションはその新しいノードから継続する。

40

【0044】

この方法は単純であるが、それぞれの親のフルブロックから、ほとんど空のメモリブロックが多数、おそらくは数百個も作成されることにつながる可能性が高いため、非効率的である。これは、占める空間に関しては、問題とみなされないかもしれない。というのは、ディスクドライブは驚くほど大量のデータを保持することができるからである。しかし、この方法は、インデックスをナビゲートするために必要なディスクI/O操作が多くなるので、インデックスパフォーマンスを損なう可能性がある。

【0045】

1つの好ましい手法は、新しいブロックを作成した後、親ブロック（すなわち、フルプロ

50

ック)を実質的に均等に2つのブロックに分割することである。この手法では、ブロックを最適に二分割するノードを見つけるために、元のブロックが質問される必要がある。これは、ブロック内のノードの最初からの解析から、または性質上より統計的な手法によって、行われるかもしれない。したがって、例えば、分割のためのテストノードを選び、そのノード選択から生じるインデックスの分割を計算し、その結果に基づいてテストノードの選択を修正し、以上をおよそ均等な分配が達成されるまで繰り返す、という再帰的アルゴリズムが使用され得る。

【0046】

この第2の手法は、各インデックスブロックが相当数のノードを含むという利益を有する。これは、大幅により効率的なメモリの使用につながる。

10

【0047】

パターンマッチング

本明細書で前述のように、従来技術のシステムは、所与のノードに関連する探索キーの全体を記憶する。

【0048】

本発明の第1および第2の態様は、ノードが探索キーを記憶せず、キーデータはツリーの構造体に内在するようなツリー構造体を開示した。

【0049】

本発明の第3の態様によれば、インデックスおよびデータを有するデータベースに質問する方法が提供される。この方法において、インデックスは、探索基準に合致するデータを見つけるために探索キーを用いて問合せされる。この方法は以下のことを特徴とする。すなわち、インデックスは決定ノードの階層的構造体であり、構造体は結論セットに到達するまでナビゲートされ、ノードはキーのサブシーケンスを含むことが可能であり、とるべきアクションを定めるために候補キーがそのサブシーケンスと比較されることが可能であり、インデックスのN+1番目のレイヤ内のノードに記憶されるキーの部分は、N番目のレイヤ内の先行するノードに記憶される部分と無関係である。

20

【0050】

本明細書で用いられる場合、(N+1)番目のノードにあるキーの部分は、そのキー部分が名目上異なる方式でキーの連続する部分を選択することによっては計算されないという意味で、N番目のノードにあるキーの部分とは無関係である。

30

【0051】

したがって、各ノードはキー全体ではなくキーのサブシーケンスのみを記憶するので、決定ノードを比較的「小さく」維持することができるようなデータベースを提供することが可能となる。この構造体は、パターンマッチングによるデータベースのナビゲーション、挿入および検索を容易にする。

【0052】

本発明の第4の態様によれば、インデックスおよびデータを有するデータベースが提供される。このデータベースにおいて、インデックスは、探索基準に合致するデータを見つけるために探索キーを用いて問合せされる。この方法は以下のことを特徴とする。すなわち、インデックスは決定ノードの階層的構造体であり、構造体は結論セットに到達するまでナビゲートされ、ノードはキーのサブシーケンスを含むことが可能であり、とるべきアクションを定めるために候補キーがそのサブシーケンスと比較されることが可能である。

40

【0053】

拡張パターンマッチング

パターンマッチングのプロセスにおいて、パターンマッチングにおける計算負荷は、照合されるべきパターンの長さの関数であり、特に、文字または記号の短いストリングに対する照合は、長いストリングを照合するよりも計算量的に安価である。

【0054】

さらに、データベース/インデックスの効率は、キー数が各ノードにどの程度適切に分割されているか、およびシーケンスが問合せ内に現れる可能性がどのくらい高いかにも左右

50

される。

【0055】

このことは、2つのかなり極端な例を考察することによって最も良く例示される。インデックスが、ある企業が通信を交わしたい顧客および他のエンティティの名前によって編成されていると仮定する。

【0056】

頻繁に現れるパターンマッチングシーケンスは、インデックス問合せ中に走査されるキーの数を十分には減少させないであろう。例えば、パターンマッチングで用いられるシーケンスが「E」である場合、これがほとんどのキーに出現する可能性が高いことを考慮すれば、キー数を分割するのにはほとんど役に立たない。

10

【0057】

もう1つの極端な例として、まれに出現するシーケンスもまた、ほとんどの問合せで走査されなければならないキーの数を減少させないであろう。したがって、例えば、「ZZ」は問合せに現れる可能性が低いため、パターンマッチング問合せに対する良い候補ではない。

【0058】

本明細書で前述のように、シーケンスに関与する文字が多くなるほど、そのシーケンスがキーに現れる可能性は低くなる。したがって、単一の文字は多数の文字を有するキーに現れる可能性が高いが、4文字の特定のシーケンスは現れる可能性がずっと低い。

【0059】

理想的な場合には、問合せストリングは、あるノードから到達されるキーの数を実質的に均等に分割するであろう。

20

【0060】

ASCII文字セットの問題点は、セット内の全255文字を用いる適当なシーケンスを見つけるのがしばしば困難なことである。というのは、多くの文字があるけれども、いくつかの文字の現れる可能性は、多くの他の文字の現れる可能性よりずっと高いからである。

【0061】

しかし、2文字の組合せはまれであり得る。

【0062】

本発明の第5の態様によれば、データベースのインデックス内にキーまたはその一部を記憶する方法が提供される。この方法において、キーの要素は、それらの要素が取ることができる値のうちの少なくともいくつかのリストを構成する第1のセットから、その第1のセットより小さい第2のセットにマッピングされる。

30

【0063】

したがって、キー内に出現し得る相異なる文字の数は減少する。マッピングは一意的ではないが、マッピングが適当に選択される場合、第2のセットの任意の単一文字が出現する可能性を適度に高くすることができ、2個以上の文字の組合せがシーケンスのために用いられ得る。

【0064】

好ましくは、マッピングは、挿入、削除、または問合せ操作のための決定グラフのナビゲーション前に、キーに適用される。

40

【0065】

有利には、元のキー値または変換されたキー値（このキー値は、これを記憶するのに要求される空間に関して、より小さい）のいずれが結論セットに記憶されることも可能である。

【0066】

本発明の第6の態様によれば、インデックスナビゲーション前に、縮小された値の範囲にキーがマッピングされるデータベースが提供される。

【0067】

50

キー圧縮

本明細書で前述のように、いくつかの従来技術のデータベースは、各決定ノードに完全な決定／探索キーを記憶する。これは、キーがそれ自体大きくなり得ることを考慮すると、比較的大きいインデックス構造体を生成し得る。同じくすでに述べたように、本発明の実施形態を構成するデータベースでは、探索キーの一部を決定ノードに記憶することは望ましいかもしれないが、探索キーが各決定ノードに記憶される必要はない。探索キーの一部を記憶するという選択からはなお、高速にナビゲートされ得る比較的コンパクトなデータベース決定ツリーが得られる。しかし、本発明者は、キーはそのネイティブな形態でインデックスに記憶される必要はないことを認識した。したがって、キーは、小さい空間を占める形式に符号化されてもよい。

10

【0068】

本発明の第7の態様によれば、データベース内でキーを符号化する方法が提供される。この方法において、キーデータは、キーの記憶空間要件を縮小するように符号化された形態で少なくとも1つの結論セット内に記憶される。

【0069】

したがって、よりコンパクトな結論セットを提供することが可能である。このデータベースは厳密マッチングをサポートする。

【0070】

マッピング関数が良好に挙動すること、すなわち、マッピング関数が特定のキーに適用されるとき、それが常に同じ結果を返すことが重要である。

20

【0071】

しかし、マッピングが一意的であることは必要でない。実際、マッピングアルゴリズムが高度の圧縮を与える場合、マッピングが一意的となる可能性は非常に低くなる。

【0072】

有利には、少なくとも2個の相異なるマッピングアルゴリズムが使用される。相異なるマッピングの使用は、結論セットでの使用に制限され得る。

【0073】

例えばCRC32およびAdler32のような、既知の公表されているマッピングアルゴリズムが、マッピング関数を実施するために使用され得る。

【0074】

好ましくは、マッピング関数は、インデックス内に保持される相異なるキーの数と少なくとも同じサイズの、正の整数（ゼロを含む）の範囲からの値を返すように構成される。したがって、各キーはアルゴリズムにより別の値にマッピングされる可能性を有し、どのキーも長すぎることはない。

30

【0075】

例えば、マッピングは、長さNバイトのスカラから次の関数を用いて計算され得る：

$$[(K^0 \times 31^{(N-1)}) + (K^1 \times 31^{(N-2)}) \cdots + K^{N-1}] \bmod R$$

【0076】

ただし K^N は、キーKの第nバイトの値（0～255）である。Rは、インデックスに保持される相異なるキーの数のサイズより大きい、最小の2のべき乗である。

40

【0077】

推奨されるRの値は以下の通りである：

65000個より少ないキーを含むインデックスの場合、 2^{16} 。

4×10^9 個より少ないキーを含むインデックスの場合、 2^{32} 。

1. 8×10^{19} 個より少ないキーを含むインデックスの場合、 2^{64} 。

【0078】

有利には、結論セットに記憶されるキー情報は、従来技術のシステムに記憶されるような従来のキーであることも可能であるが、好ましくは、キーの別の符号化バージョンであって、適用されるマッピング関数は決定グラフでキーを符号化するために使用される関数と

50

は異なる。

【0079】

修飾子

インデックス問合せは多くのキーヒットにつながることもある。問合せから返されるそれぞれの結果は、大容量データ記憶デバイスからの関連データの検索を引き起こし得る。これは通常ハードディスクであるため、問合せから返されるそれぞれの結果ごとにディスク I/O 操作が要求される可能性がある。

【0080】

フェッチされたすべてのデータが必要であるとは限らないかもしれない。

【0081】

例えば、ある多国籍組織が、ロンドンを拠点とし、かつ特定の給料範囲内にある従業員の数を求めるために、その人員データベースに問合せをしたい場合を考える。居所または給料のみに基づくインデックスは、所与の問合せに対する多くの従業員を返す可能性がある。問合せサイズを縮小するのは、AND された問合せの結果だけである。

【0082】

このタイプの問合せに対して、インデックス内に複数のキーを記憶することによって、データベースインデックスを修正することが知られている。したがって、キーは、合成あるいは複合キーとみなすことができる。ある候補（すなわち、データベース内のエントリ）が2個のキーに合致する可能性は、ある候補が1個のキーにマッチする可能性に比べてずっと小さいため、問合せによって返されるヒットの数は減少し、必要なディスク I/O 操作も少なくなる。

【0083】

キーを互いに連結することによってこのような複合キーを作ることが知られている。簡単に言えば、インデックス作成時に第2の探索基準が第1の探索基準に追加される。しかし、個々のキーはかなり長い可能性もあり、連結キーのインデックスを生成することによりインデックスが実に非常に大きくなってしまう可能性がある。

【0084】

本出願人は、キーデータがその元の形態で記憶される必要はなく、実際には、大幅に縮小された長さのワードすなわちビットシーケンスにより符号化すなわち表現され得ることを認識した。インデックス内の最初のキーと、連結キー内のあらゆる後続のキーは、このような修正された形態で表現され得る。

【0085】

本発明の第8の態様によれば、データベースを編成する方法が提供される。この方法において、インデックスキーは、少なくとも1つのキーを含む複合キーであり、1つまたは複数のキーが、圧縮された形態で表現される。

【0086】

好ましくは、複数のキーが圧縮形態で提供される。

【0087】

好ましくは、キーは、非圧縮形態から圧縮形態にキーをマッピングすることにより圧縮される。このマッピングは一意的である必要はないが、良好に挙動するものでなければならない。適当なマッピングは、ハッシュアルゴリズムの使用により実行され得る。

【0088】

相異なるキーが同じ縮小表現により表現され得るというまさにその事実によって、ある問合せが、その問合せに確かにマッチする結果と、その問合せにマッチしない何らかの結果の両方を返すというリスクが存在する。しかし、良好なランダム化ハッシュ符号化が使用されると仮定すると、1つのキーに関するミスマッチの確率は $1/255$ であるが、N 個の修飾子に対しては、ミスマッチの確率は $(1/255)^N$ となる。

【0089】

したがって、誤り率は一般に小さい。

【0090】

10

20

30

40

50

有利には、返されるすべてのヒットは、データを妥当性検査し、誤って返された結果を除去するように検査される。

【0091】

挿入、削除または検索を実行するとき、新しいキーは、インデックスを構築するために使用されたのと同じキー変換プロセスを用いて圧縮されなければならない。

【0092】

好ましくは、連結キーの最初の要素の後のそれぞれの追加要素は「修飾子」として表現され得る。各修飾子は好ましくは長さが1バイトのみである。

【0093】

したがって、本発明を従来技術と比較すると、それぞれ8バイトからなる8個のキー要素を含む従来技術の連結キーは64バイトを占有する。本発明では、主キーは非圧縮フォーマットで表現されてよく、それにより8バイトを占有するが、7個の追加キーはそれぞれ1バイトの修飾子として表現され得る。したがって全キー長はの場合たった15バイトである。

10

【0094】

修飾子キーはオプションであり、インデックス問合せ中には、それらのキーのうちの一部または全部が提供されることもあり、全く提供されないこともあり得る。というのは、それらのキーはインデックス構造体のナビゲーションに影響しないからである。修飾子キーに対するヒットのテストは、結論セットを走査するときに実行される。

20

【0095】

したがって、理解されるように、本発明のこの態様は、Bツリーのような従来のデータベース構造体とともに使用されることも、本明細書の他の箇所に開示されている構造体とともに使用されることも可能である。

【0096】

追加的に、修飾子は、ノードに関係する情報が、ノード自体により明示的に定義されたりノード自体の内部に含まれたりするのではなく、(本発明の第1の態様の実施形態を構成するインデックスのような)ノードの位置から推論されるようなインデックスとともに使用され得る。

【0097】

本発明の第9の態様によれば、インデックスが少なくとも1つのキーを含み、1つまたは複数のキーが圧縮形態で表現されるデータベースが提供される。

30

【0098】

限定されたキー

本明細書で前述のように、データベースインデックスの効率は、それとともに使用されるキーの特性に大きく左右されることがあり、キー値が連続的に増大または減少する場合に潜在的に非効率的なインデックス付けシステムにつながる。このような無限定キーの例は、現在の時刻とともに絶えず増大する日時キーである。このようなキーがインデックスに追加されると、それにより新しいインデックスの領域が作成される。また、古いキーがインデックスから削除されると、それにより廃用になったインデックスの領域は新しいキー値のために再使用することができないような断片化を生じることがある。無限定キーの使用によりインデックスが不均衡にならないように、インデックス構造体を編成することができれば非常に有利であろう。

40

【0099】

本発明の第10の態様によれば、範囲の限界を表す第1キーと第2キーの間のキーの範囲を有するデータベースをナビゲートする方法が提供される。この方法において、キーは縮小された範囲にマッピングされ、キーの順序がマッピング中に保存されない場合、マッピングされた空間のうち大きい方のキーより大きい空間および小さい方のキーより小さい空間について探索が行われる。

【0100】

有利には、キーは、キーがデータベース全体に実質的に均等に分配されるように、ハッシュ

50

ュあるいはモジュロ関数により作用される。したがって、例えば、無限定な日時（K）（これは例えば、1970年1月1日以来のミリ秒数としてよい）は、 $(K \bmod N)$ に変換される。ただしNは、インデックスに対して選択されたモジュロである。したがって、変換されたキー値は常に範囲0～(N-1)に入ることになる。

【0101】

好ましくは、キーに適用されるモジュロ値は、インデックス内に保持されるであろう相異なるキー値の最大数以上である。しかし、この条件は必須ではないことは指摘しておかなければならない。

【0102】

有利には、結論セット（ツリーインデックスの終端のノード）は、マッピングされていない形態のキーを含む。これは誤りの確率を減少させる。というのは、その場合、探索キーは結論セット内のキーデータと、両方ともそれらのネイティブな形態で比較されることができるため、マッピング誤りを回避することができるからである。

10

【0103】

本発明は、従来のBツリーまたは他のインデックスに適用され得る。

【0104】

本発明の第11の態様によれば、データベース内で使用するための無限定キーを符号化する方法を含むデータベースが提供される。このデータベースは、各キーが、限定された範囲にキーをマッピングする演算子により処理されることを特徴とする。

20

【0105】

一時キー

データの妥当性または有益性がある一定の時間範囲またはその他の値範囲に制限されるデータを処理することが必要になることがある。従来技術のデータベースでは、このような「一時」データは通常のデータと全く同様に処理され、結果として通常、廃用になった一時キーデータを明示的に削除すること、またはこのような廃用になったキーデータが存在するインデックスのセクションを除去することが必要である。このような操作は非常にプロセス集約的であることがあり、データベース内で相当のパフォーマンスオーバーヘッドを被るか、または設計制約を課することがある。

【0106】

本発明の第12の態様によれば、データベース内でキーを管理する方法が提供される。この方法は、キーが、そのキーおよびそれに関連するデータがデータベース内に維持されるべき持続期間を示すデータフィールドを含むことを特徴とする。

30

【0107】

したがって、結論セットの一部または全部は、キーおよびそれに関連する属性が妥当であるとみなされ得る持続期間を示すデータを含むように修正され得る。

【0108】

好ましくは、期限切れのキーは結論セットから能動的に除去されるのではなく、いったん期限が切れた後は、新しいデータにより上書きするために利用可能となる。したがって、一時キーを明示的に削除する必要はなく、それゆえ削除プロセスに伴うパフォーマンスオーバーヘッドはない。むしろ、いったんマーカが、データがもはや妥当でないことを示すと、そのデータが占有する空間は再使用のために利用可能となる。

40

【0109】

情報が結論セット内にもはや検出されないであろうデータに対して明示的な日時マーカを提供することは可能であるが、この形式の符号化は、許容できないほど大量の空間を占める可能性がある。本発明の好ましい実施形態では、キーは、その現在のエイジの尺度を与えるエイジユニットと、データおよびキーがもはや妥当でなく上書きのために利用可能となるエイジを示すエイジ限界に関連付けられる。

【0110】

好ましくは、エイジユニットは、その長さが単一のエイジユニット内の秒数を表す変数である。エイジ限界は、その後にはキーがインデックスから除去されてもよいエイジユニッ

50

ト数を表す。

【0111】

有利には、各結論セットは、そのセット内でいちばん最近に挿入／更新されたエントリの日時スタンプであるエイジベースを含む。結論セットがアクセスされると、各エントリのエイジが計算され、有利にはその計算の結果がエントリエイジフィールドに記憶される。エントリがいつ上書き可能かを判定するために、エントリエイジとエイジ限界の間で比較を行うことが可能である。

【0112】

スカラーデータの重複の判定

本明細書において前述のように、従来技術（Bツリー）の決定グラフは、キー値およびポインタ情報を保持するリーフブロックへとナビゲートされる。決定インデックスをナビゲートすることは、選択されたキーを見つけるために1回または複数回のディスク入出力操作を必要とすることがある。インデックスが一意的なキー組合せの存在を記録するために使用され、重複組合せが頻繁である場合、インデックスのディスク構造体は、高いパフォーマンスのシステムにとって不適当なことがある。

10

【0113】

本発明の第13の態様によれば、決定インデックスおよびキーインデックスを有するようなデータベースを編成する方法が提供される。この方法において、キーインデックス内のキーは圧縮された方式で記憶され、キーを用いて決定インデックスに質問する前にそのキーが存在するかどうかを確かめるチェックがキーインデックスに対して行われ得る。

20

【0114】

したがって、直接にキーを求めてインデックス自体を探索する必要性を回避することができるよう、キーの存在をインデックスで判定する、メモリに基づく効率的な方法を提供することが可能となる。

【0115】

したがって、本発明は、符号化されたキー値を半導体メモリ内に保持することにより、標準的なインデックス構造体を効果的に補足する。キーの存在は、インデックスを探索する前にメモリに対してチェックされる。メモリ内にマッチするエントリが見つかった場合、キーは重複として拒否され、そうでない場合、エントリをインデックスに挿入する試みがなされる。

30

【0116】

効率的でコンパクトな構造体を提供するために、キーインデックスは、ハッシュ関数のような一方向性マッピング関数を使用する。ハッシュ関数または他のマッピング関数の特徴は次の通りである：

【0117】

・マッピング関数が同じキー値に対して複数回呼び出された場合には必ず、そのキー内の情報が変更されていないことを条件として、マッピング関数は一貫して同じ値を返さなければならない。

【0118】

・2つのキーが等しいとみなされる場合、その2つのキーのそれぞれに対するマッピング関数の呼出しは同じ結果を生成しなければならない。

40

【0119】

・2つの等しくないキーが相異なる値にマッピングされることは要求されないが、等しくないキーに対して異なる結果を提供することはインデックスの効率を改善するであろう。

【0120】

CRC32およびAdler32のような公表されているハッシュ符号アルゴリズムは、適当な機能を提供するので、使用され得る。

【0121】

キーインデックスの実施態様において、メモリは、4バイトの要素からなる均質な配列として構成され得る。この配列は単位配列として知られている。好ましくは、単位配列内の

50

要素の数は、インデックス構造体内に保持される一意的なキー要素の数の1.1倍以上である。しかし、単位配列は、少なくともインデックスと同じ大きさでありさえすれば十分であるということになる。2つのハッシュ関数AおよびBが、この配列とともに使用される。これらの関数は次のように選択される：

【0122】

$A(K) = A(L)$ であるような任意の2つの等しくないキーKおよびLに対して、 $B(K) = B(L)$ となる可能性は低い。

$B(K) = B(L)$ であるような任意の2つの等しくないキーKおよびLに対して、 $A(K) = A(L)$ となる可能性は低い。

【0123】

ハッシュ関数Aは、次式を用いてキーKに対する単位配列内の要素オフセット(0～N)を計算するために使用される：

要素番号 = $ABS(A(K)) \bmod N$

【0124】

ただし、 $ABS(\cdot)$ は、数の符号なし絶対値を返し、Nは単位配列内の要素の数である。

【0125】

この関数は、要素関数E(K)として知られている。

【0126】

単位配列内の各要素はB(K)を記憶する。ただし、その要素オフセットはE(K)により与えられる。キー(K)は、オフセットE(K)にある要素がE(K)である場合、重複とみなされる。要素が任意の他の値である場合、その重複を判定するためにはインデックス構造体が探索されなければならない。

【0127】

キーKをインデックスに挿入しようと試みる場合、以下のイベントのシーケンスが発生する。まず、E(K)が計算される。次にB(K)が計算される。単位配列内のE(K)にある要素がB(K)である場合、挿入は重複Kとして拒否される。しかし、単位配列内のE(K)にある要素がB(K)でない場合、その要素はB(K)で上書きされ、キーが重複であるかどうかを判定するためにインデックスに対して探索が行われる。

【0128】

単位配列の使用は、キーのリストを保持するのに必要な記憶領域を大幅に縮小する。データベースが100万エントリを有し、各キーの長さが32バイトである場合、インデックスをメモリにキャッシュするには32Mbを超えるメモリを必要とするであろう。しかし、等価な単位配列は、4.4Mbのメモリしか占有しないであろう。

【0129】

階層によるインデックスの編成

本明細書に述べられているように、従来技術のインデックスは通常、4個のキーデータからなる1つまたは複数の項目を含むノードを有するツリーグラフを使用する。ツリーグラフ(決定グラフ)は、キー値と、記憶されているデータ構造体へのポインタ情報とを保持するリーフブロック(結論セット)へとナビゲートされる。インデックスをナビゲートすることは、選択されたキーを見つけるために1回または複数回のディスク入出力操作を必要とすることがある。ディスクアクセス操作は一般に、インデックスの最大のパフォーマンスオーバーヘッドを表す。というのは一般に、ディスクアクセスはメモリアccessに比べて遅く、データボトルネックが発生することがあるからである。このオーバーヘッドを軽減するため、インデックスは分割されることが多く、各分割を異なる物理ディスクに割り当てることにより、ディスクI/Oがインデックス操作中にオーバーラップされて、インデックスの全体のスループットを増大できるようにしている。インデックスを細分する方法は、インデックス分割のフラットな一次元的見方を提示する。

【0130】

本発明の第14の態様によれば、インデックスがその作業負荷の一部を1つまたは複数の

10

20

30

40

50

他のインデックスに代行させ得るような、階層的構造体に分割されたデータベースインデックスが提供される。

【0131】

好ましくは、代行インデックスはそれ自体、その作業負荷の一部を1つまたは複数の他のインデックスに代行させ得る。

【0132】

このようなインデックスの階層に参加するインデックスは、ある範囲のサブキーを担当する。このようなインデックスは、別のインデックスに代わってあるサブ範囲のキーを処理している場合、それは代行インデックスとして知られ、そのキーサブ範囲はキー目録(manifest)として知られている。

10

【0133】

好ましくは、インデックスのキー目録は、キー内の連続するバイトのサブセットを、可能な値の連続する範囲に制限することにより定義される。インデックスキー目録に適合しないキーを挿入、削除もしくは更新または探索しようとするいかなる試みもインデックスにより拒否され、それ以外の試みの場合には、キー操作はインデックスまたはその代行インデックスの1つにより処理される。階層は、各インデックスが0個、1個または複数の代行インデックスを有し得るように構成されてもよいが、各インデックスが他の1つのインデックスの代行であるだけであってもよい。

【0134】

インデックスが1つまたは複数の代行インデックスを有する場合、キーを挿入、更新または削除する操作は、適当なキー目録を有する代行インデックスに提出される。いずれの代行インデックスも適当なキー目録を有していない場合、操作はインデックス自体により処理されなければならない。

20

【0135】

インデックスが1つまたは複数の代行インデックスを有する場合、キー範囲を探索する操作は、適当なキー目録を有するすべての代行インデックスに提出され、探索はインデックス自体により行われる。探索代行インデックスからの問合せ結果は、インデックス自体の探索からの結果と組み合わせられる。すべての問合せは同時に実行可能である。したがって、インデックス構造体を、その中のさまざまなサブインデックスにわたり作業が細分され得るように修正することが可能である。各サブインデックスは、関連する物理記憶デバイスを指すことが可能であることにより、複数の並行したディスクアクセスが発生することが可能となる。これは、ディスクボトルネックを減少させ、全体としてデータベースのより高速な操作を可能にする。

30

【0136】

以下、本発明は、添付図面を参照し、例を用いてさらに説明される。

【0137】

図1は、本発明の実施形態を構成するインデックスと、データ記憶装置とを組み込んだデータベースを図式的に示す。インデックス2は、決定グラフ4および複数の結論セット6、8、10、12および14を備える。各結論セットは、決定グラフを通して、1つの、かつただ1つのパスにより到達される。しかし、個の場合、各結論セットは、データ記憶装置16内の関連するエントリを指す。

40

【0138】

図2は、概括的に20で示される決定グラフの構造を図式的に示す。決定グラフは原点22から始まる。決定グラフを通るすべてのナビゲーションは原点から出発しなければならない。原点は、決定グラフ内のさらなるノードまたは結論セットを指す0個（例えば、データベースが新規であるとき）、1個または2個の決定ポイントを有し得る。他のそれぞれの決定ノードは、0個、1個または2個の決定ポイントを含むことが可能であり、それぞれの決定ポイントは別の決定ノードまたは結論セットのいずれかを指す。決定ノードにある決定ポイントを本明細書では「ローポイント」および「ハイポイント」と呼ぶことにする。

50

【0139】

決定グラフ内の任意の決定ノードにある決定ポインタは、同じ決定グラフ内の他の1つの決定ノードか、または単一の結論セットのいずれかのみを指すことができる。任意の決定グラフノードは厳密に、同じ決定グラフ内の他の1つの決定ノードによって、または原点によって指されなければならない。同様に、任意の結論セットは厳密に、決定グラフ内のただ1つの決定ノードによって指されなければならない。したがって、任意の結論セットは、原点から決定グラフを通り単一かつ一意的なパスをたどることによってのみ到達されることができる。この一意的パスはナビゲーションパスとして知られる。

【0140】

図3は、概括的に40で示される決定ノードの論理構造をさらに詳細に図式的に示す。決定ノードは、ローポインタタイプ、ローポインタ、ハイポインタタイプおよびハイポインタを有する。ローポインタタイプ41は、ローポインタ42の目的を示す。ローポインタタイプ41は、ローポインタ42が決定ノードまたは結論セットのいずれを指すかを示すことができる。ローポインタ42は、それが指す決定ノードまたは結論セットのアドレスを与える。ポインタが存在しないことを示すためにゼロ値を挿入することができる。同様に、ハイポインタタイプ44は、ハイポインタ45が決定ノードまたは結論セットのいずれを指すかを示すことができる。ハイポインタは、それが指す決定ノードまたは結論セットのアドレスを示す。こちらも、ポインタが存在しないことを示すためにゼロ値を使用することができる。これにより、決定グラフの「ツリー」状構造が、データプロセッサおよびそのメモリを用いて表現され記憶されることが可能となる。

【0141】

図4は、結論セットの構造を図式的に示す。結論セットは、複数のキーおよび属性エントリ60、62、64、66および68を備え、それぞれ互いに結論セット内の次のエントリのアドレスを与える有向リンクにより連結される。図5は、結論セット内の各エントリの論理構造を図式的に示す。エントリは、3個のフィールド、すなわちリンクフィールド80、キーフィールド82および属性フィールド84からなる。リンクフィールドは、次の結論セットエントリを指す。ゼロ値で、それ以上エントリがないことを示すことができる。キーフィールド82は、キーの厳密な値を保持し、属性フィールド84は、そのキーに関連する属性、例えばキーフィールド82に対応するデータのアドレスを保持する。インデックスの構成要素を定義したので、次にインデックスを使用しナビゲートする方法について論ずる。

【0142】

インデックスに沿ってナビゲートするため、ナビゲーションパスに沿って訪問される各決定ノードはキー内の1つのビットまたはビット群を参照する。このビット群は「決定群」として知られる。決定群内のビット数は、単一ビットとキー内の全ビットの間の任意のビット数を含むように設定可能である。

【0143】

決定群が取る可能な値の範囲は「決定範囲」として知られ、決定範囲最小値および決定範囲最大値によって限定される。決定範囲最小値は、決定群で全ビットがセットされたものより小さい任意の値であることが可能であり、同様に決定範囲最大値は、決定範囲最小値より大きい任意の値であることが可能である。便宜上、最小値および最大値は、符号なし絶対値記数法で表され得る。

【0144】

各決定ノードは関連する決定値を有する。決定値は、その決定ノードを通してナビゲートするときに、ローポインタまたはハイポインタのいずれがたどられるかを定める。キー内の検査された決定ビット群の値が決定値より大きいときにはハイポインタが使用され、そうでないときはローポインタが使用される。

【0145】

決定値の比較は、有利には、符号なし絶対値記数法を用いて実行される。決定値は、システム管理者、設計者またはオペレータにより選択される任意の値に設定され得るが、有利

10

20

30

40

50

には、次のうちの1つから選択される：

1. 決定群のすべての可能な値の数学的メジアン。
2. 決定群のすべての期待値の数学的メジアン。
3. 決定群のすべての期待値の数学的平均。
4. インデックスの作成時に指定される任意値。
5. 現在の決定値（すなわちいちばん最近に使用された決定値）とそれに先行する決定値のメジアン。

【0146】

訪問される決定ノードにおける決定群の選択は、有利には（しかし必須ではない）、次のうちの1つまたは複数に基づくことが可能である：

- i. 決定群はすべての決定ノードについて同じであり得る。
- ii. 決定群は決定ノードへの訪問ごとに変化し得る。
- iii. 決定群は、前の決定ノードからの決定値が現在の決定群の決定範囲最大値または決定範囲最小値に達するかまたは近づくときに、変化し得る。
- iv. 決定群は、連続するノード間で決定値が1単位未満だけ、または他の何らかの所定しきい値未満だけ変化するときに、変化し得る。

【0147】

決定ノードにある決定群のサイズは、1つまたは複数のパラメータに基づいて定められ得る。決定群のサイズは、例えばすべての決定ノードに対して固定されてもよい。しかし、このサイズは、新しい決定群が選択されるときに前の決定群から増大してもよく、あるいはこのサイズは、新しい決定群が選択されるときに前の決定群から減少してもよい。

【0148】

決定ノードにおける、キーに対する決定群の位置は、次のうちの1つまたは複数に基づくことができる。決定群の位置は、すべての決定ノードに対して固定されてもよい。追加的におよび／または別法として、決定群の位置は、新しい決定群が選択されるときに前の決定群からの固定オフセットであってもよい。

【0149】

有利には、インデックスが範囲マッチングによるキー検索をサポートすることになる場合には、さらなる制約が課され得る。特に、最上位ビットが、原点のような階層的に上位のノード内に含まれなければならない。その場合、キー全体の内部における決定群の桁位置は、桁位置が単調に減少するように変化すべきである。

【0150】

インデックスがどのようにナビゲートされ得るかについていくつかの例を考察することは有利である。図6は、厳密マッチング探索で4バイトキーがどのようにナビゲートされ得るかを示す。この例では、キーはASCII符号化で記憶され、各キーに対して示されている数字は、16進記数法でのその値を示す。この例では、各決定群は4ビットの固定サイズを有する。決定群は、訪問されるそれぞれの新しいノードにおいて、次の下位4ビットに順次移動される。この例では、最上位ビットは0で指定され、次に上位のビットは1であり、次に上位のビットは2であり、以下同様である。さらに、簡単のため、あらゆるノードで適用される決定値は4である。したがって、決定群が値0、1、2、3または4を有する場合、グラフは左にトラバースされる。しかし、決定群が5～15の範囲の値を有する場合、グラフは右にトラバースされる。

【0151】

図6に示すように、いくつかの名前がデータベースに提示される。これらの名前すなわちキー値は、Fred、John、Bill、Zoe、EricおよびPeteである。各キーに等価な16進数が名前の隣りに配置されている。これから説明されるナビゲーションプロセスは、挿入、削除またはマッチングに適している。

【0152】

最初に、キーは第1の決定ノード100に提示され、そこで、ビット0～3がテストされる。各16進ワードは1バイトを占有するため、明らかにビット0～3はキーの最初の文

10

20

30

40

50

字に対応する。したがって、Fred、John、BillおよびEricはすべて「4」で始まるので、それらのキーは枝102に沿って左方へ伝搬し第2レベルのノード104に至る。しかし、ZoeおよびPeteは両方とも最初の4ビットが0～3の範囲外に出るため、結果としてそれらのキーはハイポインタパス106に沿ってノード108へと伝搬する。

【0153】

ノード104および108は決定グラフの第2レベルを占有し、結果としてそれらの決定は、キー内の次の4ビットに基づく。これは、16進表現において次の文字を見ることに対応する。したがって、ノード104で、Billは、キー内のビット4～7が値2を符号化しているため、ローレベルポインタ110に沿って後続ノード（図示せず）に渡され、一方、Fred、JohnおよびEricは、それらのキー内のビット4～7が16進記数法でそれぞれ値6、Aおよび5を符号化しているため、ハイレベルポインタ112に沿ってさらなるノード（図示せず）に渡される。同様に、ノード108で、キーPeteは、その中のビット4～7が値0を符号化しているため、ローレベルポインタ114に沿って渡され、一方、Zoeは、ビット4～7が値「A」を符号化しているため、ハイレベルポインタ116に沿って渡される。このプロセスは、キーが結論セットに到達するために十分なだけ探索されるまで繰り返すことができる。

10

【0154】

図7は、範囲マッチング探索で4バイトキーがどのようにナビゲートされ得るかを図式的に示す。図6と同様に、キーはASCII符号化で記憶され、各キーに対するASCII値は16進記数法でのその値を示す。

20

【0155】

この例では、決定群は8ビットの固定サイズを有する。決定群は、決定範囲が尽くされたとき、すなわち、決定値が決定範囲のいずれかの境界に到達したときにのみ、次の下位8ビットに移動される。この例では、図6に示した例と同様に、最上位ビットはビット0で指定され、次に上位のビットは1であり、などとなる。あらゆるノードにおける決定範囲は、ASCIIフォーマットで表現されたすべての数字および大文字をカバーするために、16進で30～50である。あらゆるノードにおいて新しい決定群のために使用される決定値は、16進で48であり、その後、その同じ群に対してそれぞれの後続ノードにおいて、同じ決定群の前の決定値の平均であるように変更される。前の決定値が1つだけの知られている場合、直前に左または右のいずれにナビゲートしてきたかに応じて、決定値範囲の最小限界または最大限界が使用される。

30

【0156】

前の例と同じキーを使用すると、キーは第1ノード118に提示され、第1ノード118は、最初の8ビットすなわち0～7を検査し、それらを決定値48（16進）と比較する。図示したキーの場合、Fred、BillおよびEricはローポインタに沿ってノード122に進み、一方、John、ZoeおよびPeteはハイポインタ124に沿ってノード126に進む。

【0157】

ノード122で、決定値を変更することが必要となる。この新しい決定群のために使用される決定値は、同じパスに沿って到達される前の決定群のメジアン値に変更される。この場合のように、前の決定値が1つだけ知られている場合、決定値範囲の最小または最大限界が使用される。ノード122にはローポインタ120によってナビゲートされたことを考慮すると、決定範囲の下限が使用される。

40

【0158】

したがって、新しい決定値を計算するためには、30（16進）～48（16進）の範囲内の数のメジアンを求めることが要求される。明示的には、この数の群は30、31、32、33、34、35、36、37、38、39、3A、3B、3C、3D、3E、3F、40、41、42、43、44、45、46、47および48である。これから、メジアン値は3Cであることがわかる。これを決定値として設定すると、Fred、Bill

50

および E r i c はすべてハイポインタ 1 2 7 に沿って次のノード（図示せず）に進む。次のノードの決定値は、3 C（1 6 進）から 4 8（1 6 進）までにわたる範囲のメジアン値となるであろう。

【0 1 5 9】

同様に、ノード 1 1 8 からハイポインタ 1 2 4 に沿って到達されるノード 1 2 6 は新しい決定値を計算しなければならないが、この場合それは 4 C（1 6 進）となる。このノードでキーを適用すると、J o h n は、値 4 A を含み、これは決定値 4 C より小さいため、左（ローポインタ）に進む。一方、Z o e および P e t e は、それぞれ値 5 A および 5 0 を有し、これらは決定値 4 C より大きいため、右側（ハイポインタ）に進む。

【0 1 6 0】

探索のもう 1 つの例は、文字探索に関して見ることができる。つまり、探索は以下の基準を用いて実行される：

- ・ A S C I I 文字キーに関する厳密マッチインデックスで、各文字は 8 ビットを占有し、「A」～「Z」の範囲にのみある。
- ・ 8 ビットの固定された決定群。
- ・ 決定グラフの原点は、上位 8 ビット（第 1 文字）。
- ・ 決定群は、各決定ノードごとに 8 ビット（1 文字）ずつ進む。

【0 1 6 1】

「A」および「Z」に対する A S C I I 符号はそれぞれ 6 5 および 9 0 である。したがって、この群に対する理に適った決定値はメジアン値（すなわち 7 8）である。

【0 1 6 2】

この例では、決定群は、新しい決定ノードに到達するたびに 1 文字（8 ビット）ずつ進む。いずれのポインタをたどるかを定めるために、キー内の決定群内のビットの値と、固定された決定値（7 8）との比較が行われる。

【0 1 6 3】

より洗練された手法は、新しい決定ノードへのそれぞれの移動ごとに決定値を変更することを含む。例えば、決定群の決定範囲は、決定範囲最小値および決定範囲最大値、すなわちそれぞれ 6 5 ～ 9 0 によって制限される。次の決定ノードに移動すると、オペレータは、現在の決定値と前の決定値のメジアンを使用することにより新しい決定値を選択することを選び得る。これが新しい決定群であり、前の決定値が利用可能でないときには、決定範囲の最小値または最大値（それぞれローポインタまたはハイポインタのいずれをたどったかによって決まる）を使用し得る。新しい決定群の選択は、決定値が 1 または他の何らかのあらかじめ定義されたしきい値未満だけ変化したときに行われ得る。決定値がどのように変化するかの例を以下に示す。

【0 1 6 4】

キー内の決定群は値 8 2 を有する。新しい群に対する開始決定値は 7 8 である。決定範囲は 6 5 ～ 9 0 である。決定群は、決定値が 1 未満だけ変化したときに新しい群が選択されるように設定されている。

【0 1 6 5】

【表 1】

10

20

30

40

決定値	決定値の変化	アクション
7 8	—	ハイポインタをたどる ($82 > 78$)。新しい決定値 = $(78 + 90) / 2$
8 4	6	ローポインタをたどる ($82 < 84$)。新しい決定値 = $(84 + 78) / 2$
8 1	3	ハイポインタをたどる ($82 > 81$)。新しい決定値 = $(81 + 84) / 2$
8 2. 5	1. 5	ローポインタをたどる ($82 < 83$)。新しい決定値 = $(82.5 + 81) / 2$
8 1. 7 5	0. 7 5	決定群を変更する

10

【0166】

範囲マッチングによるキー検索をサポートすることが要求されないインデックスは、それが要求されるインデックスほど厳しく制約されないことに留意すべきである。特に、範囲マッチングをサポートすることが要求されないインデックスは、その原点で任意の決定群を有することが可能であり、決定群の位置を任意に変更可能であり、また決定値が決定範囲限界、すなわち決定範囲の最大値および最小値に近づいたとき（これは、システム設計者により定義されてもよく、また、ユーザまたは何らかの管理機能により選択されてもよい）、または決定値があらかじめ選択された量未満だけ変化したときに決定群を任意に変更することも可能である。

20

【0167】

候補キーを挿入、削除または検索する目的でナビゲーションパスをトラバースするとき、候補キーの決定群は、訪問される各決定ノードにおいて、決定ノードの決定値と比較され、ローポインタまたはハイポインタをたどって次の決定ノードまたは結論セットに進む。結論セットに到達すれば、ナビゲーションパスに沿った行程は完了する。ナビゲーションパスに沿った行程はまた、決定群の値が、存在しない決定ポインタの使用を示しているときにも完了する。

【0168】

結論セットは、一連のキーおよび関連する属性からなる。結論セット内に含まれるキーの数は、次のうちの1つまたは複数の組合せによって決められる上限により制限される：

30

1. 固定された限界。
2. インデックスが作成されるときに指定される限界。
3. インデックスの有効期間にわたり指定可能な限界。
4. インデックス増大の関数として変動する限界。

【0169】

図8は、キー（K）を属性（Q）とともに挿入する手続きを示す。手続きは、ステップ150から始まり、ここではインデックスにその原点から入る。デフォルトすなわち初期決定群DGは設計者により設定される。開始決定値DV₀を計算するためのアルゴリズムが実施される。次に制御はステップ152に移り、キー内の決定群が決定値より大きいかどうかをテストする。テストが満たされる場合、制御はステップ154に移り、ハイポインタHPが検査され、新しい決定値DV₁が作成される。

40

【0170】

ステップ154から制御はステップ156に移り、ポインタ（これがハイポインタであるかローポインタであるかにかかわらず）が存在することを確認するチェックが実行される。同様に、ステップ152が、キー内の決定群が決定値より大きくないことを示している場合、制御はステップ158に移り、ローポインタが検査され、新しい決定値DV₁が作成される。制御はステップ158からステップ156に移る。ポインタが存在する場合、ステップ156から制御はステップ160に移る。ステップ160は、ポインタがさらなる決定ノードを指しているかどうかを確認するテストを行う。ポインタが決定ノードを指している場合、制御はステップ162に移り、次の決定ノードにテストを進める。ステッ

50

プ 1 6 2 から制御はステップ 1 6 4 に移り、決定群が変更されるべきかどうかを確かめるテストを実行する。このテストへの答えが Y E S である場合、制御はステップ 1 6 6 に移り、上述の基準に従って新しい決定群の値を計算する。その後制御はステップ 1 6 4 または 1 6 6 のいずれかからステップ 1 5 2 に戻る。

【 0 1 7 1 】

ステップ 1 5 6 が、ポインタが存在しないと判定した場合、制御はステップ 1 7 0 に移り、新しい結論セットの作成を引き起こした後、ステップ 1 7 2 に移り、新しい結論セットを指すようにポインタを更新する。その後、制御はステップ 1 7 4 に移り、キーおよびその属性を結論セットに追加する。

【 0 1 7 2 】

さらに、ステップ 1 6 0 でポインタが決定ノードを指していない場合、それは結論セットを指していなければならない、制御はステップ 1 7 6 に移り、関連する結論セットへのジャンプが行われる。そこからステップ 1 7 4 に進み、キーおよびその属性が結論セットに追加される。

【 0 1 7 3 】

ステップ 1 7 5 で、結論セットがそのサイズ限界を超えたかどうかを確かめるテストが行われる。結論セットが大きくなりすぎている場合、制御は、キー挿入手続きからの出口を表すステップ 1 7 8 に移るが、結論セットがそのサイズ限界に達している場合、制御はステップ 1 8 0 に移り、ポインタのない新しい決定ノードを作成する。ステップ 1 8 0 から制御はステップ 1 8 2 に移り、前の決定ノードのポインタを、旧結論セットではなく新しい決定ノードを指すように更新する。その後、制御はステップ 1 8 4 に移り、旧結論セット C S (S) が空であるかどうかを確かめる検査を行う。旧結論セット C S (S) が空でない場合、制御はステップ 1 8 6 に移り、結論セット C S (S) 内の最初のキーおよび属性エントリに進み、このエントリが配置されるべき新しい結論セットを見つけるためにインデックスを再ナビゲートする。いったんエントリが結論セットに再配置されると、ステップ 1 8 8 でそのエントリはその現在の結論セット C S (S) から除去される。その後、制御はステップ 1 8 4 に戻り、結論セット C S (S) が空になるようなときまでプロセスは繰り返される。その後、制御は、キー挿入手続きからの出口点を表すステップ 1 9 0 に移る。

【 0 1 7 4 】

こうして一般に、キーおよびその関連する属性をインデックスに挿入する手続きは、挿入されるべきキーに対する決定パスに沿ったナビゲーションを必要とする。ナビゲーションパスが結論セットに到着することなく終了する場合、新しい結論セットが作成されなければならない、決定ポインタはその新しい結論セットを指すように更新されなければならない。新しい決定ポインタは、決定グラフにおいて、決定グラフのトラバースが終了した点に配置されなければならない。しかし、ナビゲーションパスが結論セットで終了する場合、キーおよびその属性はこの結論セットに挿入される。結論セットのそれぞれの修正は、そのセットのサイズを変化させる。したがって、新しいキーが結論セットに挿入され、これによりそのセットがその上限を超えたか、または超えることになる場合、新しい決定ノードが作成されなければならない。新しい決定群および／または決定値は、新しい決定ノードに関連付けられなければならない、フルの結論セットを以前に指していた決定ポインタは新しい決定ノードを指すように補正されなければならない。その後、旧（今はフルの）結論セットからのあらゆる項目が、インデックスに沿ったナビゲーションパスをたどって、1 つまたは複数の新しい結論セットに再挿入されなければならない。

【 0 1 7 5 】

データベース内のデータは時間とともに発展し、この発展はキーおよびデータの削除を含むであろう。その削除の理由は、データがデータベースにとってもはや関連性がないか、または、データベースの「現行」バージョンが無関係または廃用になったデータで混乱しないようにする保管の目的で除去される必要があるかのいずれかである。図 9 は、キーを削除する手続きを図式的に示す。ステップのうちのいくつかは図 8 に関して説明したもの

10

20

30

40

50

と同一であるので、これらのステップには同じ参照番号が与えられている。したがって、ステップ150～166は、すでに説明したものと同一である。ただし、ステップ156で行われるテストが、ポインタが存在しないことを示している場合、制御は、削除手続きからの出口を表すステップ200に移る、という点が例外である。

【0176】

ステップ160におけるテストが、ポインタが決定ノードを指していない（キーを挿入する手続きの場合にそうであったように）と判定している場合、ステップ160は制御をステップ176に移す。ステップ176から制御はステップ202に移り、結論セットCS（S）が空であるかどうかを確かめるテストが実行される。結論セットが空である場合、制御は、手続きからの出口を表すステップ204に移り、そうでない場合、制御は、結論セット内のエントリ全体の走査を実施するステップ206に移る。走査は、結論セット内の最初のキー（L）から始まる。その後、制御はステップ208に移り、キー（L）が削除キーに等しいかどうかを確かめるテストが行われる。等しい場合、制御はステップ210に移り、キー（L）およびその属性が結論セットから削除される。ステップ210から制御はステップ214に移り、キー（L）が結論セット内の最後のキーであるかどうかを確かめるテストが行われる。また、制御は、ステップ208から、キー（L）が最後の削除キーと同じでない場合にもステップ214に移る。その後、制御はステップ216に移り、次のキーが選択された後、制御はステップ208に移る。そうでない場合、制御はステップ214から、この手続きの出口を表すステップ218に移る。

【0177】

したがって一般的に言えば、キーおよびその関連する属性をインデックスから削除することは、削除されるべきキーに対するナビゲーションパスに沿ったナビゲーションを必要とする。ナビゲーションパスが結論セットに到着することなく終了する場合、キーはインデックス内に存在しないとみなされる。しかし、ナビゲーションパスが結論セットで終了する場合、削除されるべきキーに厳密に等しいすべてのキーを求めて、結論セット全体で走査が開始されなければならない。次に、これらのキーおよびそれらに関連する属性が結論セットから除去される。

【0178】

図10は、キーの厳密な問合せを行い、その結果を返す手続きを示す。この手続きは、キーを削除する手続きと非常に多くの類似点を共有し、ステップ150～206は図9を参照して説明した通りである。しかし、追加ステップ151がステップ150と152の間に挿入され、このステップでは、結果セット（R）が空状態にリセットされる。結論セットまでのインデックスのナビゲーションの後、ステップ202で、制御はステップ206に移り、結論セット全体の順次走査を開始する。ステップ206から制御はステップ230に移り、現在検査中のキー（L）が探索キーに等しいかどうかを確かめるテストが行われる。YESである場合、制御はステップ232に移り、そうでない場合、制御はステップ234に移る。ステップ232で、キーおよびその属性が探索問合せ結果リスト（R）に追加されて、その後、制御はステップ234に移る。ステップ234は、現在検査中のキーが結論セット内の最後のキーであるかどうかを確かめるテストを行う。NOである場合、次のキーが検査され、制御はステップ230に移る。キーが最後のキーである場合、制御はステップ234から、このルーチンからの出口を表すステップ236に移る。

【0179】

図11は、最小キー値（I）および最大キー値（A）を有する範囲問合せを実行し、その結果（R）を返す手続きを図式的に示す。手続きはステップ300から始まり、ここでは結果セット（R）がヌルすなわち空状態にリセットされる。その後、制御はステップ302に移り、決定群および決定値が計算され、そこから制御はステップ304に移り、範囲問合せにマッチする結果を見つけるためにインデックスがトラバースされる。インデックスをトラバースする手続きは図12にさらに詳細に示される。この手続きはステップ350から始まり、その開始ノードから手続きに入る。その後、制御はステップ352に移り、最小キーが決定値より小さいまたは決定値に等しい（または省略）かどうか、かつ、最

大キーがその関連する決定値より大きい（または省略）かどうかを確かめるテストが行われる。このテストの結果が、両方の条件が満たされているようなものである場合、制御はステップ370に移り、そうでない場合、制御はステップ354に移る。ステップ370は、ローポインタが設定されているかどうかを確かめるチェックを行う。設定されている場合、制御はステップ372に移り、ローポインタが結論セットを指しているかどうかを確かめるチェックが行われる。YESである場合、制御はステップ400に移る。NOである場合、制御はステップ374に移り、決定群および決定値が再計算され、ポインタを介して到達されるノードからインデックスをトラバースするために使用される。手続きは自分自身を呼び出す能力を有し、したがって再帰的である。その後、制御はステップ376に移り、ハイポインタが存在するかどうかを確かめるチェックが行われる。NOである場合、制御は、出口を表すステップ378に移る。YESである場合、制御はステップ390に移り、ハイポインタが結論セットを指しているかどうかを確かめるテストが行われる。YESである場合、制御はステップ400に移るが、ハイポインタが結論セットを指していない場合、制御はステップ392に移り、新しい決定群および決定値が計算され、新しいノードを開始ノードとしてインデックスのさらなるトラバースが実行される。

10

【0180】

手続きの最初の近くに返ると、ステップ352が制御をステップ354に移した場合、ステップ354は、最小キー値および最大キー値の両方が決定値より小さいまたは決定値に等しいかどうかを確かめるテストを実行する。その通りである場合、制御はステップ356に移り、そうでない場合、制御はステップ358に移る。ステップ356は、ローポインタが存在するかどうかを確かめるテストを実行し、存在しない場合、制御は、手続きからの出口を表す357に移り、そうでない場合、制御はステップ360に移る。ステップ360は、ローポインタが結論セットを指しているかどうかを確かめるテストを行う。YESである場合、制御はステップ400に移る。そうでない場合、制御はステップ402に移り、決定群および決定値を再計算して、トラバース手続きを再帰的に呼び出す。ステップ358に戻って、ハイポインタが存在するかどうかを確かめるテストが実行され、YESである場合、制御はステップ362に移り、そうでない場合、制御は、手続きからの出口を表すステップ364に移る。ステップ362は、ハイポインタが結論セットを指しているかどうかを確かめるテストを実行し、YESである場合、制御はステップ400に移る。そうでない場合、制御はステップ364に移り、新しい決定群および決定値を計算し、これらの新しい値から開始して、インデックスのさらなるトラバースが実行される。

20

30

【0181】

ステップ400は、結論セット（S）が空であるかどうかを確かめるテストを実行し、YESである場合、制御は、手続きからの出口を表すステップ404に移る。結論セットが空でない場合、結論セット内のエントリの順次走査が、ステップ406から出発して開始され、結論セット内の最初のキーが選択される。その後、制御はステップ408に移り、キー（L）が最小キー値および最大キー値により定義される範囲内にあるかどうかを確かめるテストが実行される。テストの結果が「YES」である場合、ステップ410で、キーおよびその属性が結論セット（S）に追加されるが、ステップ408のテスト結果が否定的である場合、制御はステップ412に移り、現在考慮中のキーが結論セット内の最後のキーであるかどうかを確かめるテストが行われ、その通りである場合、制御は、手続きからの出口を表すステップ414に移る。ステップ410もまた制御をステップ412に移す。ステップ412のテストが、考慮中のキーが最後のキーでないと判定している場合、ステップ416で、結論セット内の次のキーが選択され、制御はステップ408に戻る。

40

【0182】

したがって一般的に言えば、オプションの最小探索キー値およびオプションの最大探索キー値を用いた範囲マッチによってキーおよびそれらの範囲属性を検索することは、最小探索キーおよび最大探索キー内の決定群ビットを各決定ノードにおける決定値と継続的に比較して、決定グラフの原点から再帰的にトラバースすることを必要とする。決定群内の最

50

小探索キービットの符号なし絶対値が決定値より小さいか、または決定値に等しく、かつ決定群内の最大探索キービットの符号なし絶対値が決定値より大きい場合、第1に、最小探索キーを用い、最大探索キーを省略して、ローポインタから到達される決定グラフをトラバースするアクションがとられる。そして第2に、最大探索キーを用い、最小探索キーを省略して、ハイポインタから到達される決定グラフをトラバースするアクションがとられる。

【0183】

決定群内の最小探索キービットおよび最大探索キービットの符号なし絶対値が両方とも決定値より小さいか、または決定値に等しい場合、最小探索キーおよび最大探索キーの両方を用いてローポインタから到達される決定グラフのトラバースが実行される。決定群内の最小探索キービットおよび最大探索キービットの符号なし絶対値が両方とも決定値より大きい場合、最小探索キーおよび最大探索キーの両方を用いて、ハイポインタから到達される決定グラフのトラバースが実行される。

10

【0184】

複数回の反復された決定グラフのトラバースが要求されることは明らかである。トラバースから生じるすべてのナビゲーションパスが結論セットに到着することなく終了する場合、マッチするキーがインデックス内に存在しないとみなされる。結論セットで終了するナビゲーションパスに対しては、結論セット内のすべてのキーにわたる走査が実行され、探索範囲内の要求される属性にマッチするすべてのキーが探索結果として返される。

20

【0185】

データプロセッサおよび記憶システム内でのデータベースの実施の一部として、結論セットのサイズが記憶デバイスブロック転送サイズの整数倍でなければならないとすることは、必須ではないが有利である。これは、物理メモリの最適な使用法を提供する。また、決定グラフが、記憶デバイスブロック転送サイズの整数倍であるサイズのブロックに分割されることもまた、必須ではないが有利であり、これは、決定ブロックがフルになったとき、新しいブロックが作成され、新しい決定ポインタが適宜新しいブロックを指すことができるという利点を有する。

【0186】

データベースの高速動作を達成するため、決定グラフは全体として、または部分的に、使用時にデータ処理システムの半導体メモリ内に保持されることにより、ディスクアクセスに関連する時間オーバーヘッドを避けることが有利である。さらなる改善として、結論セットが半導体メモリ内に記憶されること、または少なくとも、いったん結論セットが識別された後は、結論セットの順次走査を素早く実行することができるように、結論セットが半導体メモリ内にスワップされることもまた好ましい。

30

【0187】

すべてのキータイプの範囲マッチングのための最適な構成は、固定された決定値0を有する1ビットの決定ビット群を使用することである。原点における決定群は最上位ビットであり、決定群は、訪問される決定ノードごとに1ビットずつ（最下位ビットへ向かって）進む。数値または日時の厳密キーマッチングの場合、最適なビット構成は、固定された決定値0を有する1ビットの決定ビット群を使用することであると思われる。原点における決定群は最下位ビットであり、決定群は、訪問される決定ノードごとに1ビットずつ（最上位ビットへ向かって）進む。ASCII系キーの探索の場合、厳密マッチングのための最適なビット構成は、8ビットの決定群を使用することである。最適な決定値は、期待される文字の範囲に左右されるであろう。

40

【0188】

パターンマッチング

これまでの図を参照して説明したデータベース構造体は、決定ノードが何らのキーデータも含まないような構造体に関するものである。しかし、1つまたは複数のノードが部分的なキーシーケンスを含むようにデータベース構造体を修正することが可能である。これは、パターンマッチングによるスカラーデータのインデックス付けおよび検索を容易にする

50

。ツリー構造は依然として従来技術のデータベースに比べて小さいが、その理由は、各決定ノードは、従来技術のシステムの場合のようにキー全体ではなく比較的少量のデータしか含む必要がないからである。

【0189】

データベース構造体は、本明細書ですでに説明したものからほんの少しの修正しか必要としない。図13は、決定グラフの構造を示す。これは、多くの点で、図2に示したのと同じ配置を維持している。しかし、原点および各決定ノードは今度はビットシーケンスを含む。ビットシーケンスは、候補キーが比較される相手となるサブキーの一部を表す。各決定ノードの論理構造は図14に示されている。この構造は、図3に関して説明したものを強く想起させる。ノード（包括的に440で示す）は、決定ビットシーケンスのための記憶領域450を有する。決定ビットシーケンスは、任意の長さであることが可能である。しかし、留意すべき重要なことは、概して、決定ビットシーケンスは、決定ツリーに提示されるキーの長さよりもかなり短いであろうということである。ノードは、除外ポインタタイプ452および包含ポインタタイプ456を有する。これらは、それらの関連する除外ポインタまたは包含ポインタがさらなる決定ノードまたは結論セットのいずれを指しているかを示す。したがって、これらのポインタは、図3を参照して説明したローポインタタイプおよびハイポインタタイプ（それぞれ41および44）と同じ機能を実行する。除外ポインタ454および包含ポインタ458は、現在のノードに続く次のノードまたは結論セットのアドレスを指す。したがって、ポインタ454および458は、前述のローポインタおよびハイポインタ（それぞれ42および45）に類似している。しかし、用語の変更は、候補キーが決定ビットシーケンスを含む場合には一方のポインタがたどられるが、候補キーが決定ビットシーケンスを含まない場合には他方のポインタがたどられるということを単に明確化しているだけである。

【0190】

図15は、キーをツリー構造に挿入する手続きを示す。この手続きは、図8に関して説明したものと非常に類似しており、簡単のため同じ参照番号が同じ部分に対して使用されている。ただし、ステップ152はステップ152'となるように修正されているが、これは、テストが、決定ノードにある決定ビットシーケンスがキー内に含まれるかどうかを確かめるように変更されているからである。YESである場合、制御はステップ154'に移り、包含ポインタがたどられる。その後、制御はステップ156に移る。ステップ152が、決定ビットシーケンスが候補キー内にないと判断した場合、制御はステップ158'に移り、除外ポインタがたどられる。手続きへのもう1つの修正は、ステップ162がステップ152'の入力を直接指し、それによりステップ164および166を省略することができることである。しかし、他の点では、キー挿入手続きは前述の通りである。

【0191】

キーを削除する手続きもまた、図16に示すように、前述のものと非常に類似しており、今度も同じ参照番号が同じ部分に対して使用される。キーを挿入する手続きの場合と同様に、ステップ152、154および158は、図13を参照して説明したようにステップ152'、154'および158'となるように修正される。したがってステップ152は、候補キーが決定ビットシーケンスを含むかどうかを確かめるテストを行い、YESである場合、包含ポインタがたどられ、そうでない場合、除外ポインタがたどられる。さらに、ステップ162は今度はステップ152'の入力を直接指し、それによりステップ164および166を省略している。

【0192】

インデックスは、もちろん、キーの厳密問合せマッチングのためにもナビゲートされ得る。図17は、キーの厳密問合せを返すナビゲーション手続きを示す。この手続きは、図10を参照して説明したものと非常に類似しており、前と同様、繰り返しを避けるために同じ参照番号が同じステップに対して使用される。ただし、ステップ152、154および158が修正されることは例外である。すなわち、ステップ152'は今度は候補キーを決定ビットシーケンスと比較し、決定ビットシーケンスが候補キー内にある場合、制御は

ステップ154'に移り、包含ポインタがたどられ、そうでない場合、制御はステップ158'に移り、除外ポインタがたどられる。さらに、ステップ162は今度はステップ152'の入力を直接指す。

【0193】

図18は、パターンリスト(L)を有するノードからトラバースして結果(R)を返す手続きを示す。手続きはステップ500から出発し、開始ノードに入る。次に制御はステップ502に移り、決定ビットシーケンスがパターンリスト(L)内に含まれるかどうかを確かめるテストが行われる。このテストの結果が否定的である場合、制御はステップ504に移り、決定ノードからの除外ポインタが存在するかどうかを確かめるテストが行われる。除外ポインタが存在しない場合、制御はステップ506に移り、ポインタが別の決定ノードを指しているかどうかを確かめるテストが行われ、指している場合、制御はステップ508に移り、そうでない場合、制御はステップ510に移る。ステップ508により、次の決定ノードが選択され、ステップ512でトラバース手続きが再帰的に呼び出される。ステップ512から制御はステップ514に移る。さらに、ステップ502でのテストが、決定ビットシーケンスがパターンリスト内に含まれると結論した場合、制御はステップ514に移り、同様に、ステップ504が、除外ポインタが存在しないことを示した場合にも、制御はステップ514に移る。

【0194】

ステップ510に戻ると、このステップによって、ステップ506で検査されたポインタにより指される結論セットへジャンプが行われ、その後ステップ516で、結論セット全体の探索が行われる。探索手続きは、図19を参照して後述される。ステップ516から、制御はステップ514に移る。ステップ514は、包含ポインタが存在するかどうかを確かめるテストを実行し、存在する場合、制御はステップ518に移り、そうでない場合、制御は、手続きの出口を表すステップ520に移る。ステップ518は、ポインタがさらなる決定ノードを指しているかどうかを確かめるテストを行い、その通りである場合、制御は、手続きを再び再帰的に呼び出すステップ522に移り、そうでない場合、制御はステップ524に移り、ポインタにより指される結論セットにジャンプする。ステップ524から、ステップ526で結論セットの探索が実行される。これはステップ516と同じである。ステップ526および522は両方とも、手続きの終了を表す出口ステップであるステップ528を指す。

【0195】

図19は、ステップ516および526の探索手続きを図式的に示す。ステップ530で手続きに入り、そこから制御はステップ532に移る。ステップ532は、結論セットが空であるかどうかを確かめるテストを実行し、空である場合、手続きはステップ534で終了し、そうでない場合、制御はステップ536に移る。ステップ536は、結論セット内の最初のキーを選択した後、制御をステップ538に移し、キーがパターンリスト内のあらゆるパターンを含むかどうかを確かめるテストを実行する。その通りである場合、制御はステップ540に移り、そうでない場合、制御はステップ542に移る。ステップ540は、キーおよびその属性を結論セットから結果リストに追加した後、制御をステップ542に移す。ステップ542は、現在のキーが結論セット内の最後のキーであるかどうかを確かめるチェックを実行し、その通りである場合、手続きはステップ544で終了し、そうでない場合、制御はステップ546に移る。ステップ546は、結論セット内の次のキーを選択し、制御をステップ538に戻す。

【0196】

図20は、範囲探索が本発明の実施形態を構成するデータベースを用いてどのように実行され得るかを図式的に示す。ノード560はその中に記憶されたビットパターンREを有し、ノード562はビットパターンOHを有すると仮定する。ノードがred、Fred、BillおよびJohnという単語で問合せされると、単語redおよびFredは包含パスに沿ってさらなるノード564へ向かって送られるのに対して、単語BillおよびJohnは除外パスに沿ってノード562へ送られる。ノード562は単語Billを

10

20

30

40

50

除外パスに沿って送り、単語 J o h n を、それがビットパターン「O H」を含むので、包含パスに沿って送る。

【0197】

ここでデータベースがワイルドカード探索を用いて、例えば探索ストリング「* r e d *」を用いて（ただし*は複数文字ワイルドカードを表す）問合せされる場合、ノード560は、562へ向かう除外パスは探索される必要がないため、探索はノード564を含むパスに制限されることが可能であると判定することができる。しかし、インデックスが探索項目「* l a p *」を用いて問合せされる場合、ノード560は、ビットパターンR Eが探索ストリング内にあるかどうかを判定することができず（それがワイルドカード部分の中に含まれているかもしれないため）、その結果、インデックスは包含および除外パスの両方

10

【0198】

キー内のビットパターンの位置が、データベース内でノードからノードへナビゲートするときにキーに沿って順次的に移動する必要がないことは留意されるべきである。

【0199】

キー圧縮

従来のデータベースシステムでは、キーの全体が各決定ノードに記憶される。これに対して、上記の本発明では、キーの一部が各決定ノードに記憶された。ただし、その部分の長さに制約はない。出願人は、キーの全体またはキーの一部をそのネイティブな形式で記憶するのではなく、マッピングプロセスの結果から導出されるそのキーの表現を記憶することによって、さらなる空間節約が得られることを認識した。C R C 3 2 および A d l e r 3 2 のようなハッシュ符号アルゴリズムは既知であり、キー値を符号化するマッピングの機能を提供する。これらのアルゴリズムは、長いビットシーケンスに作用して、ずっと短いビットシーケンスを導出する。マッピングアルゴリズムは、入力ビットシーケンスが常に同じであれば導出されるビットシーケンスが常に同じであるという点で決定論的である。しかし、マッピングは、得られる高度の圧縮に起因して一意的ではあり得ず、結果としていくつかの入力シーケンスがずっと短い1つの出力シーケンスにマッピングされることになる。それにもかかわらず、この結果を受け入れることにより、インデックスの設計を大幅に縮小することができる。こうして、候補キーKに対してツリーグラフをナビゲートするとき、Kの代わりに値A（K）が使用されることを除いては、従来のトラバース方法に従う。ただし、Aはそのツリーグラフに対して使用されるマッピング関数である。

20

30

【0200】

2つ以上のキーが同じ関数をマッピングすることがあるため、結論セットは、もしマッピング関数が使用されていなかったら到達されなかったはずの情報を含む可能性がある。したがって、マッピングなしのキーに対応しないものを除外するために、結論セット内のこれらのエントリを検査することが必要である。これは、結論セット内にマッピングなしのキーを保持し、もとのマッピングなしの探索キーにマッチしない項目を除外するためにこれらのキーをもとのマッピングなしの探索キーと比較することによってなされ得る。しかし、結論セット内のキーを第2の符号化アルゴリズムB（ただし、BはAとは異なる）で符号化することによっても、空間節約は達成され得る。両方のマッピングアルゴリズムの下で、2つの同一のマッピング値へ探索キーが不正にマッピングされる可能性は実に非常に小さく、結果として、誤ったマッピングが得られる可能性は非常に低い。実際、誤ったマッチの確率はおよそ R^2 分の1である（ただし、Rは関数マッピングの値域サイズ）。マッピングされたキーAおよびBが両方とも結論セットに記憶される。というのは、これにより、結論セットが最大サイズに達した場合にその結論セットの再マッピングが容易になるからである。

40

【0201】

キーをインデックスに挿入するため、キーは第1のハッシュ関数を用いて符号化される。その後、インデックスは、上記のナビゲーション方法または従来の方法により、符号化されたキーを用いて、結論セットに到達するまでナビゲートされる。その後、キーは、非符

50

号化形式で、または第2のハッシュ関数を用いて符号化されて、結論セットに挿入され得る。

【0202】

削除も同様であり、キーはそのネイティブな形式で、または第2のハッシュ関数を用いたキーへの作用の結果として符号化された値の形式を用いて、結論セットから削除される。

【0203】

したがって、データベースキーを、それらのサイズが大幅に縮小されるような形式で符号化する効率的な方法を提供することが可能であり、この方法では、相異なる関数を用いた二重符号化の使用は、誤ったマッチのリスクを許容可能なほど小さい頻度に縮小する。

【0204】

限定されたキー

図21は、本発明の実施形態を構成するインデックスを図式的に示す。概念的には、インデックスは従来のBツリーインデックスのもののように見える。しかし、ネイティブなキー全体が決定ツリーの各ノード内に含まれるのではなく、キーの符号化バージョンが使用される。ここでキーはモジュロ算術関数を用いてマッピングされる。また、キーの一部のみが無限定である場合、キーの一部のみが本発明に従ってマッピングされればよいということになる。したがって、キーが探索、削除または挿入のためにインデックスに提示されるとき、その使用前に標準的なモジュロ関数がキーに適用される。モジュロを文字ストリングキータイプに適用するには、ストリングを複数バイトの非常に大きい符号なし絶対値整数として扱い（最初の文字がその整数値の最上位バイトである）、それからモジュロを適用することが必要である。

【0205】

したがって、キーをインデックスに挿入するには、キーはインデックスに対して選択されたモジュロを用いて符号化される。その後、インデックス構造体は、標準的なインデックス方法、または本明細書に提示される方法により、符号化されたキー値を用いて結論セット（リーフブロック）まで、前述のようにナビゲートされる。その後、キーは、そのネイティブな形式、またはその符号化された形式のいずれかで、結論セットに挿入される。削除および探索も同様に実行される。つまり、探索において、結論セットに到達した後、キーは、結論セット内のすべてのキー値と、それらのネイティブな形式、または符号化された形式のいずれかで比較される。

【0206】

図22は、オペレータが、上記のような厳密マッチではなく範囲マッチを実行したいときに起こり得る可能性を図式的に示す。範囲の最小および最大探索キーはそれぞれL（下位）およびU（上位）と表される。これらのキーは、図22に示すように、無限定すなわちマッピングなしの探索空間580に存在する。次にキーは、インデックスの限定された範囲内へそれらを変換するために、モジュロプロセス582を通じてマッピングされる。キーの順序がマッピングプロセスの後に保存されることは保証され得ない。これは2つの可能性を生じる。第1に、図式的に584で示されるマッピングされたセットに示されるように、マッピングされた下位キーはマッピングされた上位キーより小さい値を有する。この場合、インデックスは、標準的なインデックスナビゲーション方法を用いて結論セットまでナビゲートされる。しかし、同じく予想され、マッピングされたセット586に図式的に示されているように、マッピングされた下位キーMLは、マッピングされた上位キーMUより大きい値を有することがある。この場合、探索は、符号化された最小キーML以上の符号化値を有するすべての結論セットに対して行われなければならない。この探索は標準的なインデックスナビゲーション方法を用いて実行される。さらに、探索は、符号化された最大キーMU以下の符号化値を有するすべての結論セットに対して行われなければならない。この場合も、これは標準的なナビゲーション方法を用いて実行される。

【0207】

この技法は、キーどうしの差がモジュロ符号化方式の定義域より小さいときにのみうまく働き得ることが留意されるべきである。したがって、自明な例として、キーがモジュロ1

10

20

30

40

50

0を用いてマッピングされた場合、探索範囲内の下位キーと上位キーの間の差は10より小さくなければならない。

【0208】

一時キー

図23は、本発明の実施形態を構成する結論セット／リーフブロック600のデータ構造を図式的に示す。結論セットはエイジベース602を含む。エイジベース602は、結論セット600へのいちばん最近の更新の日時を表す。結論セットはまた、複数のキーエントリ604、606および608も含む。簡単のためキーエントリ608だけを考えると、これはさらに3つの部分、すなわちエントリエイジ608A、エントリキー情報608Bおよびエントリ属性情報608Cに分割される。

10

【0209】

一時エントリが作られると、一時エントリはそれに関連するエントリエイジデータ608Aを有する。このデータは、エイジユニットおよびエイジ限界を指し、これらは、インデックスに対して全体として設定されることも可能であるが、随意選択的に、個々の結論セットごとに設定されることも可能である。エイジユニットは、1エイジユニット内の秒数を指定する。例えば、60という値は1分を意味し、一方86400は1日を意味する。エイジ限界は、その後にキーがインデックスから失われてもよいエイジユニット数を表す。これは通常、1～254の範囲にあり、それにより8ビットワード内に収まることが可能となる。つまり、キーが31日後に期限切れになるよう設定される場合、エイジユニットは86400秒に設定され、エイジ限界は31に設定される。キーが1日後に期限切れになるためには、エイジユニットおよびエイジ限界はそれぞれ3600秒および24に設定されればよい。

20

【0210】

各エントリは、効率のために1バイトで表されるエントリエイジを含み、これは、この例では、値0と255の間に設定されることが可能であるが、他の値範囲も適宜選択され得る。0はエントリが新しいこと、またはエイジベースよりも1エイジユニット未満しか経過していないことを示す。255はエントリが廃用になっていることを示す。

【0211】

結論セットにおけるそれぞれの更新および挿入ごとに、以下のシーケンスがたどられる。まず、現在の日時が記録され、エイジベースと現在の日時の間に経過したエイジユニット数が計算される。次に、結論セット内のあらゆるエントリで、そのエントリエイジが、経過したエイジユニット数だけインクリメントされる。計算されたエントリエイジが255以上であるエントリは、そのエントリエイジが255に設定される。このようなエントリはその後、再使用のために利用可能となる。次に、エイジベースが、手続きの最初に記録された日時に設定される。挿入されるべきキーが、結論セット内の利用可能なスロットに配置され、エントリエイジが0に設定される。利用可能なスロットは、まだ割り当てられていないスロット、または（そのエントリエイジが255に達したために）廃用になったスロットのいずれでもよい。さらに、更新されたエントリはそのエントリエイジが0にリセットされる。

30

【0212】

上記の実施形態では、エントリエイジは単一バイトとして記憶された。これは、あるタイプのデータが廃用になり得る254種類の期間の範囲を与える。1バイトより多くのバイトを使用して、より広い範囲のエイジ限界を与えることも可能であるが、記憶領域オーバーヘッドが増大する。別法として、エントリエイジとして1バイト未満を使用することも可能であり、それにより記憶領域オーバーヘッドは減少するが、個別に定義可能なエイジ限界の範囲もまた減少する。

40

【0213】

こうして、改善されたデータベースを提供することが可能である。

【0214】

決定グラフの分割

50

前述のように、決定グラフ（インデックス）の全体を半導体メモリ内に保持することが好ましい。これは高速アクセスを可能にするからである。しかし、データベースを動作させるハードウェアプラットフォームの能力またはデータベースのサイズが、インデックスの全体を半導体メモリなどの中に保持することが可能でないようなものであることがある。これは、例えば、マシンがマルチタスク動作を行い、他の機能を実行することが予想されるからであるかもしれない。

【0215】

ハードディスクドライブのような大容量記憶デバイスは、所定サイズのブロックに分割され、ブロックサイズ未満のいかなるサイズのファイルでも1ブロックを占有することは既知である。したがって、物理メモリは、ブロックサイズの整数倍単位で使用される。決定グラフの読み出しおよび書き込みをこのようなメモリで行うとき、決定グラフの全体、または可能な限り大部分が、単一ブロックに書き込まれるように、決定グラフが構造化されることが好ましい。こうして、決定グラフは、1ブロックのメモリ内に存在する可能性が高くなる。しかし、インデックスがますます増大すると、キー数が増えるにつれて決定グラフは大きくなる。それゆえ、決定グラフを単一ブロック内に収容することが不可能となるときの来る。したがって、複数ブロックにわたる決定グラフを収容する方法が要求される。

10

【0216】

図24を参照して、メモリブロック700はコンピュータシステムのハードディスク上の1ブロックの物理サイズを表し、ブロックは個々の部分702、704、706および708を含むように細分され、とりわけ、その各部分は決定グラフ内の関連するノードに関するデータに関係すると仮定する。さらに、インデックスは今、ブロック700が完全にフルであるようなサイズに達していると仮定する。

20

【0217】

ここで、データベース挿入操作の結果として、さらなるノードを決定グラフに挿入することが要求されていると仮定する。ノード挿入はいかなる時点でも起こり得る。前述のように、各決定ノード（図3）は、決定値と、グラフ内の次のノードを示すローポイントおよびハイポイントとを備える。結果として、いかなる既存ノードの後にでも新しいノードを挿入することができるようなグラフに対する局所的補正を行うことが可能である。例えば、挿入の結果として、データ項目706によって表されるデータを有するノードの後に新しいノードを挿入することが要求されていると仮定する。まず、新しいノードが挿入されると、ノード706の出力ポイントの1つが新しいノードを指すように修正される。挿入プロセスの一部として、新しいノードの出力ポイントもまた、決定グラフが破壊されないように正しく設定される。しかし、新しいノード710に関係するデータをメモリブロック700に記憶することがもはや不可能である場合、新しいノード710に関係する詳細は別のブロック712に記憶されなければならない。その新しいノードに関係するさらなるノードもまた作成され、情報はメモリブロック712に記憶され得る。

30

【0218】

後に、より以前から存在するノード708に関係する新しいノードを挿入することが要求される場合、再びメモリブロック700はフルになり、その結果、さらなる新しいノード714の詳細はメモリブロック700の外部に記憶されなければならない。このプロセスにより、決定グラフは増大することが可能となるが、それぞれのフル（親）ブロックから、ほとんど空のメモリブロックが多数生じる可能性がある。

40

【0219】

図25は、インデックスを増大させる代替方法を示す。再び、メモリブロック700はフルであり、今、新しいノードを挿入することが要求されていると仮定する。前と同様、新しいメモリグラフブロックが作成される。しかし今度は、親の決定グラフが均等な部分に分割される。一方の部分は旧ブロック700内に配置されるが、他方の部分は新ブロック712内に配置される。

【0220】

50

図25を参照して、最初にインデックス720はメモリブロック700内に記憶されており、そのブロック700が現在フルであると仮定する。インデックス挿入により新しいノード722が作成され、このプロセスでインデックスはメモリブロック700のサイズを超えたと仮定する。しかし、インデックス増大を制御する方法では、インデックスの構造が、それをインデックスブロック間で実質的に均等に分割するように解析される。例えば、図25に示すインデックスの部分を取ると、ノード724から出発して、一点鎖線726で囲まれる10個のノードは、ノード724の左に延びるパスによって到達することができ、これに対して一点鎖線728で囲まれる12個のノードは、ノード724の右に延びるパス中に見出されることがわかる。インデックスを分割するための適当な候補ノードを見つけるために、テストノードを選択した後、それにぶら下がっているノードを計算することによって、アルゴリズムはインデックス構造体を再帰的に走査する。図25に示される例では、ノード724および線726で囲まれるノードは旧メモリブロック700内に配置されるが、一点鎖線728で囲まれるノードは新メモリブロック712内に配置されることになる。このプロセスは、各メモリブロックが最適かつ公平に利用され、実質的に空のメモリブロックの急増を止めることを保証する。

10

【0221】

修飾子

しばしば、インデックス内の探索項目は、2個以上の探索基準を用いて実行されることが所望される。これらの基準は、一般にブール代数の原理に従って組み合わせられる。

20

【0222】

問合せを個々の部分に分割し、1つの部分を実行してから次の部分を片づけ、最後に結果を組み合わせるようにすることが可能である。しかし、これは非常に計算量集約的となり、ハードディスクのようなブロックデータ記憶装置に対する多くの読み書き操作を必要とすることがある。例えば、100ヒットを返した1つの問合せがある場合、インデックス問合せにより指されるデータをフェッチするために、さらに100回の遅いディスクI/O操作を必要とする可能性がある。

【0223】

問合せによりフェッチされるすべてのデータが必要であるとは限らない。例えば、データベースに対する問合せが、1997年に登録されたメルセデス製のすべての車を見つけるために行われた場合、1997年に登録された車に対する問合せは何千もの結果を返すであろう。しかし、これらのうちの少数のみがメルセデス製のものである。さらに、多くの問合せについて、不要な結果の数を減少させるように問合せの順序を変更することは可能でないかもしれない。

30

【0224】

この問題は、連結キーを使用することにより克服することができる。しかし、従来、このような連結キーは、主探索キーおよび副探索キーの両方の全体を、組み合わせられた連結キー内に維持しており、結果として、組み合わせられたキーは特に長くなる。

【0225】

出願人は、主探索キーに修飾子を関連付けること、特に修飾子を付加することによって、探索パフォーマンスを改善することができることを認識した。修飾子は、第2のすなわち後続の探索項目を直接表すのではなく、むしろその後続項目の短縮形式である。本発明では、修飾子は、それぞれの後続の探索項目に対してハッシュ関数を実行することにより生成される。データベースに対するデータの挿入または削除を行うために使用されるのと同じハッシュ関数が、データベースに問合せを行うために使用される。したがって、データベースの作成中、インデックスは主キーとともに（前と同様）、および0個以上の修飾子キーとともに作成される。

40

【0226】

使用時に、インデックスは主キーを使用することによりナビゲートされ、データは結論セットに通常のように記憶される。しかし、各修飾子キーは単一バイトに、すなわち0～255の範囲内の数にハッシュ符号化され、結論セットにキーエントリとともに記憶される

50

。

【0227】

問合せ中に、主キーには0個以上の修飾子キーが提供される。そしてインデックスは、前と同様に主キーを用いてナビゲートされる。最後に、いったん正しい1つまたは複数の結論セットに到達した後、結論セットは、主キー値および修飾子キーハッシュ符号を結論セット内のエントリと照合することによって走査される。主キーの全体およびすべての修飾子ハッシュ符号にマッチするエントリのみが返される。

【0228】

複数の修飾子キーが同じ単一バイトのハッシュ符号にマッピングされる可能性があることが留意されるであろう。したがって、不要なデータが返される小さいが有限の確率が存在する。しかし、良好なランダム化ハッシュ符号化が使用されると仮定すると、誤ったマッチの確率は255分の1（すなわち0.5%未満）である。本発明は、すべての正しい結果を返すことを保証するが、小さい割合の不正な結果も返すことがわかる。したがって、返されるすべてのキーヒットをその後データ記憶装置内で検査して、不要なエントリがあればそれを除去しなければならない。しかし、インデックスのサイズの縮小および問合せプロセスの単純化は、返されるヒットの追加的解析を埋め合わせて余りある。この利点は、修飾子の数が増えるほど不正なヒットの確率が減少することを考慮すると、より多くの修飾子を使用されるにつれて増大する。誤った結果の確率は、理想的な状況の下では、 $(1/255)^N$ である。つまり、2個の修飾子（ $N=2$ ）の場合、誤った結果の確率は65025分の1すなわち0.002%未満である。3個の修飾子の場合、不正なデータ項目を返す確率は約1400万分の1にまで減少する。

【0229】

探索キーのサイズ、および結論セット内のデータ記憶要件が大幅に縮小されることが理解されるであろう。したがって、各項目の長さが8バイトまでになり得る8個の項目を探索しようとしたとすると、従来ならばこれは各探索キーごとに64バイトのデータの記憶領域を必要とするであろう。本発明では、この要件はたった15バイトのデータまで縮小される。

【0230】

データベースに問合せを行うとき、各修飾子キーはオプションであり、それらのキーのうちの一部または全部が提供されることもあり、全く提供されないこともあり得る。というのは、それらのキーはインデックス構造体のナビゲーションに影響しないからである。修飾子キーは、結論セットを走査するときに返されるヒットの数にのみ影響する。したがって、提供される修飾子の数が多いほど、問合せ結果は正確になる。

【0231】

この修飾子技法は、任意のインデックス構造体、例えばBツリーのような既知の構造体に適用可能であることが留意されるべきである。図26は、本発明のこの態様による探索キーのフォーマットを図式的に示す。キーの最初の部分は主キーデータを含み、キーの残りの部分は修飾子Q1、Q2などからQ7までを含み、これらのそれぞれは、もとの追加的キーにハッシュ関数を作用させることにより生成される、長さが1バイトのワードである。

【0232】

パターンマッチング

しばしば、データベース内でパターンマッチング問合せを行うことが所望される。例えば、探索キーの全体に対する探索ではなく、キーの一部を用いて探索が実行される。

【0233】

このような探索の効率は、キー数が決定グラフ内の各ノードにどの程度うまく分割されているかに左右される。

【0234】

ASCII文字セットの問題点は、それは255個の文字を含むが、一部の文字の現れる可能性が非常に高く、それにもかかわらず二重文字の組合せは特にまれであることである

10

20

30

40

50

。

【0235】

出願人は、ASCII文字セット（または実際には同等のセット）がより小さいセットにマッピングされれば、パターンマッチングを大幅に改善することができることを認識した。

。

【0236】

出願人は、単一文字が4ビット（0～15）に収まる一方で、2文字シーケンスを単一バイト内に収めることができるように、セットを16文字に縮小するいくつかのマッピングを定義した。

【0237】

そして、これらの修正された文字セットは、単一のインデックスでキー数を分割するために使用可能である。

【0238】

文字セットのマッピングは、挿入、削除または問合せ操作のための決定グラフのナビゲーションの直前にキーに適用される。正確な問合せ結果が要求されるか、それとも蓋然的な問合せ結果が要求されるかに応じて、もとのキー値または変換されたキー値（より小さいサイズである）のいずれが結論セットで記憶されることも可能である。2つの文字セットの例を以下に与える：

【0239】

【表2】

音声セット

ASCII文字	マッピングされた値
母音（A、E、I、O、U）	除去
数字（0、1、2、3、4、5、6、7、8、9）	除去
句読文字	除去
空白、タブ	0
B	1
C、K、Q	2
D	3
F	4
G、J	5
H	6
K	7
L	8
M、N	9
P	10
R	11
S、Z	12
T	13
V	14
W	15
X、Y	除去

【表3】

類似性セット

A S C I I 文字	マッピングされた値
空白、タブ	0
句読文字	1
数字 (0、1、2、3、4、5、6、7、8、9)	2
母音 (A、E、I、O、U)	3
B、D	4
C、K	5
F、P	6
G、J	7
H、Y	8
K、L	9
M、N	1 0
P、Q	1 1
R、S	1 2
T	1 3
V、W	1 4
X、Z	1 5

10

【0240】

これらの方式のそれぞれにおいて、問合せが大文字と小文字を区別しないように、文字は大文字に変換される。さらに、改行および復帰のような非印字文字はマッピングから除去される。

20

【0241】

この縮小セットを利用することにより、パターンマッチングのパフォーマンスを大幅に改善することができる。キーが、挿入、問合せまたは削除操作の前にマッピングされた後、マッピングされたキーを用いてインデックスがナビゲートされる。

【図面の簡単な説明】

【図1】 本発明の図式的概観である。

【図2】 決定グラフの論理構造の図式的表示である。

【図3】 決定ノードの構造を図式的に示す。

30

【図4】 結論セットの構造を図式的に示す。

【図5】 結論セット内のエントリの構造を図式的に示す。

【図6】 厳密探索手続きを図式的に示す。

【図7】 範囲探索手続きを図式的に示す。

【図8】 キーを挿入する手続きを図式的に示す。

【図9】 キーを削除する手続きを図式的に示す。

【図10】 厳密キー問合せの手続きを図式的に示す。

【図11】 範囲問合せの手続きを図式的に示す。

【図12】 決定群G、決定値Vを有するノードを最小および最大キー範囲までトラバースし、結果を返す手続きを図式的に示す。

40

【図13】 決定グラフの一部の構造を示す。

【図14】 修正決定ノードの論理構造を示す。

【図15】 キーを挿入する手続きを示す。

【図16】 キーを削除する手続きを示す。

【図17】 厳密問合せマッチの手続きを示す。

【図18】 パターンリストを有するツリーをトラバースし結果を返す手続きを示す。

【図19】 パターンリスト(L)の探索の手続きを示す。

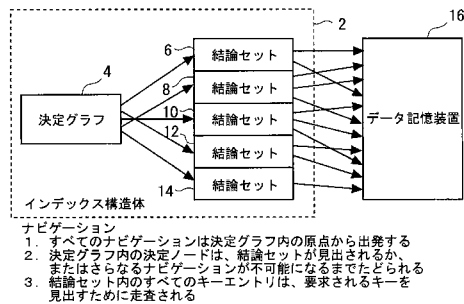
【図20】 パターンマッチプロセスがデータをソートする際にどのように役立つかの例である。

【図21】 限定インデックスを有する決定グラフの図式的例示である。

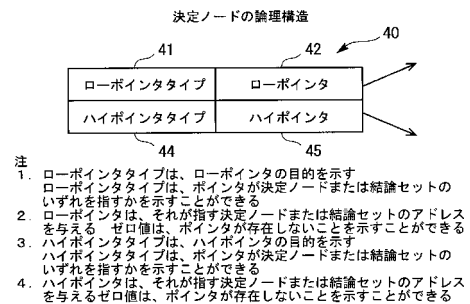
50

- 【図 2 2】 修正された結論セットの構造を示す。
 【図 2 3】 決定グラフを分割する第 1 の方法を図式的に示す。
 【図 2 4】 決定グラフを分割する第 2 の方法を図式的に示す。
 【図 2 5】 内部に修飾子を有する複合キーの構成を図式的に示す。
 【図 2 6】 探索キーのフォーマットを図式的に示す。

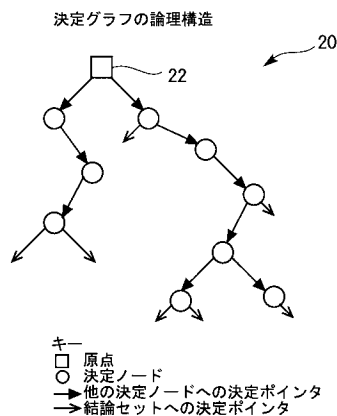
【図 1】



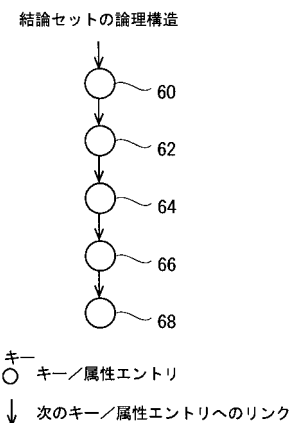
【図 3】



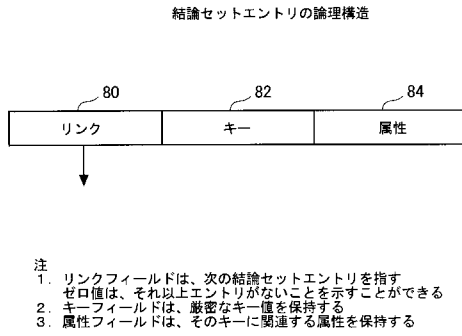
【図 2】



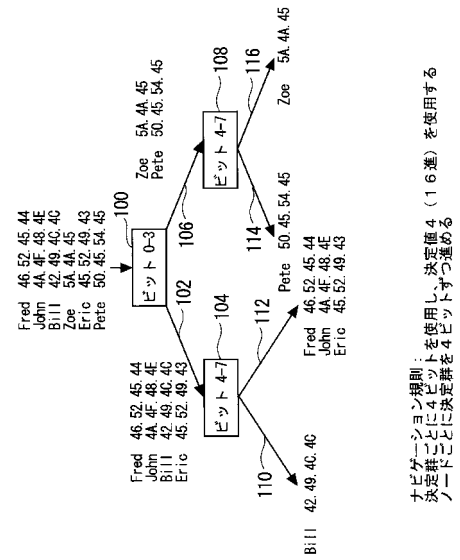
【図 4】



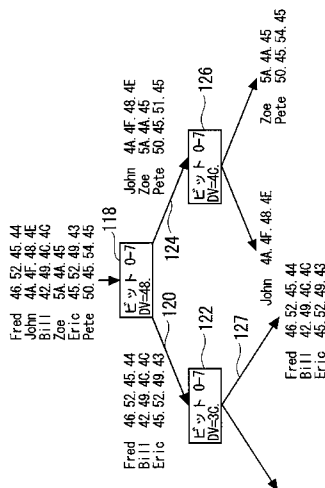
【図 5】



【図 6】

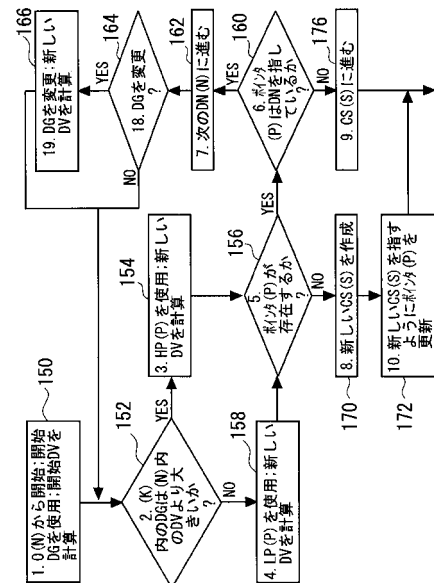


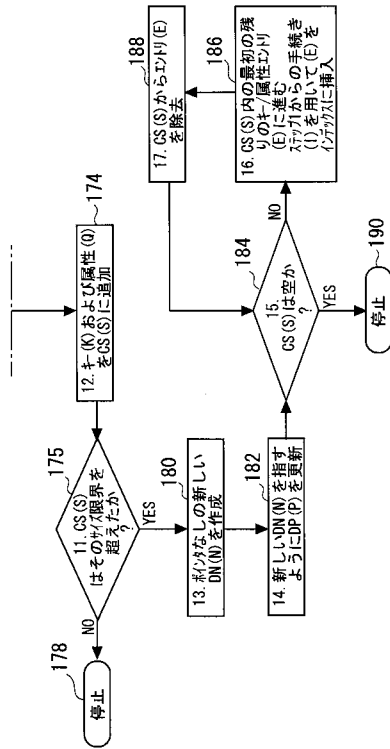
【図 7】



【図 8】

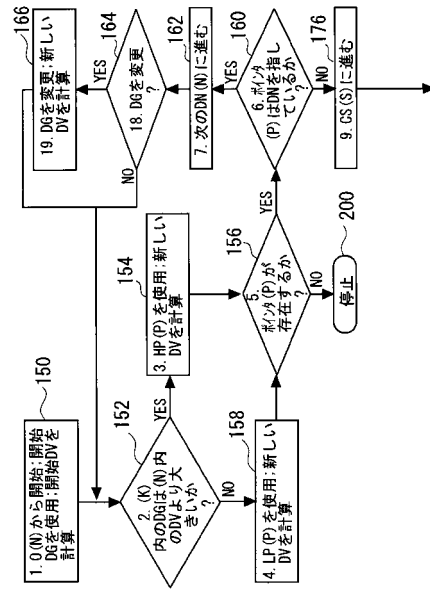
キー (N) を属性 (N) とともに挿入する手続き (I)





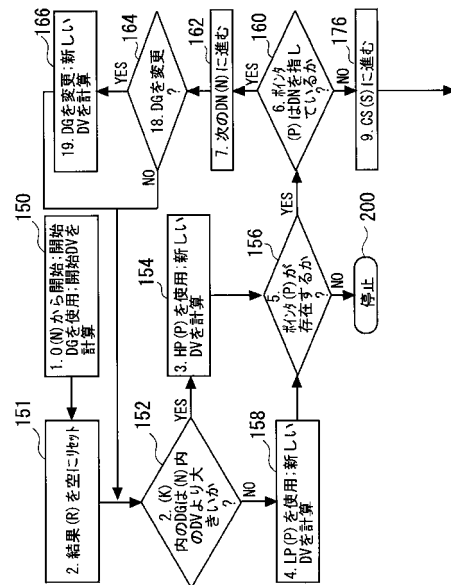
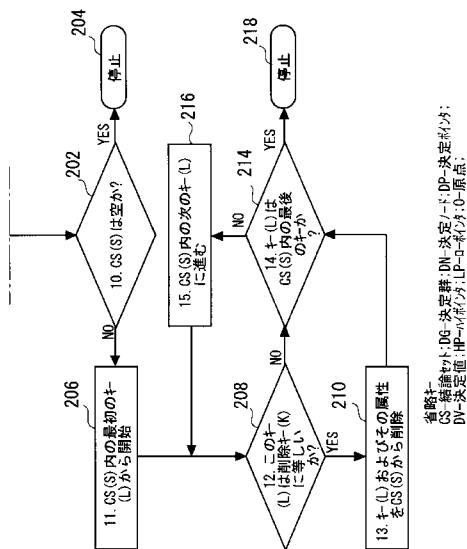
【図 9】

キー (K) を削除する手続き (D)

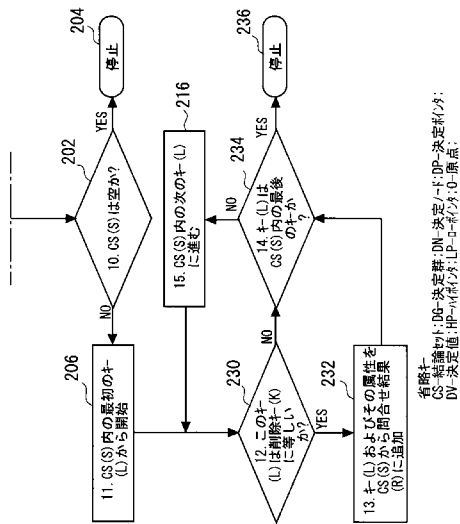


【図 10】

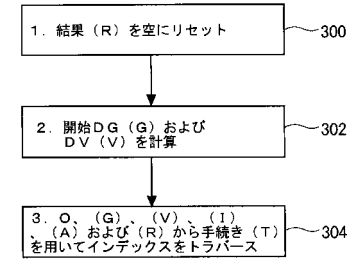
キー (K) の秘密問合せを行い、結果 (R) を返す手続き (E)



【図 11】

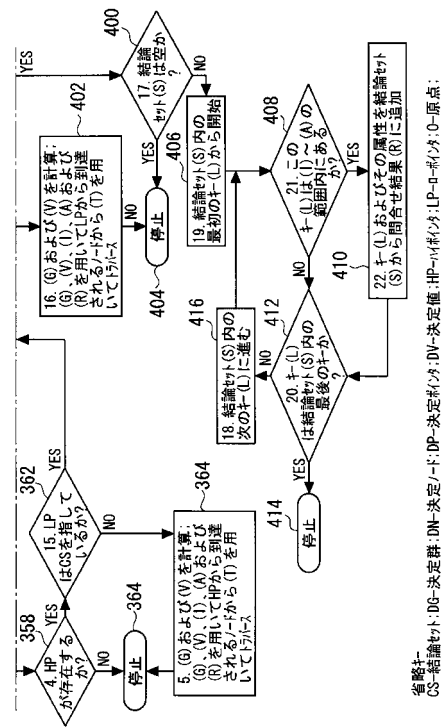
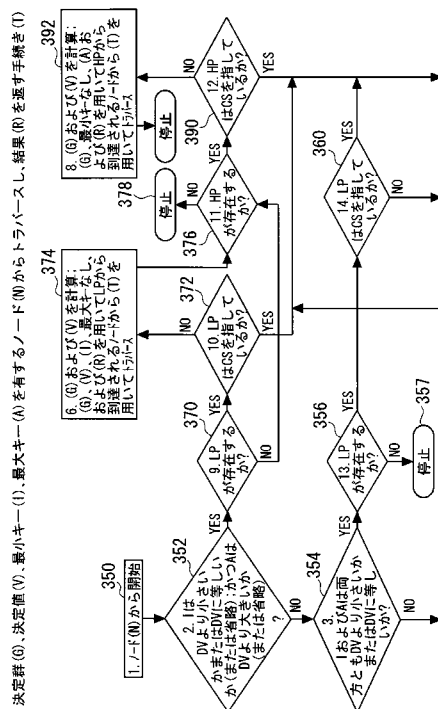


最小キー (I)、最大キー (A) の範囲問合せを行い、
結果 (R) を返す手続き (G)

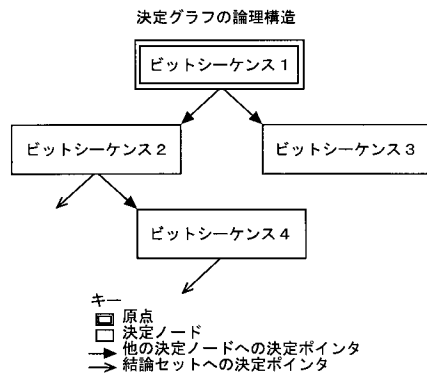


有路キー
CS: 結論セット; DG: 決定群; DN: 決定ノード; DP: 決定ポイント;
DV: 決定値; HP: ハイポインタ; LP: ローポインタ; 0: 原点;

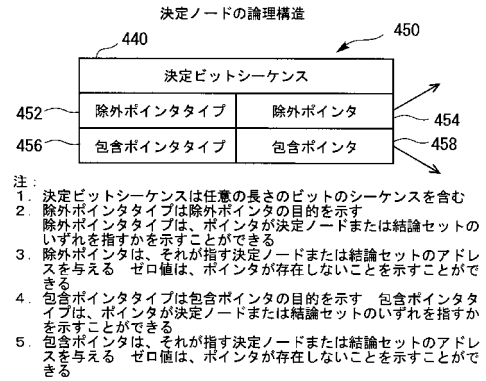
【図 12】



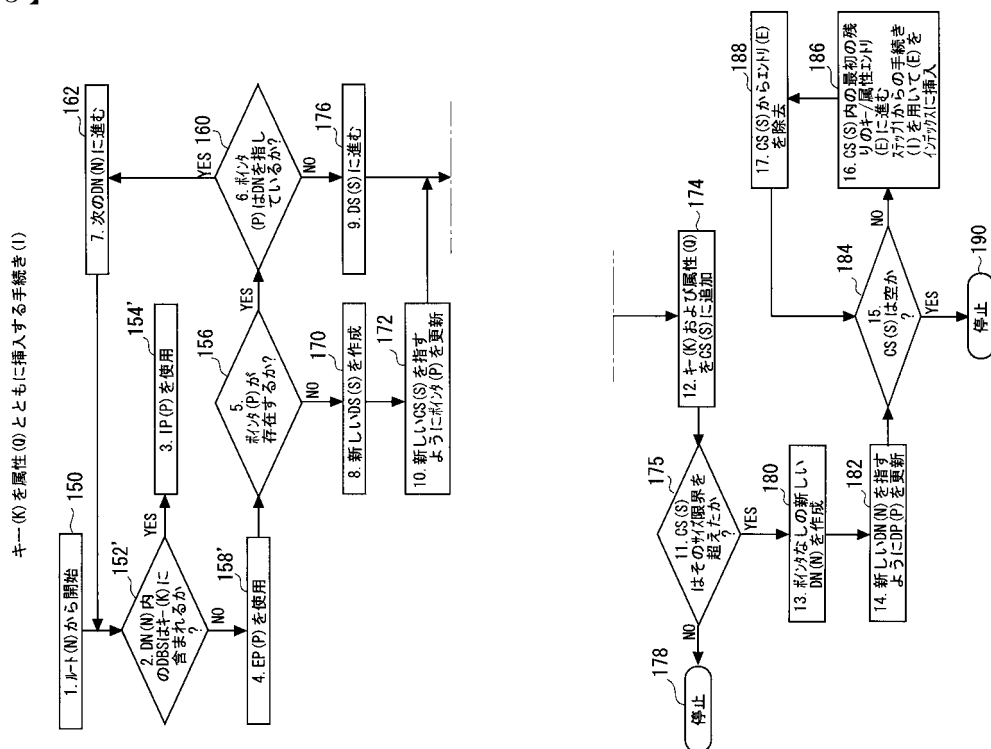
【図 13】



【図 14】

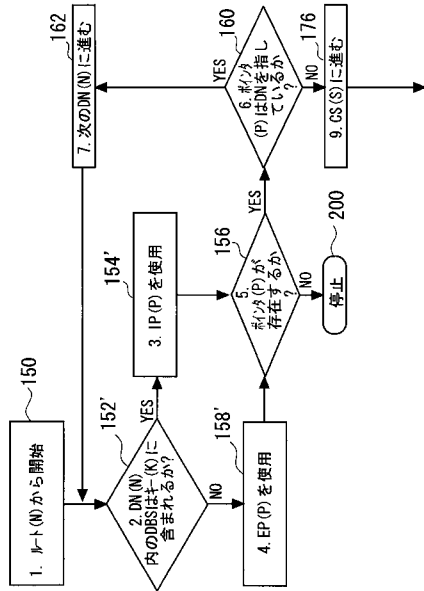


【図 15】



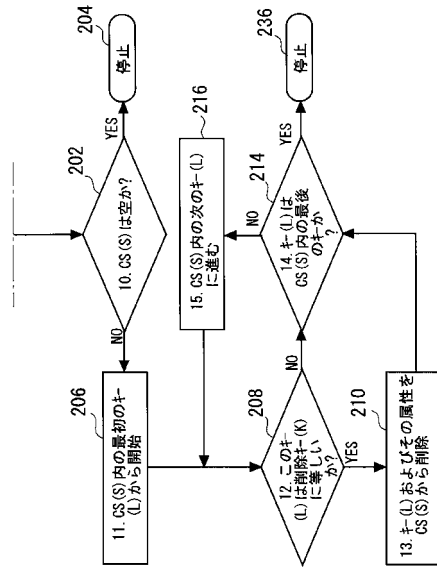
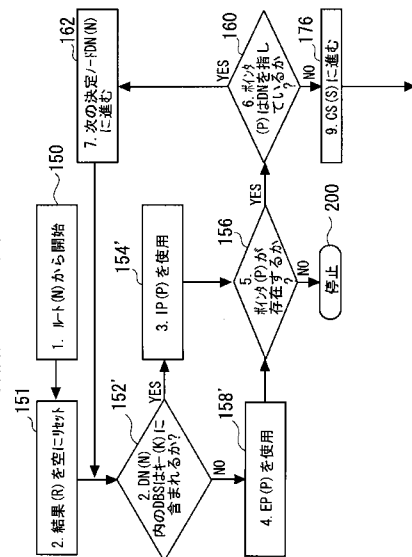
【図 16】

キー (K) を削除する手続き (D)

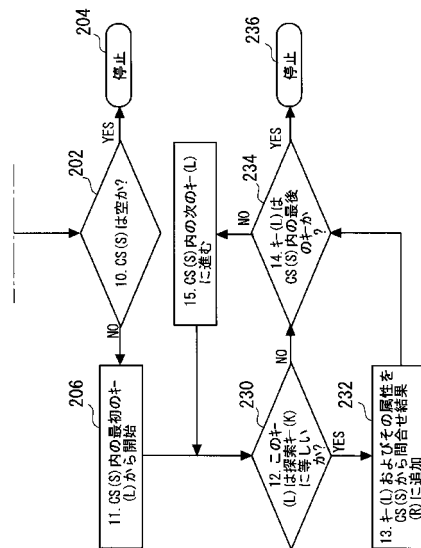


【図 17】

キー (K) の秘密問合せを行い、結果 (R) を返す手続き (E)

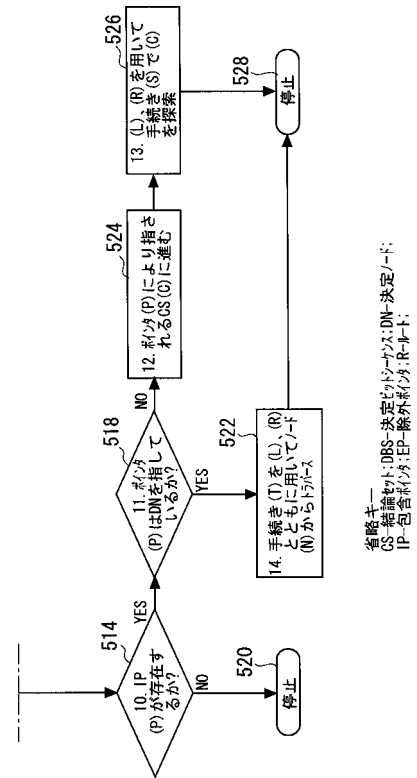
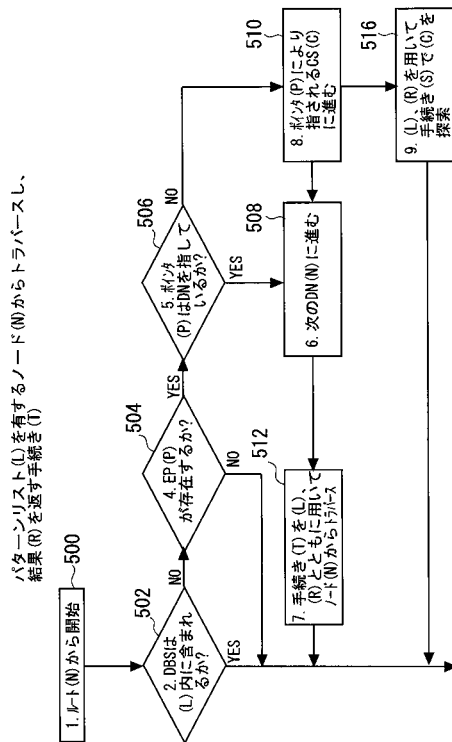


省略キ
CS-結果セット: DBS-決定ノードノード: DN-決定ノード: IP-包含キー: EP-除外キー: R-結果:

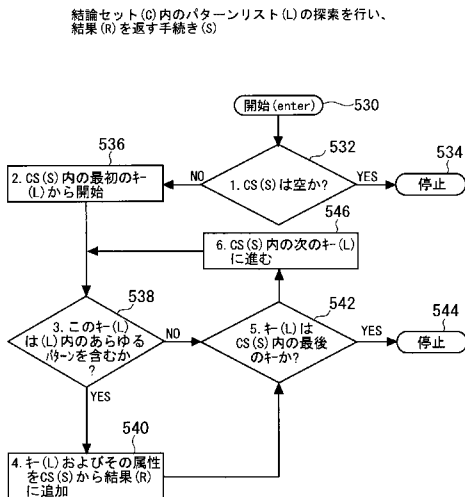


省略キ
CS-結果セット: DBS-決定ノードノード: DN-決定ノード: IP-包含キー: EP-除外キー: R-結果:

【図 18】

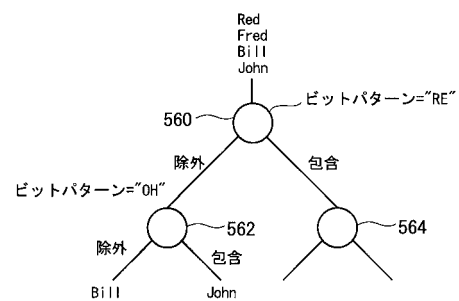


【図 19】

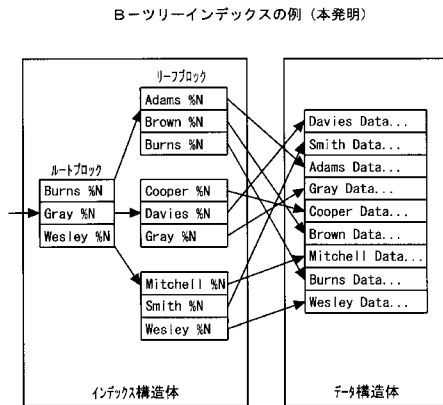


省略キー
 CS-結論セット; DBS-決定ビットパターン; DN-決定ノード;
 IP-包含ポインタ; EP-除外ポインタ; R-ルート;

【図 20】



【図 2 1】



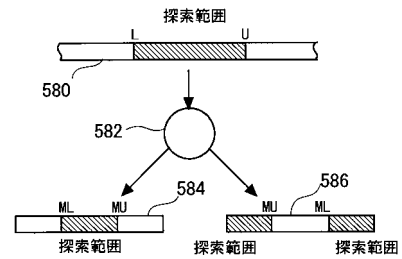
ナビゲーション

1. すべてのナビゲーションはルートブロックから出発する。
2. すべての非リーフインデックスブロックは比較のためにモジュンを用いて符号化されたキー値を含む。
3. 各インデックスブロックを通してナビゲートするには、探索キーよりも大きいかまたはそれに等しい最初のキーが、ブロックに含まれるキー全体の線形探索により見つけれられる。そのブロック・エントリに関連するポインタがとられる。
4. リーフインデックスブロックは、データ構造体へ進む前の最終比較のためにモジュンを用いて符号化されたキー値、またはレギュラーなキー値を含むことができる。

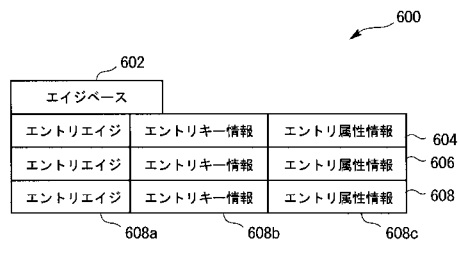
注

1. インデックスがN個のキーを収容するまで増大したら、その後に入力されるキーは既存のリーフブロックを共有することになる。

【図 2 2】



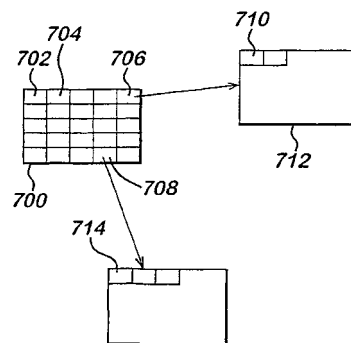
【図 2 3】



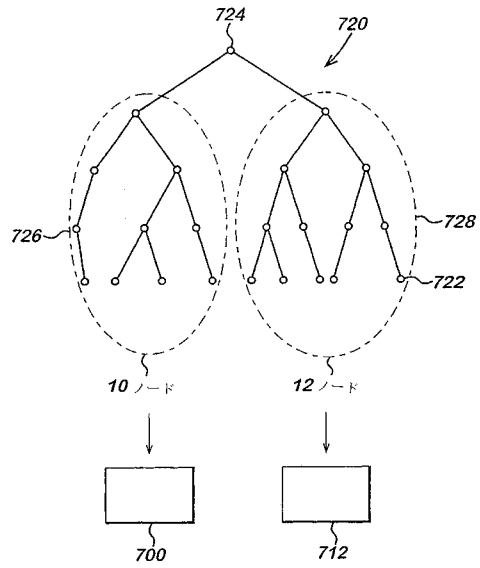
ナビゲーション

1. エイジベースはいちばん最近の挿入または更新の日時を記録するエントリエイジ (0 ~ 255) はエイジベースに宛するエントリのエイジをエイジユニット単位で記録する。値 0 は最新のエントリを示す。255 は、その空間が新しいエントリのために再使用可能な廃用エントリを示す。
- 2.

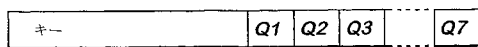
【図 2 4】



【図 2 5】



【図 2 6】



フロントページの続き

(72)発明者 ポーリー, ダンカン ガンサー
イギリス エスエヌ13 8エーエー ウィルトシア ボックス ボックス ハウス コッパーア
イ リミテッド

合議体

審判長 長島 孝志

審判官 加内 慎也

審判官 飯田 清司

(56)参考文献 特開2000-76106 (JP, A)
特開平5-120339 (JP, A)
特開平3-3083 (JP, A)
特開昭63-285629 (JP, A)
特開平5-334153 (JP, A)
特開平10-40255 (JP, A)
特開平9-36747 (JP, A)

(58)調査した分野(Int.Cl., DB名)

G06F12/00

G06F17/30