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

特許第4126843号
(P4126843)

(45)発行日　平成20年7月30日(2008.7.30)　(24)登録日　平成20年5月23日(2008.5.23)

(51)Int.Cl.　　F I
G06F 12/00 (2006.01)　　G06F 12/00 505
G06F 17/30 (2006.01)　　G06F 12/00 518A
　　G06F 17/30 414A

請求項の数 2　(全 16 頁)

(21)出願番号　特願2000-101211 (P2000-101211)
(22)出願日　平成12年3月31日 (2000.3.31)
(65)公開番号　特開2001-282599 (P2001-282599A)
(43)公開日　平成13年10月12日 (2001.10.12)
審査請求日　平成16年8月24日 (2004.8.24)

(73)特許権者　000005108
株式会社日立製作所
東京都千代田区丸の内一丁目６番６号
(74)代理人　100100310
弁理士　井上　学
(72)発明者　原　憲宏
神奈川県川崎市幸区鹿島田８９０番地　株式会社日立製作所　ビジネスソリューション開発本部内

審査官　上嶋　裕樹

(56)参考文献　特開平０４－１１２２４０（ＪＰ，Ａ）
特開平０８－０８７５１１（ＪＰ，Ａ）
特開昭６１－１８４６３５（ＪＰ，Ａ）
最終頁に続く

(54)【発明の名称】データ管理方法および装置並びにデータ管理プログラムを格納した記録媒体

(57)【特許請求の範囲】
【請求項１】
一つのキーが、該キーを含むデータに関する識別情報をデータポインタとして管理し、キー値に基づいて上記キーに関連するデータポインタを見つけるためにＢ木インデクスを用いたデータ管理装置において、
上記一つのキーと同一のキーを含む複数のデータに関する複数のデータポインタを格納するデータ構造であり、そのキーを含むインデクスエントリが格納されるリーフノードとは異なるデータ構造であり、そのデータ構造はそのキーを含むインデクスエントリからポイントされ、前記同一キーを含む複数のデータに関する複数のデータポインタへのアクセスの際に排他を取得することは無い重複データ構造を有する記憶手段と、
あるキーに関連するデータポインタを追加もしくは削除する際に、そのキーに関連する複数のデータポインタを格納する前記重複データ構造に対し、そのデータポインタを追加もしくは削除するための情報としてテーブルデータ関連情報と前記重複データ構造へのポインタと前記キーを含む第１のログレコードを取得する手段と、
リーフノードに格納されたそのキーを含むインデクスエントリを更新する手段と、
前記重複データ構造へのそのデータポインタの追加もしくは削除が行われたことを示す情報を含む第２のログレコードを取得する手段と、
前記重複データ構造に、そのデータポインタを追加もしくは削除する手段と、
上記データポインタの追加もしくは削除を伴う処理結果を取り消す際に、
上記取得したログレコードを受け取り、受け取ったログレコードが前記第２のログレコ

ードであった場合には、その前記第２のログレコードを保持し、受け取ったログレコードが前記第１のログレコードであった場合には、第２のログレコードを保持しているかを確認し、第２のログレコードを保持している場合には、前記重複データ構造へのそのデータポインタの追加もしくは削除が完了したと判断する手段と、
上記判断において受け取ったログレコードが前記第１のログレコードであった場合にもかかわらず第２のログレコードを保持していないことを確認することにより前記重複データ構造へのそのポインタの追加もしくは削除が完了していないと判断した場合、第１のログレコードに含まれる前記重複データ構造へのポインタが示す前記重複データ構造をアクセスし、第１のログレコードに含まれる前記テーブルデータ情報を追加もしくは削除する手段を有することを特徴とするデータ管理装置。

【請求項２】
一つのキーが、該キーを含むデータに関する識別情報をデータポインタとして管理し、キー値に基づいて上記キーに関連するデータポインタを見つけるためにＢ木インデクスを用いたデータ管理プログラムを記録した計算機読み取り可能な記録媒体において、
上記一つのキーと同一のキーを含む複数のデータに関する複数のデータポインタを格納するデータ構造であり、
そのキーを含むインデクスエントリが格納されるリーフノードとは異なるデータ構造であり、そのデータ構造はそのキーを含むインデクスエントリからポイントされ、前記同一キーを含む複数のデータに関する複数のデータポインタへのアクセスの際に排他を取得することは無い重複データ構造を有し、
あるキーに関連するデータポインタを追加もしくは削除する際に、
そのキーに関連する複数のデータポインタを格納する前記重複データ構造に対し、
そのデータポインタを追加もしくは削除するための情報としてテーブルデータ関連情報と前記重複データ構造へのポインタと前記キーを含む第１のログレコードを取得する機能と、
リーフノードに格納されたそのキーを含むインデクスエントリを更新する機能と、
前記重複データ構造へのそのデータポインタの追加もしくは削除が行われたことを示す情報を含む第２のログレコードを取得する機能と、
前記重複データ構造に、そのデータポインタを追加もしくは削除する機能とを有し、
上記データポインタの追加もしくは削除を伴う処理結果を取り消す際に、
上記取得したログレコードを受け取り、受け取ったログレコードが前記第２のログレコードであった場合には、その前記第２のログレコードを保持し、受け取ったログレコードが前記第１のログレコードであった場合には、第２のログレコードを保持しているかを確認し、第２のログレコードを保持している場合には、前記重複データ構造へのそのデータポインタの追加もしくは削除が完了したと判断する機能と、
上記判断において受け取ったログレコードが前記第１のログレコードであった場合にもかかわらず第２のログレコードを保持していないことを確認することにより前記重複データ構造へのそのポインタの追加もしくは削除が完了していないと判断した場合、第１のログレコードに含まれる前記重複データ構造へのポインタが示す前記重複データ構造をアクセスし、第１のログレコードに含まれる前記テーブルデータ情報を追加もしくは削除する機能とを、コンピュータへ実行させるデータ管理プログラムを記憶した計算機読み取り可能な記憶媒体。

【発明の詳細な説明】
【０００１】
【発明の属する技術分野】
本発明はデータ管理技術に関し，あるインデクスのキーに対して複数のデータを管理する場合に適用して有効な技術に関するものである。
【０００２】
【従来の技術】
近年インタネットを利用した業務システムの急速な普及に伴い、システムを支えるデータ

ベース管理システム(DBMS)の適用分野も拡大している。さらにそれに伴いDBMSの扱うデータ量も年々増加している。そのシステム中には、ワークフローにおける「業務推移状態」など、値域の狭いデータに対してDBMSの最も代表的な高速検索手段であるB木インデクスを利用するようなアプリケーションも多く、Ｂ木インデクスの安定した性能の提供が重要となっている。

【０００３】

Ｂ木インデクスに関しては、例えば、文献(Jim Gray and Andreas Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publishers, 1993)に、基本的はデータ構造、複数処理からの同時アクセス方法、および回復に関する基本方式について開示している。

【０００４】

実システムとしての高速なＢ木インデクスを実現するためには、次の課題を解消する必要がある。

【０００５】

（ｉ）複数処理による同時実行性の向上

レコードを含むデータベーステーブルに対して複数のトランザクションが同時にアクセスしてくる場合、問題が起こる。具体的には、あるトランザクションが１つのレコードを更新しようとしたとき、同時に他のトランザクションが同一レコードにアクセスしようとするとき、競合状況が発生する。競合問題の解決法の１つとして、レコードまたはＢ木インデクスのある部分（Ｂ木インデクス全体、ノード、インデクスエントリ、キー値など）に対するロッキング方式（排他的アクセス方式）がある。ロッキング方式は、データにアクセスする前に強制的にトランザクションにロックを取得させ。このとき他のトランザクションによってロックがかけられている場合、そのロックが衝突するために、必要なロックを取得できないことがある。ロッキング方式は自トランザクションの更新結果や参照結果を確実に保証してくれるが、ロックを取得中は他のトランザクションのアクセスをロック解除まで待たすので、システムの多重度(同時実行性)を高めスループットを上げるためには、１つのトランザクションが同時に取得するロックの数やロックによる影響範囲（粒度）を最小限に抑えることが重要である。

【０００６】

（ii）回復処理制御

実システムにおいては、トランザクションの中断、システムダウン、媒体障害など種々の障害に関して、トランザクションの原子性(Atomicity)を保証し、かつＢ木インデクス内のデータの整合性を保証する必要がある。そのためのログレコード取得方式およびログレコードを使用した回復処理制御が重要となる。ログレコードに取得方式には、インデクスを構成するノードの更新前・更新後の物理イメージをログレコードの情報をして取得する物理ログ方式などがある。物理イメージを用いた回復では、更新を行ったトランザクションの完了まで、他トランザクションはそのページへのアクセスが制限されることになる。すわち、回復処理制御は排他制御と密接に関係し、同時実行性にも影響を与える。

【０００７】

図１１に、従来の代表的なＢ木インデクスの構造を示す。

【０００８】

Ｂ木インデクスは、１つのルートノードを頂点に、ルートノードから多数のレベルにわたりノードが枝分かれしている。枝分かれしたノードの内末端のノードをしばしばリーフノードと呼ぶ。またルートノードを含むリーフノード以外のノードをしばしば上位ノードと呼ぶ。各ノード内は以下に示す情報から構成される複数のインデクスエントリを持つ。リーフノード内のインデクスエントリは、データベース中のレコード(テーブルデータ)を示すポインタと、そのレコード(テーブルデータ)の特徴の１つを表すキー値から構成される。また上位ノード内のインデクスエントリは、次の下位レベルにあるノード（子ノード）を示すポインタと、その子ノードから枝分かれして最終的にリーフノードで管理されているキー値の範囲を示す１キー値から構成される。上位インデクスエントリ内のキー値は、

Ｂ木インデクスへのアクセス・プログラムがルートノードから目的とするレコードへのポインタを含むインデクスエントリが格納管理されているリーフノードへと辿っていく際の道しるべ（判定要素）の役割を担う。通常Ｂ木インデクスの１ノードは、データベース上では、アクセス単位であるページによって実現される。

【０００９】

【発明が解決しようとする課題】

Ｂ木インデクスに対する排他制御方式の一つにインデクスの排他資源の粒度を「キー」とするキー値排他方式もしくはキーレンジ排他方式がある。これらインデクスのキーに対する排他制御を用いた場合、同一キーを持つデータ(行)に対する検索・更新処理はインデクスキー排他によりシリアライズされ、同時実行性が低下する。現在急速な普及をみせているWEB環境下におけるトランザクションは、従来のＯＬＴＰのトランザクションの長さに較べ長く、キーに対する排他制御方式では高スループットのサービスの提供は困難である。

【００１０】

また、図１１に示すように多数の同一キー有するＢ木インデクスに関して、以下のような問題もある。図１１のＢ木インデクスでは、あるキーに関連するテーブルデータ情報を１つのリーフエントリ１６で管理している。リーフエントリ１６は、キー１４、そのキー値に関連するテーブルデータ情報１８(データレコード識別子)、そのキー値に関連するテーブルデータに関する情報１８の個数１７（超複数）を有する。テーブルデータ情報１８は重複数分リスト形式にて昇順に配置されている。このような格納方式では、あるキーに対する重複数が多くなると、リーフエントリ自体の長さがリーフノード１２に収まりきれなくなる。結果的に、１つのキーに対するテーブルデータ情報を複数のリーフエントリとして、複数のノードに分けて格納しなければならない。図１１の例では、キー「２８」のエントリがリーフノードＮ４、Ｎ５、Ｎ６と３つのノードに分かれて格納されている。この構造では、範囲検索に対応するためのリーフリーフノードの水平方向のポインタを用いたスキャンにおいて、キー「２８」のエントリがネックとなる。またさらに、ルートノード１０および上位ノード１３に格納されている上位エントリ１３に含まれるキー１４は、「２８」に加えテーブルデータ情報でキーとしなければならず、ルートノード１０および上位ノード１３の数を増大する要因となっている。結果的に、重複キーは、重複キーそのものに対するアクセス処理だけでなく、Ｂ木インデクス全体のアクセス性能の低下および格納効率の低下の要因となっている。

【００１１】

GrayはＢ木構造外へのデータ構造にてテーブルデータ情報を管理する方法を開示しているが、それに対する同時実行制御および回復制御方法に関しては開示されていない。また、物理ログによる回復処理方式について開示しているが、先に示したような同一ページに対する同時実行を実現することができない。このような問題は、同一キーに対するインデクスアクセスを行う場合、一般的に発生する。

【００１２】

本発明の目的は上記問題を改善し、複数のアクセスが行われる環境下において、同一キーに対するインデクスアクセスを好適に行うことが可能な技術を提供することにある。

【００１３】

【課題を解決するための手段】

前記課題を以下の手段により改善する。

【００１４】

複数ユーザから同時にアクセスされる環境下で、多数の同一キーを持つＢ木インデクスを管理する方法において、ある一つのキーに関する複数のデータポインタを格納するデータ構造であり、そのキーを含むインデクスエントリが格納されるリーフノードとは異なるデータ構造であり、そのデータ構造はそのキーを含むインデクスエントリからポイントされ、複数処理による同時アクセスが可能であるデータ構造を有し、あるキーに関連するデータポインタを追加もしくは削除する際に、そのキーに関連する複数のデータポインタを格

納する前記データ構造に対し、そのデータポインタを追加もしくは削除するための情報を含む第１のログレコードを取得し、リーフノードに格納されたそのキーを含むインデクスエントリを更新し、前記データ構造へのそのデータポインタの追加もしくは削除が行われたことを示す情報を含む第２のログレコードを取得し、前記データ構造に、そのデータポインタを追加もしくは削除することにより上記課題を改善する。

【００１５】

【発明の実施の形態】

複数ユーザからの同時アクセス環境下において、同一キーに対するインデクスアクセスを効率的に行うことが可能な一実施形態のデータベース処理システムについて説明する。

【００１６】

まず、本発明の概念を図１を用いて簡単に説明する。

【００１７】

本実施形態のデータベース管理システムにおけるＢ木インデクス１は、図１に示すようにキーを用いて効率的にテーブルデータに関連する情報を取得するためのＢ木構造１０１と、キーごとにそのキーに関連付けられた多数のテーブルデータ情報を管理するデータ構造１０２(以降「重複キーデータ構造」と呼ぶ)から構成される。

【００１８】

Ｂ木構造１０１は、ルートノード１０、中間ノード１１、リーフノード１２により構成され、１つのルートノードを頂点に、ルートノードから多数のレベルにわたりノードが枝分かれしている構造をもつ。ルートノード１０および中間ノード１１には、キー１４とそのキー値に関連する下位レベルへのノードを示すポインタ１３とを有する上位エントリ１３が格納される。それぞれのノード内において、上位エントリ１３およびリーフエントリ１６は、そのインデクスエントリ１３、１６が有するキー値の順に格納管理されている。本実施例では、図の左から右方向へキー値が昇順に格納されている。それぞれのノード内において、それらソートされたインデクスエントリをバイナリサーチすることにより、効率的かつ安定して必要なインデクスエントリにアクセスすることができる。一つの上位インデクスエントリがポイントする子ノードには、その上位インデクスエントリ内キー値よりも小さいキー値を持つインデクスエントリが格納されている。

【００１９】

キー１４に関連するテーブルデータが多数の場合のリーフエントリ２０は、キー１４、そのキー値に関連するテーブルデータに関する情報の個数１７、及び、テーブルデータに関する情報を格納する重複キーデータ構造１０２へのポインタ１９を有する。図１の例では、キー値「２８」に対して、関連するテーブルデータ情報数すなわち重複数が「35,678」であることを示している。それら35,678個の情報は、ポインタ１９「Nx1」にて指される重複データ構造１０２に格納管理されている。

【００２０】

一方、キー値「２８」以外のインデクスエントリのように図１の例では、重複数すなわちテーブルデータに関する情報数があまり多くない場合、重複キーデータ構造１０２を持たないリーフエントリ１６として格納される。リーフエントリ１６は、キー１４、そのキー値に関連するテーブルデータに関する情報１８(データレコード識別子)、そのキー値に関連するテーブルデータに関する情報１８の個数１７を有する。データレコードの追加もしくは更新によって、そのキーに関連するテーブルデータ情報の重複数がある敷居値を超えた際に、上記重複キーデータ構造１０２をポイントするリーフエントリ２０に移行する。

【００２１】

また、多数のテーブルデータに関連する情報を格納管理する重複キーデータ構造１０２は以下の特徴を持つ。

【００２２】

(1)アクセスの際に排他を取得することなく、複数処理による同時アクセスが可能である。

【００２３】
(2)通常処理のアクセスを制限することなく取り消し処理が可能である。
【００２４】
ここで、矢印１０３はデータレコードの更新に伴うＢ木インデクス１に対するキー値「２８」、テーブルデータ関連情報「Ｐ１８」の追加更新処理の様子を示している。まず、この追加更新処理は、キー値「２８」を用いてＢ木構造１０１のルートノード１０（Ｎ１）からサーチを開始し、中間ノード１１（Ｎ２）を経由し、更新対象のキー値「２８」を有するリーフエントリ２０が格納されているリーフノード１２（Ｎ５）に辿り着く。そして、リーフエントリ２０を確定し、第１のログレコード４１を取得し、エントリ内の重複数１７をインクリメントする。第１のログレコード４１には、重複キーデータ構造１０２に対する更新処理を続行するために必要な情報が含まれている。さらに、エントリ内の重複キーデータ構造１０２へのポインタ１９（Ｎｘ１）を取得し、重複キーデータ構造１０２へのアクセスを行う。そして、第２のログレコード４２を取得し、重複キーデータ構造１０２に対し、テーブルデータ関連情報「Ｐ１８」を追加する。第２のログレコード４２には、重複キーデータ構造１０２に対しテーブルデータ関連情報１８の追加が完了したことを示す情報が含まれている。
【００２５】
図１の例を用いてテーブルデータ関連情報の追加に関し説明したが、テーブルデータ関連情報の削除に関しても同様である。リーフエントリ２０をアクセスし、第１のログレコード４１を取得し、エントリ内の重複数１７をデクリメントする。そして、重複キーデータ構造１０２をアクセスし、第２のログレコードを取得した後に、重複キーデータ構造１０２からテーブルデータ関連情報１８の削除を行う。
【００２６】
以上のように、リーフエントリの更新から重複キーデータ構造への変更順序が一定であり、インデクス１を構成するデータリソースへの排他制御を必要としないことから、複数処理の同時実行が可能である。また、一連の変更処理結果に対し取り消し要因が発生した場合、もしくはリーフエントリに対する更新直後に処理の中断要因が発生した場合でも、前述の第１のログレコード４１および第２のログレコード４２を使用することにより、同時実行性を失うことなしに取り消し処理を実施することが可能である。
【００２７】
具体的には、取り消し処理時、第１のログレコード４１と第２のログレコードがペアで取得されている場合、一連の変更処理は完了していると判断し、図１の例では、キー値「２８」、テーブルデータ関連情報「Ｐ１８」の削除処理を行う。取り消し処理時、第１のログレコード４１しか取得されていない場合は、リーフエントリ２０に対する変更処理しか行われていないので、第１のログレコードに含まれる情報を用いて重複キーデータ構造２０１に対するテーブルデータ関連情報「Ｐ１８」の追加処理を行った後、改めてキー値「２８」、テーブルデータ関連情報「Ｐ１８」の削除処理を行う。取り消し処理においてもＢ木インデクス１に対する変更の順序は常に片方向であるため、他処理との同時実行性を損なうことはない。以上のように、取り消し処理を交えても高い同時実行性を提供することが可能である。
【００２８】
さらに、本重複キーデータ構造２０１をリーフエントリ２０からポイントし、リーフノード１２の外に格納管理することから、リーフエントリ２０が格納されているリーフノード１２内には、すべてのテーブルデータ関連情報１８をリーフエントリ１６の有する方式に較べ、より多くの他のキーを有するリーフエントリの格納が可能となる。そしてリーフエントリを格納するリーフノード１２の必要数を抑え、最終的に上位ノード１１の必要数も抑え、Ｂ木インデクス全体の格納容量削減にもなる。多くのテーブルデータ情報を持たないキーに対するアクセスに関しても、Ｂ木構造の容量が抑えられるため、Ｂ木構造を辿るすべての検索処理および変更処理に対し安定した性能を提供することが可能である。
【００２９】

次に図２に本実施形態のデータベース管理システムの概略構成を示す。ユーザが作成したアプリケーションプログラム６と、問い合わせやリソース管理などのデータベースシステム全体の管理を行うデータベース管理システム２がある。上記のデータベース管理システム２は、論理処理部２１、物理処理部２２、システム制御２３と、データベースアクセス対象となるデータを永続的にあるいは一時的に格納するデータベース３、そしてシステムログ４を有する。また、上記データベース管理システム２はネットワークなどを介して他のシステムと接続されている。

【００３０】

上記論理処理部２１は、問合せの構文解析・意味解析を行う問合せ解析２１１と、適切な処理手順を生成する最適化処理２１２と、処理手順に対応したコードの生成を行うコード生成２１３を具備している。また、上記生成されたコードに基づき、コードを解釈しデータベース処理実行の指示を物理処理部２２に対して行うコード解釈実行２１４を具備している。

【００３１】

上記物理処理部２２は、データベース３に格納されているテーブルデータ５に対し、データの格納、削除、更新、検索処理をデータベースバッファ２４を介して行うテーブルデータ管理部２２１を具備する。また、データベース３に格納されているインデクス１を管理するＢ木インデクス管理部２５を具備している。インデクス１は、キーを用いて効率的に関連情報にアクセスするためのＢ木構造と、キーごとにそのキーに関連付けられた多数のテーブルデータ情報を管理する重複キーデータ構造から成る。

【００３２】

Ｂ木インデクス管理部２５は、テーブルデータ５の更新に伴い、それに関連するＢ木インデクス１に対し、Ｂ木構造に対する変更を行うＢ木構造アクセス処理２６の機能、および重複データデータ構造に対する変更を行う重複キーデータ構造アクセス処理２７の機能を有する。また、インデクス１をアクセスすることにより、検索条件であるキーから関連するテーブルデータを高速に検索するための機能をＢ木構造アクセス処理２６と重複キーデータ構造アクセス処理２７内に有する。

【００３３】

上記物理処理部２２は、また、システムで共用するリソースの排他制御２２３を具備している。本実施例では、データに対する整合性は、テーブルデータ５に対する排他制御(排他リソース：テーブルデータ格納ページID,テーブルデータIDなど)のみで実現し、Ｂ木インデクス１のリソース(インデクスID,インデクスページID,キー値など)に対する排他制御は行わない。

【００３４】

システムログ４は、データベース３へのデータ挿入、更新、削除などの更新履歴情報を蓄積する。その蓄積される情報には、Ｂ木インデクス１に対する変更に関する履歴情報も含まれる。そして、トランザクションの取り消しやシステムダウンなどの種々の状況において、Ｂ木インデクス１のデータ整合性を回復するために使用される。

【００３５】

上記システム制御２３は、ユーザからの問い合わせ入力、問い合わせ結果の返却、コマンド受付によるデータベース保守などを行う。また、前述のシステムログ４の管理を行い、上記物理処理部２２と連携し、データベース３変更の際に履歴情報であるログレコードの取得２６を行う。そして、システム制御２３は、読み込んだログレコードがＢ木インデクスに対する変更処理に関する場合に、物理処理部２２のＢ木インデクス管理部２５にそのログレコードを渡し、取り消し処理を依頼する。依頼されたＢ木インデクス管理部２５は、ログレコード内の情報を基に変更取り消し処理を行う。

【００３６】

ユーザは、アプリケーションプログラム６を介し、データベース管理システム２を用いて、データベース３内のテーブルデータ５を構成するデータレコードを探索、アクセスし、変更することができる。データベース管理システム２はデータレコードへの効率のよいア

クセスを実現するために、Ｂ木インデクス１を利用する。

【００３７】

データベース３内のＢ木インデクス１あるいはテーブルデータ５は、アクセス単位であるページ単位に、データベースバッファ２４に読み込んでくる(GETしてくる)ことによりアクセスが可能になる。バッファ上のページを参照（検索）した後、バッファをRELEASEすることにより使用していたバッファに対する利用終了を宣言し、他処理からのそのバッファの利用を許可する。また、バッファ上のページに対して更新を行った後、バッファをPUTすることにより当該バッファ上のページがデータベースに反映することを宣言し、使用していたバッファの他処理からの利用を許可する。そのバッファ上のページはすぐにはデータベースに反映されず、ある契機にデータベースに反映される。バッファにGETしたページに他のトランザクションが不当にアクセスしないようにバッファに対してラッチをかける。ラッチは一種のロックであるが通常のロックよりもはるかに取得期間が短く、獲得と解除を安価に行うことができる。本実施例では、インデクスノードを参照する際に共用ラッチを、更新する際に排他ラッチをかける。排他ラッチに関する処理はシリアライズされ、共用ラッチ間ではバッファに対する同時参照を可能にする。そして、バッファ解放の際に獲得中ラッチを解除し、排他ラッチを獲得しようとしている他の処理のアクセスを許可する。

【００３８】

図３は本実施形態のコンピュータシステムのハードウエア構成の一例を示す図である。コンピュータシステム３０００は、ＣＰＵ３００２、主記憶装置３００１、磁気ディスク装置等の外部記憶装置３００３及び多数の端末３００４で構成される。主記憶装置３００１上には、図２を用いて先に説明したデータベース管理システム２が置かれ、外部記憶装置３００３上にはデータベース管理システム３が管理するテーブルデータ５とインデクス１を含むデータベース３が格納される。また、システムログ４も外部記憶装置３００３上に格納される。さらに、データベース管理システム２を実現するプログラム３１００も外部記憶装置３００３上に格納される。

【００３９】

図８は本実施形態のインデクスのログレコードの一例を示す図である。

【００４０】

図８の４１に示す様に、図１にて説明したリーフエントリ変更時に取得される第１のログレコード４１は、その変更処理が属するトランザクションの情報などを含むログレコードヘッダ４１０、変更を示す操作コード４１１、キー４１２、変更対象であるテーブルデータ情報４１３、そして変更されたリーフエントリが指す重複キーデータ構造へのポインタ４１４により構成される。操作コード４１１には、追加処理や削除処理を示すコードが設定される。図８の例ではその設定コードは追加処理を示す「INSERT」である。ログレコードヘッダ４１０には変更対象のＢ木インデクスのインデクスIDやそのインデクスが格納されるエリア情報も含まれる。キー４１２に設定される値は変更対象であるリーフエントリ２０内のキー１４の値と同値である。図８の例ではその値は「２８」である。

【００４１】

重複キーデータ構造へのポインタ４１４は、リーフエントリ変更は終了したが重複キーデータ構造変更が完了していない状況にて変更結果取り消し要因が発生した場合に、重複キーデータ構造に対する変更処理を続行するための情報である。

【００４２】

また図８の４２に示す様に、図１にて説明したリーフエントリ変更時に取得される第２のログレコードは、その変更処理が属するトランザクションの情報などを含むログレコードヘッダ４２０、変更を示す操作コード４２１、そして変更対象であるテーブルデータ情報４２２により構成される。４１１と同様に操作コード４２１には、追加処理や削除処理を示すコードが設定される。図８の例ではその設定コードは追加処理を示す「INSERT」である。また、テーブルデータ情報４２２には、第１のログレコード４１の４１３と同じ値が設定されることになる。第２のログレコード４２は、先に図１を用いて説明したように、

重複キーデータ構造に対する変更処理が終了しているかを示すマーカの役割を担う。

【００４３】

図４は本実施形態のＢ木インデクス管理部２５の処理手順を示すフローチャートである。図４ではＢ木インデクスに対するテーブルデータ情報追加の場合のＢ木構造アクセス処理２６の処理内容を表している。

【００４４】

まず、ステップ４０１において、キーを用いて、Ｂ木構造をルートノードから上位ノードへと辿り、リーフノードを確定後、そのリーフノードをデータベースバッファに読み込みバッファ固定(GET)を行う。次にステップ４０２において、リーフノード内に格納されているリーフエントリをサーチし、変更対象であるリーフエントリの確定を行う。そして、ステップ４０３においてそのリーフエントリのキーに関連するテーブルデータ情報数(重複数)をインクリメントする。ここで、ステップ４０４にて、リーフエントリ内に含まれる重複キーデータ構造を指すポインタを取得する。さらに、図８を用いて説明した第１のログレコードを作成し（ステップ４０５）、作成した第１のログレコードを取得する（ステップ４０６）。最後にリーフノードをデータベースに書き出すように予約し、バッファ固定を解除(PUT)する（ステップ４０７）。以上でリーフエントリ変更処理を終了し（ステップ４０８）、引き続き重複キーデータ構造に対する変更処理に進む。

【００４５】

本フローチャートでは、リーフエントリ変更（４０４）、ログレコード取得（４０５）の順に処理が行われているが、実際のデータベースへのリーフノードに対する変更結果の反映は、ＷＡＬ(Write-ahead log)プロトコルに従ってログレコードのシステムログへの書き出しが実施された後に行われる。

【００４６】

図５は本実施形態のＢ木インデクス管理部２５の処理手順を示すフローチャートである。図５ではＢ木インデクスに対するテーブルデータ情報追加の場合の重複キーデータ構造アクセス処理２７の処理内容を表している。

【００４７】

重複キーデータ構造に対する変更処理は、図４のフローチャートで説明したリーフエントリに対する変更処理に引き続き行われる。

【００４８】

まず、ステップ５０１において、先に図４のステップ４０４にて取得したポインタを用いて重複キーデータ構造へアクセスし、ステップ５０２において、テーブルデータ情報を追加する位置をサーチする。そして、図８を用いて説明した第２のログレコードを作成し（ステップ５０３）、作成した第２のログレコードを取得する（ステップ５０４）。ここで、重複キーデータ構造に追加領域があるかを判定する（ステップ５０５）。追加領域が無い場合、ステップ５０６に進み、新規領域の割り当てを行い、新規領域追加に伴う重複キーデータ構造の変更を行う(ステップ５０７)。具体的には重複キーデータ構造が複数のページから構成される場合は、新規ページ割り当てが行われる。その後ステップ５０８へ進む。また、ステップ５０５の判定結果が領域有りの場合、ステップ５０８へ進み、重複キーデータ構造へテーブルデータ情報を追加し、重複キーデータ構造に対する変更処理を終了する（ステップ５０９）。

【００４９】

図６は本実施形態のＢ木インデクス管理部２５の処理手順を示すフローチャートである。図６ではＢ木インデクスに対するテーブルデータ情報削除の場合のＢ木構造アクセス処理２６の処理内容を表している。

【００５０】

まず、ステップ６０１において、キーを用いて、Ｂ木構造をルートノードから上位ノードへと辿り、リーフノードを確定後、そのリーフノードをデータベースバッファに読み込みバッファ固定(GET)を行う。次にステップ６０２において、リーフノード内に格納されているリーフエントリをサーチし、変更対象であるリーフエントリの確定を行う。ここで、

ステップ６０３において、リーフエントリ内に含まれる重複キーデータ構造を指すポインタを取得する。そして、ステップ６０４において、そのリーフエントリのキーに関連するテーブルデータ情報数(重複数)が、１かどうかを判定する。重複数が１の場合、ステップ６０５に進みリーフエントリの削除を行い、ステップ６０７に進む。ステップ６０４の判定結果が重複数が１より大きい場合、ステップ６０６に進み、そのリーフエントリのキーに関連するテーブルデータ情報数(重複数)をデクリメントする。

【００５１】

さらに、図８を用いて説明した第１のログレコードを作成し（ステップ６０７）、作成した第１のログレコードを取得する（ステップ６０８）。最後にリーフノードをデータベースに書き出すように予約し、バッファ固定を解除(PUT)する（ステップ６０９）。以上でリーフエントリ変更処理を終了し（ステップ６１０）、引き続き重複キーデータ構造に対する変更処理に進む。

【００５２】

図７は本実施形態のＢ木インデクス管理部２５の処理手順を示すフローチャートである。図７ではＢ木インデクスに対するテーブルデータ情報削除の場合の重複キーデータ構造アクセス処理２７の処理内容を表している。

【００５３】

重複キーデータ構造に対する変更処理は、図６のフローチャートで説明したリーフエントリに対する変更処理に引き続き行われる。

【００５４】

まず、ステップ７０１において、先に図４のステップ６０３にて取得したポインタを用いて重複キーデータ構造へアクセスし、ステップ７０２において、削除対象テーブルデータ情報の存在位置をサーチする。そして、図８を用いて説明した第２のログレコードを作成し（ステップ７０３）、作成した第２のログレコードを取得する（ステップ７０４）。

【００５５】

次に、ステップ７０５において、重複キーデータ構造からのテーブルデータ情報削除を行う。ここで、テーブルデータ情報の削除によって、回収可能な重複キーデータ構造を構成する領域が発生したかを判定する（ステップ７０６）。判定の結果、回収可能領域が発生した場合、ステップ７０７へ進み領域の回収を行い、領域回収に伴う重複キーデータ構造の変更を行う(ステップ７０８)。具体的には重複キーデータ構造が複数のページから構成される場合は、ページの回収が行われる。回収されたページは他の処理における新規ページ割り当てにより再利用されることとなる。ステップ７０８後ステップ７０９にて重複キーデータ構造に対する変更処理を終了する。また、ステップ７０６における判定結果が、回収可能領域発生なしの場合、そのままステップ７０９にて重複キーデータ構造に対する変更処理を終了する。

【００５６】

図９は本実施形態のインデクス管理部のインデクス更新結果取り消し処理の処理手順を示すフローチャートである。

【００５７】

まず、ステップ９０１において、システム制御からＢ木インデクス変更に関するログレコードを受け取る。ステップ９０２において、処理すべきログレコードが存在するかを判定する。処理すべきログレコードが存在しない場合、ステップ９１３にてインデクス更新結果取り消し処理を終了する。処理すべきログレコードが存在する、すなわち受け取りログレコードが存在する場合、ステップ９０３へ進みログレコードの種別を判定する。ここでまずログレコードが第２のログレコードかどうかを判定する（ステップ９０３）。第２のログレコードである場合、ステップ９０４へ進み第２のログレコードを保持する。そして、ステップ９０１へ戻り次のログレコードの処理を行う。ここで次のログレコードは、第１のログレコードのはずである。ステップ９０３の判定において、第１のログレコードではない場合、さらにステップ９０５にて、ログレコードが第１のログレコードかどうかを判定する。第１のログレコードでない場合、ステップ９０６にてその他のログレコードに

対応する更新結果取り消し処理を行い、ステップ９０１へ戻り、次のログレコードの処理を行う。第１のログレコードである場合、以下の処理を行う。まず第２のログレコードが保持されているかを判定し（ステップ９０７）、保持されている場合には、リーフエントリおよび重複キーデータ構造に対する一連の変更処理が完了していると判断し、保持されている第２のログレコードを廃棄し（ステップ９０８）、ステップ９１０へ進む。第２のログレコードが保持されていない場合、リーフエントリの変更処理は終わっているが対となる重複キーデータ構造に対する変更処理が完了していないと判断し、ステップ９０９にて重複キーデータ構造の対する更新処理を続行する。その後ステップ９１０へ進む。ステップ９１０では、処理中のログレコードが通常処理にて取得されたものであるかどうかを判定する。通常処理にて取得されたログレコードである場合、関連する変更結果を取り消す必要がある。ステップ９１１にてその取り消し処理を行う。また、通常処理時取得ではない、すなわち取り消し処理時における取得の場合、取り消し処理はステップ９０９を含めすでに完了していると判断し、一連のＢ木インデクス更新結果の取り消し処理を終了し、ステップ９０１へ戻り、次のログレコードの処理を行う。

【００５８】

図１０は本実施形態の重複キーデータ構造１０２の構成の一例を示す図である。

【００５９】

図１０の例では、重複キーデータ構造１０２は、実際にテーブルデータ情報を格納するページ１００２と、そのページ１００２を指すポインタを有する管理エントリ１００３を格納するページ１００１より構成される。

【００６０】

ページ１００１に格納される管理エントリ１００３は、ページ１００２を指すポインタ１００５、およびそのページ１００２内に格納されるテーブルデータ情報の最大値１００４を有する。ページ１００１内の管理エントリ１００３は、テーブルデータ情報の最大値１００４の順にソートされている。

【００６１】

また図１０に示すように、各ページ１００１は、そのページ１００１内に格納される管理エントリ１００３内の最大情報１００４よりも大きな最大情報を持つ管理エントリが格納されるページ１００１へのポインタを有する。リーフエントリ２０からは、最も小さいテーブルデータ情報を格納するページ１００２をポイントする管理エントリが格納されてるページ１００１がポイントされる。

【００６２】

以上のデータ構造により、図５および図７にて説明した重複キーデータ構造内のサーチを効率的に行うことが可能である。

【００６３】

また、ページ１００１もしくはページ１００２の新規割り当て・回収は、データベースバッファへのＧＥＴ・ＰＵＴにより容易に実現可能である。新規割り当て・回収の際の第２のログレコードは、図８の例で示した構成情報の他にページ１００１もしくはページ１００２のページ識別子を付け加えることにより、取り消し処理における変更続行処理が容易に可能である。

【００６４】

以上示したフローチャートの処理は、図３で例として示したコンピュータシステムにおけるプログラムとして実行される。しかし、そのプログラムは図３の例の様にコンピュータシステムに物理的に直接接続される外部記憶装置に格納されるものと限定はしない。ハードディスク装置、フロッピーディスク装置等のコンピュータで読み書きできる記憶媒体に格納することができる。また、ネットワークを介して図３のコンピュータシステムとは別のコンピュータシステムに接続される外部記憶装置に格納することもできる。

【００６５】

このようにすることにより、同一キーを持つテーブルデータ情報が、アクセスの際に排他を取得することなく複数処理による同時アクセスが可能であり、且つ通常処理のアクセス

を制限することなく取り消し処理が可能である重複キーデータ構造で管理されるため、同一キーに対して同時にアクセスする複数の処理に対し、取り消し処理をも交えた高い同時実行性を提供することが可能である。

【００６６】

その重複キーデータ構造はリーフエントリからポイントされ、リーフノードの外に格納管理されることから、重複キーを有するリーフエントリが格納されているリーフノード内により多くの他のキーを有するリーフエントリの格納が可能となる。その結果、Ｂ木インデクスの格納容量が抑えられ、安定したアクセス性能を提供することが可能になる。

【００６７】

以上のように、多数の重複キーを有する場合でも同時実行性に優れ、格納効率のよいＢ木インデクスを用いたデータ管理方法を提供することができる。

【００６８】

本Ｂ木インデクスを用いたデータ管理方法は、ワークフローにおける「業務推移状態」など、値域の狭いデータに対してＢ木インデクスを利用するようなアプリケーションに対して非常に有効である。そのアプリケーションが利用するＢ木インデクスでは、値域が狭いため一つのキーに関連するデータ重複数が膨大であり、さらに「業務推移状態」などの変化によってＢ木インデクスに関して頻繁に更新が発生するためである。

【００６９】

以上、Ｂ木について説明してきたが、本発明は、複数のアクセスが行われる環境下において、同一キーに対するインデクスアクセスが発生する場合に同様に適用可能であり、同一キーに対するインデクスアクセスを好適に行うことが可能なデータ管理方法および装置を提供することが可能である。

【００７０】

【発明の効果】

本発明によれば、複数のアクセスが行われる環境下において、同一キーに対するインデクスアクセスを好適に行うことが可能なデータ管理方法および装置を提供することができる。

【図面の簡単な説明】

【図１】本発明の概念図である。

【図２】本実施形態のＢ木インデクスを有するデータベース処理システムの機能ブロックを示す図である。

【図３】本実施形態のコンピュータシステムのハードウエア構成の一例を示す図である。

【図４】本実施形態のＢ木インデクス管理部２５のインデクス更新処理の処理手順を示すフローチャートである。

【図５】本実施形態のＢ木インデクス管理部２５のインデクス更新処理の処理手順を示すフローチャートである。

【図６】本実施形態のＢ木インデクス管理部２５のインデクス更新処理の処理手順を示すフローチャートである。

【図７】本実施形態のＢ木インデクス管理部２５のインデクス更新処理の処理手順を示すフローチャートである。

【図８】本実施形態のＢ木インデクスのログレコードの一例を示す図である。

【図９】本実施形態のＢ木インデクス管理部のインデクス更新結果取消し処理(回復処理手順)の処理手順を示すフローチャートである。

【図１０】本実施形態の重複キーデータ構造の構成の一例を示す図である。

【図１１】従来の多くの重複キーを有するＢ木インデクスの構成を示す図である。

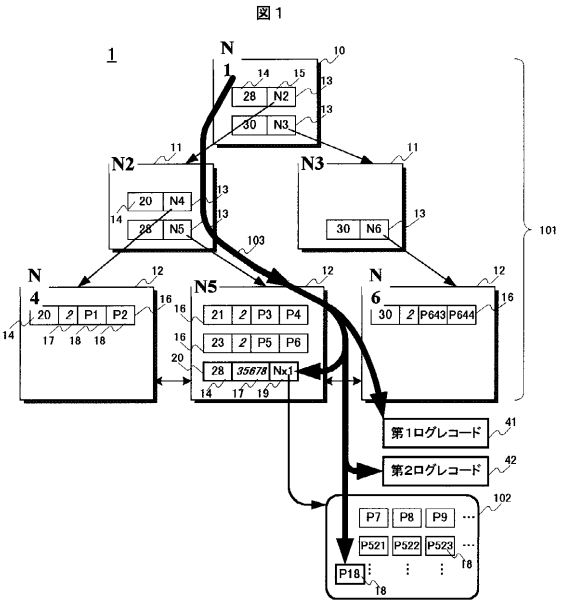
【符号の説明】

１…Ｂ木インデクス、２…データベース管理システム、３…データベース、４…システムログ、５…テーブルデータ、６…アプリケーションプログラム、２１…論理処理部、２２…物理処理部、２３…システム制御、２５…Ｂ木インデクス管理部、２６…Ｂ木構造アクセス処理、２７…重複キーデータ構造アクセス処理、１０１…Ｂ木構造、１０２…重複キ

ーデータ構造、４１…第１ログレコード、４２…第２ログレコード。

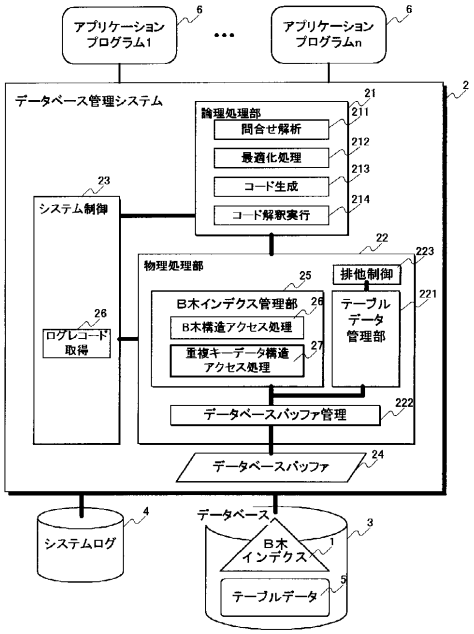
【図１】

図１



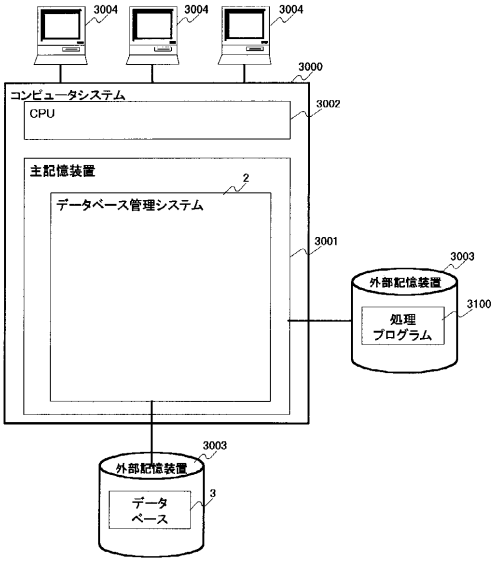
【図２】

図2



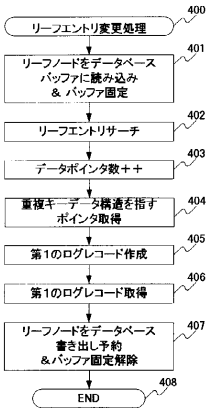
【図３】

図3



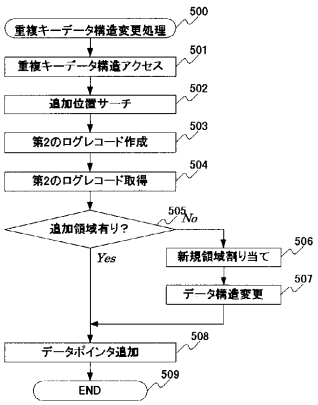
【図４】

図4



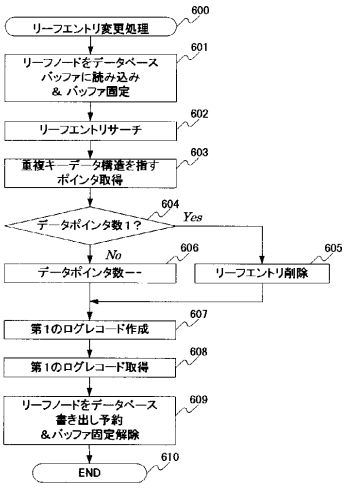
【図５】

図5



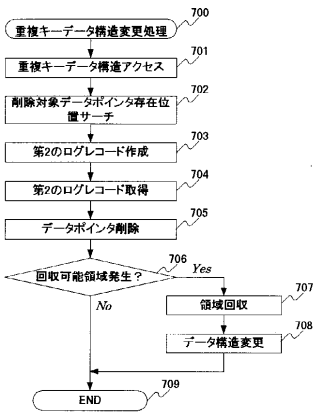
【図６】

図6



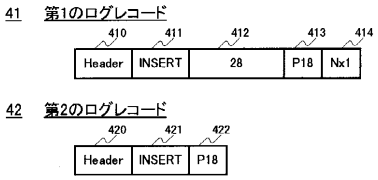
【図７】

図７



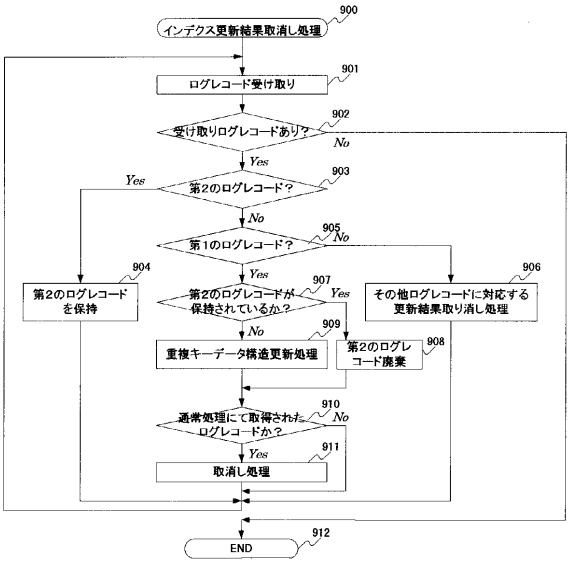
【図８】

図８



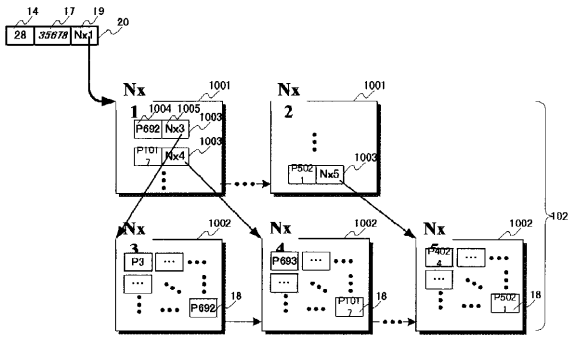
【図９】

図９



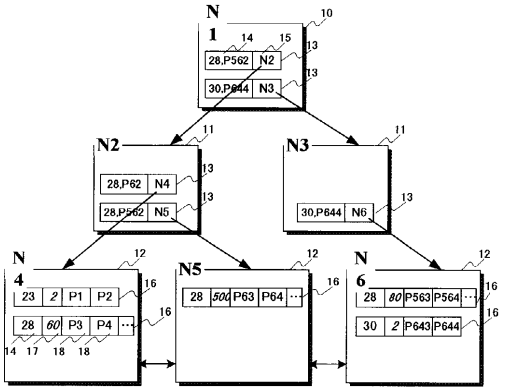
【図１０】

図１０



【図１１】

図１１



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
JSTPlus(JDreamII)