時間的に変化するデータに対する索引技術

天笠 俊之　奈良先端科学技術大学院大学情報科学研究科　amagasa@is.aist-nara.ac.jp
有次 正義　群馬大学工学部情報工学科　aritsugi@cs.gunma-u.ac.jp
金森 吉成　群馬大学工学部情報工学科　kanamori@cs.gunma-u.ac.jp

データベースの応用の1つに，時間的に変化するデータ（時制データ; temporal data）の管理が挙げられる．時制データベースではデータの更新履歴を検索・操作するため，データの更新や削除は仮想的に行われ，データベース内に蓄積されるデータ量は常に増え続けるという特徴がある．このため，時制データを高速に検索するための索引は，この特徴を考慮したものになる．本稿では，データベースにおける時間の考え方について述べた後，これまでに提案されてきた代表的な索引手法を紹介する．

データベースでの時間って？

データベースは実世界の組織や物事に関する情報を管理する．別の言い方をすると，実世界のある側面を計算機中に再現した写し絵ともいえる．実世界は静止しているわけではなく，時間の経過とともに刻々とそのありようが変わっていく．このため，データベースの内容を新鮮に保つためには，実世界の変化にあわせて，データベースの内容も常に更新し続けなければならない．

初期のデータベースシステムでは，計算機の計算能力や記憶容量は現在のそれと比べると非力だったため，基本的にデータ更新の際には最新の情報しか保存されていなかった．つまり，過去のデータは上書きされ，捨てられてしまっていたのである．しかしながら，アプリケーションによっては，最新の情報だけではなく，データの変化の履歴が重要な意味を持つことも多い．また，データ更新の履歴を後から解析することによって，有益な情報が得られることも少なくない．

たとえば，ある会社で過去に在籍したすべての社員の在籍期間と給与のデータが蓄積されているならば，「社員Aの入社時点での給与はいくらか？」，「過去に在籍した社員の年度別の平均給与はいくらか？」，「社員の平均給与が初めて27万円を超えたのはいつか？」といったさまざまな問合せを処理することができる．また，音や映像を扱うマルチメディアデータベースも時間に関連した応用である[9]．近年特に注目されている応用としては，時系列データからのデータマイニングが挙げられる．これは株価や気象データなど，過去に蓄積された膨大なデータから特徴を抽出して，今後の値を予測したり，有益な情報を抽出する技術である．

このように時間の経過に伴って変化するデータの操作・検索が可能なデータベースを時制データベース（temporal database）と呼ぶ．時制データベースは，この二十数年来活発に研究が行われてきた分野であり，その範囲は問合せ言語（temporal query language），問合せ最適化手法，索引手法など，多岐に渡る．本稿では特に，時制データに対する高速な検索処理を可能に

する，これまでに提案された代表的な索引手法を紹介する．

2つの時間軸
―有効時間とトランザクション時間―

まず，時間をどのようにモデル化するかを考えよう[8),6)]．時間軸のモデルとしては，単一時間軸モデルと枝分かれモデルがある．前者は，すべての事象が単一の時間軸上に現れるのに対し，後者では軸が途中で次々と分岐していく．さらに時間軸は，時間の値を整数値とするか実数値とするかによって，連続モデルと離散モデルに分けることができる．データベースの分野では，一般に単一時間軸かつ離散モデルが使われることが多い．時間軸は（相対的な）開始時刻と現在時刻によって両端が閉じられている．現在時刻は，特別な記号*now*で表される．*now*は時間の経過とともに増え続ける．離散モデルにおいて，時間軸はそれ以上分割することのできない固定長の区間の連続した集まりであるとされる．この最小の単位をクロノン（chronon）と呼ぶ．クロノンの大きさ（粒度; granularity）は，応用によって秒，日，分のように異なる．任意の時点（instant）は1つのクロノンに相当する．時間軸上の2つの時点t_sとt_e ($t_s \leq t_e$) を定めると，時区間$[t_s, t_e)$が得られる．時区間は時間軸上の連続した時点の集合である．

時間軸は，実世界を中心に据えるか，データベースを中心に据えるか，利用者を中心に据えるかによって以下の3種類が考えられる[7)]．

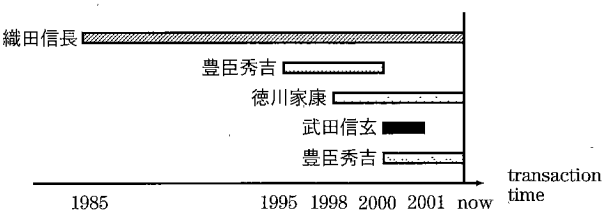
トランザクション時間（transaction time）ある事実がデータベース内に存在していた時間．
有効時間（valid time）ある事実が実世界において有効だった時間．
ユーザ定義時間（user-defined time）利用者によって与えられシステムが関与しない時間．

このうち，ユーザ定義時間はシステムが関与しない時間であるため，以下ではトランザクション時間と有効時間のみを議論する．

時制データベースという用語は，広義では時制データをサポートするデータベースを意味するが，厳密にはサポートする時間軸によってさらに詳細に分類される．一般にデータベースは刻々と変化する実世界のある一瞬をモデル化したものであると考えられる．この断面をスナップショット（snapshot）と呼び，時間をサポートしないデータベースはスナップショットデータベースと呼ばれる．さらに，トランザクション時間のみをサポートするものがトランザクション時間データベース，有効時間のみをサポートするものが有効時間データベース，有効時間とトランザクション時間の両方をサポートするのがバイテンポラルデータベースと呼ばれる．

ID	名前	年代（有効時間）	トランザクション時間
001	織田信長	1534..1582	1985..*now*
002	豊臣秀吉	1536..1597	1995..2000
003	徳川家康	1542..1616	1998..*now*
004	武田信玄	1521..1573	2000..2001
002	豊臣秀吉	1536..1598	2000..*now*
...			

表-1　武将データベース

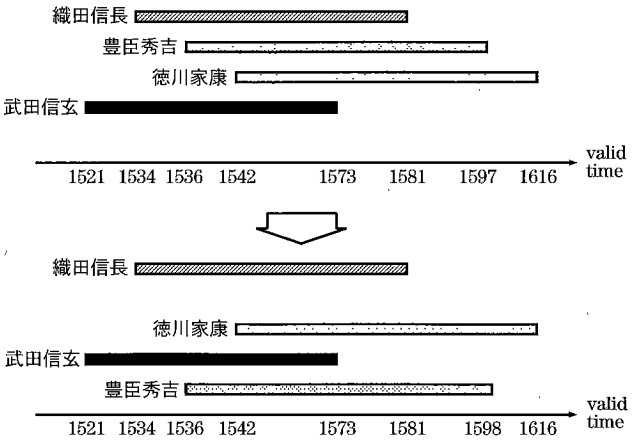


トランザクション時間DBでは，データの削除や更新は論理的に表現され，物理的なデータは保存される．

図-1　武将DBのトランザクション時間表現

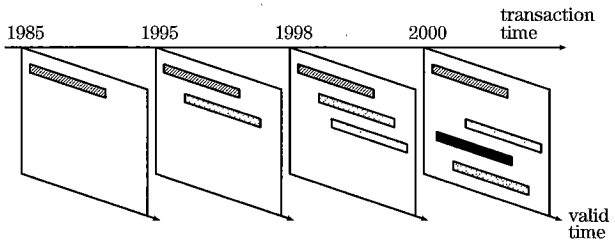
あるオブジェクトがデータベースに挿入された時刻をtとすると，そのトランザクション時間は時区間$[t, now)$によって表される．その後，時刻t'にそのオブジェクトがデータベースから削除されると，トランザクション時間は$[t, t')$に更新される．ここで注意したいのは，トランザクション時間においてオブジェクトは論理的に削除されることはあっても，実際に対応するレコードがデータベースから消え去ることはないという点である（削除されたオブジェクトも，トランザクション時間の終了時間を変更された上でデータベース内に保存される）．これにより，現在の状態だけでなく過去の履歴も操作することができる．

図-1はトランザクション時間データベースの例である．これは**表-1**に示した武将データベースの各レコードのトランザクション時間に対応している．たとえば，「豊臣秀吉」の没年を1597年から1598年に訂正しているが，これは既存オブジェクトの修正と新たなオブジェクトの挿入によって表されている．また2001年に削除



有効時間データベースでは，常に最新の情報だけが保存される．過去のデータは更新とともに捨てられてしまう．

図-2　武将DBの有効時間表現



有効時間とトランザクション時間を組み合わせて用いることによって，データベースの履歴をトランザクション時間上のスナップショットの系列として表現することができる．

図-3　バイテンポラルデータ

された「武田信玄」のデータも，実際には削除されたという事実がデータベースに存在することが分かる．さらに，データベースの更新は常に最新の状態に対して行われるので，後から挿入されるオブジェクトの開始時刻は，それ以前のものと比べて単調に増加する．トランザクション時間データベースでは，1) オブジェクトの操作履歴の格納，2) 最新のデータに対するデータの追加，削除，更新，および3) これらのデータへの高速なアクセスと検索処理が求められる．

一方，有効時間はトランザクションとは直交した概念であり，実世界におけるデータの有効性に関連した時間である．上の例でいうならば，各戦国武将たちが実世界に存在していた時間に対応する．つまり，有効時間データベースで格納の対象となるのは時区間となる．**図-2**を例にとると，それぞれの武将が実世界に存在していた時区間がデータベースに格納されている．このとき，有効時間データベースではデータの過去の履歴は保存されない．図-2は有効時間データベースにおけるデータ更新の様子を表している．ここでは，豊臣秀吉の没年の訂正により，そのデータの更新が行われた時点で過去の情報が失われる．有効時間データベースでは，1) 最新状態を表す時区間の格納，2) 時区間の追加，削除，更新，および3) これらの時区間への高速なアクセスと検索処理が求められる．

実世界の様子をより忠実に表現するには，トランザクション時間と有効時間を両方組み合わせて用いればよい．データベースは，トランザクションの実行とともに変化していく．この様子は時間軸上をスナップショットデータベースが遷移していく様子としてモデル化することができる（**図-3**）．図中の時区間は，実世界の対応する事象の各時点での最新の様子を表している．さらに，それらがデータベースに挿入された時間はトランザクション時間によって表現されている．

時間情報の索引技術

時制データの検索あれこれ

問合せ処理の視点から時制データを見ると，トランザクション時間データベースも有効時間データベースも，ともに時区間の集合と見なすことができる．このとき，時制データの検索には以下のようなものが考えられる．

1. ある連続した時区間$T=[t_s, t_e)$が与えられたときに，T内に存在したオブジェクトをすべて検索する（例：1600年から1605年に存在した武将を検索せよ）．
2. オブジェクトの持つ属性値に対して，検索キーが範囲$K=[k_s, k_e)$で与えられ，同様に時間の範囲Tが与えられるとき，属性値がK以内でかつT内に存在したオブジェクトをすべて検索する（例：石高5,000石以上で，1600年から1605年に存在した武将を検索せよ）．
3. 検索キーの範囲Kに関して，この範囲内の属性を持つオブジェクトの存在した時間をすべて検索する（例：石高5,000石以上の武将が存在した時間を検索せよ）．

時間軸に対する範囲検索は，時間Tによってデータベ

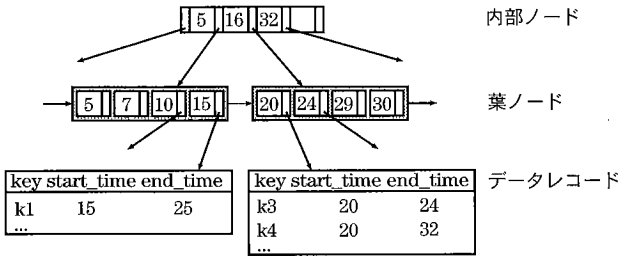


図-4 AP-tree

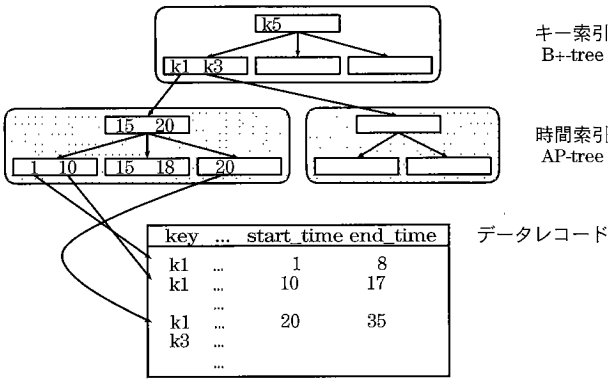


図-5 ST-tree

ースを切り出すことに相当するので，タイムスライス検索（timeslice query）とも呼ばれる．また，範囲検索の特殊な場合として，TやKを範囲ではなく点で与えることもできる．これを点検索（point query）と呼ぶ．トランザクション時間，有効時間，属性値はそれぞれ直交した軸なので，検索キーを与える際，各軸に与える条件の組み合わせ方は自由である．

時制データのための索引構造

時制データに対する質問処理を高速に処理するために，これまでに数多くの手法が提案されてきた．これらは，1）時区間をB+-treeに代表される1次元索引に格納する手法，2）時区間を2次元以上の多次元空間に写像し，R-treeなどの多次元索引を用いて検索処理を行う手法，および3）データベースの更新ログを用いるもの[3]に大別される．以下では，この中から1と2に分類される代表的な索引手法を紹介する．

1次元索引を利用した時間索引

AP-Tree（Append-Only Tree）

AP-tree[2]はB+-treeに基づいたシンプルな索引手法である．AP-treeは，トランザクション時間に対する範囲検索，点検索をサポートし，属性値は対象にしない．B+-treeはデータベースシステムで提供される一般的な索引構造であるので，それを使うことにより容易に実現することができる．

AP-treeでは，トランザクション時間データのみを対象とするので，物理的にはデータの追加のみが発生し，更新，削除は行われない（前述の通り，論理的なデータの更新，削除はデータの追加として表現される）．すべてのオブジェクトはその開始時刻の順にデータベースに挿入される．また，挿入時には必ず終了時間が確定していなければならない．終了時間の確定していないデータ，すなわち終了時間が*now*であるようなオブジェクトは，終了時間が確定してから索引に挿入される（図-4）．

葉ノードは(t, b)の形をしており，tは時刻，bはバケットへのポインタである．バケットはtの直前のエントリからtまでに発生したすべてのオブジェクトのレコードを保持している．前述の通り，データの更新は常に木の右端のノードに起きるので，これを意識した挿入アルゴリズムを採用している．つまり，挿入時に右端のノードが一杯の場合は，ノードの分割は行わず，新たなノードを作成して，適切な親ノードの子とする．つまり，AP-treeは平衡木ではないが，ノードの利用効率は右端のノードを除いて常に100%となる．また，データの更新は定数時間で行うことができる

しかしながら，検索処理の効率はよくない．時刻tに存在したオブジェクトを検索したいとすると，まずtを含む葉ノードまで木を辿る．ここで，tよりも右側のエントリは開始時刻がtより大きいためtを含むことはない．しかし，左側のエントリはtを含む可能性があるため，すべてのエントリを調べなければならない．

AP-treeから派生した索引として，属性キーと時間を両方サポートするST-treeが提案されている．これは属性キーに関するB+-treeを第1段に，そして各属性値に対応する時制データのAP-treeを2段目に配置する（図-5）．1段目のキー索引からは，対応するオブジェクトの最新のデータレコードへのポインタとAP-treeへのポインタが張られている．これによって属性値のみの検索と，属性と時間による検索の両方を高速に処理することができる．しかし，各検索キーごとにAP-treeが

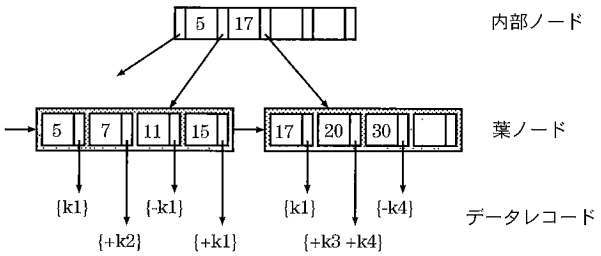


図-6　タイムインデックス

存在するため，索引のサイズが大きくなり，索引のメンテナンスに多大なコストがかかってしまうことがある．

タイムインデックスと単調（monotonic）B+-tree

タイムインデックス（Time Index）[8]の18章はAP-treeと同様に，B+-treeをベースにした索引手法であるが，AP-treeに比べて以下の特徴がある．

- データの追加だけでなく，任意の時刻における時区間の追加を行うことができる．つまりトランザクション時間だけでなく，有効時間のための索引としても使うことができる．
- 終了時間が確定していない時区間も格納することができる．
- 検索処理（点検索，範囲検索）をより高速に処理することができる．
- 空間効率は劣る．しかし，あらかじめトランザクション時間が対象だと分かれば，データ挿入アルゴリズムを工夫することによって，効率のよい格納が可能となる（単調（monotonic）B+-tree）．

時制データの索引を考えるとき，最も単純で高速な検索ができる構造は，すべての時刻において，その時に存在していたオブジェクトのコピーを保持する索引である．このとき，点検索に要する時間的複雑さは$O(\log_B n)$（nはオブジェクトの挿入，削除，更新の総数，Bはディスクページサイズ）である☆1．しかしながら，この方法は空間効率が非常に悪く現実的ではない．これは以下に示すデータの冗長性に由来する．1）オブジェクトの追加，削除のない時刻が連続している場合には，最新の変化が起こった時刻の状態をそのまま保持する．2）一般にオブジェクトの変更は一部のオブジェクトに対して行われるため，その他のオブジェクトは直前の状態と同一である．

タイムインデックスでは次の方法でこれらの無駄を省いている．まず索引点（indexing point）という概念を導入している．索引点とは，新しいオブジェクトの開始時刻，もしくはオブジェクトが削除された時刻の直後の時刻である．たとえば，T=[10, 15）の場合，10と16（=15+1）が索引点である．索引点は，実世界において何らかの変化が生じた時間なので，この点に関してのみオブジェクトの生起を記録すれば十分である．索引点はB+-treeの検索キーとして用いられ，葉ノードに配置される．葉ノードはディスクページへのポインタを保持する（図-6）．

ディスクページにはオブジェクト識別子が格納されるが，これは次のルールにしたがう．

1. 各葉ノードの先頭エントリから指されるページは「充填バケット（full bucket）」と呼ばれ，そのときに存在していたすべてのオブジェクト識別子が保持される．
2. 続くエントリは「増分バケット（incremental bucket）」と呼ばれ，充填バケットからの差分のみが格納される．

つまり，各葉ノードの先頭エントリは，その時点で存在していたすべてのオブジェクトを記録し，続くエントリは直前の索引点からの差分のみを記録する．これによって，記録に必要な容量を大幅に削減しているのである．

空間効率が優れる反面，すべての時点におけるオブジェクトを保持していないので，検索には若干余計な手間が必要である．点検索を行う場合，まず与えられた時刻tが索引点であるかどうかを調べる．索引点でなければ，tの直前の索引点t'を求める．次に，t'が充填バケットであるかどうかを調べ，もしそうであれば，その内容が答えとなる．そうでない場合は，先頭エントリのバケットに対してt'までのオブジェクトの増減を計算して，最終的な答えを得る．範囲検索もほぼ同様に計算を行うことができる．

タイムインデックスも，AP-tree同様，時間データのみを対象とした索引であるが，複数の索引を多段階に

☆1 本来はこれに加えてディスクページをメモリにロードする時間を考える必要があるが，ここでは無視する．

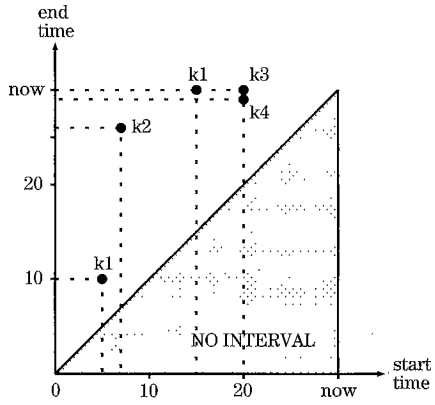


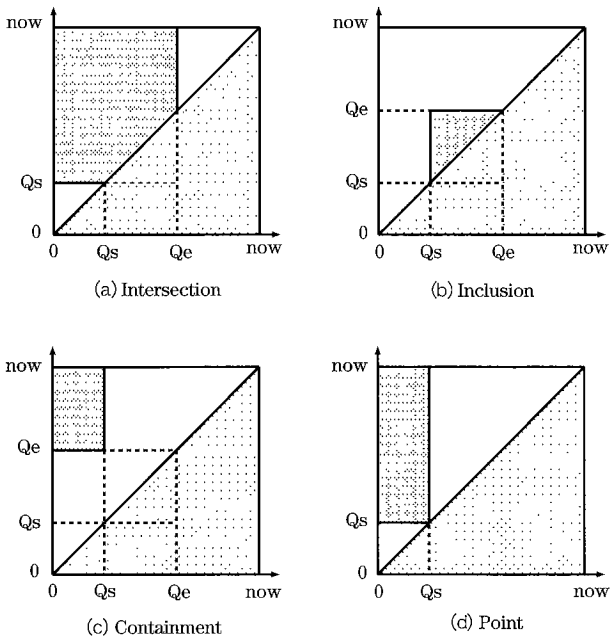
図-7　時区間の2次元空間への写像

重ねることによって，属性キーと時間を組み合わせた検索が可能になる．これを2レベル属性／タイムインデックス（two-level attribute/time index）と呼ぶ．構成はST-treeと同様なので，ここでは省略する．

多次元空間索引を用いた時間索引

ここでは，時区間の集合を多次元空間中の点または領域に写像し，R-treeに代表される多次元空間索引を用いて索引付けする手法をとりあげる[1), 4)]．この手法の大きな利点の1つは，次元数を増やすことによって，有効時間，トランザクション時間，属性等の複数の値を単一の索引で扱うことができることである．前節で説明した1次元の索引構造は，本質的に1つの索引で1つの次元しか扱うことができない．このため，同じことをするためには複数の索引を組み合わせる必要がある．

多次元空間を利用する際には，各次元にどの値を割り当てるかが重要である．これによって時区間がどのように多次元空間に写像されるかが変わってくる．よく用いられるのは，2次元空間のX軸，Y軸にそれぞれ時区間の開始時刻と終了時刻を割り当てる方法である☆2．これによって，時区間は2次元空間中の点に写像される（**図-7**）．図において，右下は開始時刻が終了時刻よりも大きくなるため時区間が写像されない領域である．上辺は*now*を表しており，時間の経過に伴って空間は広がっている．また，現在でも存在しているオブジェクトは上辺に常に張り付いている（終了時刻が*now*に等しい）．



(a) Qと重なりを持つ時区間を検索する場合，(b) Qに含まれる時区間を検索する場合，(c) Qを含む時区間を検索する場合，(d) 時刻Q_sを含むすべての時区間を検索する場合（点検索）

図-8　2次元空間での問合せ範囲

この索引を用いると，検索処理は質問区間で規定される2次元空間中の領域に含まれる点を検索する処理に置き換えられる．**図-8**（a）～（d）は，質問区間を$Q = [Q_s, Q_e)$としたときの範囲検索と点検索に対応する領域を図示したものである．

このバリエーションとしては以下のものがある．

- 上の手法に，さらに3つ目の次元として属性値を表す次元を加える方法．これにより，属性値と有効時間，トランザクション時間を同時に扱うことができる．
- X軸に有効時間（またはトランザクション時間），Y軸に属性を割り当てる方法（**図-9**）．これは2D R-treeとも呼ばれる．これによって，有効時間（トランザクション時間）と属性の両方を同時に扱うことができる．
- X軸とY軸にそれぞれ有効時間とトランザクション時間を割り当てる．その結果，任意のオブジェクトの時間属性は，2次元空間中の多角形に写像される．この索引では，バイテンポラルデータを単一の索引で扱うことができる（**図-10**）．

☆2 これと似た方法にY軸に時区間の長さ（終了時刻－開始時刻）を割り当てる方法もある．この場合，得られる空間の形が変わるが，本質的な違いはない．

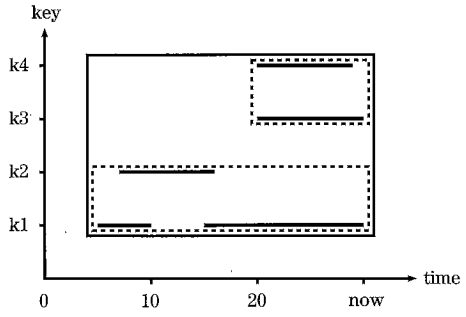


図-9　2D R-tree

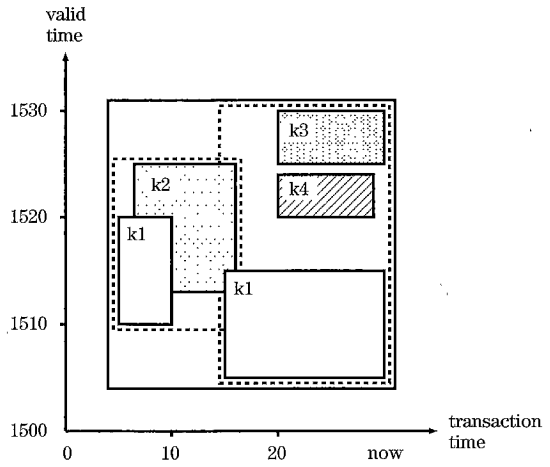


図-10　バイテンポラルインデックス

これらの手法にもいくつか問題がある．1つは，時間の経過に伴う空間の拡大である．時制データでは，時間の経過と共に常に*now*が増加し続ける．その結果，索引の定義域が広がっていくのである．これに対し，R-tree（やその他の多次元索引）では，空間の大きさがあらかじめ決まっているとしてその操作が定義されているため，時制データを扱う場合必ずしも最適な性能が得られるとは限らない．

また，軸ごとのデータの分布の違いがある．たとえば，有効時間，トランザクション時間，属性の3次元空間を考える．このとき，トランザクション時間に対応する軸では，常に座標の最大の場所にデータが挿入される．このため，木の構成が偏ってしまう可能性がある．また，現在も存在しているオブジェクトは，*now*の軸にくっついたまま，時間の経過に従って移動するので，これも索引の空間分割に悪影響をおよぼす恐れがある．

さらに，軸ごとの単位（セマンティクス）の違いもある．上の例でいうと，時間の軸は同じ単位であるが，属性の軸に関しては単位は何なのだろうか．また，座標的に近い場所にあったとして，それは時間が近いことと何の関連もないかもしれない．

TP-Index（Time-Polygon Index）

上で述べたような問題に対処した索引手法がTP-Index[5]である．TP-Indexは有効時間のタイムスライス検索のための索引構造である（**図-11**）．他の空間索引と同様，多次元空間中（ここでは2次元）の点集合を，領域の再帰的な分割によって行うのだが，R-treeのような矩形を用いず，時制データの特徴を活かした分割を行うのがTP-Indexの特徴である．

まず，すべての時区間$[t_s, t_e)$を2次元空間中の点$(t_s, t_e - t_s)$に写像する．これにより，すべての時区間はX軸，Y軸，時間の前線（time-front）で囲まれた三角形の領域に収められる．時間の前線は，タイムラインとも呼ばれ，時区間の存在し得る境界を表している．タイムラインは時間の経過とともに拡張し続ける．

領域に収められる点の数が一定値を超えると空間分割を行う．このとき矩形ではなく，時区間の特徴を考慮した包囲多角形（bounding polygon）を用いる．包囲多角形は，三角形の空間をそれぞれX軸とタイムラインに並行な線で分割して得られる形状であり，いくつかの基本型から構成される．この基本型は，空間全体を隙間なく埋め尽くすことができる．また，時制データの特徴である時間の経過に伴う空間の拡張やオブジェクトの追加も，基本型の組合せによって容易に表現することができる．領域を分割する際には，分割軸としてX軸かタイムラインを選択することになるが，その指針としては，1）分割後の領域に含まれる点集合の数がなるべく均等になり，かつ2）分割後の領域のサイズがなるべく等しくなる方の軸を選択する．

TP-Indexにおけるタイムスライス検索は，すでに説明した多次元索引を用いた方法とほぼ同様に行うことができる．すなわち，質問区間を点$Q = [Q_s, Q_e - Q_s)$に写像し，検索したい条件に合う領域を求める．さらに，TP-Indexを根ノードから再帰的に辿り，質問領域に含まれる点を検索する．

有効時間に加えて属性値を索引付けしたい場合は，第3の次元として属性の次元を設ければ，有効時間に加えて属性値も同時に索引付けすることができる．すなわち，任意のオブジェクトは，3次元空間中の点に写像

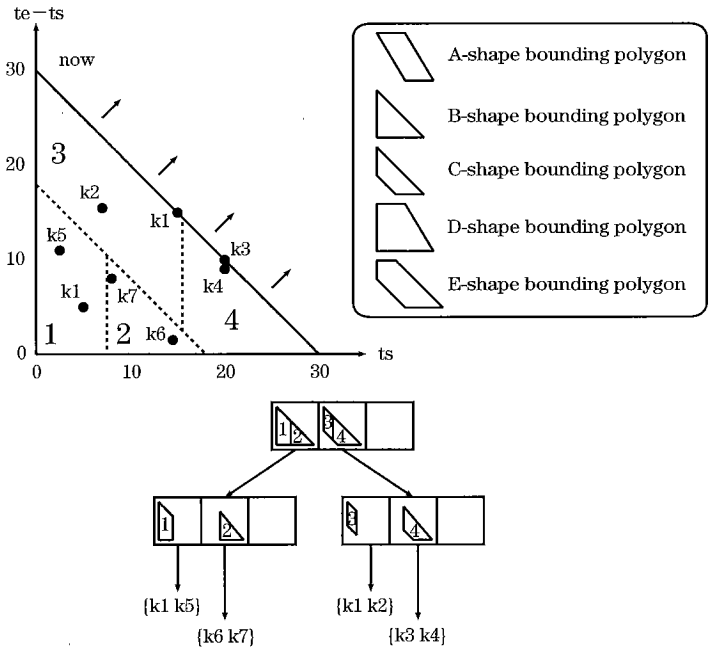


図-11 TP-Index

される．検索質問としては時間$Q=[Q_s, Q_e)$に加えて属性キー$K=[K_s, K_e)$を指定する．これにより，3次元の質問領域が規定され，これに含まれる領域を3次元のTP-Indexを根ノードから辿ることによって，質問処理を行うことができる．

ま と め

本稿では，時間の経過とともに変化するデータのための時制データベースについて，その基本的事項と時制データへの高速なアクセスを可能にする索引手法を紹介した．

データは本質的に時間の経過とともに変化するものである．この十年来，時制データベースに関する研究は活発に行われてきた．しかし現在のところ，有効時間やトランザクション時間をサポートする真の時制データベースは商用ではほとんど存在しない．1999年に制定されたISO/ANSI標準のSQL99では，TSQL2[6]の成果がPart 7: Temporal (SQL/Temporal) として取り込まれたこともあり，今後，商用システムでのサポートが期待される．

さらに，計算機技術の過去数十年の長足の進歩とともに，計算機の応用範囲は飛躍的に広がった．これに合わせて，データベースが扱うデータの種類も同様の広がりをみせている．ディジタル放送等のストリーミング系のメディア，XML文書に代表される構造化文書，GPSを利用した位置情報のトラッキング等の新しい応用も時間と密接な関連を持っている．今後は，このような時制データに対する高速な検索を可能にする索引技術の開発が望まれる．

参考文献
1) Bertino, E., Ooi, B. C. , Sacks-Davis, R., Kian-Lee Tan, Zobel, J., Shidlovsky, B. and Catania, B.: Indexing Techniques for Advanced Database Systems, Kluwer Academic Publishers (1997).
2) Gunadhi, H. and Segev, A.: Efficient Indexing Methods for Temporal Relation, IEEE Trans. on Knowledge and Data Engineering, Vol.5, No.3, pp.496-509 (1993).
3) Jensen, C. S., Mark, L., Roussopoulos, N. and Sellis, T.: Using Differential Techniques to Efficiently Support Transaction Time, VLDB Journal, Vol.2, No.1, pp.75-111 (1992).
4) Salzberg, B. and Tsotras, V. J.: A Comparison of Access Methods for Temporal Data, Technical Report TR-18, TimeCenter (June 1997) , http://www.cs.auc.dk/research/DP/tdb/TimeCenter/publications2.html
5) Shen,H., Ooi, B. C. and Lu, H.: The TP-Index: A Dynamic and Efficient Indexing Mechanism for Temporal Databases, In Proc. of the 10th Int'l Conf. on Data Engineering, pp.274-281 (1994).
6) Snodgrass, R., Ahn, I., Ariav, G., Batory, D., Clifford, J., Dyreson, C., Elmasri, R., Grandi, F., Jensen, C., Käfer, W., Kline, N., Kulkarni, K., Leung, T., Lorentzos, N., Roddick, J., Segev, A., Soo, M. and Sripada, S.: The TSQL2 Temporal Query Language, Kluwer Academic Publishers (1995).
7) Snodgrass, R. T. and Ahn, I.: Temporal Databases, IEEE COMPUTER, Vol.19, No.9, pp.35-42 (1986).
8) Tansel, A. U., Clifford, J., Gadia, S., Jajodia, S., Segev, A. and Snodgrass, R.: Temporal Databases: Theory, Design, and Implementation, The Benjamin/Cummings Publishing Company, Inc. (1993).
9) 増永良文: マルチメディアデータベースと時間, 情報処理, Vol.36, No.5, pp.369-377 (May 1995).

(平成13年8月10日受付)