データベースシステム　122－55
(2000. 7. 28)

知識獲得を用いて圧縮されたデータベースのための索引機構の実現

工藤 祐介† 春本 要‡ 西尾 章治郎†
†大阪大学大学院工学研究科 情報システム工学専攻
〒565-0871 大阪府吹田市山田丘 2-1
TEL: 06-6879-7817
E-mail: {kudo,nishio}@ise.eng.osaka-u.ac.jp

‡大阪大学 サイバーメディアセンター
〒567-0047 茨木市美浦ヶ丘 5-1
TEL: 06-6879-8792
E-mail: harumoto@cmc.osaka-u.ac.jp

あらまし
近年，電子データの取り扱い量が膨大になり，大規模なデータベースの使用が一般的なレベルまで広がっている．このような状況に対し，これまでに筆者らの研究グループでは，データベースからの知識獲得によって得られるルールを利用し，データベースを圧縮する手法を提案している．しかし，これまで圧縮されたデータベースに対する索引機構について考慮していなかった．索引は，特に大規模なデータベースシステムにおいては非常に重要な役割を果たす．本稿では，圧縮されたデータベースのための索引機構として，各分割表に索引を付加する分割索引と，複数の分割表を一つの表とみなして索引を付加する仮想索引の二つの機構について述べ，その実現方法および評価実験結果を示す．

キーワード　索引機構，データベース圧縮，データベースからの知識獲得，関係データベース

Realization of Indexing Mechanisms for Databases Compressed with Knowledge Discovery Techniques

Yusuke KUDO† Kaname HARUMOTO‡ Shojiro NISHIO†

†Department of Information Systems Engineering,
Graduate School of Engineering, Osaka University
2-1 Yamadaoka, Suita, Osaka 565-0871
TEL: 06-6879-7817
E-mail: {kudo,nishio}@ise.eng.osaka-u.ac.jp

‡Cybermedia Center, Osaka University
5-1 Mihogaoka, Ibaraki, Osaka 567-0047
TEL: 06-6879-8792
E-mail: harumoto@cmc.osaka-u.ac.jp

Abstract
Recently, very large databases have commonly been used to manage large volumes of electronic data. As a result, the storage cost is increasing. To cut down the storage cost, we have proposed a database compression mechanism using knowledge discovery techniques. However, so far we have not considered the indexing mechanism for compressed databases, which plays an important role especially in large databases. In this paper, we show two indexing mechanisms, i.e., the divided indexing and the virtual indexing, for databases compressed with knowledge discovery techniques. The divided indexing mechanism adds an index to each fragmented table, while the virtual indexing mechanism adds a single index to the fragmented tables as if they were a single table. We also give evaluation of these two indexing mechanisms.

key words　indexing mechanism, database compression, knowledge discovery in database, relational database

1 はじめに

近年，計算機の普及とネットワーク環境の整備により，電子データの取り扱い量が膨大になり，大規模なデータベースの使用が一般的なレベルまで広がっている．近年の技術発展により，そのような大規模なデータを蓄える二次記憶装置は低価格化しているが，それでもまだデータの保持にかかるコストは重要な要素である．このような環境に対して，データベースのサイズを圧縮することが有効であると考えられる．

これまでに筆者らの研究グループでは，データベースからの知識獲得手法 [5, 6] を用いたデータベース圧縮手法 [2, 3, 7, 8, 9] を提案してきた．この手法ではデータベースから獲得した知識を用いて，重複するデータを演繹データベースのルールに置き換えることによりデータベースを圧縮する．また，ルールを表として記録し，ビュー機能を用いることにより関係データベースシステム上でこの手法を実現した [10, 11]．この手法では，関係データベースから重複するデータをルールとして抽出し，その重複データを，ルールと置き換え，複数の表群に分割することにより圧縮を行う．その際，ルールは関係データベースに表として格納し，ビュー機能を用いて元の表を復元する．これによって，通常の表と全く同じように，ビューに対して選択操作を行うことができる．また，圧縮された表に対する挿入，削除，更新等の操作は分割表群への操作に変換することで実現することができる [11]．

しかし，これまで圧縮データベースの問合せの高速化と整合性制約に関して考慮していなかった．通常のデータベース管理システムでは，このような目的のために索引が利用されるが，提案してきた手法により圧縮された表は，構造の異なる複数の表群に分割されるため，従来の索引をそのまま使用することはできない．そこで，本研究では知識獲得を用いたデータベース圧縮手法のための索引機構を設計・実装し，その評価を行った．

2 知識獲得によるデータベース圧縮手法

知識獲得によるデータベース圧縮手法では，データベースからの知識獲得手法を用いて，表からデータの重複をルールとして抽出し，ルールを対応表として関係データベースに格納する [10, 11]．ルールの抽出にはアプリオリ・アルゴリズム [1] を用いる．例として，図1の“売上表”を考える．この表にしきい値2として，アプリオリ・アルゴリズムを適用すると

- (日付 = '0131')
- (日付 = '0201')
- (コード = '99990001')
- (コード = '33330001')
- (数量 = 1)
- (数量 = 2)
- (数量 = 3)
- (日付 = '0131', 数量=1)
- (日付 = '0131', 数量 = 2)

といったルール候補が抽出される．

抽出したルール候補を表に対して適用する．このとき，ルール候補からルールを選択して適用する順序によって適用されるルールの数および種類が異なるため，圧縮率も変化する．ルール適用順序を決定する手法については文献 [4] で議論されている．

この例ではデータの減少量が最大になるルール候補から順に適用することにする．日付のデータサイズを4バイト，コードのデータサイズを8バイト，数量のデータサイズを8バイトとして計算すると，この表に対しては以下の順序でルールが適用される．

1. (日付 = '0131', 数量 = 1)
2. (日付 = '0131', 数量 = 2)
3. (コード = '99990001')

これらのルールを選択，適用して圧縮した結果を図2に示す．対応表は，ルールとそれに対応する分割表の表名を格納した表である．分割表名のサイズを4バイトとすると，この例では 160 バイトから 126 バイトに圧縮されている．

関係データベースシステムのビュー機能を用いて圧縮前の表をビューとして定義することにより，圧縮した表に対する問合せを実現する．この例では，以下の SQL 文で定義したビュー“売上表”に対して問合せを行うことで，元の表に対する問合せを実現することができる．

```
CREATE VIEW 売上表 AS
  SELECT '0131' AS 日付, コード, 1 AS 数量
  FROM 分割表1
  UNION ALL
  SELECT '0131' AS 日付, コード, 2 AS 数量
  FROM 分割表2
  UNION ALL
  SELECT 日付, '99990001' AS コード, 数量
  FROM 分割表3
  UNION ALL
  SELECT 日付, コード, 数量
  FROM 分割表4
```

このようにして定義したビュー“売上表”を用いることにより，圧縮したままでの問合せを実現できる [11]．

3 圧縮された表のための索引

2節で示した通り，提案してきた圧縮手法では，圧縮によって複数の構造の異なる表に分割される．したがって，圧縮表に対して既存の索引をそのまま使用することはできない．

索引を使用することによって，特定の属性を条件とした検索を高速化することができる．索引にはキー値でソート

売上表

日付	コード	数量
0131	11110001	1
0131	22220001	1
0131	33330001	1
0131	44440001	2
0131	55550001	2
0131	99990001	3
0201	99990001	2
0201	33330001	3

図 1: 元となる関係

対応表

日付	コード	数量	分割表名
0131	-	1	分割表 1
0131	-	2	分割表 2
-	99990001	-	分割表 3

分割表 1

コード
11110001
22220001
33330001

分割表 2

コード
44440001
55550001

分割表 3

日付	数量
0131	3
0201	2

分割表 4

日付	コード	数量
0201	33330001	3

図 2: 圧縮前の関係と圧縮後の関係（売上表）

されたデータレコードに直接作られる主索引と，それ以外の属性に作られる二次索引がある．本研究では二次索引を実装し評価を行った．一般に二次索引は，キー値とその値をもつタプルを一意に識別するタプル ID の組を，高速な探索が可能なデータ構造に記録することにより実現される．このデータ構造としては B ツリーまたはその変種やハッシュなどが用いられる．

索引を付加した属性を条件として，タプルを検索するときは，まず索引を探索して，条件を満たすタプルのタプル ID を取得する．こうして取得したタプル ID を用いて，表のタプルに直接アクセスする．索引の探索は高速であるため，表の検索を高速化することができる．また，“ある属性の値がある値をとるタプルが存在するか” という問合せには，対象となる属性に二次索引を付加しておけば，索引の探索のみで答えることができる．したがって索引は属性の一意性制約の実現にも利用される．

このように，索引は問合せ処理の高速化に有効であり，

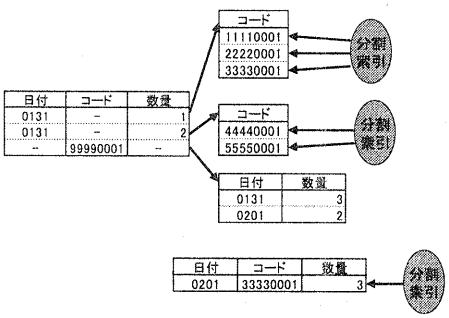

図 3: 分割索引

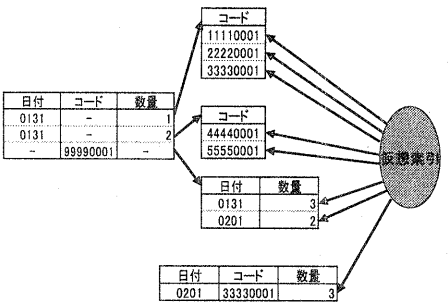

図 4: 仮想索引

実用的なデータベースシステムには不可欠である．ところが提案してきた手法を用いて圧縮された表は，構造の異なる複数の表に分割されるため，既存の索引をそのまま使用することはできない．そこで，圧縮された表のための索引手法として二つの手法を考える．

3.1 分割索引

まず第一に，元の表が分割されたのにしたがって，索引も分割して付加するという手法が考えられる．この手法による索引を分割索引と呼ぶ．概念図を図 3に示す．分割索引では，各分割表に対する索引として既存のものが使用できるため，既存のデータベース管理システムを用いた実現が容易である．また，多数のルールに関係する属性に索引を付加する場合，索引を付加すべき属性をもたない分割表には索引を付加する必要がないため，全体の索引サイズの減少が期待できる．しかし，表の検索や更新の際には，索引の管理が複雑となり，すべての分割表の索引に対して走査，更新が必要となるので，問合せ処理に対する効果は低いと考えられる．

3.2 仮想索引

もう一つの手法として，索引を分割せずに，仮想的に元の表に対する索引を作成するという手法が考えられる．この手法による索引を仮想索引と呼ぶ．概念図を図4に示す．仮想索引では，元の表のタプルと分割表のタプルの対応をとり，仮想的に元の表に対して索引を付加する．この手法では，原理的には圧縮前の表に対する索引と同一の構造となる．したがって索引サイズおよび索引の探索時間ともに，圧縮前と同等である．また，一つの索引であるため，索引の管理が単純であり，分割表全体にまたがった属性値の一意性制約も，容易に実現可能である．

4 索引の構成と実装

4.1 実装環境

データベース管理システムとして PostgreSQL6.5.2 を使用し，この索引機構を拡張した．また PostgreSQL では索引のデータ構造として B ツリーやハッシュなどが利用できるが，ハッシュ索引では比較演算子として大小比較演算子が使用できない，複数の属性の組に対する索引をサポートしないなどの制限があるため，今回実装したのは B ツリー索引のみである．

4.2 分割索引の構成と実装

4.2.1 索引の作成

圧縮済みの表に索引を作成するときは，以下のようにして索引の作成文 (CREATE INDEX 文) を変換する．

1. 索引を作成する対象の表が，圧縮表であるか調べる．
2. 圧縮表であれば，ルールを記録した対応表を走査し，各タプルごとに，索引を付加する属性がルールの一部となっているか調べる．
   (a) ルールの一部となっている場合
   対応する分割表には索引を付加する属性がないので何もせずに，次の分割表を調べる．
   (b) ルールの一部となっていない場合
   その分割表の同じ属性に索引を付加する．
3. ルールに対応しない分割表に対して索引を付加する．

例として，2節で圧縮の例を示した“売上表”の属性“コード”に索引を付加する場合を考える．まず，対応表を走査し，対応表の各タプルごとに，属性“コード”がルールの一部となっているか調べる．ルールの一部となっていない場合には，対応する分割表に対して，属性“コード”に索引を付加する．ルールの一部となっている場合には，対応する分割表には索引を付加すべき属性“コード”がないので，この分割表に対しては索引を作成しない．最後にルール未対応の表“分割表4”には通常の索引を付加する．

4.2.2 タプルの検索

圧縮表に対する問合せを実行するために，まずビュー定義文を実行して元の表を復元し，その結果に対して問合せを実行した場合，分割索引は利用されない．これは，復元された表には利用できる索引は存在せず，さらに元の表の復元のために実行されるビュー定義文には，索引を付加した属性は関係しないためである．分割索引を利用するためには，各分割表ごとに問合せを実行したのち，元の表に対する問合せ結果を合成しなければならない．したがって，分割索引の実現のためには，問合せ時に圧縮表に対する SQL 文を分割表に対する SQL 文に変換する機構が必要となる．

索引を付加された圧縮済みの表の検索を行うときには，以下のようにして検索文 (SELECT 文) を変換する．

1. 検索対象の表に圧縮表を含むか調べる．
2. 検索対象に圧縮表を含んでいれば，対応表を走査し，各タプルごとに，選択属性および検索条件の属性がルールの一部となっているか調べる．
   (a) ルールの一部となっている場合
   検索対象の圧縮表を 対応する分割表に置き換え，選択属性，および検索条件中のそれらの属性を，対応表から得たルールの属性値に置き換える．
   (b) ルールの一部となっていない場合
   選択属性がルールの一部となる属性を含んでいなければ，検索対象の圧縮表を対応する分割表に置き換える．
3. ルールに対応しない分割表を対象とする部分的な問合せを作成する．
4. これらの各分割表に対するすべての部分的な問合せの和をとる．

圧縮後に属性“コード”に索引を付加した“売上表”から，コードが '99990001' である商品が売れた日付を調べる問合せを考える．この問合せを SQL で書くとこのようになる．

```
SELECT 日付
FROM 売上表
WHERE コード = '99990001'
```

この問合せは，以下のように分割表ごとの部分問合せの和 (UNION) に変換される．

```
SELECT '0131' AS 日付
FROM 分割表1
WHERE コード = '99990001'
UNION
SELECT '0131' AS 日付
FROM 分割表2
WHERE コード = '99990001'
UNION
SELECT 日付
FROM 分割表3
WHERE '99990001' AS TEXT = '99990001'
UNION
```

```
SELECT 日付
FROM 分割表4
WHERE コード = '99990001'
```

このようにして等価なSQLに変換し，圧縮した状態での問合せを実行する．

4.3 仮想索引の実装

仮想索引を実装するためには，圧縮前の元の表のタプルと，圧縮後の表の各分割表のタプルを対応付ける機構が必要となる．本研究では，分割表間にまたがってタプルを一意に識別できる以下のような仮想IDを用いて索引を実装した．

- 仮想IDは14ビットの分割表番号，18ビットの分割表内ブロックID，16ビットのブロック内オフセットで構成される．このIDのサイズはPostgreSQL6.5.2のタプルIDのサイズと同じであるため，この拡張によって索引エントリのサイズは圧縮前後で変わらない．
- 分割表番号を記録し，分割表内ブロックIDを16進数3ffff，ブロック内オフセットを0としたものを，その分割表全体を指す特殊な 仮想IDとする．

4.3.1 索引の作成

圧縮済みの表に索引を作成するときには，仮想IDを用いて以下のような手順で索引を作成する．

1. 索引を作成する対象の表が，圧縮表であるか調べる．
2. 圧縮表であれば，対応表を走査し，各タプルごとに索引を付加する属性がルールの一部となっているかを調べる．
   (a) ルールの一部となっている場合
   対応する分割表の番号から，分割表全体を指す特殊な 仮想IDを合成し，索引のエントリを作成する．
   (b) ルールの一部となっていない場合
   対応する分割表を走査し，各タプルのブロックID，ブロック内オフセットを調べ，元のタプルの仮想IDを合成し，索引のエントリを作成する．
3. 最後に，ルールに対応しない分割表を走査して，同様に各タプルの仮想IDを求め，索引のエントリを作成する．
4. これらのエントリに対してBツリーによる索引を作成する．

例として，図2に示した圧縮された“売上表”の属性“コード”に索引を付加する場合を考える．

対応表の各タプルごとに，索引を付加する属性“コード”がルールの一部となっているか調べる．最初のルールには属性“コード”が含まれていない．よって，対応する分割表1のすべてのタプルに対してタプルIDを調べ，そのタプルIDと分割表番号1から仮想IDを合成し，索引のキーとなる属性の値との組を索引のエントリとして作成する．3つめのルールには属性“コード”が含まれている．よって，分割表番号から分割表全体を指す特殊なIDを合成し，それと索引のキーとなる属性の値から索引のエントリを作成する．このようにすべてのルールについてエントリを作成し，Bツリー構造に記録する．

4.3.2 タプルの検索

索引を付加された圧縮済みの表の検索を行うときには，以下のようにして元のタプルを復元する．

1. Bツリーを走査して，検索するタプルの仮想IDを求める．
2. 仮想IDから分割表番号を求め，ルールを記録している対応表から，対応するルールの属性値を求める．
   (a) 仮想IDが分割表全体を指す場合
   その分割表を走査して全タプルの属性値を求め，ルールの属性値と合成することにより圧縮前の表のタプルを復元する．
   (b) 仮想IDが分割表全体を指さない場合
   その分割表内で，仮想IDの分割表内ブロックIDと，ブロック内オフセットから分割表の属性値を求め，ルールの属性値と合成することにより圧縮前の表のタプルを復元する．

圧縮後，属性“コード”に索引を付加した，“売上表”に対して分割索引の例と同様に，属性“コード”の値が’99990001’である商品が売れた日を求める問合せを考える．この場合，まず属性“コード”に付加された仮想索引のBツリーを走査し，キーの値’99990001’に対応する仮想IDを求める．

このとき，分割表3全体を指す特殊な仮想IDが得られるので，分割表3を全走査して，ルール以外の属性“日付”，“数量”の値を求め，対応表にルールとして記憶しているルールの属性“コード”の属性値と合成することにより元のタプルを復元する．このようにして，検索条件を満たす元の表のタプルを求めることができ，これに対して属性“日付”で射影をとることで，問合せの結果を得る．

5 実験による評価

5.1 実験条件

実験には，書店の売上表を想定した合成データベースを用いた．売上表(SALE)は属性として日付(DATE)，書籍コード(CODE)，売上数(AMOUNT)をもち，タプル数は100,367，テーブルサイズは8.09MBである．この表を圧縮し，圧縮前後の表に対する各索引のサイズ，各索引の作成時間，問合せ処理時間を測定した．圧縮によって，186個のルールが発見され，ルールに対応しないタプルを記録した分割表と合わせて187個の分割表に分割された．これら186個のルールのうち“書籍コード”に対するルールは0個，“日付”に対するルールは78個，“数量”に対

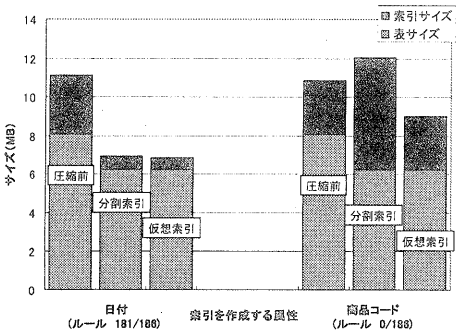

図 5: 索引のサイズの比較

するルールは 5 個，“日付” と “数量” の組に対するルールは 103 個であった．実際の売上表を想定し，属性 “書籍コード”，“日付” に対して索引を付加した．

5.2 索引のサイズ

圧縮前の売上表の各属性に付加した通常の索引および，圧縮後の売上表の各属性に付加した分割索引・仮想索引それぞれのサイズを測定した．結果を図 5に示す．

多数のルールに関係する属性 “日付” に索引を作成した場合，仮想索引，分割索引はそれぞれ 4.281 MB(約 38.40%)，4.227 MB(約 37.91%) サイズが減少している．これは多くのルールに関係する属性に索引を付加するとき，仮想索引では索引エントリの重複を減少させることができ，また分割索引では索引を付加すべき属性をもつ分割表の数が減少するためである．

また，ルールに関係しない属性 “書籍コード” に対する索引では，表と索引の合計サイズの減少量は最大でも 1.844 MB(約 16.95%) しか減少せず，また分割索引の場合には，逆に 1.164 MB (約 10.70%) サイズが増加している．これはルールに関係しない属性に索引を付加するとき，仮想索引では索引のエントリ数，エントリサイズおよび索引の構造が圧縮前の表に対する索引と変わらないのに対して，分割索引ではエントリ数とエントリサイズは変わらないが，索引の構造が大きく変わり領域の使用量が増えるためである．

5.3 索引の作成時間

圧縮前の売上表の各属性に通常の索引を作成したときの作成時間と，圧縮後の売上表の各属性に分割索引・仮想索引を作成したときの作成時間を測定した．結果を表 1に示す．分割索引では，索引を付加しなければならない分割表が多数あるとき，索引の作成が非常に遅くなっている．また，仮想索引の作成にはルールを記録した対応表の走査を行い，対応する分割表の番号と分割表内のタプル ID から仮想的なタプル ID を合成するのに多少の時間がかかるため，ルールに関係しない属性 “書籍コード” に索引を付加するためには圧縮前の索引の作成よりも若干遅くなっている．一方，ルールに関係する属性に対する索引を付加するときは，分割索引，仮想索引ともに圧縮前の索引よりも短時間で作成できる．

表 1: 索引の作成時間の比較

	索引を付加する属性	
	日付	書籍コード
圧縮前	25.99 秒	26.47 秒
分割索引	25.29 秒	732.67 秒
仮想索引	8.70 秒	27.76 秒

5.4 問合せ処理時間

売上表に対して問合せを実行し，選択されるタプルの割合とその問合せにかかる処理時間を計測した．各属性について，選択されるタプルの割合と問合せ処理時間の関係を図 6，7に示す，

図 6は最も多くのルールに関係する属性 “日付” に索引を付加した場合の結果を示している．図中の点線で示された 2 つのグラフはそれぞれ圧縮前の表に対する索引を用いた場合と，索引を用いない場合の実験結果である．この結果から分割索引では圧縮後の索引を用いない場合と比べて，問合せ処理の高速化がほとんど行われていないことがわかる．一方，仮想索引では圧縮前の索引を用いた場合と同等の処理時間を実現している．

また，図 7はどのルールにも関係しない属性 “書籍コード” に索引を付加した場合の結果を示している．分割索引は圧縮後の索引がない場合より問合せに時間がかかっている．索引サイズと問合せ処理時間の実験結果から，分割索引は圧縮表に対する索引として適していないことが明らかである．よって以下では仮想索引のみを議論の対象とする．

仮想索引では選択されるタプル数が少数であれば，圧縮前に索引を用いた場合と近い効果が得られている．しかし，選択されるタプルが多数になるにしたがって，圧縮前に索引を用いた場合との差が拡大している．どのルールにも関係しない属性に対する仮想索引であれば，その構造は圧縮前の索引と同じ構造であり，索引の探索時間に差はないはずである．ここで，圧縮前の表に対する索引と問合せ処理時間に差が出ているのは，仮想索引では索引を付加される分割表のタプルがそのまま求める結果とはならず，対応するルールの値と合成して圧縮前の表のタプルを復元する必要があり，この処理に時間がかかるためであると考えられる．これは，現在はまだ問合せの最適化を考慮しておらず，選択されたすべてのタプルに対して復元操作を行っているためである．しかし，例えば複数の条件をも

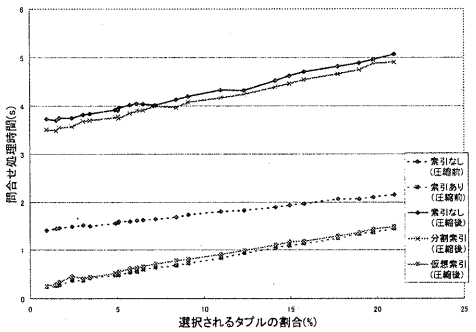

図 6: “日付” を条件とする問合せの選択タプルの割合と問合せ処理時間

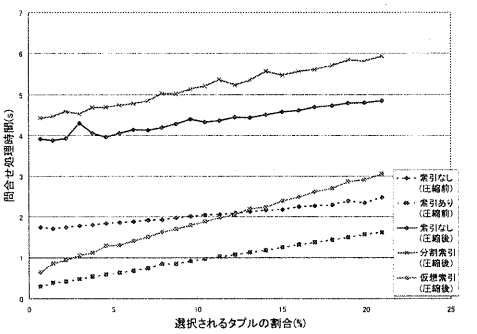

図 7: “書籍コード” を条件とする問合せの選択タプルの割合と問合せ処理時間

つ問合せを処理する場合，他の条件を満たさなければタプルを合成しないことによってタプルの合成処理の回数を減少させることができる．また，検索条件と射影属性のどちらにも関係しない属性は合成する必要がないので，タプル合成処理の時間を短縮することもできる．このような状況を考慮にいれた問合せ最適化を行うことによって，提案した索引機構を用いた問合せ処理をより高速化できると考えられる．

5.5 サーバの環境が問合せ処理時間に与える影響

図 8，9はサーバのバッファサイズが小さい場合に “日付” および “書籍コード” に仮想索引を付加し，その属性を条件とする問合せ処理時間をグラフにしたものである．多くのルールに関係する属性 “日付” に仮想索引を付加した場合は，アクセス速度はバッファサイズの大小に関係しない．一方，関係するルールの少ない属性 “書籍コード” に付加した仮想索引では，バッファが少ないときにグラフの傾きが増加している．これは，仮想索引を用いた場合に対応表と分割表の複数のブロックに同時にアクセスしなければならない状況が頻繁に生じるためである．このときサーバのバッファサイズが小さいと，ディスク入出力が多数発生し，アクセス速度が低下する．一方，多くのルールに関係する属性に付加された仮想索引では，同じキー値をもつタプルが一つの分割表にまとまっているため，同時アクセスされるブロック数は少なく，小さなバッファ領域しかなくても，ディスク入出力が発生しにくいため，問合せ処理速度があまり変化しない．

5.6 大規模データベースに対する性能

図 10，11 は前の実験で用いた表の 10 倍，100 万行 93MB の表に対して同様の実験を行った際の選択されるタプル数とその問合せ処理にかかる時間の関係をまとめたものである．圧縮により表のサイズは，89.0 MB から 62.5 MB へ 26.5MB(約 29.8%) 減少した．圧縮時に発見された

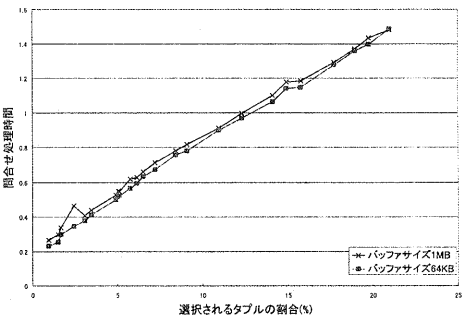

図 8: バッファサイズの大小による “日付” に関する仮想索引の問合せ処理速度の比較

ルールは 934 個で，属性 “日付” に関係するルールは 928 個，“書籍コード” に関係するルールは 0 個であった．なお，実装上の問題により，圧縮後の索引を使用しない場合の実測はできなかった．この実験でも前の実験と同様の結果が得られている．

この結果から，使用するデータベースの属性値の分布特性が同じであれば，大規模なデータベースでも同様の効果が得られるものと考えられる．

6 おわりに

本研究では，データベースからの知識獲得手法を用いて圧縮されたデータベースのための索引機構について設計・実装し，実験によって評価を与えた．実験の結果から，圧縮表に対する索引としては分割索引は適しておらず，問合せ処理の高速化に対する効果が得られないことが明らかとなった．これに対して仮想索引は問合せ処理の高速化に対して効果があり，また，索引のサイズは圧縮前の表に対

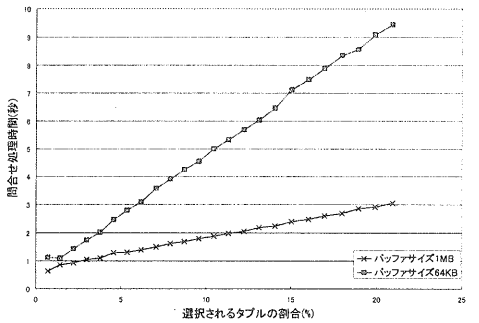

図 9: バッファサイズの大小による “書籍コード” に関する仮想索引の問合せ処理速度の比較

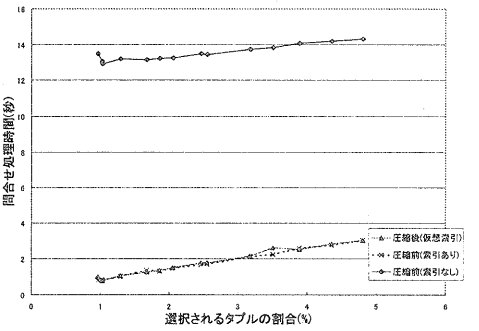

図 10: 大規模な表に対する属性 “日付” を条件とした問合せ処理時間

する索引のサイズをより大きくなることはなく，重複を減少させることによって索引サイズが減少することがある．そしてこの手法は比較的大規模なデータベースについても有効である．しかし，分割索引でも分割表が小さい場合には索引を付加しないなどの工夫をすれば，索引サイズの減少と索引探索時間の短縮を実現できると考えられる．

今後の課題としては，表の更新に伴う索引の更新処理を実装すること，分割索引や仮想索引の特性を考慮した問合せ最適化機構を実現することなどがある．また，ルールとなっている属性に対して索引を付加した場合には，仮想索引のサイズは圧縮前の表に対する索引のサイズより減少するので，圧縮時に索引を付加することがわかっている属性を，できるだけルールとして選択するようにすると，索引を含めたデータサイズの減少量が最大となるようなルール選択手順が考えられる．このように，索引サイズの減少量も考慮した圧縮時のルール選択手法を設計することも今後の課題である．

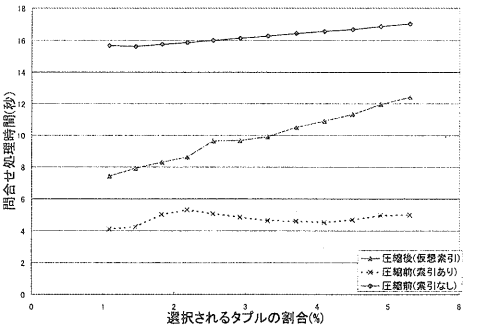

図 11: 大規模な表に対する属性 “書籍コード” を条件とした問合せ処理時間

参考文献

[1] R. Agrawal and R. Srilant: “Fast algorithms for mining association rules,” in *Proc. the 20th VLDB Conf.*, pp. 487–499 (Sept. 1994).

[2] 相坂一樹, 呉乾禮, 塚本昌彦, 西尾章治郎: “ データベースからの知識獲得による圧縮機構をもつデータベースシステムの設計と実装,” 電子情報通信学会第 8 回データ工学ワークショップ (DEWS'97) 論文集, pp. 251–256 (Mar. 1997).

[3] 相坂一樹, 呉乾禮, 塚本昌彦, 西尾章治郎: “データベースからの知識獲得を用いたデータベース圧縮システム REDUCE1,” 情報処理学会 第 55 回全国大会講演論文集 (3), pp. 407–408 (Sept. 1997).

[4] 相坂一樹, 塚本昌彦, 春本要, 西尾章治郎: “知識獲得を用いたデータベース圧縮のためのルール選択方法について,” 情報処理学会研究報告 (データベースシステム研究会 99-DBS-117), vol. 99, no. 6, pp. 65–70 (Jan. 1999).

[5] M. -S. Chen, J. Han, and P. S. Yu: “Data mining: An overview from a database perspective,” *IEEE Trans. on Knowledge and Data Eng.*, vol. 8, no. 6, pp. 866–883 (Dec. 1996).

[6] U. M. Fayyad, G. Piatetsky-Shapiro, and R. Uthurusamy: *Advances in Knowledge Discovery and Data Mining*, AAAI Press / The MIT Press (1996).

[7] C. -L. Goh, K. Aisaka, M. Tsukamoto, K. Harumoto, and S. Nishio: “Database compression with data mining methods,” in *Proc. the 5th Int. Conf. on Foundations of Data Organization (FODO'98)*, pp. 97–106 (Nov. 1998).

[8] C. -L. Goh, M. Tsukamoto, and S. Nishio: “Compressing databases with knowledge discovery techniques,” *Technical Report, ISE-TR-97-001, Department of Information Systems Engineering*, Osaka University (1997).

[9] 呉乾禮, 塚本昌彦, 谷口伸一, 西尾章治郎: “データベースからの知識獲得を用いたデータベース圧縮について,” 人工知能学会 知識ベースシステム研究会 (第 35 回), pp. 1–6 (1996).

[10] 春本要, 塚本昌彦, 西尾章治郎: “ビューを用いたデータベース圧縮手法,” 電子情報通信学会データ工学研究会報告, DE97-13, pp. 45–50 (July 1997).

[11] 小西孝重, 相坂一樹, 春本要, 西尾章治郎: “関係データベースのための圧縮機構と問合せ処理機構の実現,” 電子情報通信学会第 9 回データ工学ワークショップ (DEWS'98) 論文集 (CD-ROM) (Mar. 1998).